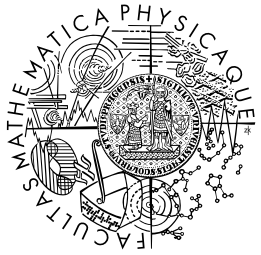


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

**DIPLOMOVÁ PRÁCE**

Bc. Michaela Štolová

# Brooom – závodní hra na košťatech v Unity

Katedra distribuovaných a spolehlivých systémů

Vedoucí diplomové práce: Mgr. Pavel Ježek, Ph.D.

Studijní program: Informatika – Vizuální výpočty a vývoj počítačových her

Praha 2025

Prohlašuji, že jsem tuto diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne .....  .....

Podpis autora

Tímto bych ráda z celého srdce poděkovala svému vedoucímu, Mgr. Pavlu Ježkovi, Ph.D., za to, že byl po celou dobu neutuchajícím zdrojem cenných rad, ochoty a nadšení. Bez jeho kompetentního vedení a konstruktivní kritiky by práce nikdy nemohla vzniknout. Velký dík patří také všem playtesterům, kteří obětovali svůj čas a poskytli velmi přínosnou zpětnou vazbu.

—

Dále bych chtěla poděkovat mamince, Lucince a Pavlíkovi za shovívavost, když jsem na ně neměla moc času. Pejsátkům Michellce, Airince, Cassience a Villience za stoprocentně účinné zlepšování nálady, ačkoliv v případě Michellky už bohužel nikdy víc (moc mi chybíš, kočičko!).

<3

A nakonec mé nejúžasnější babičce, která se vždy rozdala pro druhé a záleželo jí jen na tom, aby se ostatní kolem ní měli dobře. I mě vždy maximálně podporovala, a to až do poslední chvíle. Babi, mám Tě moc ráda a chybíš mi. Tohle je pro Tebe, děkuji za vše!

<3

Název práce: Brooom – závodní hra na košťatech v Unity

Autor: Bc. Michaela Štolová

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí diplomové práce: Mgr. Pavel Ježek, Ph.D., Katedra distribuovaných a spolehlivých systémů

Abstrakt: Tato práce představuje návrh a implementaci demo verze 3D závodní hry nazvané Brooom. Hráč v ní na koštěti prolétává obručemi, sbírá bonusy, sesílá kouzla a snaží se porazit své soupeře. Vývoj začal sepsáním Game Design Documentu (GDD), po němž následovala implementace klíčových herních mechanik pomocí herního enginu Unity. Tyto mechaniky zahrnují například procedurální generování tratí a umělou inteligenci řídící chování protivníků. Obtížnost hry se dynamicky přizpůsobuje schopnostem hráče. Součástí je také tutoriál, který hráče seznamuje s ovládáním a zásadními herními prvky. Po dokončení dostatečně obsáhlé verze jsme provedli řadu experimentů zaměřených na srozumitelnost ikonek, herní zážitek a také odladění různých parametrů. Tím jsme získali cennou zpětnou vazbu, kterou jsme částečně zapracovali. Výsledkem je finální demo verze hry, jež tvoří hlavní výstup této práce.

Klíčová slova: Unity engine, procedurální generování, závodění na košťatech, sesílání kouzel, rubber banding

Title: Brooom – Broom Racing Game in Unity

Author: Bc. Michaela Štolová

Department: Department of Distributed and Dependable Systems

Supervisor: Mgr. Pavel Ježek, Ph.D., Department of Distributed and Dependable Systems

Abstract: This thesis presents the design and implementation of a demo version of a 3D broom racing game titled Brooom. In this game, players fly through hoops, collect bonuses, cast spells, and strive to defeat their opponents. Development began with a Game Design Document (GDD), followed by the implementation of key game mechanics using the Unity game engine. These mechanics include procedural track generation and artificial intelligence governing opponent behaviour. The game dynamically adapts its difficulty based on the player's skills. A built-in tutorial introduces players to the controls and fundamental game features. Once a sufficiently extensive version was complete, we conducted a series of experiments to evaluate the clarity of the icons, the overall game satisfaction and the balancing of various parameters. Moreover, valuable user feedback was gathered and partially integrated, resulting in the finalised demo version, which forms the primary output of this thesis.

Keywords: Unity engine, procedural generation, broom racing, spell casting, rubber banding

# Obsah

# 1 Úvod

Počítačové hry jsou velmi komplexní formou umění, která v sobě sdružuje vizuální, auditivní i narativní složky. Jejich efekty na hráče pak mohou být velmi různorodé, od her sloužících čistě pro pobavení, až k hrám, které zpracovávají vážnější témata a jejich účelem může být hráče vzdělávat. Stejně tak existuje celá řada herních žánrů, každý se svými vlastními specifiky. V této práci bychom se chtěli zaměřit na závodní hry zasazené do kontextu magického světa. Dle našich znalostí totiž zatím neexistuje hra, která by kombinovala všechny následující herní prvky:

- Závodění na košťatech.
- Režim kariéry, tj. jednotlivé závody na sebe navazují, nejedná se jen o krátké a zcela oddělené zážitky. Hráč postupuje někam dál a buduje si kariéru.
- Volný pohyb ve 3D prostoru.
- Možnost sesílat kouzla za letu.
- Procedurálně generované tratě.

V této diplomové práci bychom tedy chtěli takovou hru navrhnout a implementovat. Pojmenujeme ji *Brooom*[1], plným názvem pak *Brooom: Doleť až na vrchol!* (a v angličtině *Brooom: Race your way up!*). Bude se jednat o 3D závodní hru, ve které hráč v sérii procedurálně generovaných tratí vylepšuje svou techniku létání na koštěti a pomocí vlastního talentu a sesílání kouzel se snaží stát světovou jedničkou. V rámci tvorby práce bychom si chtěli sami vyzkoušet co nejvíce aspektů herního vývoje. Pokusili bychom se tedy omezit využití assetů třetích stran (např. modelů, generátorů terénu) a raději bychom upřednostnili jejich vlastní přípravu.

Celou práci bychom rozdělili do několika fází. Nejprve provedeme průzkum souvisejících děl. Na základě toho vytvoříme *Game Design Document* (zkráceně GDD), tedy dokument popisující kompletní návrh výsledné hry. Následně budeme návrh postupně implementovat, čímž vznikne první verze hry. S touto verzí provedeme řadu experimentů pro vyhodnocení různých aspektů hry. Nakonec zapracujeme podnětnou zpětnou vazbu, provedeme ještě několik dalších vylepšení a výslednou hru zveřejníme na platformě `itch.io`, která je jedním z běžných kanálů pro distribuci indie her.

## 1.1 Související díla

V této sekci projdeme některé podobné herní tituly. Tím získáme lepší představu o současném stavu herního trhu, která nám pomůže následně specifikovat detailní návrh naší hry. U každého titulu si popíšeme, o čem je, a uvedeme některé

[1] Jedná se o složeninu slov „broom“ (anglicky koště) a „vrooom“ (pro naznačení rychlosti), což zachycuje jak téma, tak žánr. Navíc je název dostatečně univerzální, aby jej nebylo třeba překládat.

zásadní prvky, které by nám mohly sloužit jako inspirace (ať už pozitivní, či negativní). To nám umožní následně lépe stanovit cíle práce a požadavky kladené na hru.

Podobné herní tituly pro analýzu volíme na základě zpracovávaného tématu a herních mechanik. Snažíme se tedy najít takové hry, ve kterých se létá na koštěti a nejlépe také závodí. Zařazujeme jak tituly od nezávislých tvůrců, tak ty od velkých a významných herních studií.

## Broom Race (Harha Studios)

Hra *Broom Race* [1] od Harha Studios (2018) je zdarma dostupná přes Google Play pro zařízení s operačním systémem Android. Hlavním cílem je vyhnout se na trati všem překážkám a doletět co nejrychleji do cíle. K tomu napomáhají červené kruhy po cestě (na obrázku 1.1), které hráče dočasně zrychlí, pokud jimi proletí. Nic se ovšem nestane, pokud je hráč mine. Podle výsledného času se hráči umisťují v žebříčku.


**Obrázek 1.1** Snímek obrazovky ze hry *Broom Race* od Harha Studios během probíhajícího závodu. Viditelný je červený zrychlující kruh.

Uvedeme nyní několik hlavních bodů, ze kterých pak budeme vycházet při návrhu naší hry:

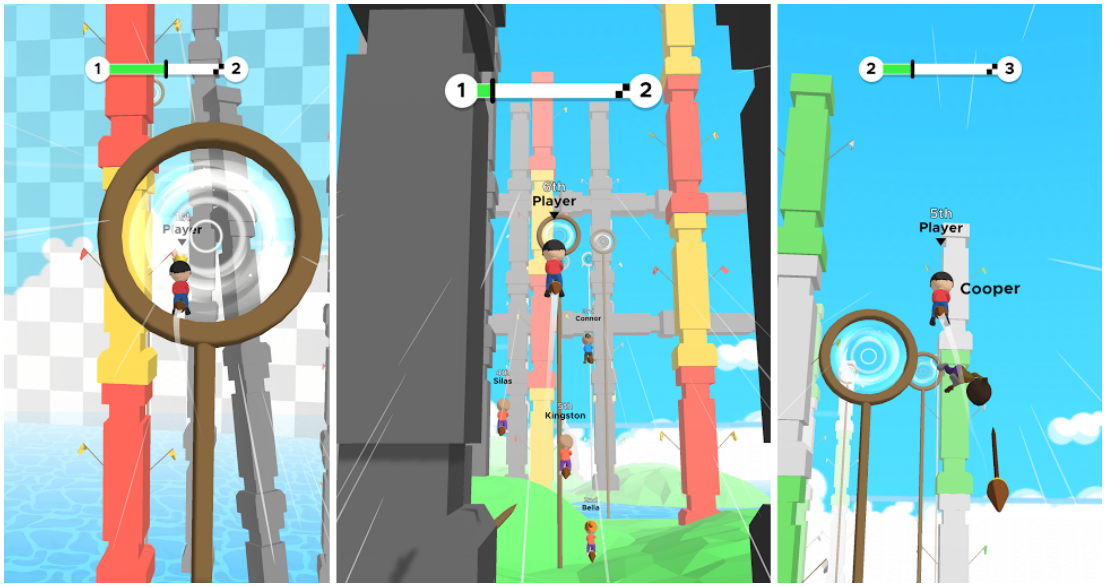
- Hráč sice letí na koštěti, ale nemá příliš velkou svobodu pohybu. Letí neustále dopředu a může se pouze posouvat do stran.
- Pokud hráč narazí do překážky, závod okamžitě končí. Z počátku je tak hráč motivovaný spíše se vyhýbat zrychlujícím kruhům, protože ve vyšší rychlosti snáz narazí.
- Ve hře je možné přepínat mezi dvěma různými pohledy. První je pohled zezadu, kdy je viditelný celý hráč i s koštětem, což však výrazně omezuje viditelnost např. v trati odehrávající se v jeskyni. Druhý je pak z pohledu hráče, kdy je ovšem část obrazovky skryta kapucí.
- Když hráč proletí zrychlujícím kruhem, kamera se mírně oddálí, což posiluje pocit vyšší rychlosti.

- Zrychlující kruhy jsou průhledné, ale mají zvýrazněné okraje a rotují.
- Nikde na obrazovce se nezobrazuje čas závodu, přestože je důležitý a na základě něj se hráč umisťuje.
- Hráč je na trati sám, nemá žádné soupeře.

## Broom Race (Dotpixel)

Další hra stejného jména, *Broom Race* [2] od Dotpixel (2021), je opět zdarma dostupná z Google Play[2] pro zařízení s operačním systémem Android.

Jelikož se nám však nepodařilo hru nainstalovat na žádné z našich zařízení, nemohli jsme ji otestovat. Hodnotíme tak pouze vizuální podobu na základě dostupných snímků obrazovky (např. na obrázku 1.2). Zdá se, že hráč společně se svými soupeři prolétává obručemi, vyhýbá se překážkám a snaží se dostat až do cíle. Naražením do soupeřů je může shodit z koštěte a tím pravděpodobně trochu zdržet.




**Obrázek 1.2** Snímky obrazovky ze hry *Broom Race* od Dotpixel, převzaté z Google Play stránky hry.

Uvedeme opět pár zásadních bodů, na kterých následně vystavíme návrh naší hry:

- Jedná se o jednoduchou low-poly grafiku s většími jednobarevnými plochami. Vše je barevné a hravé, ne realistické.
- Hráč prolétává obručemi na trati, kolem jsou překážky, kterým se musí vyhýbat.
- Je možné shazovat soupeře z koštěte.

[2]Než byla tato práce dokončena, byla již hra bohužel z obchodu odebrána. Pomocí Wayback Machine (`https://web.archive.org/`) se však dá zobrazit alespoň snapshot z 5. 7. 2021 (`https://web.archive.org/web/20210705093926/https://play.google.com/store/apps/details?id=co.dotpixel.broomrace`).

- Na obrazovce se vykreslují čáry zdůrazňující, že hráč letí, a to vysokou rychlostí.
- Nahoře se zobrazuje ukazatel postupu v trati.

## Hex Rally Racers

Ve hře *Hex Rally Racers* [3] od SMU Guildhall (2022), dostupné zdarma na Steam, hráč závodí na koštěti společně s 8 dalšími závodníky. Na začátku si zvolí jedno ze 4 košťat, které kromě letových vlastností ovlivňuje také nabídku dostupných kouzel. Během závodu pak sbírá předměty (na obrázku 1.3), podobně jako ve hře *Mario Kart* [4], díky kterým může používat kouzla pro ovlivnění sebe nebo svých soupeřů.


**Obrázek 1.3** Snímek obrazovky ze hry *Hex Rally Racers* od SMU Guildhall. Na trati před hráčem je několik sebratelných předmětů, každý je spojen s určitým kouzlem. Vlevo nahoře je aktuální pořadí hráče, vpravo nahoře je zobrazeno právě aktivní kouzlo, vpravo dole je mapa závodu.

Sepíšeme si nyní několik zásadních pozorování:

- Je možné létat pouze do stran, ne nahoru a dolů. Důležité je pouze udržet se na trati, takže koště nemá žádnou speciální funkci, podobá se to jen obyčejným závodům autíček.
- Hráč může v jednu chvíli držet jen jeden bonus, dokud se nepoužije, teprve pak lze sebrat další.
- K dispozici je velké množství kouzel se zajímavými efekty, např. dočasné zmatení soupeře, dočasné znemožnění používat magii, štít odrážející kouzla zpět.
- Na trati se nachází různé překážky, které hráče po nárazu zpomalí.

- Koště za sebou nechává stopu jako vizuální efekt.
- Během zrychlení se zdá, že se nepatrně mění pozice kamery. Přiblíží se a sníží, což posiluje dojem rychlosti.
- Pokud hráč letí špatným směrem, je na to upozorněn.
- Závody jsou spíše jednorázové a zcela oddělené zážitky.
- Měl by být dostupný také lokální režim hry více hráčů, avšak nepodařilo se nám ho nijak zprovoznit.

## Bibi Blocksberg™ – Big Broom Race 3

Hra *Bibi Blocksberg™ – Big Broom Race 3* [5] od Independent Arts Software (2018) je dostupná přes herní platformu Steam. Jelikož je však placená, nemohli jsme si ji přímo vyzkoušet, a tak vycházíme čistě z dostupných videí a snímků obrazovky (např. na obrázku 1.4). Hráč má na výběr z 8 postav a 8 různých košťat, poté závodí stylem podobným hře *Mario Kart*. Během závodu sbírá různé lektvary, kterými pak může ovlivňovat soupeře.


**Obrázek 1.4** Snímek obrazovky ze hry *Bibi Blocksberg™ – Big Broom Race 3* od Independent Arts Software, převzatý ze stránky hry na Steam. Na trati kus před hráčem jsou dva sebratelné lektvary. Vlevo nahoře je aktuální pořadí hráče, vlevo dole je zobrazen právě aktivní efekt lektvaru, vpravo dole je mapa závodu.

Opět uvedeme důležitá pozorování relevantní pro návrh naší vlastní hry:

- Hráč si může vybrat z 8 různých košťat, kdy každé má nějaké přednastavené vlastnosti.
- Ve hře je několik pevně daných tratí, které vedou velmi zajímavými prostředími, např. tunelem.

- Zdá se, že pohyb je omezený a nedá se létat nahoru a dolů. V podstatě se jen zatáčí, jako by to bylo autíčko, koště je tak čistě kosmetická záležitost.
- Podobně jako v *Mario Kart* smí mít hráč aktivní pouze jeden lektvar. Dokud ho nepoužije, nemůže sebrat další. Navíc se při sebrání náhodně zvolí, jaký efekt bude daný lektvar mít.
- Některé lektvary je možné použít směrem dopředu i dozadu. Hráč si to může zvolit během jejich použití.
- Jednotlivé závody jsou spíše oddělené zážitky. Hráč si pouze postupně odemyká nové tratě.
- K dispozici je také režim hry více hráčů (pouze lokální, ne přes síť).

## Little Witch Academia: VR Broom Racing

Hru *Little Witch Academia: VR Broom Racing* [6] od UNIVRS, Inc. (2021) jsme si bohužel nemohli zahrát, jelikož nevlastníme žádný set pro virtuální realitu (VR). Při ohodnocení tak vycházíme pouze z dostupných materiálů, jako je popis hry a recenze na Steam [7], snímky obrazovky a videa ze hry.

Příběh a postavy ve hře vychází z anime Little Witch Academia. Hráč se stává dočasným studentem akademie a současně se účastní turnaje v závodění dvojic na košťatech. Během závodů prolétává obručemi (na obrázku 1.5), vyhýbá se překážkám a sbírá předměty pro možnost seslání kouzla. Na základě výsledku závodu pak hráč získá odměnu, za kterou si může vylepšit koště. Na výběr má z několika základních košťat (ovlivňují vzhled a výchozí hodnoty vlastností koštěte), ke kterým navíc může přidávat různé krystaly (dále ovlivňující vlastnosti).

Jako u předchozích titulů opět poznamenáme několik důležitých bodů:

- Hra má nějaký příběh, který hráč postupně odkrývá dialogem s postavami. Tím se také odemykají nové tratě.
- Hráč se ve hře může pohybovat vcelku volně, navíc se zřejmě může otáčet nezávisle na směru letu. Ovládání ve VR se však nedá srovnat s hraním na PC.
- Obručemi není nutné prolétávat, ale hráč za ně má pak větší odměnu.
- K dispozici jsou pouze dva typy kouzel (zrychlení sebe sama a vyčarování pavučiny pro zpomalení soupeře). Stejně jako v *Mario Kart* se kouzlo zvolí náhodně po sebrání předmětu a dokud ho hráč nepoužije, nemůže získat další.
- Na trati jsou umístěny koule dvou barev, které ve hře nazývají překážkami. Zelenou by měl hráč proletět (zrychlí ho), ale fialové se naopak vyhnout (zpomalí ho).
- I když se hráč umístí poslední, získává odměnu.
- Ve hře je navíc tzv. Free Flight Mode, který umožňuje hru více hráčů online. Nedá se v něm ovšem závodit, pouze společně létat na pozemcích akademie a kouzlem střílet na plážové míče vznášející se ve vzduchu.


**Obrázek 1.5** Snímek obrazovky ze hry *Little Witch Academia: VR Broom Racing* od UNIVRS, Inc., převzatý ze stránky hry na Steam. Přímo uprostřed je obruč na trati, vpravo dole od ní je pak počet prolétnutých obručí v řadě, nahoře je aktuální čas v daném kole. Vpravo je vidět postava, se kterou hráč závodí ve dvojici.

## Hogwarts Legacy

Ve hře *Hogwarts Legacy* [8] od Avalanche Software a Warner Bros. Games (2023) se hráč vžívá do role kouzelníka, který postupně odkrývá svou moc a ovlivňuje svět kolem sebe. Učí se rozličná kouzla, která pak může používat v rozsáhlém prostředí. Létání na koštěti však netvoří ústřední část, hráč jej může využít pouze jako způsob dopravy nebo v něm závodit na pár tratích. Během závodu se hráč snaží co nejrychleji proletět všemi obručemi (na obrázku 1.6) až do cíle a překonat tak rekord trati.

Ačkoliv je létání odsunuté stranou, můžeme se inspirovat celou řadou prvků:

- Hráč může létat také nahoru a dolů, což mu dává mnohem větší svobodu a současně to dělá závody složitějšími.
- Za letu je možné se myší rozhlížet, ale ovlivňuje to také směr letu.
- Během letu se používají různé efekty, např. čáry a mlha na okraji obrazovky pro zdůraznění rychlosti, motion blur, mírné přiblížení kamery při zrychlení a také šumění větru.
- Pokud hráč mine některou z obručí, je penalizován 3 s navíc.
- Vlevo dole na obrazovce je minimapa a během závodu se na ní zakreslují také obruče.
- Po trati jsou rozmístěné žluté bubliny, po jejichž sebrání je hráč dočasně zrychlen.
- S hráčem na trati létá také jeden soupeř, který reprezentuje rekord trati.

- Kolem nepřístupných oblastí je ochranná bariéra, která je zpočátku neviditelná a až teprve po přiblížení nebo nárazu se zviditelní a zavlní. Zakresluje se také na minimapě.
- Hráči se postupně zpřístupňují vylepšení koštěte.
- Ve hře jsou sice dostupná kouzla, ale není možné je používat během letu na koštěti. Nelze tak pomocí kouzel jakkoliv ovlivňovat průběh závodu.
- Pokud hráč neletí na koštěti, má k dispozici celou řadu kouzel, rozdělených do několika kategorií dle účelu. Ta, která chce používat, si musí dosadit do slotů.
- Každé kouzlo má nějakou dobíjecí dobu, po kterou ho není možné seslat znovu.


**Obrázek 1.6** Snímek obrazovky ze hry *Hogwarts Legacy* od Avalanche Software a Warner Bros. Games. Přímo před hráčem je vidět obruč a o kus dál zrychlující bubliny. Nahoře je zobrazen stav závodu (aktuální čas, nejlepší čas, počet prolétnutých obručí), vlevo dole je minimapa se zvýrazněnou obručí, vpravo dole je ukazatel zrychlení.

## Harry Potter: Quidditch Champions

Hra *Harry Potter: Quidditch Champions* [9] od Unbroken Studios a Warner Bros. Games (2024) zpracovává známý kouzelnický sport *famfrpál* ze světa Harryho Pottera. Hráč si může vyzkoušet postupně všechny různé pozice v rámci týmu, tj. střelec, brankář, odrážeč, chytač. K dispozici jsou tréninkové režimy, ale také režim hry více hráčů.

Tato hra vyšla vcelku nedávno, konkrétně v září roku 2024, tedy ve chvíli, kdy už jsme měli dokončenou značnou část implementace naší vlastní hry. Navíc se nejedná přímo o závodění. I přesto však dává smysl ji zde uvést, jelikož je od významného tvůrce, létá se v ní na koštěti a může tak nabízet něco navíc.

Identifikovali jsme několik důležitých prvků a některými z nich jsme se také zpětně inspirovali:

- Po celou dobu zápasu je hráč omezen pouze na plochu hřiště. Také maximální výška je omezená.
- Létat lze do všech směrů, ale není možné se rozhlížet kolem sebe. Pohled je vždy zafixovaný ve směru pohybu.
- Pomocí pravého tlačítka myši se může hráč v roli brankáře automaticky vrátit zpět do vycentrované pozice před brankovými obručemi.
- Pomocí klávesy F se může hráč zaměřit na objekt, který je pro danou pozici zajímavý (např. v roli střelce se pohled automaticky otočí směrem k camrálu).
- Hráč je vizuálně upozorněn (na obrázku 1.7), pokud se k němu blíží potlouk, aby měl možnost se mu včas vyhnout.


**Obrázek 1.7** Snímek obrazovky ze hry *Harry Potter: Quidditch Champions* od Unbroken Studios a Warner Bros. Games. Nahoře je zobrazeno aktuální skóre, vpravo dole jsou právě dostupné akce, vlevo dole je přehled spoluhráčů a uprostřed je pak varování před blížícím se potloukem.

### 1.1.1 Shrnutí

Uvedené hry se nejvíce liší ve třech kritériích – zda se v nich závodí, jaká je volnost pohybu a jakým způsobem se používají kouzla.

Když už se v některé hře závodí na košťatech, je tam obvykle nějakým způsobem omezený pohyb. Takové hry se pak víceméně nijak neliší od závodů motokár, protože košťata jsou spíše jen kosmetická záležitost. My bychom chtěli naopak poskytnout maximální svobodu pohybu, kdy bude možné natáčet se nejen do stran, ale také nahoru a dolů. Tak by se skutečně využila myšlenka létajícího koštěte.

Obvykle také platí, že když už je někde dostatečná volnost pohybu a rozmanitost kouzel, není možné kouzlit za letu. My bychom chtěli povolit obojí a dokázat tak, že lze mít hru, ve které se létá volně ve 3D prostoru a současně kouzlí, a že to může být zábavné. Nechtěli bychom však kouzla zpřístupnit pomocí náhodných předmětů na trati. Místo toho by měl hráč sám možnost zvolit si několik kouzel, která by směl používat v následujícím závodě (s určitými omezeními jako je cena seslání a dobíjecí doba).

Závody jsou navíc často zcela separátní zážitky. V naší hře by však měly na sebe navazovat. Hráč by si budoval kariéru, postupně se zlepšoval a posouval dál. Každý závod by měl na jeho závodní kariéru vliv. Měl by to být jeden velký ucelený zážitek. Jednorázové závody pak mohou být dostupné jako vedlejší herní režim.

Tratě ve zmíněných podobných hrách jsou vždy pevně dané a ručně vytvořené. V naší hře bychom se ale chtěli pokusit je procedurálně generovat. Sice tak nebude snadné dosáhnout zajímavých prostředí (např. s průletem pod mostem), ale zato poskytneme hráči pokaždé nový zážitek, takže nikdy nepoletí stejnou trať dvakrát. Díky tomu se hráč naučí lépe zvládat samotné herní mechaniky a bude motivován se zlepšovat, nejen se naučit trať nazpaměť.

Předchozí koncepty budou představovat zásadní herní mechaniky v naší hře. Mimo ně bychom se však inspirovali také ostatními body, které jsme identifikovali u podobných her. Naše hra by se mohla popsat jako kombinace prvků z *Little Witch Academia: VR Broom Racing* (tj. volný pohyb, závodění, obruče, soupeři, bonusy, nějaký pokrok) a z *Hogwarts Legacy* (tj. volný pohyb a ovládání, obruče, bonusy, mechanika kouzel s vybavováním do slotů a s dobíjecí dobou), doplněná o procedurálně generované tratě. Měla by tak být dostatečně jedinečná a odlišná od již existujících her.

## 1.2 Nastínění hry

Na základě analýzy podobných herních titulů a jejich významných prvků nyní můžeme blíže nastínit, jak bude vypadat hra, kterou budeme vytvářet. Detailnější návrh je pak popsán v kapitole 2.

*Brooom* bude 3D závodní hra vyvinutá pomocí herního enginu Unity [10] pro PC s operačním systémem Windows. Hráč se v ní bude účastnit závodů v letu na koštěti a snažit se stát nejlepším závodníkem na světě. Jeho výkon bude ohodnocen v několika kritériích a na základě tohoto hodnocení se pak bude umisťovat v globálním žebříčku. Ve hře budou implementovány základní herní mechaniky popsané v sekci 1.1.1.

Pro vyhrazení trati, aby byla jasně určená a hráč se na ní neztratil, pak využijeme kombinaci různých řešení. Na trati budou obruče, kterými bude hráč prolétávat. Díky nim budeme také schopni snadno zjistit, jak dobře hráč zvládá ovládat koště (tj. jak často obruče míjí). Dále bude kolem trati ochranná bariéra, která hráče nasměruje, pokud by neviděl následující obruč, a současně mu znemožní letět za hranice levelu. Nakonec bude mít hráč k dispozici také minimapu se zakreslenými obručemi a bariérami.

S hráčem bude navíc na trati také několik soupeřů řízených umělou inteligencí tak, aby se přizpůsobovali schopnostem hráče a neustále se drželi v dostatečně

blízkém okolí. Tak bude mít hráč nejen lepší přehled o situaci kolem, ale také to bude posilovat napětí a soutěživost. Hráč by měl být více vtažen do hry.

Mezi závody bude hráči k dispozici obchod, ve kterém si může zakoupit různá vylepšení za odměny ze závodů. Jednou možností jsou vylepšení různých aspektů koštěte. Druhou možností jsou pak rozličná kouzla, pomocí kterých může hráč ovlivňovat průběh závodu a také interagovat se soupeři. Díky tomu by mělo být ve hře dost prostoru pro různé strategie.

Jelikož hru nebude možné dokončit během jednoho krátkého sezení, bude se stav hry ukládat perzistentně, aby hráč nepřišel o dosažený pokrok po jejím zavření.

Abychom hru zpřístupnili co nejširšímu publiku, připravíme ji rovnou na možnost lokalizace do různých jazyků.

## 1.3 Cíle práce

Hra nastíněná v předchozí sekci je poměrně rozsáhlá a její vývoj od úplného začátku až do hotového díla by byl velmi časově náročný. V této práci se tedy prozatím zaměříme pouze na demo verzi. Nebude se tak jednat o plně dokončenou hru, ale současně v ní budou již implementovány základní herní mechaniky a bude obsahovat vše potřebné pro to, aby bylo možné ji otestovat v rámci menšího experimentálního průzkumu.

Demo verze tak bude plně funkční a již publikovatelná jako hra s předběžným přístupem. Bude však postrádat některé dodatečné funkce a nebude obsahovat tak rozmanité možnosti obsahu a pokročilé efekty (např. vizuální efekty, animace, adaptivní audio). Hra bude nicméně implementována tak, aby byla snadno rozšiřitelná.

### Požadavky

Nyní si uvedeme některé základní požadavky, které by měla demo verze hry splňovat. Pro přehlednost je rozdělíme do několika skupin. Detailnější popis návrhu hry v kapitole 2 pak bude představovat již specifičtější naplnění těchto požadavků.

#### Obecné

**P1** Hru vytvoříme pomocí herního enginu Unity.

**P2** Hra poběží na cílové platformě, tj. PC se systémem Windows.

**P3** Stav hry se bude perzistentně ukládat mezi sezeními.

**P4** Hra bude lokalizována do češtiny a angličtiny, přičemž bude umožňovat snadnou podporu dalších jazyků.

**P5** Ve hře bude základní audio jako je hudba, zvuky prostředí a zvukové efekty. Mělo by přitom odpovídat žánru a upevňovat zážitek ze hry, nenarušovat ho.

**P6** Uživatelské rozhraní bude přehledné a intuitivní a bude se přizpůsobovat různým velikostem zobrazení.

**P7** Minimalizujeme využití již hotových řešení třetích stran a pokusíme se co nejvíce obsahu vytvořit sami.

### Gameplay

**P8** Hráč bude ve hře závodit na koštěti.

**P9** Jednotlivé závody na sebe budou navazovat, hráč si bude postupně budovat kariéru a vylepšovat schopnosti.

**P10** Za dobré umístění v závodě hráč dostane finanční odměnu, za kterou si pak může v obchodě vybírat ze sortimentu vylepšení.

**P11** Výkon hráče v závodech bude ohodnocen v několika kritériích. Na základě hodnocení se pak bude umisťovat v globálním žebříčku závodníků.

**P12** Hra skončí, jakmile se hráč stane nejlepším závodníkem na světě.

**P13** Hra se bude obtížností přizpůsobovat hráči.

### Létání

**P14** Při létání na koštěti bude mít hráč možnost volného pohybu ve 3D prostoru (tj. bude moci letět dopředu, brzdit, zatáčet do stran a naklánět se nahoru a dolů).

**P15** V obchodě si hráč bude moct vylepšovat konkrétní aspekty koštěte.

### Trať

**P16** Součástí závodní trati budou obruče, kterými budou závodníci prolétávat. Dále bude vyhrazená ochrannou bariérou, skrz kterou nebude možné proletět.

**P17** Na obrazovce bude mít hráč k dispozici minimapu zakreslující obruče, cíl, ochranné bariéry a závodníky.

**P18** Trať včetně okolního prostředí bude procedurálně generovaná.

**P19** Obtížnost tratí se bude přizpůsobovat schopnostem hráče v souladu s **P13** (dle ohodnocení jeho výkonu z **P11**).

**P20** Na trati budou také sebratelné bonusy s rozličnými pozitivními efekty.

### Soupeři

**P21** S hráčem bude na trati závodit také několik soupeřů řízených umělou inteligencí.

**P22** Soupeři se budou přizpůsobovat schopnostem hráče v souladu s **P13** (na základě jeho výkonu z **P11**) a budou se snažit držet v dostatečně blízkém okolí.

### Kouzlení

**P23** V obchodě si hráč bude moci zakoupit kouzla ze široké nabídky. Pomocí nich pak bude mít možnost ovlivňovat průběh závodu.

**P24** Kouzla bude možné sesílat za letu během závodu.

- **a** Pro možnost zamířit kouzla bude možné rozhlížet se kolem sebe nezávisle na směru pohybu.
- **b** Aby se mohl hráč snadno reorientovat, bude k dispozici reset pohledu do výchozí pozice jediným stisknutím tlačítka.

**P25** Hráč bude mít k dispozici pevně daný počet slotů, do kterých si dosadí již zakoupená kouzla, která bude chtít používat během závodu.

**P26** Sesílání kouzel bude omezené cenou seslání (hráči se bude doplňovat mana) a dobíjecí dobou (po seslání kouzla nebude nějakou dobu možné seslat ho znovu).

**P27** Některá kouzla bude možné sesílat na ostatní závodníky. Tím se umožní interakce hráče se soupeři.

**P28** K dispozici bude vizuální indikace, pokud se nějaké kouzlo blíží k hráči. Bude zachycovat, z jakého směru letí a jak je daleko.

### Doplňující funkce

**P29** Kromě hlavního režimu kariéry bude k dispozici také režim rychlého závodu.

**P30** Součástí bude také oddělený generátor tratí pro demonstrační a testovací účely.

## Cíle práce

V této práci si klademe hned několik cílů, které také mimo jiné naznačují, jak bude vývoj hry probíhat v čase:

**C1** Vytvořit návrh hry v podobě GDD (Game Design Document).

**C2** Vytvořit demo verzi hry, která:

- **a** splňuje vytyčené požadavky **P1**–**P30**,
- **b** je snadno rozšiřitelná (z hlediska nové funkcionality i nového obsahu).

**C3** Vytvořit pro hru tutoriál.

**C4** Provést experimenty, sbírat během nich herní analytiky a zapracovat zpětnou vazbu.

**C5** Zveřejnit demo verzi na `itch.io`.

# 2 Návrh hry

V předchozí kapitole jsme již nastínili, o čem by měla být hra, kterou budeme v této práci implementovat, a co by měla zhruba obsahovat. Nyní je třeba popsat návrh o něco detailněji. V příloze A.1 se pak nachází kompletní GDD. Ten jsme začali sepisovat dle cíle C1 již na začátku samotného vývoje, abychom měli zaznamenáno vše, co by hra měla obsahovat. Následně jsme ho upravovali s tím, jak se návrh postupně vyvíjel. Kromě finální vize zachycuje navíc také aktuální stav vývoje pomocí různobarevných značek. V této kapitole uvedeme z návrhu to nejdůležitější a vysvětlíme také některá rozhodnutí.

Jak jsme naznačili již v sekci 1.3 předchozí kapitoly, hra obsahující všechny zamýšlené funkce by byla velmi rozsáhlá, takže se v této práci zaměříme pouze na omezenou demo verzi. Kdykoliv tedy v této kapitole budeme popisovat něco, co je již nad rámec demo verze, označíme to následujícím rámečkem:

**Nad rámec demo verze**

Popis obsahu, který je zamýšlený do budoucna a nebude tak součástí demo verze z této práce.

Návrh hry popíšeme shora dolů. Začneme obecným popisem dění ve hře a postupně budeme zacházet do konkrétních detailů.

## 2.1 Gameplay

V sekcích 1.1.1 a 1.2 jsme již stručně popsali zamýšlenou hru. Pro přehlednost to však uvedeme také zde. *Brooom* je 3D závodní hra, ve které hráč prolétává obručemi, sbírá různé bonusy a sesílá kouzla, kterými může ovlivnit průběh závodu. V sérii procedurálně generovaných tratí se setkává se soupeři řízenými umělou inteligencí a vylepšuje svou techniku létání na koštěti tak, aby se nakonec stal světovou jedničkou. Obtížnost hry se přitom přizpůsobuje hráčovým schopnostem.

### Klíčové aspekty

Nyní identifikujeme několik základních pilířů, na kterých bude hra vystavěna. Jedná se o zavedený koncept v návrhu her [11], pomocí kterého popíšeme to nejdůležitější na hře, co značně ovlivňuje herní zážitek a čemu musíme věnovat dostatečnou pozornost při implementaci. Veškeré herní mechaniky a obsah ve hře pak musí vycházet z těchto pilířů a být v souladu s nimi.

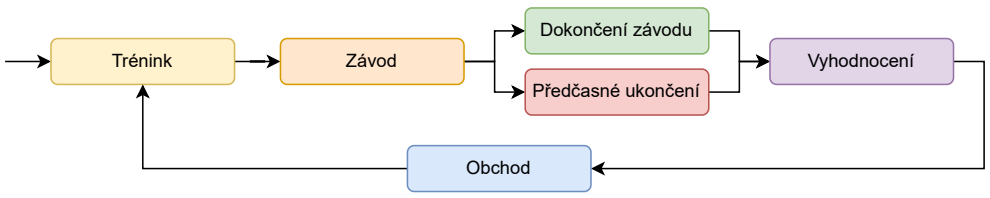
- Létání na koštěti (více v sekci 2.5) – Hráč by měl mít možnost volného pohybu ve 3D prostoru. Jelikož létáním stráví většinu času ve hře, je důležité zajistit, aby bylo plynulé a již samo o sobě poskytovalo uspokojivý pocit.
- Procedurálně generované tratě (více v sekci 2.4.3) – Každý závod se bude odehrávat na procedurálně generované trati. Tím se zajistí, že se obtížnost tratí bude přizpůsobovat hráčovým schopnostem. Navíc to poskytne dostatečnou různorodost a donutí hráče zlepšit se obecně v herních mechanikách, ne v konkrétních tratích.

- Sesílání kouzel (více v sekci 2.7) – Během závodu bude mít hráč možnost sesílat kouzla na sebe, své soupeře nebo další objekty. Kouzla by měla být smysluplná a hráč by měl mít pocit, že záleží nejen na jejich volbě, ale také na načasování seslání. Průběh závodu by se jimi měl významně ovlivňovat. Volba cíle kouzla by měla být intuitivní.

### Herní smyčka

Po spuštění hry si hráč vytvoří svou postavu, projde úvodním tutoriálem a následně vstoupí do herní smyčky (na obrázku 2.1). Ta sestává z několika hlavních kroků:

1. Spuštění závodu – Hráč spustí závod, na pozadí se procedurálně vygeneruje trať (obtížností přizpůsobená schopnostem hráče) a do ní je pak hráč umístěn.
2. Trénink – Hráč má nejprve možnost seznámit se s tratí, přičemž má neomezený počet pokusů na její prolétnutí (prozatím bez soupeřů).
3. Závod – Kdykoliv má hráč možnost přejít z fáze tréninku již do skutečného závodu. V tu chvíli se přemístí na start, kolem něj se objeví soupeři a začne odpočet.
4. Konec závodu – Závod může být ukončen jedním ze dvou způsobů:
   (a) předčasné ukončení – hráč může závod vzdát ještě před jeho dokončením,
   (b) dokončení závodu – hráč doletí do cíle a ukáže se mu výsledek závodu (čas, pořadí apod.).
5. Vyhodnocení – Hráči je ukázán globální žebříček závodníků s případnou aktualizací stavu po předchozím závodě. Navíc se ohodnotí hráčův výkon, aby se mohla hra dál přizpůsobit.
6. Obchod – Hráč si může vylepšit kostě nebo odemknout nová kouzla za mince získané z dobrého umístění v závodě.




**Obrázek 2.1** Zjednodušený nákres herní smyčky.

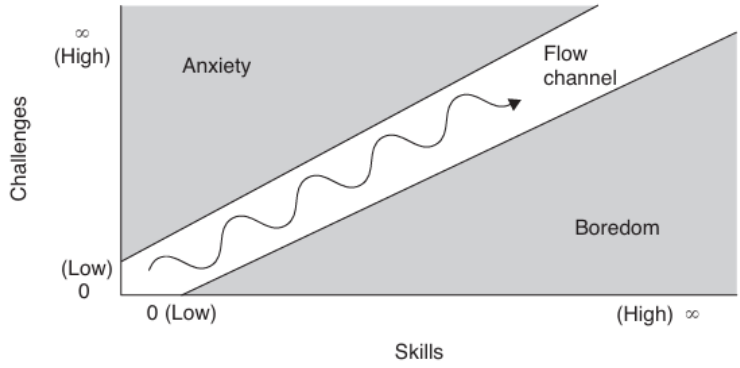
Po ukončení jednoho závodu může hráč přejít do dalšího, čímž se smyčka uzavře. Kdykoliv ji však může opustit ukončením hry.

### Herní zážitek

Nyní se zaměříme na to, jaký herní prožitek bychom chtěli hráči nabídnout a jakými prostředky se ho pokusíme dosáhnout.

Po celou dobu hraní by měl být hráč motivovaný pokračovat dál. Měl by mít pocit, že se ve hře zlepšuje a přibližuje se tak svému cíli. Měli bychom mu tedy poskytnout nějaký přehled, který by ukazoval aktuální ohodnocení jeho schopností. Obtížnost hry by se jim pak měla přizpůsobovat (dle požadavku **P13**), ale tak, aby nebyla příliš snadná a neustále představovala pro hráče výzvu.

Hráč by se měl dostat do tzv. stavu *Flow*, který M. Csikszentmihalyi poprvé popsal ve své knize *Beyond Boredom and Anxiety* [12]. Tento koncept se dá rozšířit také do oblasti návrhu videoher, jak ukazuje J. Schell ve své knize *The Art of Game Design* [13]. Popisuje zážitek, kterého chceme dosáhnout, kdy je hráč plně ponořen do hry a má pocit, že má nad hrou kontrolu. Toho se dosáhne optimálním nastavením obtížnosti hry (Challenges) přiměřeně ke schopnostem hráče (Skills), na obrázku 2.2. Pokud by byla hra příliš jednoduchá, hráč by se nudil (Boredom), a pokud by byla naopak příliš těžká, hráč by pociťoval úzkost (Anxiety). Cíle ve hře by měly být jasně definovány, hráč by měl být průběžně odměňován a získat okamžitou zpětnou vazbu na své akce.




**Obrázek 2.2** Graf zakreslující stav *Flow* jakožto optimální balanc mezi obtížností hry (Challenges) a schopnostmi hráče (Skills). Převzato z knihy *The Art of Game Design* [13] od J. Schell.

Současně by měl hráč rozumět tomu, co je ve hře jeho konečným cílem a jakým způsobem ho dosáhnout. V rámci úvodního tutoriálu bychom tak hráči představili děj hry. Kdykoliv by se pak po závodě hráči zobrazila obrazovka s přehledem, viděl by také, jaké je jeho aktuální umístění, a tedy jak blízko je svému cíli.

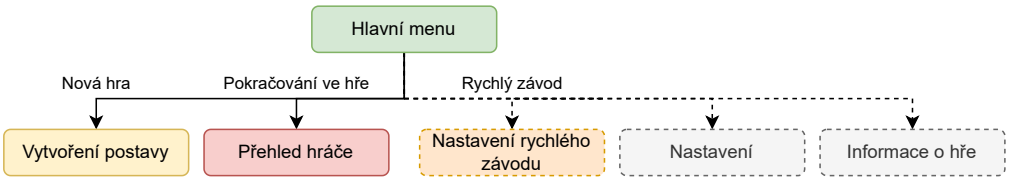
Hráč by měl být postupně seznamován s dalším obsahem ve hře tak, aby se udržela jeho pozornost a současně se necítil příliš zahlcený. Kdykoliv se objeví něco nového, hráč by s tím měl být obeznámen např. formou tutoriálu.

Hráč by měl mít také možnost provádět nějaká rozhodnutí a mít pocit, že na nich záleží. Toto bychom umožnili prostřednictvím obchodu s vylepšeními koštěte a kouzly. Hráč by tak mohl adoptovat různé strategie. Navíc by měl k dispozici jen omezený počet slotů pro kouzla použitelná v závodě, což přidává další vrstvu rozhodování.

## 2.2 Obrazovky

Již v sekci 2.1 jsme naznačili průběh hry. Nyní se však podíváme na jednotlivé části o něco detailněji. Ve hře bude několik obrazovek, mezi kterými bude hráč moci přecházet. Projdeme si tedy nyní tři různé průchody hrou.

Když hráč spustí hru, zobrazí se mu hlavní menu s nabídkou možností (zjednodušeně zakresleno na obrázku 2.3). Základní možností je zahájení nové hry, které hráče navede do obrazovky pro vytvoření postavy. Pokud má však hráč již rozehranou hru, pak bude mít také možnost v ní pokračovat, čímž se přesune na obrazovku s přehledem hráče. Kromě hlavního kariérního režimu je pak k dispozici také možnost zahrát si jeden rychlý závod. V tom případě se nejprve zobrazí možnosti nastavení rychlého závodu, než se závod skutečně spustí. Nakonec si hráč bude moct z menu zobrazit nastavení hry nebo informace o hře.




**Obrázek 2.3** Zjednodušený nákres přechodů mezi obrazovkami na základě jednotlivých možností v hlavním menu. Plné šipky a rámečky reprezentují načtení nové scény (včetně načítací obrazovky), přerušované šipky a rámečky pak značí zobrazení obsahu v té samé scéně.

Popíšeme si nyní tři konkrétní průchody hrou, které dohromady pokryjí všechny obrazovky vyskytující se ve hře a možné přechody mezi nimi.

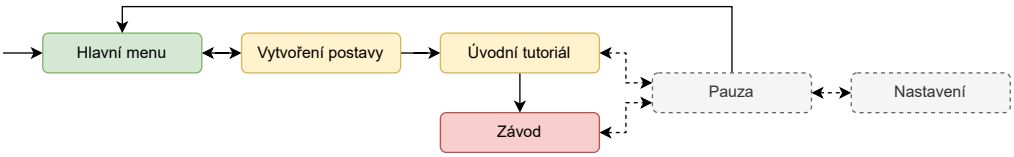
### Nová hra

Pokud hráč ještě nemá žádnou rozehranou hru nebo se rozhodne začít od začátku, přejde z hlavního menu do scény s možnostmi pro vytvoření postavy, která jej bude reprezentovat ve hře. Na výběr bude mít z několika možností vzhledů a navíc bude moct postavu pojmenovat.

Následně se ve speciální scéně spustí úvodní tutoriál, ve kterém bude hráč seznámen s premisou hry, se základním ovládáním a také s hlavními prvky trati (např. obručemi, bonusy). Poté hráč přejde do svého prvního závodu, stále v rámci tutoriálu. Po jeho dokončení už pak vstoupí do základní herní smyčky, kterou popíšeme v následující části.

Kdykoliv během tutoriálu nebo závodu bude mít hráč možnost hru zapauzovat. Ze zobrazeného menu bude moci přejít do nastavení, vrátit se zpět do hry, nebo úplně odejít do hlavního menu.

Veškeré přechody mezi obrazovkami jsou zakresleny na obrázku 2.4.




**Obrázek 2.4** Zjednodušený nákres přechodů mezi obrazovkami při spuštění nové hry. Po dokončení závodu se pak vstupuje do hlavní herní smyčky. Plné šipky a rámečky reprezentují načtení nové scény (včetně načítací obrazovky), přerušované šipky a rámečky pak značí zobrazení obsahu v té samé scéně.

### Pokračování ve hře

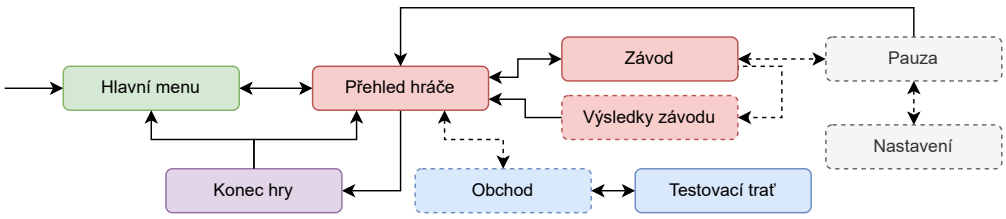
V okamžiku, kdy má hráč již rozehranou hru z dřívějška, bude v hlavním menu možnost pokračovat v této hře. Pokud ji hráč zvolí, vstoupí tím do hlavní herní smyčky (červeně na obrázku 2.5), kterou nyní popíšeme z hlediska různých obrazovek, které ji tvoří.

Nejprve se zobrazí obrazovka přehledu hráče s přehledem všech možných informací, např. počet mincí (více v sekci 2.8), aktuální ohodnocení a umístění hráče (více v sekci 2.3), vybavená kouzla (více v sekci 2.7) a získaná ocenění (více v sekci 2.10). Z obrázku 2.5 je patrné, že se jedná o centrální obrazovku, do které se bude hráč často vracet. Měla by tedy být dostatečně přehledná.

Z přehledu hráče bude moct hráč přejít do dalšího závodu. Po jeho dokončení se mu pak zobrazí výsledky závodu a možnost návratu zpět do přehledu hráče. Tím se uzavře smyčka. Kdykoliv během závodu pak bude mít hráč možnost zapauzovat hru a ze zobrazeného menu přejít do nastavení, vrátit se zpět do hry, nebo vzdát závod, čímž se vrátí zpět do přehledu hráče.

Hlavní smyčku bude možné kdykoliv opustit návratem z přehledu hráče zpět do hlavního menu. Případně bude možné z přehledu hráče zobrazit obchod s nabídkou vylepšení koštěte (více v sekci 2.5) a kouzel (více v sekci 2.7). Z obchodu bude také možnost vstoupit do testovací trati, ve které si může hráč otestovat nové vybavení ještě před závodem.

Jakmile hráč po dokončení závodu dosáhne takového ohodnocení, aby se stal nejlepším závodníkem na světě, z přehledu hráče se automaticky přejde do scény pro konec hry. Hráč se pak bude moct rozhodnout, zda chce dál pokračovat v závodění (v tom případě se vrátí do přehledu hráče), nebo chce odejít do hlavního menu (z něj však může ve hře zase pokračovat).

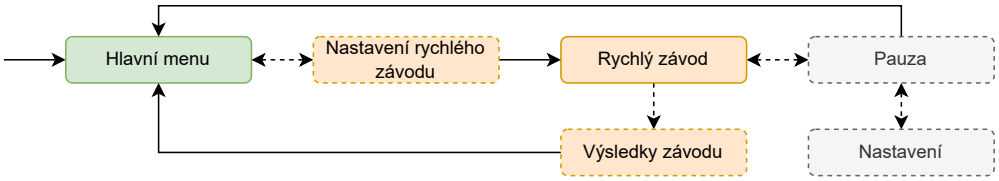



**Obrázek 2.5** Zjednodušený nákres přechodů mezi obrazovkami při pokračování v rozehrané hře. Červeně je zakreslena hlavní herní smyčka, modře pak odbočka ze smyčky a šedě pomocné obrazovky. Plné šipky a rámečky reprezentují načtení nové scény (včetně načítací obrazovky), přerušované šipky a rámečky pak značí zobrazení obsahu v té samé scéně.

### Rychlý závod

Kromě režimu kariéry bude z hlavního menu k dispozici také režim rychlého závodu. Pokud ho hráč zvolí, zobrazí se mu obrazovka s nastaveními rychlého závodu (více v sekci 2.10), např. obtížností. Z této obrazovky se bude moci vrátit zpět, nebo své volby potvrdit a přejít do rychlého závodu.

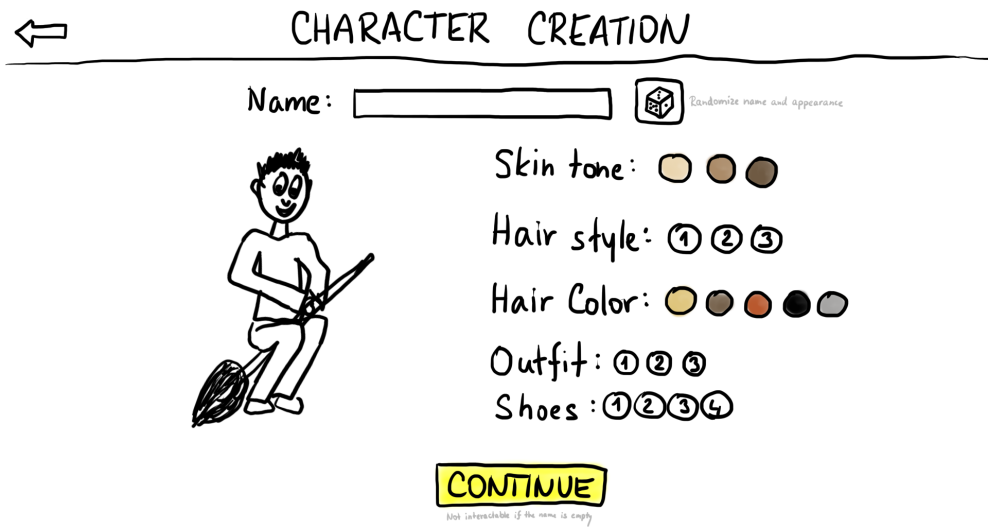
Po dokončení se hráči zobrazí výsledky závodu a následně bude moci přejít zpět do hlavního menu. Během závodu pak bude možné hru zapauzovat a ze zobrazeného menu přejít do nastavení, vrátit se do závodu, nebo závod opustit a odejít do hlavního menu. Veškeré přechody mezi obrazovkami jsou zakresleny na obrázku 2.6.




**Obrázek 2.6** Zjednodušený nákres přechodů mezi obrazovkami při spuštění rychlého závodu. Plné šipky a rámečky reprezentují načtení nové scény (včetně načítací obrazovky), přerušované šipky a rámečky pak značí zobrazení obsahu v té samé scéně.

## 2.3 Hráč

Hráč bude ve hře reprezentován postavou, která na koštěti prolétává závodními tratěmi. Na začátku hry si bude moci zvolit jméno a také vzhled z nekolika předdefinovaných variant. Na výběr bude několik možností barvy vlasů, účesů, odstínů pleti a barev oblečení (návrh rozhraní je na obrázku 2.7).




**Obrázek 2.7** Návrh grafického uživatelského rozhraní pro výběr postavy na začátku hry.

Díky této možnosti by se mohl hráč cítit s postavou ve hře více propojený. Pokud by však chtěl co nejdříve přejít k závodění, bude mít možnost jméno i vzhled inicializovat náhodně.

**Nad rámec demo verze**

V budoucnu by mohl mít hráč více možností přizpůsobení postavy. Mohl by např. kromě barvy oblečení volit také styl, přidávat různé doplňky či měnit obličej.

Veškerý pokrok ve hře, společně se zvoleným jménem a vzhledem postavy, se pak bude pro hráče automaticky perzistentně ukládat (dle požadavku **P3**), aby se mohl načíst při příštím spuštění a hráč tak mohl pokračovat z dříve uložené pozice.

### 2.3.1 Statistiky

Již v sekci 1.2 jsme naznačili, že výkon hráče bude ohodnocen v několika kritériích. Toto ohodnocení by pak bylo možné využít pro naplnění požadavku **P13**, kdy by se měla hra přizpůsobovat obtížností. Kritéria by tak měla popisovat, jak dobrý závodník hráč je. Rozlišíme celkem 5 kategorií, které nazveme *statistiky*. Každá bude nabývat hodnoty od 0 do 100, kdy vyšší je lepší. Nyní každou statistiku stručně představíme:

1. *Vytrvalost* – Bude zachycovat, jak dlouhé tratě je hráč schopný dokončit.
2. *Rychlost* – Bude popisovat, jak dobře dokáže hráč využívat maximální možnou rychlost koštěte, tedy jakou maximální rychlostí létá a po jak velkou část závodu. Pokud tedy bude hráč často brzdit a tápat, hodnota bude nižší.
3. *Obratnost* – Bude reprezentovat, jak dobře hráč zvládá náhlé změny směru, tedy jak reaguje na prudké zatáčky nebo změny výšky v trati.
4. *Přesnost* – Bude zachycovat, s jakou přesností hráč dokáže létat, tedy prolétávat obručemi na trati bez vynechání, sbírat dostupné sebratelné bonusy a vyhýbat se překážkám (prvky jsou více popsány v sekci 2.4).
5. *Magie* – Bude popisovat, jak moc hráč využívá magii, tedy kolik různých kouzel má odemčených, jak různorodá kouzla sesílá a jak efektivně využívá manu (více vysvětleno v sekci 2.7).

Po každém dokončeném závodě se z něj spočítají aktuální hodnoty statistik, které se následně zkombinují s předchozími hodnotami. Způsob výpočtu je pak více popsán v sekci 2.6. Hodnoty statistik se budou využívat na několika místech. Např. se jim bude přizpůsobovat generovaná trať (dle požadavku **P19**, více v sekci 2.4.3) a také chování soupeřů (dle požadavku **P22**, více v sekci 2.3.2).

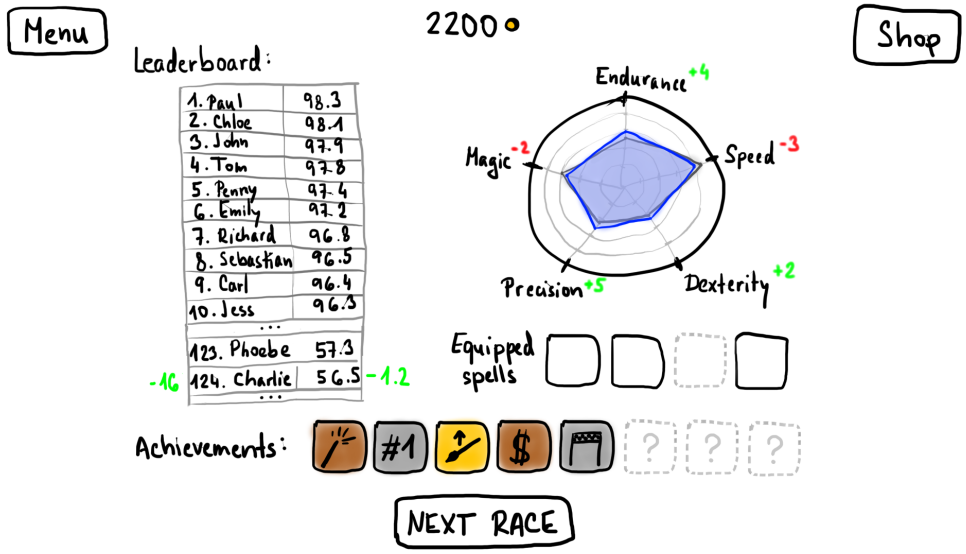
#### Globální žebříček závodníků

Hráč se na základě svých statistik bude umisťovat v globálním žebříčku závodníků (dle požadavku **P11**), který bude sloužit jako ukazatel toho, jak blízko je hráč k cíli hry (dle požadavku **P12**). Ze všech statistik se spočítá vážený průměr

a dle výsledné hodnoty se určí pořadí. Především bude nutné zajistit, že je cíl hry dosažitelný. První závodník v žebříčku by tak měl mít takový průměr, aby jej mohl hráč s dostatkem času, úsilí a schopností překonat. Nemělo by to záviset jen na štěstí.

Váha jednotlivých statistik bude určena jejich důležitostí a objektivitou výpočtu (více v sekci 2.6). *Přesnost* a *rychlost* budou mít váhu 3, protože je bude možné spočítat vcelku objektivně a budou dávat dobrou představu o tom, jak dobře hráč zvládá létat. *Vytrvalost* bude mít váhu 2, protože o skutečné zdatnosti hráče toho nebude moc říkat, ale současně může přirozeně měnit výsledné skóre dle toho, jak daleko ve hře hráč je. *Magie* bude mít také váhu 2, protože kouzla by měla být významnou součástí ohodnocení, ale ne důležitější než obecně schopnost létat. Nakonec *obratnost* bude mít nejnižší váhu 1, protože vzhledem ke způsobu výpočtu nemusí tolik odpovídat skutečné obratnosti hráče.

Vždy mezi závody se hráč dostane do obrazovky s přehledem všech důležitých informací (na obrázku 2.8). Uvidí tam své aktuální hodnoty statistik zakreslené v grafu a umístění v žebříčku, právě zvolená kouzla (viz sekce 2.7) a získaná ocenění (více v sekci 2.10). Z této obrazovky bude navíc možné přejít do obchodu (popsán v sekci 2.8).




**Obrázek 2.8** Návrh grafického uživatelského rozhraní obrazovky s přehledem hráče, která se bude zobrazovat mezi závody.

### 2.3.2 Soupeři

Jak bylo řečeno v požadavku P21, s hráčem budou závodit také soupeři řízení umělou inteligencí. Dle požadavku P22 by se navíc měli přizpůsobovat schopnostem hráče. Závody s nimi by měly být pro hráče zábavou, ale také dostatečnou výzvou. Po celou dobu závodu by se mělo něco dít, soupeři by se neměli od hráče příliš vzdálit, neměli by být ani příliš dobří, ani příliš špatní. Hráč by měl mít pocit, že se během závodu opravdu musí snažit, aby skončil na prvním místě, ale současně, že je první místo dosažitelné.

Chování soupeřů by tedy mohlo vycházet z hráčových statistik (v sekci 2.3.1), ale navíc se přizpůsobovat aktuální situaci v závodě a snažit se soupeře udržovat

v okolí hráče, aby se tím zvýšilo napětí. Měli by se chovat podobně jako hráč, tj. přesvědčivě prolétávat tratí, využívat její prvky (popsané v sekci 2.4.1) a sesílat kouzla (více v sekci 2.7). Navíc ale na základě svých přiřazených *statistik* budou dělat různé chyby (např. minou obruč, neseberou bonus, poletí pomaleji).

**Nad rámec demo verze**

Soupeři by měli být schopní přizpůsobovat se měnícím podmínkám na trati. Např. pokud někdo kouzlem změní prostředí, soupeři by to měli brát v úvahu.

Celkem bude na trati 5 soupeřů, přičemž by mělo zhruba platit, že 2 budou o něco lepší než hráč, 1 bude srovnatelný, 1 bude o trochu horší a 1 pak ještě horší. Tím by se zajistilo, že by měl hráč šanci zvítězit, avšak ne příliš snadno, a současně by jen s malou pravděpodobností skončil poslední. Soupeři budou mít náhodně zvolené všechny vlastnosti, jako je jméno, vzhled, vylepšení koštěte (popsaná v sekci 2.5) a vybavená kouzla (více popsaná v sekci 2.7). Je však třeba zajistit co největší férovost, aby neměli oproti hráči přílišnou výhodu.

- V případě vylepšení koštěte se tedy zajistí, že jich budou mít vždy odemčených stejný počet jako hráč ($\pm 1$), ale úrovně se rozloží náhodně mezi různé aspekty. Současně se však zajistí, že úroveň vylepšení stoupání bude vždy minimálně taková, jakou ma hráč, aby mohli soupeři vystoupat nad terén ve všech dostupných oblastech (viz sekce 2.4.2).

- S vybavenými kouzly je to ovšem složitější. Ideálně by měla být zhruba stejné hodnoty, jako má hráč. Současně by mohlo být dobré povolit i kouzla, která hráč nemá odemčená, protože pak by měl možnost pozorovat jejich efekt a třeba se rozhodnout zakoupit si je jako další. Na druhou stranu by ale mohl být hráč zahlcený spoustou neznámých efektů. Povolíme tedy pouze kouzla, která má hráč již zakoupená, a přidáme do výběru jen jedno další (aby hráč nebyl zahlcen, ale současně měl možnost vidět i něco dalšího). Toto kouzlo by se však vybíralo deterministicky, aby bylo pořád stejné, dokud ho hráč neodemkne, a neměnilo se s každým závodem.

## 2.4 Level

Každý závod se bude odehrávat v *levelu*, který bude tvořit trať umístěná v nějakém prostředí. Aby bylo možné levely procedurálně generovat (více rozepsáno v sekci 2.4.3 dle požadavku **P18**), bude třeba je sestavovat z prvků z předem dané množiny. Bude však třeba zařídit dostatečnou různorodost, aby levely nepůsobily na hráče příliš monotónně. Nyní si tedy určíme přesněji, jak by to mělo vypadat.

### 2.4.1 Trať

Tratě ve hře budou představovat nějakou cestu mezi startem a cílem, nebude se tedy jednat o okruhy. Měly by ale být dostatečně přehledné, aby hráč vždy věděl, kterým směrem má letět dál. Jelikož se však pohybuje ve vzduchu na koštěti, bude třeba využít různé prvky pro vymezení trati (zmíněné také v požadavku **P16**).

Již v sekci 1.1, věnující se podobným herním titulům, bylo patrné, že často využívaným konceptem jsou obruče na trati. Ty soustředí závodníky do konkrétního bodu a díky tomu je mohou navádět správným směrem. Obecně se však dají zvažovat dva způsoby využití:

1. Obručemi by bylo třeba prolétávat a jejich minutí by se trestalo. Tak by obruče jasně udávaly trať, bylo by díky nim snazší se zorientovat. V naší hře by však mělo být také kouzlení (dle požadavku **P23**). Kdyby ale hráč musel věnovat přílišnou pozornost obručím, nemohl by se dobře soustředit ještě na sesílání kouzel.

2. Obruče by mohly být jen volitelné a za jejich prolétnutí by byl závodník naopak odměněn. Obruče by pak nemusely být nutně podél hlavní trajektorie trati, ale odměna by měla hráče motivovat je vyhledávat a odchýlit se z ideální trasy. Hráč by se tak mohl soustředit na kouzlení, když by zrovna chtěl, a obručemi prolétávat jen ve volných chvílích. Pak by ale mohlo být těžší se na trati orientovat a bylo by třeba doplnit nějaké další dobře viditelné prvky vyhrazující trať.

Nakonec jsme se rozhodli pro první variantu (druhou můžeme zvažovat do budoucna). Současně však zajistíme, že obruče budou dostatečně daleko od sebe a kouzla snadno použitelná, aby hráč stíhal obojí. Trať by tedy byla vymezená následujícími prvky:

- Podél trajektorie trati budou v pravidelných intervalech rozmístěné *obruče*. Těmi budou muset závodníci prolétávat. Za každou minutou se jim pak k času dokončení závodu přičte určitá penalizace. Ta by měla být nastavená tak, aby se nevyplatilo obruče míjet a letět vždy nejkratší možnou cestou kolem nich.

- Některé z obručí budou tzv. *kontrolní body*, které budou větší a vizuálně odlišené. Jejich prolétnutí bude povinné, takže pokud závodník některý mine, bude se k němu muset vrátit zpět. Jedná se tak o koncept checkpointů běžný ze závodních her (např. *Trackmania* [14]).

- Za poslední obručí na trati bude *cílová čára* jako poloprůhledná zabarvená plocha s pohybujícími se pruhy, která bude tím výraznější, čím víc se k ní hráč přiblíží. Jakmile jí hráč proletí, dokončí závod.

- V určité vzdálenosti kolem trajektorie trati bude *ochranná bariéra*, skrz kterou nebude možné proletět. Bude viditelná pouze částečně, ale po kolizi závodníka s ní se zvýrazní. Trať přitom nebude nutné omezovat shora, protože maximální stoupání koštěte bude omezeno úrovní vylepšení (viz sekce 2.5).

**Nad rámec demo verze**

Trať by mohla být vyhrazená také okolním prostředím, např. by mohla vést kaňonem nebo dokonce tunelem pod zemí. Také by mohly být kolem rozmístěné další prvky navádějící hráče, např. řady praporků na vysokých stožárech.

Kromě toho by na trati byly rozmístěné také *bonusy* (dle požadavku **P20**), které po sebrání udělí závodníkovi nějaký pozitivní efekt. Budou mít podobu poloprůhledné koule se zabarvenými okraji. Pokud bude bonus některým závodníkem sebrán, zmizí a až po chvíli se objeví znovu. Rozlišíme celkem 4 druhy bonusů, které se budou lišit barvou a efektem:

- *Doplnění many* – Po sebrání se závodníkovi doplní určité množství many, kterou může používat na sesílání kouzel (více v sekci 2.7). Těchto bonusů bude na trati nejvíc, ale budou se objevovat až ve chvíli, kdy bude mít hráč odemčené alespoň jedno kouzlo.
- *Zrychlení* – Po sebrání se dočasně zvýší maximální rychlost závodníka. Tyto bonusy se budou v trati vyskytovat vcelku často.
- *Dobití kouzel* – Po sebrání bonusu se závodníkovi okamžitě dobijí všechna kouzla (více v sekci 2.7). Těchto bonusů bude na trati méně a budou se objevovat až ve chvíli, kdy bude mít hráč odemčené alespoň jedno kouzlo.
- *Zviditelnění trajektorie* – Po sebrání se zobrazí linka zvýrazňující trajektorii k několika dalším významným bodům (tj. obručím nebo kontrolním bodům). Hráči tak výrazně usnadní navigaci, ale v trati se bude vyskytovat jen zřídka.

Tímto způsobem by však mohlo dojít k zásadnímu problému, kdy bude mít závodník na prvním místě obrovskou výhodu, obzvlášť v případě zrychlujícího bonusu. Stačí se dostat ke zrychlujícímu bonusu jako první a pak už má téměř jistotu, že bude jako první i u každého dalšího. Promýšleli jsme několik možností, jak tento problém redukovat, či úplně odstranit.

1. Bonusy by se mohly umisťovat někam stranou, aby bylo těžší je sebrat. Pak by se ale nemuselo zrychlení tolik vyplatit, hráči by nemuseli být motivováni je sbírat a bonusy by se staly zbytečnými.
2. Možnost sbírat bonusy by se mohla omezit pro prvního závodníka. Např. by se před ním bonusy ani neobjevovaly, nebo by je nemohl sbírat. To by ovšem vytvářelo neférové podmínky.
3. Po sebrání bonusu by se mohla začít odpočítávat doba, po kterou nelze znovu sebrat ten samý typ. Doba by však musela být dostatečně dlouhá a ani tak by nebylo zajištěno, že závodník o svou výhodu bude moci přijít. Navíc by si hráč musel udržovat přehled o tom, kdy bonusy smí sbírat, aby je využil co nejlépe.
4. Efekt bonusu (délka trvání, síla efektu) by se mohl přizpůsobovat umístění závodníka. Na prvního by měl nejmenší vliv, na posledního největší. To by ovšem opět mohlo působit neférově a efekt by se mohl zdát nepředvídatelný.
5. Zrychlující bonusy by se mohly úplně odstranit, protože způsobují až moc velkou nerovnováhu. Místo toho by se mohl zavést koncept boostu, který by se postupně nabíjel a závodníci by jej mohli spotřebovávat. Tím by se však ještě více komplikovalo ovládání hry.

Chtěli bychom dosáhnout férové hry, kdy by měli všichni závodníci stejné podmínky a hráč by nebyl trestán za to, že se mu daří. Tím se vyřadí možnosti 2 a 4. Současně bychom nechtěli hru příliš zesložiťovat, protože už samotné létání (viz sekce 2.5) a sesílání kouzel (viz sekce 2.7) bude od hráče vyžadovat velké soustředění. Neměli bychom tedy zvažovat ani možnosti 3 a 5. Možnost 1 má pak zřejmou nevýhodu zmíněnou již výše. Jakmile si však hráč vylepší koště (viz sekce 2.5) a přidají se kouzla, kterými se mohou závodníci navzájem ovlivňovat, výhoda závodníka na prvním místě se potlačí. Nemuselo by tedy být nutné situaci řešit nějakými speciálními prostředky. Navíc můžeme výhodu omezit také jednoduše tím, že zajistíme umístění skupinky několika stejných bonusů na jednom místě, aby mělo více závodníků možnost je sebrat.

**Nad rámec demo verze**

V budoucnu by byly na trati také prvky specifické pro konkrétní tematickou oblast (více popsané v sekci 2.4.2). Mohlo by se jednat např. o nějaké překážky, které by dobře zapadaly do okolního prostředí.

### 2.4.2 Tematické oblasti

Prostředí kolem trati by mělo být dostatečně různorodé a zajímavé. Měl by jej tvořit členitý terén s rozmanitými prvky prostředí.

**Nad rámec demo verze**

Prostředí by mělo být živé, tzn. že by mělo obsahovat také různé dynamické prvky, pohybující se stvoření, vegetaci ohýbající se ve větru. Mimo to by v něm měly být prvky, se kterými může hráč interagovat, např. překážky, které se mu staví do cesty a je možné do nich narazit.

Abychom zajistili co nejvyšší rozmanitost, definujeme několik *tematických oblastí*. Ty se mezi sebou budou lišit různými barevnými odstíny, parametry terénu a také prvky prostředí. Level tedy bude rozdělen na tyto oblasti, přičemž každá z nich se v něm může vyskytovat vícekrát. Současně bude platit, že se oblasti budou hráči představovat postupně a v každém levelu bude vždy maximálně jedna nová oblast, kterou hráč doposud nenavštívil. Pro odemčení oblastí pak budou platit různé podmínky – některé se odemknou s konkrétní hodnotou *vytrvalosti*, jiné s konkrétním stupněm vylepšení stoupání koštěte (více v sekci 2.5). Tak se bude nabídka oblastí přirozeně rozšiřovat s postupem ve hře.

Různé oblasti mohou ovlivňovat jak vzhled okolního prostředí, tak vlastnosti trati. Rozlišíme tedy tematické oblasti dvou typů:

- *Oblasti terénu* – Tyto oblasti budou ovlivňovat jak podobu terénu (výšku, zabarvení, prvky prostředí), tak vlastnosti trati (výšku trati dle prostředí).
- *Oblasti trati* – Tyto oblasti nebudou nijak ovlivňovat terén, ale pouze vlastnosti trati (výšku). Budou se tedy moci vyskytovat v určité části levelu společně s oblastmi terénu (např. nad/pod nimi), přičemž oblast trati pak bude mít finální vliv na vlastnosti trati (bude je přepisovat svými).

**Nad rámec demo verze**

V budoucnu budou oblasti terénu ovlivňovat také překážky na trati.

Představíme si nyní kompletní seznam zamýšlených tematických oblastí s jejich charakteristikami.

**Nad vodní plochou** Jedná se o oblast terénu, která bude přístupná již od samého začátku. Převládat bude modrá barva. Terén bude jen velmi mírně zvlněný a pod velkou vodní plochou. V prostředí se budou vyskytovat lekníny na hladině a kameny na dně. Bude možné letět také pod hladinu.

**Kouzelný les** Jedná se o oblast terénu, která se zpřístupní až po dokončení tutoriálu, který hráče seznamuje s obchodem (více v sekci 2.10). Převládat bude zelená a hnědá barva. Terén bude mírně kopcovitý, ve střední nadmořské výšce. V prostředí budou stromy, keře, rostliny, kameny a houby.

**Vyprahlá poušť** Jedná se o oblast terénu, která se zpřístupní po dosažení 1/4 maximální hodnoty *vytrvalosti*. Převládat bude žlutá a okrová barva. Terén bude jen velmi mírně zvlněný a v nízké nadmořské výšce. Prostředí bude poměrně pusté s občasnými kaktusy a pyramidami.

**Rozkvetlá louka** Jedná se o oblast terénu, která se zpřístupní po dosažení 2/4 maximální hodnoty *vytrvalosti*. Převládat bude fialová a růžová barva. Terén bude mírně zvlněný, umístěný v nízké až střední nadmořské výšce. V prostředí budou rozmístěné obrovské květiny.

**Bouřlivá oblast** Jedná se o oblast terénu, která se zpřístupní po dosažení 3/4 maximální hodnoty *vytrvalosti*. Převládat bude šedá barva. Terén bude kopcovitý a umístěný ve střední až vyšší nadmořské výšce. V prostředí budou holé stromy a keře, kameny.

**Zasněžená hora** Jedná se o oblast terénu, která se zpřístupní po zakoupení prvního stupně vylepšení stoupání koštěte. Převládat bude bílá barva. Terén bude tvořený velmi členitými horami ve velké nadmořské výšce. V prostředí budou jehličnaté stromy, kameny a sněhuláci.

**Nad mraky** Jedná se o oblast trati, která se zpřístupní po zakoupení druhého stupně vylepšení stoupání koštěte. Převládat bude světle modrá a bílá barva. Trať v této oblasti povede ve velké výšce, kolem budou mraky, skrz které bude možné proletět.

**Nad rámec demo verze**

**Záhadný tunel** Jedná se o oblast trati, která se zpřístupní po dosažení 3/4 maximální hodnoty *obratnosti*. Převládat bude černá a hnědá barva. Trať v této oblasti povede pod zem, po stěnách tunelu budou rozmístěné houby, krystaly a lucerny osvětlující prostor.

**Nad rámec demo verze**

Kromě vyjmenovaných prvků terénu a trati by byly v budoucnu přidány další. Některé z nich by představovaly překážky, do kterých je možné narazit, což závodníka zdrží.

Na úplném začátku tak bude přístupná pouze oblast *Nad vodní plochou*, která je velmi přehledná a bude tak poskytovat dobré tréninkové prostředí. Hráč by v ní měl chvíli setrvat, ale ne příliš dlouho, aby se nezačal nudit. Proto se *Kouzelný les* zpřístupní po dokončení konkrétní fáze tutoriálu. Tak je jistota, že zůstane již navždy přístupný (oproti zpřístupnění na základě *vytrvalosti*, která může i klesat, viz sekce 2.6) a neobjeví se současně s jinou novou oblastí (hráč ještě nebude mít dost mincí na vylepšení koštěte, ani dostatečnou *vytrvalost*). Jesliže bude v levelu oblast, se kterou se hráč doposud nesetkal a která je něčím specifická, hráči se o ní ještě před závodem zobrazí potřebné informace.

Do základních prvků prostředí, jako jsou např. stromy a rostliny, nebude možné narazit, závodníci jimi budou moci proletět. Hráč tak nebude až moc omezovaný, nebude muset přehnaně manévrovat a navíc se tím vylepší výkon hry. Současně ale budou prvky omezovat viditelnost, protože se za nimi budou moci schovávat obruče a bonusy. Jelikož by to pro hráče mohlo být neintuitivní, bude na to upozorněn v informacích o zpřístupnění *Kouzelného lesa*. Vysvětlí se, že jsou závodníci očarováni speciálním kouzlem.

### 2.4.3 Procedurální generování

Jak bylo řečeno v požadavku **P18**, levely ve hře budou procedurálně generované. Konkrétních způsobů implementace existuje celá řada a budou blíže rozebrané v sekci 3.3. Obecně však bude třeba generovat jak trať se všemi jejími prvky, tak prostředí kolem. Bude přitom nutné zohlednit, které oblasti terénu a trati jsou hráči dostupné (dle sekce 2.4.2) a také do jaké maximální výšky může hráč vystoupat (dle sekce 2.5). Dále bude třeba zajistit korektnost generovaných tratí, např. aby se nikde nepřekřížily.

Při generování oblastí je třeba klást důraz na to, aby byly oblasti spíše větší a hráč v nich strávil netriviální množství času. Jedině tak by měly jedinečné charakteristiky jednotlivých oblastí možnost zapůsobit. Současně by neměl být v jednom levelu velký počet různých oblastí naráz, aby tím nebyl hráč příliš zahlcený.

Dle požadavku **P19** by se navíc obtížnost tratí měla přizpůsobovat schopnostem hráče. Budeme tedy vycházet z aktuálních hodnot hráčových statistik (ze sekce 2.3.1) a na základě nich přizpůsobíme související parametry trati:

- Statistika *vytrvalosti* bude ovlivňovat celkovou délku trati určením počtu obručí. S vyšší hodnotou se tak vygenerují delší tratě. Čas potřebný k dokončení trati by se měl pohybovat mezi 1 až 5 minutami.
- Statistika *rychlosti* bude ovlivňovat vzdálenost jednotlivých obručí, aby byly s vyšší hodnotou statistiky dál od sebe. Tak bude mít hráč i při vyšší rychlosti dostatek času na změnu směru.
- Statistika *obratnosti* bude určovat maximální úhly změny směru mezi po sobě

jdoucími obručemi. Pokud tedy bude mít hráč statistiku vyšší, trať bude moci více zatáčet.

- Ze statistiky *přesnosti* se určí velikost obručí, aby s vyšší hodnotou byly obruče menší a byla tak skutečně vyžadována vyšší přesnost.
- Statistika *magie* nebude mít žádný vliv na generovanou trať. To, zda se do trati umístí bonusy pro manu a dobíjení kouzel (viz sekce 2.4.1), závisí pouze na tom, zda má hráč zakoupené alespoň jedno kouzlo.

**Nad rámec demo verze**

V budoucnu by mohly být do trati přidány také dynamické překážky, které by se snažily závodníkům aktivně stavět do cesty. Frekvence jejich zásahů by pak mohla vycházet z hráčovy statistiky *obratnosti*. Když bude hráč lepší, překážky se mu budou objevovat v cestě častěji.

## 2.5 Létání

Vůbec nejdůležitější herní mechanikou ve hře je létání na koštěti (v požadavku **P8**). Hráč jím stráví většinu času, takže je třeba jej co nejlépe odladit. Pohyb by měl být plynulý, při zatáčení by se koště nemělo otáčet okamžitě, ale nejprve pomalu a postupně zrychlovat. Po uvolnění klávesy pro změnu směru by se pak mělo koště postupně dorovnat zpět do neutrální polohy. Během zatáčení do stran se koště nakloní spolu se závodníkem. Výška, do které smí koště vystoupat, bude omezená dle aktuálního stupně vylepšení. Jakmile už hráč nemůže výš, koště se automaticky dorovná.

### Ovládání

V požadavku **P14** je popsáno, jaké možnosti pohybu by měl hráč mít, tj. letět dopředu, brzdit, zatáčet do stran a naklánět se nahoru a dolů. Jelikož je hra určená pro PC, měla by se ovládat pomocí klávesnice a myši. Je tedy třeba najít pro zmíněné akce vhodné mapování.

Provedli jsme krátký průzkum, jaké ovládání se používá v jiných hrách s mechanikou létání nebo plavání. Zdá se, že se běžně používají klávesy `WASD` pro základní pohyb a k nim `mezerník` pro stoupání. Obvykle se totiž `mezerníkem` skáče, takže je přechod z chůze na let přirozený. Různé hry pak ale používají různé klávesy pro klesání. Např. *Subnautica* [15] používá klávesu `C` pro pohyb dolů pod vodou, *World of Warcraft* [16] používá klávesu `X` pro let dolů, *Hogwarts Legacy* používá `Ctrl` pro let dolů a *Ylands* [17] používá `Shift` pro plavání směrem dolů.

Mapování pohybu směrem dolů na `Ctrl` nebo `Shift` má tu výhodu, že nechává volný prostor pro využití kláves `X/C/V`, které jsou pořád snadno dostupné. Navíc nám připadá `Shift` pohodlnější pro ruku než `Ctrl`. Nakonec tedy zvolíme ovládání pomocí kláves `WASD` a k nim přidáme `mezerník` a `Shift` (viz tabulka 2.1). Hráč si ho však kdykoliv může změnit v nastavení (více v sekci 2.10).

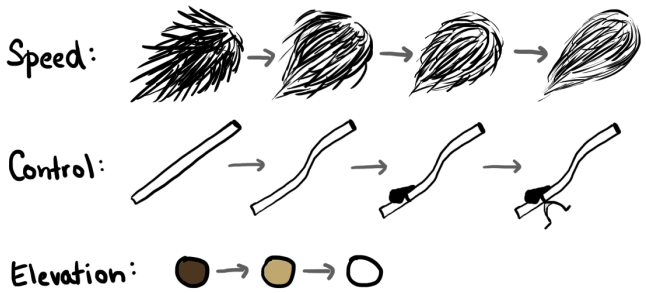
| Akce | Klávesa |
|---|---|
| Pohyb vpřed | `W` |
| Zpomalení | `S` |
| Zatočení doleva/doprava | `A/D` |
| Stoupání | `Mezerník` |
| Klesání | `Levý Shift` |

**Tabulka 2.1** Zvolené ovládání pro let na koštěti.

## Koště a vylepšení

Hráč bude létat na koštěti a v průběhu hry si jej bude moci v souladu s požadavkem P15 vylepšovat v obchodě (více v sekci 2.8). Každé vylepšení bude dostupné v několika stupních a bude postupně více upravovat vlastnosti létání. Koště se bude skládat z několika částí, které bude možné nahrazovat. Zakoupením vylepšení se pak bude měnit také vzhled koštěte (nákres je na obrázku 2.9). K dispozici budou následující vylepšení:

- *Rychlost* (3 stupně vylepšení) – Po zakoupení bude moci koště dosahovat postupně vyšších maximálních rychlostí. Vylepšení bude ovlivňovat vzhled proutí, které bude postupně uhlazenější a aerodynamičtější.
- *Ovladatelnost* (3 stupně vylepšení) – Po zakoupení bude koště rychleji reagovat na změny směru. Toto vylepšení nejprve změní tvar rukojeti na ergonomičtěji zahnutou a pak postupně přidá sedátko a opěrky pro nohy.
- *Stoupání* (2 stupně vylepšení) – Po zakoupení bude moci koště vystoupat do vyšších výšek. Díky tomu se zpřístupní nové tematické oblasti (detailně popsané v sekci 2.4.2). Každý stupeň vylepšení přitom změní materiál rukojeti.




**Obrázek 2.9** Schematické zakreslení vlivu jednotlivých vylepšení na části koštěte.

Bude však třeba dobře odladit rozdíly v rychlosti jednotlivých úrovní vylepšení. Soupeři totiž mohou mít vyšší úroveň než hráč (viz sekce 2.3.2), ale neměli by tím získat obrovskou výhodu. Rozdíly by tedy neměly být příliš velké, i za cenu toho, že pak nebudou tak znatelné.

**Nad rámec demo verze**

Do budoucna by bylo možné přidat další vylepšení. Např. *zrychlení* (to by umožnilo, že soupeři sice mohou mít vyšší maximální rychlost, ale menší zrychlení) nebo *odolnost* (vliv negativních kouzel by se mohl redukovat).

## Kamera

K hráči bude připevněná kamera, aby se pohybovala společně s ním. Výchozí pozice bude jako pohled první osoby (tedy přímo z pohledu hráče), kdy bude vidět také kousek rukojeti koštěte. Pomocí myši pak bude možné se rozhlížet kolem (především pro možnost zamíření kouzel z požadavku **P24a**), ale nebude to mít žádný vliv na směr letu. Koště poletí vždy tím směrem, kterým je natočené. Jelikož by pro hráče mohla být tato nezávislost ze začátku těžko uchopitelná, přidáme také v souladu s požadavkem **P24b** možnost pomocí pravého tlačítka myši pohled rychle resetovat do výchozí pozice (tj. dopředu ve směru pohybu).

Pokud by byl hráč během závodu na prvním místě, neměl by před sebou žádného soupeře a měl by zbytečně omezenou nabídku kouzel, která by směl používat (více v sekci 2.7). Ve hře tedy bude také možnost přepnutí na pohled dozadu. Díky němu bude moct hráč mířit za sebe a současně bude mít lepší přehled o dění kolem. V tomto pohledu však nebude možné se rozhlížet.

**Nad rámec demo verze**

Pohled dozadu by mohl fungovat jedním ze dvou způsobů, přičemž každý má své výhody i nevýhody. V demo verzi bude implementován ten první, ale do budoucna bychom chtěli přidat do nastavení možnost volby.

1. Bylo by nutné stisknout odpovídající klávesu jednou pro zapnutí a pak podruhé pro vypnutí. Hráč by tak měl volnější ruce, kdyby potřeboval přesněji zamířit kouzlo.

2. Pohled dozadu by byl aktivní, dokud by byla stisknuta odpovídající klávesa. Pokud by hráči nezáleželo na volbě konkrétního cíle, pak by bylo seslání takto rychlejší.

**Nad rámec demo verze**

Dále bychom chtěli do budoucna umožnit také přepínání mezi různými pohledy, které je v závodních hrách běžné.

- Pohled zepředu z vrcholu rukojeti koštěte (ekvivalent pohledu zepředu auta) by však v naší hře nedával příliš smysl. Hráč by pak neviděl celou řadu vizuálních indikací (např. sesílání kouzla), ale především by bylo těžké určit, kterým směrem letí.

- Pohled zezadu zpoza hráče, tedy ze třetí osoby, by mohl být přínosný. Hráč by viděl, kterým směrem je natočené koště, a mohl by sledovat dění kolem. Je však třeba vyřešit správné rozhlížení se, aby se kamera točila kolem hráče (ne na místě) a nenakláněla se v zatáčkách. Také by se muselo zajistit vhodné zaměřování kouzel.

Nakonec bychom kameře chtěli přidat také efekt, který jsme pozorovali u podobných her v sekci 1.1, např. *Hex Rally Racers* a *Hogwarts Legacy*. Kdykoliv bude na hráče působit zrychlení (z bonusu nebo kouzla), kamera se mírně přiblíží pro umocnění tohoto efektu.

## 2.6 Závodění

Jak již bylo řečeno a uvedeno také v požadavku **P8**, hráč bude ve hře závodit na koštěti. Jednotlivé závody pak na sebe budou navazovat (dle požadavku **P9**). Nyní je třeba také určit, jakým způsobem bude probíhat jeden samostatný závod.

### Trénink

Jelikož budou tratě procedurálně generované (více v sekci 2.4.3), měli bychom dát hráči možnost se s nimi seznámit ještě před závodem. Jakmile tedy hráč vstoupí do nového závodu, začne nejprve ve fázi tréninku, během které může tratí libovolně prolétávat prozatím bez soupeřů. Neuvidí však žádné interaktivní prvky (např. bonusy), na trati tak budou pouze obruče a kontrolní body. Kdykoliv během tréninku pak bude mít možnost přesunu zpět na start jediným stisknutím klávesy.

Abychom mohli rozeznat, kdy je hráč připraven závodit, musíme zavést nějaký způsob signalizace. Není možné to řešit tlačítkem na obrazovce, protože kurzor bude skrytý pro snazší ovládání. Současně bychom nechtěli vyhradit jednu klávesu speciálně pro tuto akci nebo riskovat, že ji hráč stiskne omylem. Poblíž startu tedy umístíme speciálně označenou zónu, do které hráč vletí. Pokud v ní setrvá dostatečně dlouho, přejde do fáze závodu. Tak není nutné pro signalizaci nijak přerušovat létání.

Pokud by však hráč nechtěl před závodem trénovat, pak by zbytečně ztrácel čas tím, že by musel nejprve doletět do zóny a počkat v ní. Přidáme tedy do nastavení možnost povolit přeskočení tréninku. V takovém případě se po vstupu do závodu okamžitě zahájí přímo fáze závodu.

### Závod

Na začátku závodu je hráč přesunut na start, kolem něj se objeví soupeři a zahájí se úvodní cutscene, ve které bude kamera přecházet mezi nastoupenými závodníky. Poté začne odpočet. Během závodu už má hráč k dispozici všechny akce, tj. vidí bonusy, které může sbírat, může sesílat kouzla.

Úkolem hráče je pak co nejrychleji doletět do cíle, který bude určený speciální zónou na konci trati. Přitom by měl proletět všemi kontrolními body a co nejvíce obručemi. Jelikož je průchod tratí lineární, v každém okamžiku je zřejmé, kterou obručí by měl hráč proletět jako další. Taková obruč bude tedy vždy zvýrazněná pomocí velké šipky nad ní. To by mělo hráči pomoci v navigaci na trati.

**Nad rámec demo verze**

Někdy se může stát, že jsou obruče skryté mezi prvky prostředí. Do budoucna by se tedy mohla indikace následující obruče vylepšit. Na obrazovce by se hráči neustále zobrazovaly informace o tom, jakým směrem a hlavně

v jaké výšce relativně k němu je. Bylo by však třeba řádně promyslet, kam takovou informaci umístit a jak přesně by měla vypadat.

Pokud hráč mine kontrolní bod, na obrazovce se zobrazí upozornění. Hráč pak nepostoupí dál, dokud daným kontrolním bodem neproletí. Pokud však mine jen obyčejnou obruč, obrazovka krátce zčervená, hráči se přičte časová penalizace a zvýrazní se následující obruč.

**Nad rámec demo verze**

Po prolétnutí kontrolním bodem by se mohl zobrazit čas průletu pro hráče a nejlepšího závodníka. Podobný koncept je v závodních hrách běžný, např. *Trackmania*.

V průběhu závodu budou na obrazovce všechny důležité informace. Měly by být umístěné tak, aby byly související části co nejvíce u sebe. Současně je ale potřeba zobrazit poměrně hodně informací, takže je umístíme dostatečně po stranách, aby měl hráč hlavně dobrý výhled přímo před sebe. V této hře naštěstí není třeba, aby měl hráč neustále absolutní přehled o všem, takže nevadí, že bude muset přecházet pohledem mezi několika místy. Na obrázku 2.10 je návrh takového grafického uživatelského rozhraní. Jednotlivé části se přitom budou zobrazovat pouze tehdy, pokud budou třeba (např. část s kouzly se nezobrazí, pokud nemá hráč ani jedno).

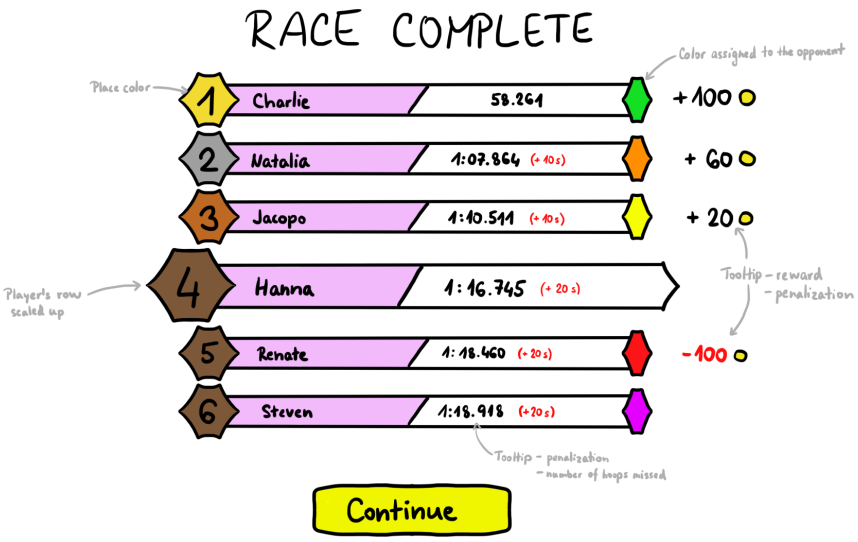
- Nahoře bude aktuální čas závodu s případnou penalizací.
- Vlevo dole bude část zaměřující se na pokrok hráče v závodě. Bude obsahovat počítadlo obručí a kontrolních bodů (zobrazující počet prolétnutých, počet minutých a celkový počet), minimapu (zakreslující soupeře, obruče, cílovou čáru a ochrannou bariéru dle požadavku P17) a aktuální umístění.
- Vpravo nahoře budou efekty právě působící na hráče (z bonusů či kouzel) včetně zbývajícího času.
- Vpravo bude část související s kouzly. Bude tedy zobrazovat aktuální množství many a vybavená kouzla ve slotech (včetně jejich ceny seslání a aktuálního stavu).
- Vpravo dole bude aktuální rychlost a výška.
- Přímo uprostřed se bude zobrazovat odpočet do začátku závodu.

Minimapa by mohla zachycovat celou trať, nebo jen menší úsek kolem hráče. Zobrazení celé trati je vcelku běžné v podobných hrách, např. *Mario Kart* nebo *Hex Rally Racers*. Dává to hráči jasnou představu o tvaru trati i o tom, kde přesně se na ní nachází soupeři. V naší hře však budou tratě procedurálně generované (viz sekce 2.4.3), takže by se musela mapa nakreslit za běhu. Současně je důležité, že na trati budou obruče, kterými by měl hráč prolétávat. Více by se tedy hodilo zobrazit jen okolí hráče, které by se navíc otáčelo dle hráčovy orientace. Díky tomu by se mohla přesně zakreslit pozice důležitých bodů (tj. obručí) a minimapa by tak hráči pomáhala především v navigaci k dalšímu takovému bodu.

Jakmile hráč splní podmínky pro dokončení závodu, tj. proletí všemi kontrolními body a doletí až do cíle, spustí se finální cutscene se záběrem zepředu na hráče,


**Obrázek 2.10** Návrh grafického uživatelského rozhraní během tréninku a závodu.

který se usměje. Po chvíli se pak zobrazí výsledky závodu (na obrázku 2.11) obsahující pro každého závodníka jeho umístění, čas dokončení závodu a finanční odměnu (více v sekci 2.8).




**Obrázek 2.11** Návrh grafického uživatelského rozhraní s výsledky závodu.

Kdykoliv během tréninku či závodu může hráč hru zapauzovat. Ze zobrazeného menu bude mít možnost závod vzdát. Pokud tak učiní, jako penalizace se mu mírně sníží hodnoty všech statistik (popsané v sekci 2.3.1). Zvažovali jsme i variantu, kdy by se penalizace aplikovala pouze při odchodu z probíhajícího závodu, ne z tréninku před ním. Tak by se mohl hráč beztrestně vrátit, kdyby si např. uvědomil, že si jen zapomněl zvolit nově zakoupené kouzlo. Současně by to ale dalo hráči možnost prohlédnout si vygenerovanou trať a odejít, pokud se mu nelíbí, čemuž bychom chtěli předejít. Mohlo by se uvažovat o nějakém kompromisu (např. žádná penalizace během prvních 10 $s$ nebo dokud se hráč nepohne), ale mohlo by být matoucí, že je chování různé. Proto jsme tyto možnosti zamítli.

## Ohodnocení hráčova výkonu

Poté, co hráč úspěšně dokončí závod, bude třeba přepočítat jeho hodnoty statistik. Nejprve se tedy číselně ohodnotí jeho výkon v závodě (dle požadavku **P11**) pro každou statistiku zvlášť a následně se tato nová hodnota zkombinuje s předchozí pomocí váženého průměru. Díky tomu se předejde náhlým výkyvům a také se budou určitým způsobem zohledňovat hodnoty ze všech předchozích závodů. Váhy použité v průměru pak budou záviset na konkrétní statistice a na pořadí hráče v závodě:

- Dle pořadí hráče v závodě se určí váha nové hodnoty. První a poslední místo povedou na nejvyšší váhu, třetí a čtvrté místo na nejnižší váhu a nakonec druhé a páté místo na střední váhu.
- V *obratnosti* a *přesnosti* bude možné dosáhnout velmi snadno vysokých hodnot hned ze začátku. Nárůst by ale měl být postupný, proto se váha původní hodnoty o něco zvýší.
- V případě *vytrvalosti* se použije rovnou nová hodnota, protože díky způsobu výpočtu bude zajištěno, že se bude měnit pozvolna.

Každá statistika pak bude mít svůj vlastní způsob výpočtu nové hodnoty. Během něj by se měla zohlednit četnost odpovídajících druhů chyb, kterých se hráč v závodě dopustil. Výpočet by ale měl být co nejjednodušší a vycházet z toho, co máme snadno k dispozici. Především se však musí zajistit, že budou hodnoty statistik ze začátku menší a porostou rozumně rychle.

- *Vytrvalost* – Hodnota vytrvalosti bude záviset na tom, jak se hráči celkově daří na trati. Pokud se hráč umístí v první polovině, pak se statistika o něco zvýší (pro první místo nejvíc). Čtvrté místo ponechá hodnotu beze změny. Zbylá umístění statistiku sníží (poslední místo nejvíc). Maximální nárůst se určí nejmenším počtem závodů, které musí i nejlepší hráč absolvovat pro dokončení hry (v takovém případě hodnota lineárně stoupá).
- *Rychlost* – V pravidelných intervalech během závodu se bude vzorkovat aktuální rychlost. Hodnota statistiky se pak spočte jako podíl součtů aktuálních rychlostí a součtů maximálních možných rychlostí, kterých lze dosáhnout s plně vylepšeným koštětem (více v sekci 2.5). Hodnota tak postupně poroste s další úrovní vylepšení rychlosti koštěte a navíc nepřímo zohlední také zrychlení pomocí bonusů nebo kouzel.
- *Obratnost* – V hodnotě této statistiky by se mělo zohlednit, jak dobře se hráč dokáže udržet na dobré trajektorii a vyhýbat se překážkám. V pravidelných intervalech se tedy bude během závodu vzorkovat vzdálenost hráče od ideální trajektorie trati a namapuje se na nějakou penalizaci. Navíc se spočítají také penalizace za každý náraz. Hodnoty se zkombinují ve vhodném poměru.
- *Přesnost* – Měly by se zohlednit různé druhy chyb jako je minutí obručí, nesebrání bonusů, nárazy do překážek a let špatným směrem (promítne se v něm také minutí kontrolního bodu, ke kterému se hráč musí vrátit). Pro obruče a bonusy se spočte podíl počtu úspěšných a celkového počtu. Různé typy bonusů však budou mít různou váhu. V případě nárazů a letů

špatným směrem se každý výskyt penalizuje. Hodnoty se nakonec zkombinují do jedné ve vhodném poměru.

- *Magie* – Bude třeba zohlednit, kolik many hráč získává (více v sekci 2.7), jak často a jak různorodá sesílá kouzla. Spočte se poměr many získané za celou dobu závodu a celkově dostupné many. Dále poměr použité many a celkově dostupné. Hodnoty se zkombinují a výsledná hodnota se pak sníží dle počtu vybavených a neseslaných kouzel (více v sekci 2.7), tj. čím méně vybavených kouzel bylo sesláno, tím více se hodnota sníží. Nakonec se ještě hodnota sníží tím více, čím méně různých kouzel ze všech ve hře hráč někdy seslal. Díky tomu bude hodnota zpočátku malá a poroste postupně.

U jednotlivých statistik se navíc bude tolerovat určité procento chyb, aby nebylo hodnocení tak přísné a vznikla rezerva pro situace, které hráč nemůže ovlivnit (např. soupeř sebere poslední bonus těsně před hráčem). Nejlepší by bylo řídit toleranci zvlášť pro jednotlivé složky výpočtu. Tak by se dalo specifikovat, v čem je třeba nechybovat a kde je naopak možné chybu udělat a jak často. Potřebovali bychom však spoustu parametrů, které by se musely ve výpočtu správně zohlednit. Zavedeme tedy toleranci až na závěr, kdy k výsledné hodnotě každé statistiky přičteme nějakou část jejího doplňku, který v jistém smyslu reprezentuje množství chyb, které se snažíme snížit.

## 2.7 Kouzlení

Jedním z klíčových aspektů hry je také sesílání kouzel. Ve hře by tedy mělo být k dispozici několik kouzel, která si může hráč dle požadavku **P23** kupovat v obchodě (více v sekci 2.8) a následně je používat během závodu dle požadavku **P24**. Nyní si tedy představíme vše, co se sesíláním kouzel souvisí.

### Kouzla

Rozlišíme 4 různé kategorie kouzel podle jejich funkce a typu cíle. Kategorie pak bude určovat také barvu pozadí ikonek příslušných kouzel.

- *Kouzla sesílaná na sebe sama* – Těmito kouzly může libovolný závodník (hráč nebo soupeř) ovlivnit přímo sám sebe.
- *Kouzla ovlivňující soupeře* – Do této kategorie patří kouzla, jejichž cílem je některý ze soupeřů (dle požadavku **P27**).
- *Kouzla pro manipulaci s prostředím* – Tato kouzla je možné využít pro ovlivnění okolního prostředí.
- *Kouzla pro vyčarování objektu* – Pomocí takových kouzel může závodník vyčarovat úplně nové objekty.

Níže uvedeme všechna plánovaná kouzla. Rozhodli jsme se volit pro ně jména odvozená od latinských slov vyjadřujících jejich účinek. Díky tomu budou jména univerzální a nebude třeba je lokalizovat. Každé kouzlo bude mít navíc přiřazenou barvu, která se bude používat pro všechny s ním spojené vizuální efekty (např. linka zůstávající za letícím kouzlem nebo particles při zasažení cíle).

Ve hře pak budou kouzla reprezentována ikonkami dostatečně znázorňujícími jejich efekty (experimentální ověření je popsáno v sekci 5.2.2 a kompletní seznam kouzel i s ikonkami je v sekci 6.2).

Každé kouzlo bude mít celou řadu parametrů (např. cena zakoupení, cena seslání, dobíjecí doba, délka trvání efektu). Bude tedy třeba všechny tyto hodnoty co nejlépe odladit.

### Kouzla sesílaná na sebe sama

- **Velox** (*Zrychlení*) – Závodník dočasně zrychlí, podobně jako po sebrání zrychlujícího bonusu (viz sekce 2.4.1).
- **Defensio** (*Štít*) – Kolem závodníka se dočasně objeví štít, který ho po dobu působení ochrání před všemi negativními kouzly přicházejícími od soupeřů.
- **Relaxatio** (*Odstranění negativních efektů*) – Veškeré negativní efekty působící na závodníka ihned přestanou působit. Jedná se např. o efekty z kouzel soupeřů, jimiž byl závodník zasažen.
- **Translatio** (*Přemístění*) – Závodník se rychle přemístí kus ve směru seslání kouzla.

### Kouzla ovlivňující soupeře

- **Confusione** (*Omráčení soupeře*) – Soupeř bude po krátkou dobu jen pokračovat přímo dopředu, nebude moci nijak ovlivnit směr letu.
- **Congelatio** (*Dočasné zmražení soupeře*) – Soupeř se okamžitě zastaví a zůstane po krátkou dobu stát na místě.
- **Flante** (*Odvanutí soupeře*) – Soupeř bude posunut kus ve směru seslání kouzla.

**Nad rámec demo verze**

- **Permutando** (*Prohození míst se soupeřem*) – Sesílající závodník si rychle prohodí místo se zasaženým soupeřem.

### Kouzla pro manipulaci s prostředím

- **Attractio** (*Přivolání bonusu*) – Závodník si k sobě přitáhne zasažený bonus, čímž ho rovnou sebere. Příště se bonus objeví zase na původním místě.

**Nad rámec demo verze**

- **Movere circulis** (*Přesunutí obruče přímo před sebe*) – Obruč, na kterou závodník zamíří, se přesune kousek před něj (ve směru jeho pohybu) a zůstane tam až do konce závodu. Nelze použít na kontrolní bod.

- **Zatím bezejmenné** (*Odstranění překážky*) – Závodník nechá úplně zmizet konkrétní překážku (ať už z prostředí, nebo vyčarovanou).

#### Kouzla pro vyčarování objektu

- **Temere commodum** (*Vyčarování náhodného bonusu*) – Závodník ve směru, kterým míří, vytvoří náhodný bonus (viz sekce 2.4.1). Ten zůstane na místě, dokud ho někdo nesebere, pak už se neobjeví znovu.
- **Materia muri** (*Vyčarování překážky*) – Kus dopředu ve směru míření se objeví zeď, do které bude možné narazit. Zeď bude natočená kolmo na směr pohybu sesílajícího a zůstane na místě, dokud do ní někdo nenarazí nebo neuplyne dostatečně dlouhá doba.

**Nad rámec demo verze**

- **Nubes maligna** (*Vyčarování oblaku s negativními účinky*) – Ve směru seslání se vyčaruje barevný oblak. Pokud některý závodník tímto oblakem proletí, zpomalí ho to, dočasně mu to částečně zatemní vidění a dokud bude uvnitř, bude mu to vyčerpávat manu. Oblak zůstane na místě po určitou dobu.

## Mana

Seslání každého kouzla bude stát určité množství many. Tu bude moct hráč získávat dvěma různými způsoby:

- Bonusy pro doplnění many – Konkrétní typ bonusů umístěných na trati (více v sekci 2.4.1) bude hráči poskytovat možnost rovnou si doplnit víc many naráz.
- Automatické doplňování many – Během závodu se bude mana hráči pomalu doplňovat sama. Díky tomu nebude muset příliš dlouho čekat na možnost seslání kouzla a nebude silně znevýhodněn, pokud se vždy dostane k bonusům až po jiných závodnících.

Současně však bude omezeno maximální množství many, kterou si závodníci mohou nahromadit. Navíc bude třeba zajistit, aby se mana doplňovala v rozumné míře. Nemělo by se stávat často, že má závodník dostatek many, ale všechna kouzla se teprve dobíjí, ani že jsou všechna kouzla dobitá, ale závodník po velmi dlouhou dobu nemá dostatek many na jejich seslání. Jelikož si hráč bude kouzla odemykat postupně, bude se i jeho spotřeba many postupně navyšovat (pokud má k dispozici více kouzel, během dobíjení jednoho může seslat jiné). Parametry tak odladíme pro pozdější hru. Tím bude mít hráč ze začátku nadbytek many, což však není na závadu, jelikož mu to usnadní počáteční seznamování s kouzlením.

## Použití

Dle požadavku **P25** bude mít hráč k dispozici 4 sloty, do kterých si může zvolit ta kouzla, která chce v závodech používat. Přiřazení si přitom bude moct měnit

kdykoliv mezi závody. Původně jsme zvažovali přidat omezení, že z každé kategorie kouzel smí mít hráč vybavené maximálně jedno, aby se podpořila rozmanitost a zmenšily se rozdíly mezi závodníky. Nakonec se však ukázalo, že je to až příliš omezující. Zakážeme tedy pouze vybavení stejného kouzla vícekrát.

Pro sesílání kouzel se bude používat myš. Jejím pohybem se bude moct hráč rozhlížet kolem sebe pro zamíření (více rozepsáno v následující části). Kolečkem bude moci přepínat mezi jednotlivými sloty a levým tlačítkem myši pak může dané kouzlo seslat. Během závodu se budou jednotlivé sloty a kouzla v nich zobrazovat na obrazovce. Právě zvolené kouzlo bude zvýrazněné rámečkem. Při přepínání se pak budou přeskakovat prázdné sloty a ty krajní budou sloužit jako zarážky.

Použití každého kouzla ale bude omezeno třemi kritérii (dvě z nich byla určena v požadavku **P26**):

- Dobíjecí dobou – Každé kouzlo bude mít pevně danou dobíjecí dobu. Po seslání jej tedy není možné po tuto dobu seslat znovu. Je však možné seslat během toho jiné.
- Množstvím many – Seslání kouzla stojí manu, závodník jí musí mít k dispozici dostatečné množství.
- Dostupností cíle – Některá kouzla vyžadují pro seslání zvolení cíle. Ten se tedy musí vyskytovat v dosahu závodníka a být zaměřený.

Aby tedy bylo možné kouzlo seslat, musí být dané kouzlo nabité, závodník musí mít k dispozici dostatečné množství many a v blízkém okolí se musí vyskytovat vhodný cíl. Zobrazené sloty s kouzly budou tyto stavy vizuálně odrážet. Slotu dobíjejícího se kouzla bude postupně nabývat výplň. Pokud na seslání kouzla nemá hráč dostatek many, cena seslání u něj bude červeně zbarvená. V obou případech pak navíc bude slot menší a částečně zprůhledněný.

## Zamíření

Některá kouzla budou vyžadovat pro seslání nějaký cíl. Pokud jím nebude přímo samotný hráč, bude moci cíl zvolit rozhlížením se kolem sebe (dle požadavku **P24a**). Také bude mít z pohledu dozadu možnost seslat kouzlo za sebe. V tom případě se však nebude moci rozhlížet a pro přesnější zaměření bude muset přizpůsobit směr letu.

Kouzla by se mohla sesílat jen konkrétním směrem a cíl zasáhnout pouze tehdy, pokud do něj po cestě narazí. Závodníci se ale pohybují poměrně rychle a přesné míření by tak mohlo být velmi obtížné. Raději tedy zvolíme možnost, že se kouzlo zaměří na jeden konkrétní cíl a bude ho následovat. Konkrétní způsob zamíření se pak bude lišit dle typu cíle zvoleného kouzla:

- Pokud bude cílem kouzla nějaký objekt (např. soupeř, bonus), tak se jako ten aktuální vybere ten, který bude dostatečně blízko hráči, bude viditelný na obrazovce a bude nejblíž jejímu středu. Obrys aktuálního cíle se zvýrazní a navíc se přes něj zobrazí zaměřovač. Pokud se nenajde žádný takový objekt, pak pro kouzlo neexistuje vhodný cíl.
- Pokud bude zvoleno kouzlo, které je třeba seslat určitým směrem, pak se přímo uprostřed obrazovky zobrazí zaměřovač a jako cílová pozice se zvolí bod v určité vzdálenosti daným směrem.

- Pokud se bude kouzlo sesílat na sebe sama, pak nebude třeba vůbec nijak vybírat cíl. Kouzlo bude možné seslat z libovolného natočení.

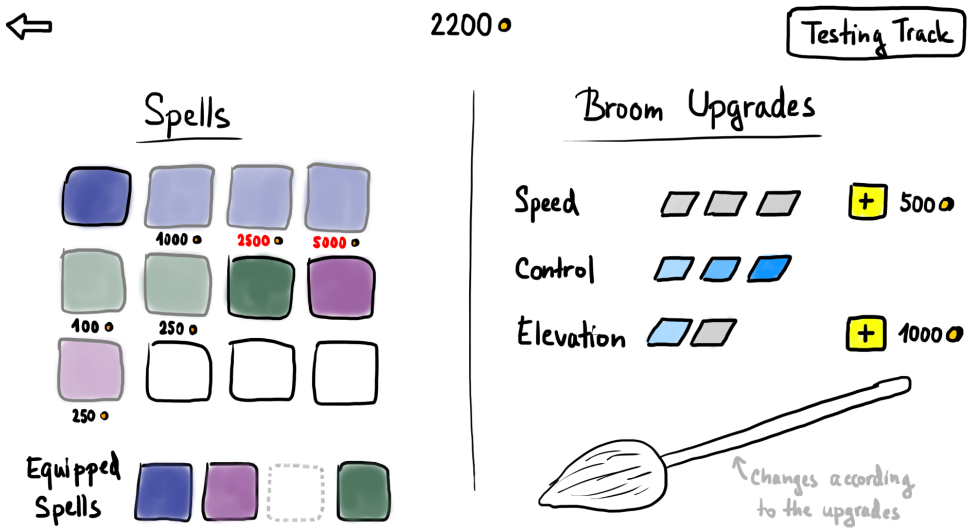
Jelikož tedy kouzlo vždy zasáhne svůj cíl, hráč se mu nemůže nijak vyhnout. Abychom mu dali alespoň nějakou možnost situaci ovlivnit, přidáme štítové kouzlo (popsané v této sekci v části *Kouzla*), které ho po určitou dobu ochrání před všemi příchozími kouzly. Pokud na hráče bude mířit nějaké kouzlo od soupeře, bude na to navíc vizuálně upozorněn dle požadavku **P28**. Na pomyslné kružnici uprostřed obrazovky se zobrazí indikátor, jehož ikonka bude odpovídat danému kouzlu. Pozice na kružnici se určí dle směru, ze kterého kouzlo přichází, a poloměr kružnice bude určen vzdáleností kouzla od hráče (čím blíž, tím menší kružnice). Indikátor také bude tím výraznější, čím blíže bude kouzlo k hráči. Díky tomu bude mít hráč možnost zareagovat použitím ochranného kouzla, pokud ho bude mít k dispozici, a nepoužívat ho jen preventivně.

V okamžiku, kdy bude hráč zasažen cizím kouzlem, se krátce zatřese kamera.

## 2.8 Nakupování

Mezi každými dvěma závody bude mít hráč přístup do obchodu z přehledové obrazovky (viz sekce 2.3.1). V něm si bude moci za mince z absolvovaných závodů pořídit vylepšení koštěte (viz sekce 2.5) dle požadavku **P15** a kouzla (viz sekce 2.7) dle požadavku **P23**. Kouzla v nabídce budou seřazená nejprve dle kategorie (více v sekci 2.7) a v rámci ní vzestupně dle ceny.

Na obrázku 2.12 vidíme návrh grafického uživatelského rozhraní obchodu. Vlevo se bude zobrazovat nabídka kouzel, pro každé bude znázorněna ikonka a pod ní cena. Již zakoupená kouzla pak budou vizuálně odlišena od těch nezakoupených. Pod nabídkou bude navíc přehled vybavených kouzel (více v sekci 2.7). Vpravo budou zobrazena možná vylepšení koštěte. Pro každé se bude znázorňovat jeho aktuální a maximální úroveň. Pod nabídkou pak bude zobrazena aktuální podoba koštěte, která se bude měnit dle zakoupených vylepšení (více v sekci 2.5).




**Obrázek 2.12** Návrh grafického uživatelského rozhraní obchodu.

## Odměna

Pokud se hráč v závodě umístí na 1. až 3. místě, získá odměnu v podobě mincí (v souladu s požadavkem **P10**). Ta by měla hráče motivovat, aby se neustále zlepšoval, pořizoval si lepší vybavení a postupoval dál.

Výše odměny za dokončený závod se odvíjí nejen od umístění, ale také od obtížnosti trati. Ta se počítá jako vážený průměr hodnot statistik, které byly použity pro její generování (dle sekce 2.4.3). *Obratnost* má váhu 3, protože je nejdůležitější a ovlivňuje míru změny směru trati. *Vytrvalost* má také váhu 3, protože určuje celkovou délku trati a díky ní neroste obtížnost příliš rychle. *Přesnost*, určující velikost obručí, má váhu 2. *Rychlost* má váhu 1, protože nepřímo určuje délku trati, která je zahrnuta už ve *vytrvalosti*. Nakonec *magie* se pro výpočet nepoužívá vůbec, protože nijak neovlivňuje obtížnost samotné trati.

## Testovací trať

Z obchodu bude také možné pomocí tlačítka přejít do testovací trati. Bude se jednat o malý a pevně daný level, který bude sloužit především k vyzkoušení nových kouzel v bezpečném prostředí mimo závod. K dispozici v něm budou všechny možné cíle kouzel a kouzla se budou moci sesílat bez spotřeby many a bez dobíjecí doby, aby si je mohl hráč vyzkoušet bez omezení.

Grafické uživatelské rozhraní (na obrázku 2.13) bude velmi podobné tomu v závodě (popsanému v sekci 2.6). Také bude obsahovat minimapu v levém dolním rohu, vpravo dole budou informace o aktuální rychlosti a výšce, vpravo nahoře budou zobrazené efekty působící na hráče a vpravo budou sloty s kouzly. Navíc se však budou zobrazovat také obecné informace o tom, jak používat kouzla, a co dělá to právě zvolené.


**Obrázek 2.13** Návrh grafického uživatelského rozhraní v testovací trati.

**Nad rámec demo verze**

V budoucnu by bylo možné také upravit rozhraní tak, aby nebyl omezen počet slotů pro kouzla, ale zobrazovala se všechna doposud odemčená.

## 2.9 Audiovizuální obsah

Ve hře samozřejmě nemohou chybět různé audiovizuální prvky jako jsou 3D modely, grafické uživatelské rozhraní a hudba na pozadí. Nyní si popíšeme naší vizi toho, jak by měly vypadat.

Kromě samotného obsahu ve hře bychom pak měli vytvořit také ikonku hry v různých velikostech. Tato ikonka by měla být jednoduchá, ale i přes to dostatečně reprezentovat hru. Zvolíme tedy zlatou obruč[1] na fialovém pozadí (na obrázku 2.14).

**Obrázek 2.14** Ikonka hry obsahující zlatou obruč na fialovém pozadí.

### Grafika

Po grafické stránce bychom se chtěli zaměřit na low-poly grafiku s většími jednobarevnými plochami. Vše by mělo být barevné a hravé. Díky zpracovávanému tématu totiž nemusí být grafika realistická a současně nám tento jednodušší styl usnadní přípravu modelů (přičemž dle požadavku P7 bychom si chtěli co nejvíce obsahu vytvořit sami).

Postava hráče bude složena z několika částí, aby bylo možné si ji přizpůsobit. Bude tedy možné nahrazovat materiály pro změnu barvy a meshe pro změnu tvaru. Obecně však bude mít postava jen jeden univerzální tvar a nebude se nijak rozlišovat pohlaví. Pro postavu se navíc vytvoří animace včetně animací obličeje.

Level obsahující trať bude celý pokrytý terénem. Ten budou tvořit body v mřížce s přiřazenou výškou, aby se vytvořila trojúhelníková síť. Stejně jako ostatní objekty ve hře bude mít znatelné hrany a velké jednobarevné plochy.

Bude potřeba také vytvořit celou řadu vizuálních efektů, které se využijí např. během sesílání kouzel (efekt při seslání, efekt během letu kouzla, efekt při zasažení). Typicky se pro to budou využívat particles a speciální shadery.

**Nad rámec demo verze**

V demo verzi se omezíme jen na některé základní efekty. Do budoucna však bude možné efekty ještě vylepšit a také přidat mnohé další (např. linku zůstávající za koštětem, čáry na obrazovce naznačující rychlost letu, písečnou bouři v odpovídající oblasti, shader pro vodní plochu a další).

### Grafické uživatelské rozhraní

Ačkoliv se pokusíme minimalizovat množství textu na obrazovce, úplně se mu nevyhneme, protože hráči bude potřeba sdělit spoustu informací. Současně

[1] Obruč je ve skutečnosti písmeno O převzaté z fontu Akronim (`https://fonts.google.com/specimen/Akronim`), který byl použit pro název hry v hlavním menu.

se hra odehrává v magickém světě. Měli bychom tedy zvolit takové fonty, které dostatečně zapadají do herního světa, ale současně jsou dobře čitelné.

Veškeré grafické uživatelské rozhraní by se pak mělo přizpůsobovat velikostem zobrazení (dle požadavku **P6**). To nejen umožní vhodné zobrazení na různých monitorech, ale také usnadní případné rozšíření na další cílové platformy.

Dle požadavku **P4** bychom chtěli hru lokalizovat do více jazyků. Již v hlavním menu by tak měla být možnost zvolit si jazyk hry (pouze pomocí ikonky, bez jakéhokoliv textu). Pak bychom měli zajistit, že se tomuto nastavení bude přizpůsobovat veškerý obsah.

### Audio

Samozřejmostí bude také přidání různých zvuků (v souladu s požadavkem **P5**), které by měly vhodně doplňovat atmosféru. Bude se jednat o hudbu na pozadí (bude hrát neustále ve formě náhodně přehrávaného seznamu skladeb), zvuky prostředí během závodu (podtrhující atmosféru) a zvukové efekty během závodu (jednorázové, např. seslání kouzla, i dlouhotrvající, např. zvuk letu koštěte) i v uživatelském rozhraní (např. kliknutí, návrat zpět). V nastavení bude možné změnit úroveň hlasitosti pro každou skupinu zvuků zvlášť.

V průběhu hry bychom pak chtěli ovládat různé aspekty audia. Budeme chtít, aby se hudba na pozadí potlačila, kdykoliv hráč vstoupí do závodu. Také by se měly zvuky prostředí přizpůsobovat oblasti, ve které se hráč nachází. Dále by se měly na audio aplikovat různé efekty, pokud je hra zapauzovaná nebo se hráč nachází pod vodou.

**Nad rámec demo verze**

Do budoucna bychom chtěli také vytvořit adaptivní hudbu, která by se přizpůsobovala dění v závodě. Mohla by tak být intenzivnější, když se hráč blíží k cíli a kolem něj jsou další soupeři. Nebo by se mohlo tempo měnit dle rychlosti letu. Také by se mohla přidat další zlověstnější vrstva, pokud na hráče působí nějaký negativní efekt. K tomu všemu by se daly použít koncepty *resequencing* nebo *reorchestration* (případně kombinace obou) popsané v kapitolách 8 a 9 knihy *Music Design for Game Development Studios* [18].

## 2.10 Další prvky

Ve hře bude mimo již zmíněné spoustu dalších prvků. V této sekci je představíme, ale jen velmi stručně, jelikož se jedná pouze o vedlejší funkce.

### Rychlý závod

Dle požadavku **P29** by měl být ve hře také režim rychlého závodu, ve kterém si hráč spustí jeden zcela nezávislý závod, jehož výsledek nebude mít žádný vliv na případnou rozehranou hru v kariérním režimu. Bude se jednat jen o jednorázovou zábavu bez ukládání stavu a bez postupu. Hráč se k této možnosti dostane z hlavního menu. Poté se mu zobrazí možnosti nastavení tohoto závodu. Jelikož je

totiž závod separátní, nelze vycházet z hráčových aktuálních statistik a je třeba říct, jak by měl závod vypadat.

Pro jednoduchost bude mít hráč možnost zvolit jednu z předpřipravených úrovní obtížnosti. Ta pak ovlivní jak hodnoty statistik, použité pro generování trati (více v sekci 2.4.3) a chování soupeřů (více v sekci 2.3.2), tak úrovně vylepšení koštěte (popsané v sekci 2.5). Každá úroveň obtížnosti bude mít tato nastavení fixně přiřazená. Navíc bude mít hráč možnost zvolit si vlastní obtížnost. Pak se zobrazí další prvky pro nastavení konkrétních hodnot individuálních statistik a stupňů vylepšení koštěte.

Dále bude moct hráč určit, zda se mají zpřístupnit kouzla, či nikoliv. Pokud je zpřístupní, pak si bude moci vybrat, která kouzla si vybaví do slotů pro použití v závodě (jako u běžného závodu, viz sekce 2.7). Na výběr však bude mít ze všech ve hře dostupných kouzel. Jestliže budou kouzla povolená, pak je budou moci stejně tak používat také soupeři.

Naposledy použitá nastavení se uloží perzistentně, takže se při příštím zobrazení budou moci načíst a rovnou použít jako výchozí. Pokud však uložená data neexistují, ale hráč má již rozehranou hru v kariérním režimu, hodnoty se inicializují co nejblíže stavu z něj (tj. zvolí se jedna z přednastavených obtížností, která je nejblíže hodnotám statistik ze stavu hry). Pokud hráč nemá rozehranou hru, jako výchozí obtížnost se zvolí ta nejsnazší.

**Nad rámec demo verze**

Do budoucna bychom mohli přidat také možnost povolit jen konkrétní kouzla. Tak by nemohli soupeři používat zcela náhodná, což by dělalo závod vyrovnanější. Takové nastavení by mohlo být obzvlášť přínosné pro nepříliš zkušeného hráče, který jednotlivá kouzla teprve postupně prozkoumává.

V režimu rychlého závodu se vždy přeskočí tréninková fáze (popsaná v sekci 2.6), aby se hra ještě urychlila a mohlo se přejít rovnou do akce. Průběh závodu je pak zcela totožný s kariérním režimem (více v sekci 2.6). Po dokončení závodu a zobrazení výsledků se však bude moct hráč pouze vrátit do hlavního menu.

## Ocenění

Jelikož je ve hře spoustu možných akcí, identifikujeme některé menší cíle, za které bude moci hráč získávat ocenění. Ta nebudou mít ve hře žádnou funkci, jejich jediným účelem bude poskytnout hráči satisfakci z jejich získání a motivovat ho k získání všech. Některá ocenění pak budou mít více úrovní, od bronzové, přes stříbrnou a zlatou až k platinové.

Přehled získaných ocenění se bude zobrazovat na obrazovce přehledu hráče (popsané v sekci 2.3.1). Dostupná budou např. ocenění za počet prolétnutých obručí nebo seslaných kouzel, za dosažení maximální hodnoty v některé statistice, za našetření určité částky nebo za umístění na prvním místě několikrát v řadě. Kompletní seznam i s ikonkami je pak uveden v sekci 6.2.

## Tutoriál

Abychom hráči usnadnili zorientování se ve hře a seznámení se s ovládáním, přidáme do hry také tutoriál (dle cíle **C3**). Rozdělíme ho do několika částí, které se budou moci spustit ve vhodné okamžiky. Tak se ukáže vždy jen to, co hráč v danou chvíli potřebuje vědět, a až tehdy, když na to narazí poprvé. Části budou na sebe navazovat, takže konkrétní část se může spustit až ve chvíli, kdy ta předchozí skončila. Bude se jednat o následující:

1. **Úvodní tutoriál** – Spustí se ihned po spuštění nové hry. Hráč se v něm seznámí se základním ovládáním (viz sekce 2.5) a s prvky trati (viz sekce 2.4.1).
2. **První závod** – Spustí se, jakmile hráč vstoupí do svého prvního závodu. Představí se koncept tréninku a startovní zóna (viz sekce 2.6). Pak bude mít hráč možnost zvolit, jestli chce nechat trénink před závodem zapnutý.
3. **Přehled hráče** – Spustí se, jakmile se hráči poprvé zobrazí obrazovka s přehledem hráče (viz sekce 2.3.1). Představí se mu žebříček závodníků a jednotlivé statistiky.
4. **Obchod** – Spustí se, jakmile má hráč dostatek mincí na zakoupení něčeho v obchodě a nachází se v obrazovce přehledu hráče. Představí se tlačítko pro vstup do obchodu (viz sekce 2.8) a základní části obchodu.
5. **Vybavení kouzla** – Spustí se, jakmile hráč zakoupí své první kouzlo. Ukáže hráči, jak si může kouzlo dosadit do slotu (viz sekce 2.7).
6. **Sesílání kouzel** – Spustí se, jakmile hráč dosadí první kouzlo do slotu. Seznámí hráče se základním ovládáním, vysvětlí manu, související bonusy a různé cíle kouzel (viz sekce 2.7).
7. **Testovací trať** – Spustí se, jakmile hráč vstoupí do obchodu po dokončení všech předchozích částí. Ukáže se tlačítko pro vstup do testovací trati a vysvětlí se její jedinečné charakteristiky (viz sekce 2.8).

Během tutoriálu se budou využívat jak statické tabulky s textem, tak interaktivní části, ve kterých bude muset hráč provést konkrétní akci. Hráč by měl mít možnost také konkrétní část tutoriálu přeskočit, případně tutoriál v nastavení úplně vypnout.

## Tooltipy

Na různých místech ve hře by se hodilo využít tooltipy, aby se po najetí kurzoru na nějaký prvek uživatelského rozhraní zobrazila tabulka s doplňujícími informacemi. Díky tomu by nebyly obrazovky plné textu a hráči by se mohlo zobrazit jen to, co v danou chvíli vyžaduje.

Hlavním způsobem využití by byla možnost zobrazit kouzla pouze jako ikonky (např. v nabídce obchodu nebo ve slotech s vybavenými kouzly, viz sekce 2.7) a až po najetí zobrazit veškeré informace (např. efekt kouzla, cenu seslání, dobíjecí dobu). Daly by se tím vysvětlit také jednotlivé statistiky u os grafu v přehledu hráče (viz sekce 2.3.1) nebo upřesnit, že po vzdání závodu z menu pozastavené hry se aplikuje penalizace a sníží se aktuální hodnoty statistik (viz sekce 2.6).

## Mapování kláves

Ovládáním jsme se inspirovali ze hry *Hogwarts Legacy*, ale najdou se hráči, kteří si na mapování v ní stěžují [19]. Proto přidáme do hry také možnost přizpůsobit si mapování kláves dle sebe. Tak si bude moci každý hráč zvolit, co mu vyhovuje.

V případě některých akcí však přemapování neumožníme. V nastavení se budou zobrazovat, aby si hráč mohl snadno zjistit, jaké je ovládání, ale nebude možné je změnit. Bude se jednat např. o dobře zažité akce (pozastavení hry pomocí `ESC`) a o akce využívající myš (jiné mapování místo pohybu myši by se nastavovalo vcelku složitě).

Hráč by měl být také upozorněn na duplicity, tzn. pokud pro nějakou akci zvolí mapování, které se používá už u jiné, mělo by se zobrazit varování.

**Nad rámec demo verze**

Do budoucna bychom chtěli zvážit také podporu herního ovladače, nejen klávesnice a myši. Pokud bychom se rozhodli přidat lokální hru více hráčů (více v sekci 2.11), pak by to bylo dokonce nezbytné.

## Cheaty

Pro možnost snadného testování hry by měly být k dispozici také cheaty, které budou umožňovat snadno a rychle měnit parametry hry. Inspirujeme se podobou cheatů ze hry *The Sims* [20]. Tam se pomocí kombinace `Ctrl+Shift+C` zobrazí konzole, do které se následně zadávají příkazy. Jelikož se v naší hře používá `Shift` jako výchozí mapování akce letu dolů, použijeme místo toho kombinaci `Ctrl+Alt+C`. Kombinace by tak současně měla být dostatečně komplexní na to, aby na ni hráč nepřišel omylem.

Měli bychom zavést cheaty pro všechny významné akce ve hře. Může se jednat např. o změnu množství peněz, změnu hodnot statistik, vylepšení/zhoršení koštěte, odemčení/zamčení kouzla, změnu aktuálního množství many. Kompletní seznam cheatů s odpovídajícími příkazy a příklady použití je pak v sekci 4.13.1. Použití cheatů však bude na vlastní riziko. Nevhodné použití může uvést hru do nestabilního stavu. Současně by ale měla jít podpora pro cheaty snadno vypnout, kdybychom je chtěli odebrat z distribuované verze hry.

## Oddělený generátor levelů

Pro generování levelů dle sekce 2.4.3 budeme využívat nějakou implementaci generátoru. Mohlo by být užitečné vytvořit také separátní scénu, do které by se daný generátor umístil (pro naplnění požadavku **P30**) a bylo by možné pomocí jednoduchého uživatelského rozhraní nastavovat různé parametry, generovat odpovídající levely a prohlížet si je volným pohybem kamery ve scéně.

Tato scéna by nebyla hráči dostupná běžnými prostředky, protože by neměla být přímo součástí hry. Místo toho by měla sloužit spíše jen pro testovací nebo demonstrační účely. Bylo by tedy možné dostat se do ní pouze pomocí cheatů (popsaných v předchozí části a v sekci 4.13.1).

Parametry, které by se nastavovaly, by mohly být přímo hodnoty statistik, ze kterých se pak odvozují jiné parametry používané generátorem. Dále by se mohlo

nastavit maximální stoupání koštěte, protože na něm závisí dostupné tematické oblasti (viz sekce 2.4.2). Takto by bylo možné generovat přesně takové tratě, jaké jsou ve hře, což by se dalo použít pro testování (např. generovat spoustu levelů za sebou a sledovat, jestli nedochází k nějakým problémům).

**Nad rámec demo verze**

Druhým možným přístupem by bylo nastavovat přímo parametry generátoru (např. nenastavovat *obratnost*, ale nastavit rovnou maximální úhly změny směru). Díky tomu by bylo možné prohlížet si i tratě, které se ve hře vyskytovat nemohou. To by mohlo posloužit návrháři, který by tak mohl ladit parametry používané přímo ve hře. Bylo by však třeba vytvořit rozhraní pro velké množství parametrů, a tak ponecháme tuto variantu do budoucna a prozatím přidáme pouze nastavování pomocí statistik a stoupání koštěte.

Mohli bychom také přidat možnost generovat jen určité části levelu (např. terén bez trati). Navíc by mohlo být užitečné mít možnost skrýt uživatelské rozhraní a uložit snímek obrazovky.

## 2.11 Budoucí plány

Rozsah plánované hry je obrovský a do demo verze, která bude výsledkem této práce, nemůžeme implementovat vše. Některé funkce tedy odložíme až do budoucna.

**Nad rámec demo verze**

### Režim více hráčů

Hry bývají obvykle zábavnější, pokud se hrají ve skupině. V závodních hrách to navíc přidává na soutěživosti. Proto bychom chtěli do budoucna přidat také nějakou formu hry více hráčů. Nabízí se však několik různých možností provedení a je třeba ještě pečlivě zvážit, která by byla nejvhodnější, případně jestli jich nepodporovat rovnou několik.

První možné rozdělení je dle způsobu propojení hráčů:

- Lokální hra více hráčů – Hráči by spolu hráli na jednom zařízení. Mohli by hrát maximálně 4 hráči, přičemž obrazovka by se pak rozdělila na odpovídající počet částí (tzv. split screen). V tomto případě by však bylo třeba přidat podporu externích herních ovladačů a možnost je přemapovat.
- Globální hra více hráčů – Hráči by se připojili přes síť, každý ze svého zařízení. Tak by mohlo hrát společně až 6 hráčů. Bylo by možné vytvářet veřejné skupiny a přidávat se do nich. Případně pomocí zvacího kódu vytvářet skupiny uzavřené.

Pak je však otázka, jakým přesně způsobem by závodění probíhalo. Opět se nabízí několik možností, z nichž několik uvedeme:

- Mohlo by se jednat o samostatný závod, který by byl podobný rychlému závodu ze sekce 2.10. To by bylo vhodné pouze pro lokální multiplayer,

protože režie potřebná pro vytvoření skupiny po síti by byla příliš dlouhá oproti hernímu času.

- Nebo by to mohla být série několika závodů, které by tvořily jakýsi turnaj (podobně jako ve hře *Mario Kart*). Hráči by za každý závod získávali nějaký počet bodů a na základě nich by se umisťovali. To by mohlo být vhodné pro lokální multiplayer i globální (pak se ovšem může stát, že se ostatní hráči odpojí a do konce zůstane jen jeden).
- Závody by mohly být oddělené, ale navazovat donekonečna jeden za druhým. To by se hodilo pro globální multiplayer, kdy se může kdokoliv kdykoliv připojit a zase odpojit. Pokud by se však hráč připojil uprostřed probíhajícího závodu, mohl by ho pouze sledovat a účastnit se až toho dalšího.

Dále by bylo třeba rozhodnout, jakým způsobem řešit vybavení hráčů. Jestli by měli všichni hráči rovnou maximálně vylepšené koště a na výběr ze všech kouzel, nebo by si mohli ze začátku zvolit něco jako obtížnost, která by to určovala (a mohli by se seskupit pouze hráči ze stejné obtížnosti), nebo by začali od nuly a mezi závody by si mohli vše postupně odemykat podobně jako v režimu kariéry. Různá řešení jsou vhodná do různých prostředí (lokální/globální) a mají různé výhody a nevýhody. Až by nadešel čas na implementaci, bylo by třeba vše řádně promyslet.

Podobně by se muselo také určit, na základě jakých parametrů by se generovaly tratě. Nejlepší by asi bylo použít nějakou přednastavenou obtížnost, podobně jako v rychlém závodě (ze sekce 2.10), nebo brát parametry zcela náhodně.

Nakonec by bylo třeba určit, zda by se měli přidávat soupeři řízení umělou inteligencí (např. aby doplnili počet do 6 závodníků jako v kariérním režimu) a jaké by měli mít schopnosti.

S přidáním režimu hry více hráčů by navíc bylo možné přidat i další funkce. Závodníci by se mohli navzájem shazovat z koštěte, nebo se alespoň odstrkovat, aby byly závody zajímavější. Bylo by možné přidat nová kouzla, která by tu dávala větší smysl (např. kouzlo, které zasaženému dočasně prohodí ovládání koštěte).

Jak je tedy zřejmé, režim více hráčů je poměrně rozsáhlý a před samotnou implementací by bylo třeba upřesnit jeho návrh.

## Další cílové platformy

Zamýšlenou cílovou platformou je dle požadavku **P2** PC s operačním systémem Windows. Do budoucna bychom však chtěli zvážit rozšíření také na WebGL. Ovládání by nebylo třeba nijak upravovat, ale tato platforma má určitá specifika, kterým by bylo nutné hru přizpůsobit. Např. audio middleware FMOD, který bychom chtěli použít (viz sekce 3.8), využívá vlákna, která však WebGL nepodporuje [21], což může způsobit zasekávání audia, kdykoliv se něco načítá.

Mohlo by být zajímavé rozšířit hru také na systém Android, ale k tomu by bylo třeba navrhnout zcela nové schéma ovládání pro dotykovou obrazovku. To však může být problematické vzhledem k velké volnosti pohybu a množství akcí ve hře. Zatím tedy tento systém vůbec nebereme v úvahu, protože by představoval jen spoustu omezení. Do budoucna by se však mohla zvážit nějaká modifikace hry pro tuto platformu.

# 3 Analýza technických řešení

V předchozí kapitole jsme popsali návrh toho, jak by hra měla vypadat a co vše by měla obsahovat. Nyní se zaměříme na jednotlivé části tohoto návrhu a probereme možnosti jejich implementace. Odůvodníme některá rozhodnutí a popíšeme potřebné nástroje.

Naši hru implementujeme v herním enginu Unity, dle požadavku P1. Během implementace tak budeme využívat v něm běžné koncepty. V následujících sekcích pak pár vybraných stručně popíšeme, ale obecně předpokládáme, že je čtenář s Unity alespoň základně obeznámen, takže nebudeme zacházet do přílišných detailů. Pro více informací je pak možné podívat se do oficiální dokumentace [22].

## 3.1 Objektový návrh v Unity

Po celou dobu práce na hře využijeme standardní postupy, které jsou běžné během vývoje v Unity a v jazyce C#. Současně budeme myslet také na výkon a používat obvyklá doporučení. V této sekci nyní popíšeme některá specifika práce v Unity a naše případná vylepšení.

### 3.1.1 Herní objekty

Z objektově orientovaných jazyků jsme zvyklí definovat různé třídy, od kterých pak vytváříme instance, a využívat hierarchii dědičnosti. Vývoj her v Unity nám však přidává další vrstvu abstrakce, kdy vytváříme *herní objekty* (reprezentované instancí třídy `GameObject`, nebo odvozené), které mají přiřazené *komponenty* (potomky třídy `MonoBehaviour`).

#### Třídy a komponenty

Pokud pak chceme vytvořit nějakou uzavřenou jednotku obsahující kód, musíme se rozhodnout, jestli stačí jen obyčejná třída, jejíž instance se někde vytvoří a uloží, nebo jestli by to měla být komponenta, kterou pak bude třeba navěsit na nějaký herní objekt ve scéně. Rozhodnutí může být různé v různých situacích. Obecně zvolíme komponentu tehdy, když vytváříme něco, co reprezentuje konkrétní funkcionalitu herního objektu a/nebo co by se mělo aktualizovat každý frame (v komponentě můžeme implementovat metodu `Update()`, která se pak automaticky volá).

#### Prefab

Kdykoliv potřebujeme použít ten samý herní objekt na několika různých místech, můžeme využít tzv. *prefab* [23]. Vytvoříme tedy daný herní objekt se všemi jeho komponentami (případně podobjekty) a s požadovanými hodnotami parametrů a následně z něj vytvoříme asset. Ten slouží jako šablona, ze které pak můžeme vytvářet instance v libovolné scéně. Dalo by se říct, že se jedná o implementaci návrhového vzoru *prototype* [24]. Veškeré změny provedené v prefabu se propagují také na všechny instance. Navíc ale máme možnost pro konkrétní instanci přepsat

zděděné vlastnosti (např. parametry komponent) nebo upravit hierarchii (tj. přidat nové podobjekty a komponenty).

Jestliže pak změníme některou instanci, můžeme z ní vytvořit tzv. *prefab variant*. Opět se bude jednat o asset, který obsahuje již připravený herní objekt s komponentami a případnými podobjekty. Bude ale odvozený od konkrétního prefabu. Poté můžeme snadno vytvářet libovolné množství instancí od této varianty. Pokud bychom změnili nějaké vlastnosti původního prefabu, změní se také v jeho variantě, pokud tyto vlastnosti nepřepisuje svými hodnotami.

V naší hře můžeme tyto koncepty využít např. pro závodníky. Vytvoříme prefab pro základ závodníka (tj. model a základní chování) a následně od něj odvodíme dvě prefab varianty, jednu pro hráče a jednu pro soupeře, aby mohla mít každá navíc své specifické komponenty.

#### ScriptableObject

Dalším běžně používaným konceptem v Unity je tzv. *ScriptableObject* [25], který je velmi užitečný, pokud chceme uložit data potřebná na více místech. Jedná se o obyčejnou třídu, která dědí od `ScriptableObject`. Instance se pak ukládají jako assety a dají se vytvářet snadno přímo v editoru. Následně k nim můžeme přistupovat z kódu skrz reference. Obvykle se používá v situaci, kdy potřebujeme několik instancí dat stejného typu, které se poté využívají napříč hrou. V našem případě tak můžeme uložit např. data pro jednotlivá dostupná kouzla.

### 3.1.2 Provázání objektů

Jakmile máme ve hře několik různých herních objektů, obvykle je potřebujeme nějak provázat, aby mezi sebou mohly interagovat (volat metody, přistupovat k datovým položkám apod.).

#### Získání reference

Pro samotné získání reference na jiný objekt, kterou pak můžeme použít v kódu, existuje celá řada způsobů, každý se svými výhodami a nevýhodami:

- Zveřejnění datové položky v Inspector okně – Veřejné datové položky a položky označené atributem `[SerializeField]`, které definujeme v komponentě, se zobrazují v Inspector okně. Následně do nich můžeme přetáhnout objekt, na který potřebujeme referenci. Tak můžeme k objektu přistupovat velmi rychle, ale Inpector okno může být nepřehledné s velkým množstvím datových položek (lze zmírnit použitím `[Header("")]` atributů) a při refaktorizaci může docházet k problematickým situacím (např. se reference může ztratit, je třeba znovu dosadit).

- Nalezení dle jména nebo tagu – Pokud potřebujeme najít objekt za runtime, můžeme využít statické metody ze třídy `GameObject`. `Find(string name)` pro nalezení podle jména nebo `FindWithTag(string tag)` pro nalezení podle tagu. Jsou velmi obecné, ale tím také pomalé. Při jejich použití se totiž prohledávají všechny herní objekty a komponenty v paměti. Vyhledáváním podle řetězce navíc může snadno docházet k chybám (přepíšeme se, přejmenujeme objekt apod.).

- Nalezení dle typu komponenty – Pomocí metody `FindObjectOfType<T>()` můžeme nalézt objekt podle typu komponenty. Opět se ovšem prohledává, takže je to pomalé.

- Nalezení potomka dle typu komponenty – Jestliže víme, že se hledaná komponenta nachází přímo na aktuálním objektu (tj. na tom, jemuž náleží komponenta, ve které je kód) nebo jeho podobjektu, můžeme použít metodu `GetComponent<T>()` nebo `GetComponentInChildren<T>`. Jejich použití je vcelku rychlé, ale i přesto se hodí výsledek cachovat do proměnné, pokud ho budeme využívat opakovaně.

- Vlastní pomocné metody – Pomocí výše zmíněných si můžeme implementovat také vlastní metody, např. pro vyhledání objektu na základě kombinace typu komponenty a tagu.

- Statická datové položka – Pokud se jedná o objekt implementující návrhový vzor singleton (více popsaný v sekci 3.9.1), můžeme při inicializaci objektu dosadit referenci do statické položky, ke které pak lze snadno přistupovat odkudkoliv.

Během implementace naší hry tak pečlivě zvážíme výhody a nevýhody jednotlivých přístupů a pro každou situaci vždy zvolíme ten nejvhodnější.

#### Komunikace mezi objekty

Velmi běžně potřebují herní objekty reagovat na nějakou událost, která se odehrála v jiném objektu. Opět existuje celá řada možností:

- Volání metod – Pokud nastane událost, objekt může zavolat na jiném nějakou konkrétní metodu, aby ho na tuto skutečnost upozornil. Potřebuje však referenci na daný objekt a v případě volání více metod na více objektech je třeba všechny explicitně uvést.

- Události a delegáti – V objektu, ve kterém dochází k zajímavé události, můžeme deklarovat delegáta, ve kterém si pak ostatní objekty mohou zaregistrovat callback na danou událost. Jakmile pak k události dojde, stačí daného delegáta vyvolat a objekty budou notifikovány. Tak je možné se kdykoliv přihlásit, či odhlásit, ale je potřeba mít referenci na objekt s delegátem.

- `UnityEvent` – Unity nabízí svou vlastní alternativu delegátů v podobě `UnityEvent`. Jedná se o velmi podobný koncept, ale callbacky na událost se dají přiřazovat nejen z kódu, ale také přehledně z editoru pomocí Inspector okna.

- Zprávy v Unity – V Unity je již implementován koncept zasílání zpráv pomocí metod `SendMessage()` a `BroadcastMessage()` (parametrem je název metody, která se vyvolá na příjemci). Pomocí něj se však dají posílat zprávy pouze v omezené části hierarchie objektů a navíc je zaslání velmi neefektivní, protože interně využívá reflection.

Každá z vyjmenovaných možností má své specifické využití. Navíc by se nám ovšem hodilo dokázat zasílat zprávy libovolnému objektu, který o ni projeví zájem. Díky tomu by nemusely být objekty úzce provázané a nevyžadovaly by žádné reference (pouze na případného prostředníka zajišťujícího zasílání zpráv, který ale může být globálně přístupný singleton, viz sekce 3.9.1). Mohly by tak být znovupoužitelné i na jiných místech nebo v jiných projektech.

Implementujeme si tedy vlastní systém pro zasílání zpráv, ve kterém by se mohly různé objekty kdykoliv zaregistrovat ke konkrétní zprávě (případně také odregistrovat). Pokud by pak nastala nějaká zajímavá událost, daný objekt by pomocí systému odeslal odpovídající zprávu. Samotný systém by se poté staral o notifikaci všech registrovaných objektů. Jednalo by se o zjednodušenou implementaci návrhového vzoru *publish-subscribe* [26], kdy bychom podporovali pouze synchronní zasílání zpráv, ne asynchronní pomocí fronty zpráv.

Tato implementace by nám umožnila snadné zavedení cheatů (popsaných v sekci 3.9.8) a analytik (popsaných v sekci 4.10.7), které by tak mohly existovat zcela nezávisle. Současně by se zjednodušila také reakce na události od objektů, kterých je ve scéně velké množství. Např. pokud hráč sebere bonus, měl by se o tom dozvědět systém pro ocenění (popsaný v sekci 4.10.2), aby bylo možné získat ocenění za počet nasbíraných bonusů. Bonusy by ale o tomto systému neměly nic vědět, aby s ním nebyly úzce svázané. Současně by ale systém pro ocenění neměl potřebovat procházet všechny bonusy ve scéně, aby se u nich zaregistroval. Zavedení nějakého správce všech bonusů jen proto, abychom potřebovali jedinou referenci, se pak zdá jako zbytečná komplikace.

## 3.2 Práce s daty

V řadě situací budeme potřebovat pracovat s různými daty. Můžeme chtít načítat nějaká externí data pro použití ve hře nebo naopak ukládat data ze hry pro použití v příštím sezení. Nyní si popíšeme, jaké možnosti máme k dispozici.

### Načítání externích dat

Pokud potřebujeme do hry dodat data v rámci nějakých souborů, existují pro to různé způsoby:

- Addressables [27] – Konkrétní assety (soubory, prefaby apod.) můžeme označit jako „addressable“. Tím se pro ně vygenerují adresy, které následně můžeme využívat v kódu místo cesty k souboru. Pokud se umístění assetu změní, nemusíme kód měnit. Hodí se to v případě, že chceme přistupovat ke konkrétnímu assetu.

- `Resources` [28] – Soubory umístěné ve složce `Resources/` (kdekoliv v projektu) se zabudují přímo do buildu jako soubor `resources.assets`, do kterého se všechny zkombinují. Pomocí třídy `Resources` je pak můžeme vyhledávat a načítat z kódu. Můžeme tak ukládat také instance potomka `ScriptableObject`, abychom pak mohli za runtime snadno získat všechny instance daného typu. Využijeme to např. pro uložení dat o všech dostupných kouzlech, která po spuštění hry načteme.

- `StreamingAssets` [29] – Slouží pro uložení souborů, které chceme zpřístupnit hráči. Stačí je umístit do `Assets/StreamingAssets/`. Ve finálním buildu hry pak budou stále dostupné jako separátní soubory v původním formátu. Díky tomu se dají snadno nahradit také po vytvoření buildu. Využijeme to např. pro soubor se jmény, která se používají při randomizaci postavy, takže si může hráč tento seznam upravit.

### Perzistentní ukládání dat

V požadavku **P3** jsme si určili, že by se měl stav hry ukládat mezi jednotlivými sezeními. Ukládali bychom tak různé informace jako zvolenou podobu postavy, hodnoty nastavení, zvolený jazyk, hodnoty statistik, zakoupená kouzla apod.

Pro uložení jednoduchých dat můžeme využít třídu `PlayerPrefs` [30]. Ta umožňuje uložení několika základních datových typů pod určitým klíčem. Data z `PlayerPrefs` se ve skutečnosti ukládají do registru.

Pokud však máme nějaká větší a složitější data, lepší by bylo použít soubory. Pomocí `Application.persistentDataPath` se můžeme snadno dostat k vhodnému adresáři. Navíc můžeme použít třídu `JsonUtility` [31], abychom mohli serializovat celé instance tříd nebo struktur do formátu JSON, který je dobře čitelný, nebo je následně deserializovat. Je však třeba dát pozor na to, že některé datové typy (např. `Dictionary`) nejsou tímto způsobem serializovatelné.

V naší hře bychom si implementovali vlastní systém pro ukládání dat v podobě třídy se statickými metodami, aby k nim byl dobrý přístup odkudkoliv. Kdykoliv by se mělo něco uložit, systému by se předala celá objektová reprezentace daného stavu. Pak už by bylo na systému, jakým přesně způsobem data uloží (tj. v jakém formátu, do jakého souboru). Díky tomu by byla skutečná implementace oddělená a mohli bychom ji snadno měnit. Veškerý stav ve hře pak rozdělíme do několika různých souborů, abychom mohli pracovat s menšími jednotkami.

Pro jednoduchost však nebudeme nijak ošetřovat nevalidní obsah souboru či chybějící soubory. Jelikož si hra bude vždy ukládat validní stav, selhání načítání bude možné jedině v případě, pokud přímo hráč modifikoval obsah souborů. Pak by tedy mělo být přijatelné, že to bude na vlastní riziko a hra se tím může dostat do nekonzistentního stavu. Do budoucna bychom však mohli alespoň hráče upozornit, že nelze stav načíst, protože data nejsou validní.

## 3.3 Procedurální generování levelů

V sekci 2.4 jsme popsali, jak budou vypadat levely, ve kterých se budou odehrávat závody, a také jsme zavedli závislost určitých parametrů trati na hodnotách statistik hráče (popsaných v sekci 2.3.1). V této sekci na to navážeme a projdeme různé způsoby, jakými by se dalo procedurální generování levelů implementovat.

Level se skládá z trati a okolního prostředí. Pokud bychom chtěli generovat skutečně zajímavé prostředí, které by se s tratí vhodně doplňovalo, pak bychom obě části měli generovat současně. Někdy by se terén tvaroval dle trati, jindy zase trať dle terénu. Tím by se však implementace velmi zkomplikovala. Jako kompromis bychom mohli manuálně připravit několik různých prostředí, do kterých bychom umístili nápovědy pro následný generátor trati (příkladem by mohlo být označení začátku a konce tunelu). Výsledná prostředí by tak mohla být velmi zajímavá,

ale už ne tolik různorodá. Zůstaneme tedy prozatím u samotného generování, které umožní různorodost, snadné rozšíření do budoucna a větší volnost při volbě parametrů trati.

Jelikož existuje celá řada možností, ale generování je jen jednou z mnoha částí této práce, pokusíme se volit spíše jednodušší postupy, kterými snadno dosáhneme přijatelných výsledků. Budeme tedy generovat trať a prostředí zvlášť (až na drobné úpravy) a pro obojí zvolíme vhodné metody. Současně však celý systém navrhneme tak, aby bylo možné v budoucnu některé části vylepšit.

Pořadí, ve kterém se budou části generovat, je velmi důležité. Pokud generujeme trať, je třeba uvažovat také terén, aby nevedla pod něj (kromě tunelu, který zatím v demo verzi neuvažujeme). Bylo by tedy dobré vytvořit terén jako první. Pak by však mohlo být těžké generovat trať tak, aby neopustila prostor vymezený terénem, a navíc by nebylo zřejmé, jak velký terén na začátku zvolit. Mohlo by tak být lepší generovat nejprve trať a podle ní teprve zvolit vhodné rozměry pro terén. V tom případě by se však terén musel přizpůsobovat trati a měli bychom menší kontrolu nad výskytem různých tematických oblastí (viz sekce 2.4.2), které by se měly vyskytovat ve specifické nadmořské výšce. Bude tedy třeba generování trati a terénu vhodně prokládat.

### 3.3.1 Návrh generátoru

Generátor levelů složíme z několika modulů, které poběží jeden po druhém a postupně budou naplňovat sdílenou objektovou reprezentaci levelu. Každý modul bude zodpovědný za dílčí část generování levelu (např. určení terénu, nebo bodů trati) a bude tak mít všechny potřebné parametry přímo u sebe. Díky tomu bude možné snadno nahradit konkrétní modul jiným, který může vytvářet to samé, ale jiným postupem. Také se bude moci měnit pořadí modulů nebo jednotlivé moduly zvlášť (de)aktivovat během testování. Úplně nakonec pak poběží moduly, které převedou objektovou reprezentaci na tu fyzickou (tj. vytvoří mesh pro terén, umístí herní objekty pro obruče apod.).

Tento návrh pomocí modulů nám také umožní snadno prokládat různé části. Můžeme tedy např. vygenerovat nejprve základní trajektorii trati, podle ní určit rozměry levelu, vygenerovat terén a nakonec přizpůsobit výšku trajektorie trati terénu.

Jelikož by mělo generování vycházet ze statistik hráče, které budou uložené ve stavu hráče, musíme nějakým způsobem zajistit správné nastavení těchto parametrů. Generátor (a jeho moduly) by mohl přímo přistupovat do stavu hráče a potřebné hodnoty si sám vyzískat. Tak by byl ale úzce svázaný přímo se statistikami a nebylo by ho možné použít mimo naši hru. Lepší by tedy bylo, aby existoval nějaký prostředník, který by nejprve získal hodnoty statistik (např. *rychlost*), které by pak převedl již na hodnoty konkrétních parametrů generátoru (např. vzdálenost obručí) a ty mu předal. Tak by byl generátor zcela nezávislý a znovupoužitelný.

### 3.3.2 Prostředí

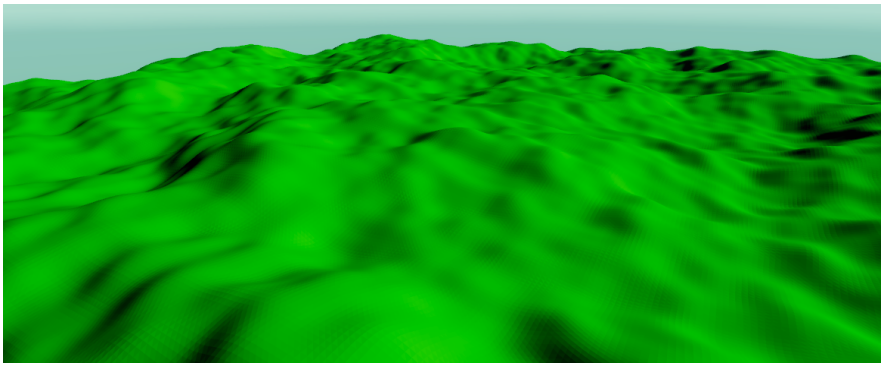
Při generování okolního prostředí nám nijak nezáleží na jeho realističnosti. Může klidně vypadat uměle, ale musí být především dostatečně zajímavé. Navíc

bude rozdělené na tematické oblasti (viz sekce 2.4.2), které by hlavně měly být velké, aby v každé hráč strávil dostatek času.

### Generování terénu

Pro vytvoření terénu bychom mohli využít již existující *Terrain systém*, který je součástí Unity. Ten je však již poměrně zastaralý a navíc vytváří hladké výškové přechody. My bychom se chtěli zaměřit na low-poly vzhled, takže by bylo lepší generovat rovnou vlastní mesh složenou z větších, jednotlivě definovaných trojúhelníků. Mohli bychom tak tedy generovat *height map*, která by se nakonec převedla na mesh.

Existuje celá řada způsobů generování height map. Nejjednodušším z nich je použití samotného *Perlinova šumu* [32], který by se jen převedl do potřebného rozsahu. Výsledný terén však není příliš zajímavý. Běžně používaným rozšířením je pak použití několika oktáv Perlinova šumu, kdy každá má svou vlastní frekvenci a amplitudu. V každém bodě se pak sečtou hodnoty ze všech oktáv. Takový terén už je zajímavější, protože mu jednotlivé oktávy dodávají různé úrovně detailů. Na obrázku 3.1 vidíme ukázku takto vygenerovaného terénu. Jelikož je výsledek v souladu s našimi představami o vzhledu terénu, použijeme uvedený postup jako základ, na kterém pak budeme dále stavět.



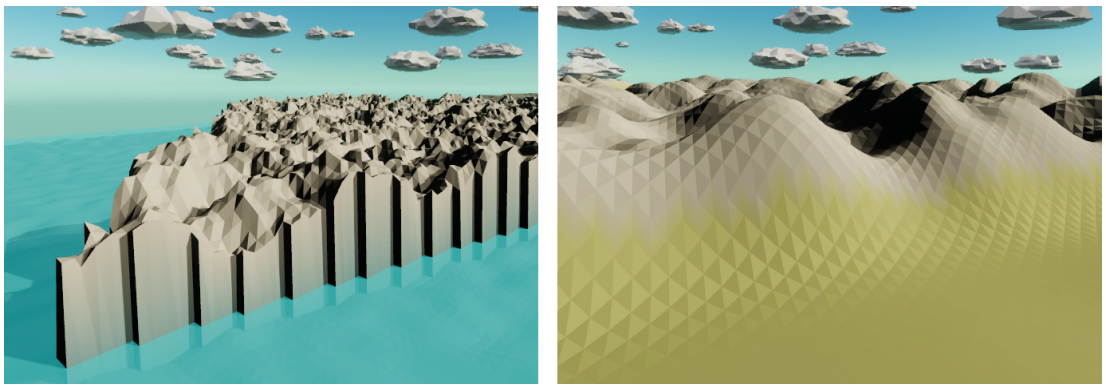
**Obrázek 3.1** Jednoduchý terén vygenerovaný pomocí Perlinova šumu s několika oktávami.

### Určení oblastí

Pro naší hru však potřebujeme generovat prostředí rozdělené do oblastí, které se navíc zpřístupňují postupně, takže musíme být schopni generovat level jen s pevně danou množinou oblastí.

Mohli bychom nejprve vygenerovat terén a teprve poté určit oblasti. Pokud bychom je však určili pomocí rozsahu výšek, nebyli bychom schopni dobře omezit výběr oblastí na konkrétní podmnožinu. Navíc by mohly být vzniklé oblasti jen velmi malé a všechny by používaly stejné parametry šumu (tj. jsou stejně kopcovité), takže by bylo nutné v konkrétních oblastech aplikovat nějaký postprocessing (např. vyhlazení). Mohli bychom také místo výšek určit oblasti nějak nezávisle na terénu. Tak by ale byly všechny oblasti velmi podobné, protože by využívaly stejné parametry šumu a nelišily se rozsahem výšek. Bylo by tedy opět třeba dodatečné zpracování.

Lepší by tak bylo postupovat opačně, tj. nejprve určit výskyty jednotlivých oblastí a až poté v rámci nich generovat height map, přičemž by se pro každou oblast využívaly jiné parametry. Díky tomu bychom mohli určit zvlášť rozsahy výšek i kopcovitost. Jen by bylo třeba zajistit, aby se nakonec vyhladily výškové rozdíly na hranicích oblastí, např. pomocí konvolučních filtrů [33] (na obrázku 3.2 vidíme vlevo, jak vypadají hranice bez vyhlazení, a vpravo, jak vypadají s vyhlazením).



**Obrázek 3.2** Ukázka přechodů mezi sousedními oblastmi, vlevo bez vyhlazení, vpravo s vyhlazením.

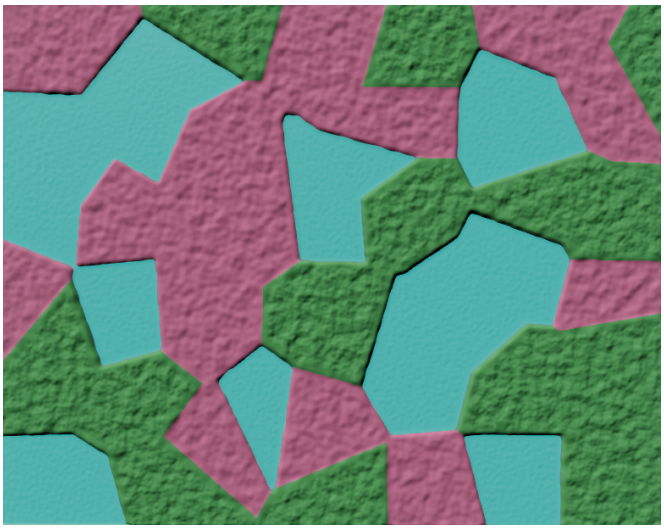
Samotné určení oblastí je pak také možné provést více způsoby. Uvedeme dva nejčastější:

- *Voroného diagram* [34] – Náhodně, ale dostatečně rovnoměrně, se určí body reprezentující středy oblastí, které se jim náhodně přiřadí. Pak se mapa rozdělí na oblasti tak, že každý bod bude náležet oblasti z nejbližšího středu. Výsledné oblasti budou podobně velké, navíc máme dobrou kontrolu nad tím, které oblasti se smějí v levelu použít a jak časté mají být. Hůře se ale určuje, které oblasti spolu mohou sousedit, pokud by bylo třeba.

- *Whittakerův diagram biomů* [35] – Nejprve se vytvoří diagram biomů, který zachycuje výskyt konkrétní oblasti v závislosti na hodnotách dvou parametrů. Tím se také určí, které oblasti spolu mohou sousedit. Pak se pomocí Perlinova šumu určují náhodné hodnoty parametrů pro jednotlivé body mapy a z diagramu se vyčte odpovídající oblast. Snadno se dá určit sousednost a frekvence výskytu, ale výsledné oblasti mají tvar nevhodný pro naši hru. Navíc je třeba nějak rozumně ošetřit případ, kdy se některá oblast v levelu nesmí vyskytovat (je třeba určit jinou).

Pro naši hru je důležité především to, aby byly oblasti velké, různorodé a abychom mohli snadno určovat, které z nich se smějí v levelu vyskytovat. Zvolíme tedy variantu, kdy nejprve určíme oblasti pomocí Voroného diagramu a následně využijeme Perlinův šum s několika oktávami pro vygenerování height map, přičemž v každé oblasti se budou používat jiné parametry tohoto šumu (např. rozsah hodnot, počet oktáv). Pro výběr středů Voroného diagramu použijeme tzv. *stratified random sampling* (popsaný např. v knize *Sampling Techniques* [36]) s pravidelnou mřížkou. Level rozdělíme do mřížky a z každého políčka pak zvolíme jeden bod náhodně jako střed. Tak budou vcelku rovnoměrně rozdělené a navíc nám to usnadní přidělování oblastí ostatním bodům. Kdykoliv budeme pro některý bod

určovat, do jaké oblasti patří, stačí hledat nejbližší střed oblasti jen v nejbližších políčkách mřížky. Na obrázku 3.3 pak vidíme výsledek takto vygenerovaného terénu s přidělenými oblastmi.



**Obrázek 3.3** Ukázka rozdělení levelu do oblastí pomocí Voroného diagramu.

### Prvky prostředí

Každá tematická oblast má také své specifické prvky, které je třeba umístit do prostředí. Toto umisťování však může být provedeno vcelku jednoduše, neboť jak už jsme uvedli dříve, nepotřebujeme dosáhnout realistických výsledků. Můžeme celý level rozdělit do dostatečně jemné mřížky a následně z každého políčka mřížky vybrat jeden náhodný bod, do kterého by se potenciálně mohl umístit nějaký prvek.

Nyní je třeba zvolit jeden konkrétní prvek. Mohli bychom tak mít seznam prvků a pro každý definovat, ve které oblasti se smí vyskytovat. Navíc by byla možnost říci, že na oblasti nezáleží. Každý prvek by navíc mohl mít nějakou váhu. Pro zvolený bod bychom tak nejprve určili všechny možné prvky a z nich vybrali na základě pravděpodobnosti určené vahou. S určitou pravděpodobností by navíc bylo možné nevybrat žádný. Díky tomu bychom mohli také nepřímo řídit hustotu zaplnění každé oblasti zvlášť.

Jednotlivé prvky by navíc měly ještě detailnější možnosti nastavení. Mohly by mít několik různých variant vzhledu a dala by se povolit randomizace velikosti a rotace. Také by se dalo říct, zda se může daný prvek vyskytovat i poblíž hranic oblastí. Příklad takto vygenerovaného prostředí vidíme na obrázku 3.4. Na různá místa jsou tam umístěny stromy, keře, rostliny, kameny a další. S takovým výsledkem jsme spokojeni, takže nastíněný postup použijeme v naší hře.

### Reprezentace oblastí

Ve hře tedy rozlišujeme několik oblastí, přičemž každá má nějaké své vlastnosti, např. barvu, rozsah nadmořských výšek, seznam prvků prostředí, podmínku zpřístupnění. Mohlo by být dobré všechny tyto informace udržovat na jednom

**Obrázek 3.4** Ukázka vygenerovaných prvků prostředí umístěných v levelu.

místě, což by umožnilo lepší rozšiřitelnost. Pro novou oblast by se jen vytvořil nový `ScriptableObject` a vše v něm se nastavilo.

Oblasti by pak ale obsahovaly také informace, které jsou již velmi specifické pro generování nebo konkrétní modul generování. Mohlo by být tedy přínosnější, aby byly takové informace spíše pohromadě pro všechny možné oblasti přímo v modulu. Tak by se dal snáz měnit jeden konkrétní aspekt generování. Jelikož pro generování levelu bude existovat celá řada parametrů, které bude třeba odladit, dá se očekávat, že tento scénář použití bude mnohem častější než přidání nové oblasti. Raději tedy zvolíme tento způsob, kdy se data oblasti rozdělí do odpovídajících modulů generátoru.

### 3.3.3 Trať

Podobně jako v případě generování prostředí, také u generování trati zvolíme spíše jednodušší postup, který budeme schopni dobře odladit. Pomocí *náhodné procházky* [37] vygenerujeme seznam důležitých bodů trati, tj. bodů, ve kterých budou nakonec umístěné obruče nebo kontrolní body. Současně však budeme brát v úvahu také již vygenerovaný terén a trať mu přizpůsobíme, aby byly obruče vždy v určité minimální výšce nad ním.

V každém kroku náhodné procházky náhodně určíme směr do dalšího bodu tak, že vyjdeme z předchozího směru, který otočíme o náhodný úhel. Parametrizujeme přitom celkový počet bodů, maximální možný úhel změny, vzdálenost následujícího bodu a maximální možnou výšku bodu (závislost na statistikách hráče je popsána v sekci 2.4.3).

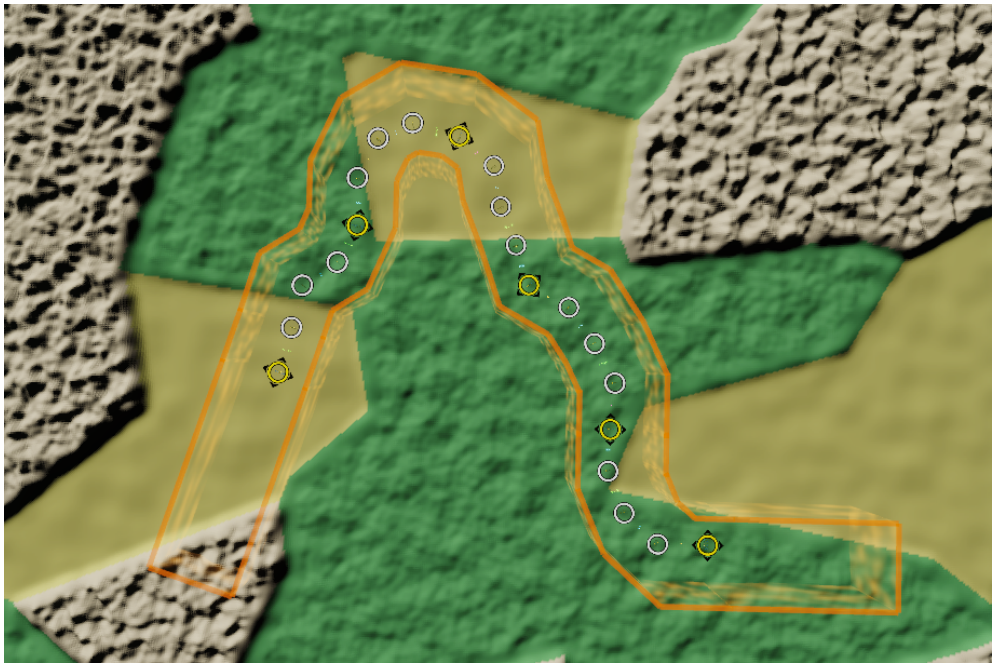
Pokud bychom další směr vybírali pokaždé jen zcela náhodně, výsledné tratě by neměly příliš zajímavý celkový tvar, protože mezi následujícími úseky neexistuje žádná závislost. Mohli bychom tak mírně podpořit zatáčení stejným směrem jako v předchozím kroku. Tím mohou vznikat větší oblouky. Pokud navíc zvýšíme pravděpodobnost volby většího úhlu oproti menšímu, výsledné tratě mají dostatečně zajímavý tvar.

### Prevence překřížení tratí

Pokud jsou generované tratě dlouhé a maximální povolené úhly změny směru velké, může často docházet k tomu, že se náhodná procházka překříží. Jelikož je však trať určená také ochrannou bariérou po stranách (viz sekce 2.4.1), překřížení by způsobilo její zablokování. Musíme tedy zvolit vhodný způsob, jak mu zabránit ještě během generování.

Vcelku přímočarým řešením by byla implementace *backtrackingu*. Pomocí výpočtu průsečíku jednotlivých úseků trati bychom detekovali, zda se trať protíná. Pokud ano, pak se vrátíme o několik kroků zpět (lze postupně inkrementovat) a zkusíme to znovu. U delších tratí by však mohlo být třeba vracet se velmi daleko a velmi často.

Dalším jednoduchým řešením je omezovat během generování rozsah úhlů pro změnu směru tak, aby k překřížení dojít nemohlo. Mohli bychom si globálně akumulovat úhel otočení a nedovolit, aby překročil 90° na kteroukoliv stranu. Tím se sice v některých případech nebude pokračovat dále, i kdyby bylo možné se z toho později zotavit, ale řešení funguje spolehlivě a rychle. Na obrázku 3.5 pak vidíme, jak takto vygenerovaná trať může vypadat. Začíná se vpravo a kvůli omezení maximálního úhlu trať nikdy nezatočí tak, že by se vracela zpět. I přesto má ale výsledná trať dostatečně zajímavý tvar. Implementujeme tedy toto řešení.



**Obrázek 3.5** Ukázka vygenerované trati umístěné v prostředí. Oranžově jsou vykresleny ochranné bariéry kolem trati, bílé kroužky jsou obruče a žluté kroužky jsou kontrolní body.

### Prevence zanoření v terénu

Jak jsme řekli již na začátku této sekce, generování terénu a trati se bude prokládat. Nakonec pak bude třeba vygenerovanou trať ještě přizpůsobit terénu, aby se nestalo, že by byla v terénu částečně zanořená. Současně by však neměla

trať pouze kopírovat tvar terénu, měla by mít možnost také nezávisle stoupat výš. Pořád je ale třeba dodržet nějaký maximální úhel změny směru.

Měli bychom se tedy podívat pro každý bod trati, jak vysoké jsou body terénu v blízkém okolí. Pokud není bod trati v dostatečné výšce nad nimi, posunout ho výš, ale současně tuto změnu výšky rozdistribuovat také do okolních bodů trati pro zajištění nepříliš velkých úhlů. Pokud navíc zajistíme, že budou body trati vždy dostatečně daleko od sebe, budou se úhly dodržovat snáz.

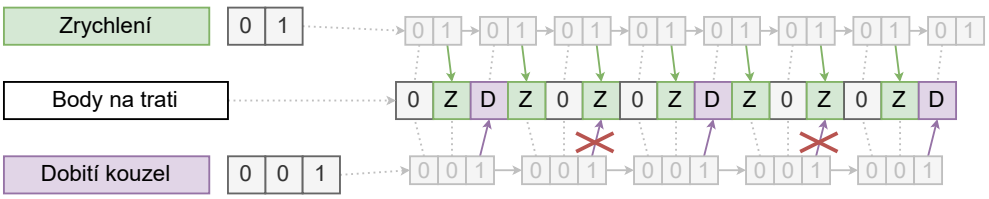
### Bonusy

Již v sekci 2.4.1 jsme si řekli, že se na trati budou kromě obručí a kontrolních bodů vyskytovat také bonusy. Jelikož bychom chtěli jejich sbírání hráči co nejvíce usnadnit, budeme je jednoduše umisťovat přímo mezi obruče pomocí lineární interpolace jejich pozice. Nakonec se pouze ujistíme, že výsledný bod není pod terénem a případně upravíme.

Nejprve tedy mezi každými dvěma obručemi (včetně kontrolních bodů) určíme pevně daný počet bodů, do kterých bude možné umístit bonusy. Pak musíme nějakým způsobem určit, jaký typ bonusu se bude vyskytovat ve kterém bodě. Pro každý typ bonusu tak popíšeme jeho výskyt pomocí něčeho, co nazveme *vzor*. Bude se jednat o pole `bool` hodnot. Následně budeme procházet potenciální body umístění bonusů a přitom neustále dokola procházet přes vzor uvažovaného bonusu. Pokud je ve vzoru `true` a aktuální bod je prázdný, bonus tam přiřadíme a pokračujeme dál. V ostatních případech rovnou pokračujeme dál.

Jednotlivé typy bonusů pak budou seřazeny dle priority. V daném pořadí se budou umisťovat. Kdykoliv už tedy v nějakém bodě bude přiřazený bonus vyšší priority, neumístí se tam žádný jiný.

Ukážeme si to na příkladu, který je schematicky zakreslený také na obrázku 3.6 (místo hodnot `false` a `true` však používá `0` a `1`). Řekněme, že máme výskyt zrychlujícího bonusu určený pomocí vzoru `[false, true]` a tento bonus má nejvyšší prioritu. Pak máme výskyt bonusu pro dobití kouzel určený vzorem `[false, false, true]`. Začneme zrychlujícím bonusem. Jelikož jsou zatím všechny body prázdné, umístíme ho jednoduše do každého druhého bodu, jak říká vzor. Poté budeme umisťovat bonus pro dobití kouzel. Jeho vzor říká, že by měl být na každém třetím bodě, ale na každém druhém už máme zrychlující bonus. Každý šestý bod tak bude obsahovat zrychlující bonus, který má vyšší prioritu. Výsledné výskyty budou `[0, Z, D, Z, 0, Z]` opakující se dokola, kde `Z` značí zrychlující bonus a `D` značí bonus pro dobití kouzel.




**Obrázek 3.6** Příklad přiřazení bonusů k vygenerovaným bodům na trati podle jejich vzorů.

#### Oblasti trati

Jak bylo řečeno v sekci 2.4.2, kromě oblastí terénu se budou určovat také oblasti trati, které ovlivňují pouze vlastnosti trati a nijak nesouvisí s terénem. V demo verzi budeme uvažovat pouze jedinou oblast, a to *Nad mraky*. Tato oblast vyniká tím, že obruče v ní jsou umístěné velmi vysoko. Stačí tedy v již vygenerované trati náhodně zvolit některou část, vyzvednout ji nahoru (se zohledněním maximální možné výšky) a nakonec dorovnat výšky okolních bodů trati, aby nebyly rozdíly příliš velké.

### 3.3.4 Optimalizace levelu

Pokud je statistika *vytrvalosti* vysoká, generované tratě jsou vcelku dlouhé a celý generovaný level je pak velmi rozsáhlý. Pokud navíc zvážíme, že naše hra se odehrává ve 3D světě, je zřejmé, že by bylo vhodné implementovat určité optimalizace, díky kterým se zefektivní renderování a hra poběží také na méně výkonných zařízeních.

Existuje celá řada možných optimalizací, které bychom mohli v naší hře zahrnout. Nyní tedy některé popíšeme a na závěr uvedeme, které z nich skutečně implementujeme.

#### Frustum culling

Velmi základní optimalizací je tzv. *frustum culling*, kdy se pro objekty mimo viditelnou oblast (tzv. *view frustum* [38]) vůbec nespouští renderování. Unity toto řeší automaticky za nás, ale pomocí parametrů můžeme vhodně nastavit minimální a maximální vzdálenost, kterou kamera vidí. Snížením maximální vzdálenosti tak můžeme značně omezit počet zpracovávaných objektů. Navíc můžeme přidat mlhu do vzdálenějších částí, abychom zakryli bod zlomu.

#### Layer cull distances

Maximální renderovanou vzdálenost nemusíme nastavovat pouze pro kameru celkově. Unity nám totiž dává možnost jemnější kontroly, kdy ji můžeme nastavit zvlášť pro každou vrstvu objektů [39]. Díky tomu můžeme např. malé objekty přestat vykreslovat v ještě menší vzdálenosti. V naší hře bychom tedy mohli menší prvky prostředí (např. kameny, houby) umístit do separátní vrstvy, pro kterou bychom pak nastavili menší maximální vzdálenost.

#### Rozdělení terénu

Pokud se celý terén skládá z jediné meshe, obsahuje velké množství trojúhelníků. Všechny se pak musí posílat na GPU pro renderování, i když je viditelná pouze malá část z nich. Mohli bychom tedy terén rozdělit do bloků, aby měl každý svou vlastní mesh. Díky tomu by mohl lépe fungovat culling celých meshí, které jsou mimo viditelnou oblast kamery, a GPU by mohla zpracovávat mnohem menší množství dat.

### Deaktivování vzdálených objektů

Renderování bychom mohli optimalizovat ještě více, pokud bychom se nespoléhali pouze na základní frustum culling (popsaný výše), ale navíc bychom úplně deaktivovali objekty, o kterých víme, že jsou dál od hráče (a tedy určitě nejsou viditelné). Tím by nebylo třeba dané objekty vůbec uvažovat.

Mohli bychom to spojit se zmíněným rozdělením terénu do bloků. Při generování levelu bychom ukládali také důležité objekty (tj. prvky prostředí, prvky trati, bonusy) do jednotlivých bloků. Pak bychom sledovali pozici hráče a při přechodu mezi bloky bychom aktualizovali viditelnost objektů náležejících blokům kolem hráče. Objekty ve vzdálených blocích bychom deaktivovali, objekty v blízkých blocích bychom naopak aktivovali.

Bylo by však třeba otestovat, že tato režie navíc nebude situaci naopak zhoršovat, aby nedošlo k zaseknutí vždy na rozhraní bloků. Také bychom museli zajistit, že (de)aktivace objektů nebude narušovat hru. Vyvolávaly by se totiž metody `OnEnable()` či `OnDisable()` a pro neaktivní objekt by neběžela metoda `Update()`. Mohlo by tak být bezpečnější deaktivovat pouze renderery a collidery.

### LOD

Předchozí postup bychom mohli ještě dále vylepšit, pokud bychom pro 3D modely zavedli několik různých úrovní detailů, které by se lišily především v počtu trojúhelníků. S rostoucí vzdáleností od hráče by se pak volila úroveň s čím dál tím menšími detaily, až by se nakonec v nějaké maximální vzdálenosti objekt úplně deaktivoval. Unity přímo pro to nabízí podporu [40], takže bychom to nemuseli řídit sami.

Tento postup bychom mohli zavést také pro terén. Pokud bychom ho rozdělili do bloků, mohli bychom pak pro různé bloky renderovat mřížku bodů v různém měřítku tak, aby bloky dál od hráče byly hrubší, zatímco ty blíž k hráči by byly jemnější. Museli bychom ovšem vhodným způsobem vyřešit přechody mezi bloky s různým měřítkem.

### Billboardy

Jelikož bychom chtěli pro naši hru použít low-poly styl, všechny modely budou už ze základu tvořené menším počtem trojúhelníků. Různé úrovně detailů modelů by pak sice jistě snížily počet renderovaných trojúhelníků, ale nemusely by způsobovat výrazné zlepšení. Mohli bychom tedy navíc jako nejnižší úroveň použít tzv. *billboard* [41], kdy se místo skutečného modelu použije pouze jeho obrázek jako textura na obdélníkové ploše tvořené dvěma trojúhelníky.

Plocha s texturou by se navíc měla neustále otáčet směrem ke kameře. Pokud by však byla v dostatečné vzdálenosti a použili bychom rozdělení levelu do bloků, mohlo by stačit plochu natočit ke středu bloku, ve kterém se právě nachází hráč. Tak by se orientace měnila pouze při změně bloku.

Obzvlášť užitečné by pak bylo, kdybychom dokázali vytvářet textury pro billboardy automaticky. Mohli bychom např. vytvořit scénu, do které bychom postupně načítali modely a pomocí vhodně umístěné kamery bychom je pak renderovali do textury. Navíc bychom mohli objekt renderovat z několika různých natočení, mezi kterými by se dalo ve hře přepínat dle aktuálního směru pohledu.

#### Neaplikovatelné optimalizace

Unity podporuje tzv. *occlusion culling*, který spočívá v tom, že se nespouští renderování pro objekty, které jsou zakryté jinými. Vývojář mu k tomu však musí poskytnout dodatečné informace. Musí říct, které objekty jsou tzv. *occluder*, tedy nějaký velký objekt, který může zakrývat jiné, a které jsou tzv. *occludee*, tedy objekt, který může být zakryt jiným. V naší hře by terén mohl být occluder, protože je poměrně velký a některé prvky prostředí mohou být schované za oblastmi s vyšší nadmořskou výškou, pokud hráč neletí příliš vysoko.

Nyní však narážíme na omezení, že pouze statické objekty mohou být occluder [42]. Objekty ale můžeme označit jako statické pouze mimo runtime, aby bylo možné předpočítat potřebná data, což pro nás není možné, protože level procedurálně generujeme za běhu.

#### Poznámky k implementaci

Nejprve jsme implementovali prototyp se všemi herními mechanikami zcela bez optimalizací, protože je hra už sama o sobě komplexní a nechtěli jsme tak zbytečně dělat předčasné optimalizace. Následně jsme hru otestovali také na dalším zařízení. Jednalo se o notebook nižší střední třídy s integrovanou grafickou kartou. Bez jakýchkoliv optimalizací byl pohyb vcelku trhaný, protože se nedosahovalo příliš vysokého počtu snímků za sekundu. Když jsme deaktivovali všechny prvky prostředí, situace se nijak nezlepšila, ale po deaktivaci terénu jsme pozorovali výrazné zlepšení. Na základě toho jsme se tedy rozhodli zavést implementaci rozdělení terénu do bloků.

Jelikož jsme pak navíc v kombinaci s vhodnými parametry pro frustum culling dosáhli velmi dobrých výsledků, již jsme v dalších optimalizacích prozatím nepokračovali. Jak je však zřejmé z výčtu možností, zůstává stále spoustu prostoru pro zlepšení. Pokud by se tedy ukázalo, že je třeba výkon ještě vylepšit, přidáme další optimalizace.

## 3.4 Systém kouzel

V sekci 2.7 jsme popsali návrh související s kouzlením ve hře. Mimo jiné jsme uvedli také seznam kouzel, která bychom chtěli implementovat. Nyní bychom si měli promyslet, co vše budeme od jejich implementace vyžadovat a jakým způsobem bychom tedy mohli celý systém kouzlení navrhnout.

Každé kouzlo má své jméno, ikonku, popis (tj. co přesně dělá), barvu (použije se pro vizuální efekty spojené s jeho sesláním) a cenu zakoupení v obchodě. Navíc patří do jedné ze čtyř kategorií. Pro sesílání během závodu je pak důležitá jeho cena seslání a dobíjecí doba. Každé kouzlo se navíc používá na určitý cíl, který může být různého typu (konkrétně sesílající závodník, soupeř, jiný objekt, nebo pozice/směr). Sesláním kouzla se aplikuje nějaký jeho efekt.

### 3.4.1 Reprezentace kouzla

Pro jednotlivá kouzla ve hře musíme zvolit vhodnou reprezentaci. Jelikož má každé kouzlo nějaká data (např. název, ikonka, cena seslání), hodilo by se vytvořit

pro ně `ScriptableObject`, takže by pak každé kouzlo mělo svou vlastní instanci jako asset v projektu a tato instance by se sdílela, kdykoliv by bylo potřeba. Současně však musíme definovat efekt kouzla, který musí být zapsaný přímo někde v kódu. Pak by mohlo být lepší vytvořit z kouzel prefab, ve kterém by byla komponenta s efektem. Z různých požadavků kladených na systém tak vyplynuly 3 možné způsoby implementace:

1. Pro každé kouzlo bychom měli prefab, na kterém by byly dvě komponenty – jedna obecná pro veškerá data kouzla, jedna jako odvozená třída od obecného efektu kouzla (např. s abstraktní metodou pro vyvolání efektu). Pak by se pracovalo s prefaby jako s kouzly, takže by se mohl zavolat efekt z komponenty.

2. Vytvořili bychom `ScriptableObject` pro samotná data. Pro každé kouzlo bychom pak měli bokem prefab a na něm komponentu pro efekt kouzla (odvozený od nějakého obecného předka) s datovou položkou pro data kouzla (jako `ScriptableObject`). Pak by se opět pracovalo s prefaby jako s kouzly, takže by bylo možné zavolat efekt z komponenty.

3. Pro každé kouzlo bychom vytvořili prefab, který by měl na sobě komponentu s konkrétním efektem kouzla. Pak bychom měli `ScriptableObject`, který by obsahoval data a navíc by měl datovou položku pro komponentu efektu, takže by se do ní dosadil prefab. Pracovalo by se tedy se `ScriptableObject`em a efekt by se volal z předané komponenty na prefabu.

První způsob je nejjednodušší a nejpřímočařejší, všechny informace jsou na jednom místě, ale v Inspectoru by přehlednost kazily další části (např. `Transform` komponenta, část pro `Tag` a `Layer`). Druhý a třetí způsob zavádí mezivrstvu navíc v podobě `ScriptableObject`u. Druhý způsob by pak byl nejméně přehledný, protože by data nebyla přímo na stejném místě. Oproti tomu třetí způsob by umožňoval hezké zobrazení `ScriptableObject`u v Inspectoru, vše by bylo hezky pohromadě, jen by nebyl vidět přímo efekt kouzla. Kdybychom se tedy chtěli podívat na parametry efektu, museli bychom si ho zvlášť otevřít, ale pak bychom zase neviděli data kouzla (jedině se dvěma instancemi Inspectoru vedle sebe). Zvolíme tedy první způsob, který nám umožní jednodušší údržbu, protože všechny potřebné parametry uvidíme rovnou v prebafu kouzla.

Pro různé druhy kouzel bychom pak mohli navíc vytvořit prefab varianty, které by mohly mít již připravené některé společné části. Mohla by tak vzniknout prefab varianta pro kouzla sesílaná na sebe sama a pro kouzla sesílaná někam pryč. Podobně by se mohla vytvořit hierarchie pro implementaci efektů kouzel. Některé efekty jsou totiž pouze jednorázové (např. vyčarování zdi), jiné mají délku trvání (např. zrychlení). Mohli bychom tak abstrahovat společné řízení.

### 3.4.2 Vizuální efekty

V různé okamžiky sesílání kouzla bychom také měli použít různé vizuální efekty. Bylo by vhodné vytvořit pro ně společný návrh, aby se se všemi pracovalo skrz stejné rozhraní. Díky tomu bychom mohli snadno dosazovat požadované vizuální efekty přímo do komponenty, která by řídila sesílání kouzla a mohla by

tak vizuální efekty spouštět ve vhodné okamžiky. Obecně bychom chtěli podporovat čtyři různé vizuální efekty:

- při seslání kouzla na sebe sama (objevil by se nějaký efekt kolem sesílajícího závodníka),
- při seslání kouzla směrem pryč (k cíli kouzla by se pohyboval objekt po určité trajektorii, zanechával by za sebou nějakou stopu),
- při nárazu do cíle,
- během působení na cílového závodníka (pokud kouzlo bude po určitou dobu ovlivňovat nějakého závodníka, mohlo by se to indikovat vizuálním efektem kolem něj).

Efekt při seslání na sebe sama by se přitom dal spojit s efektem po nárazu do cíle s tím, že by vlastně kouzlo nikam neputovalo a rovnou narazilo.

Různé vizuální efekty pak mohou mít různou podobu. Mohli bychom si však připravit hierarchii komponent, které by poskytovaly řízení některých základních chování. Příkladem by mohl být vizuální efekt, který se po skončení sám zničí. Dále třeba efekt, který má nějakou délku trvání a během ní se neustále aktualizuje (např. pro interpolaci parametrů shaderu nebo zvětšování nějakého objektu). Nebo efekt, který se ve skutečnosti stará o celý seznam vizuálních efektů, přehrává je všechny naráz a skončí, až když skončí úplně všechny. Pro konkrétní vizuální efekt bychom pak mohli mít prefab s konkrétní komponentou odvozenou od některého z obecných předků. V prefabu by navíc byly potřebné vizuální efekty vytvořené pomocí nástrojů dostupných v Unity, např. komponenta `ParticleSystem`, komponenta `TrailRenderer` a *Shader Graph*.

### 3.4.3 Seslání kouzla

Nyní si popíšeme, jak přesně bude probíhat seslání kouzla v čase. Pro jeho řízení vytvoříme separátní komponentu, která bude zodpovědná za celkový průběh od zahájení seslání až po konec účinku. Bude tedy mít na starosti spouštění vizuálních efektů ve správné okamžiky a také spuštění samotného funkčního efektu kouzla, který se bude nacházet v oddělené komponentě.

Jakmile se vyvolá seslání kouzla, vytvoří se instance prefabu daného kouzla, již s komponentou pro řízení seslání. Pokud cílem není sám sesílající, začne se provádět vizuální efekt pro seslání kouzla směrem pryč (tj. objekt kouzla bude směřovat k cíli po určité trajektorii). Jakmile je v cíli (v případě seslání na sebe sama ihned), provede se vizuální efekt nárazu do cíle (případně efekt seslání na sebe sama). Následně se vyvolá samotný funkční efekt kouzla (se zohledněním toho, zda je jednorázový, nebo má dobu působení) s případným vizuálním efektem působení (pokud kouzlo ovlivňuje závodníka po určitou dobu). Jakmile skončí sesílání kouzla, původně vytvořená instance se zničí.

#### Trajektorie kouzla

Pokud se kouzlo pohybuje ke svému cíli, nemělo by následovat přímou trajektorii, ale určitým způsobem se od ní vychylovat. Různá kouzla by pak mohla

mít různý tvar této trajektorie. Mohli bychom tedy mít speciální třídu, která by vracela odchylku od referenčního bodu na přímé trajektorii, který bychom zadali pomocí vzdálenosti od zdroje. Pak bychom se pravidelně ptali na odchylku a dle ní měnili pozici objektu sesílaného kouzla. Nakonec bychom mohli mít různé implementace výpočtu pro různá kouzla (např. pro vytvoření spirály).

Tímto přístupem sice možné trajektorie omezíme pouze na spojité cesty bez možnosti větvení, ale i přesto budeme mít dost prostoru pro dostatečnou rozmanitost.

#### Zdroj a cíl

Pro seslání kouzla je také třeba určit počáteční a cílový bod, mezi kterými se má pohybovat. Co se týče počátečního bodu, nevypadalo by nejlépe, kdybychom vycházeli čistě z počátku daného závodníka. Lepší by tedy bylo mít možnost bod určit např. relativně k němu. Podobně by mohlo být dobré umět přesněji určit také cílový bod, pokud se kouzlo sesílá na nějaký objekt.

Tyto body bychom mohli označit pomocí podobjektů, takže by se poté uvažovala jejich pozice. Tak by se ovšem zvětšila a znepřehlednila hierarchie herního objektu. Navíc bychom měli být schopní najít tento objekt dynamicky na zvoleném cílovém objektu. V případě podobjektu bychom ho však museli buď hledat podle jména, nebo si referenci uložit do nějaké komponenty a pak získávat tu podle jejího typu.

Jednodušší by tedy bylo bod rovnou označit pomocí komponenty, která by obsahovala datové položky pro zadání relativního offsetu vůči počátku. Pak bychom mohli takovou komponentu získat velmi snadno. Nevýhoda tohoto přístupu je, že by nebyl bod přímo viditelný v editoru. Pomocí třídy `Gizmos` však můžeme bod jednoduše zvýraznit.

### 3.4.4 Návrh kouzlení

Veškeré kouzlení z pohledu závodníka bychom pak mohli rozdělit na několik podjednotek. Některé by poskytovaly společný základ pro hráče i soupeře, jiné by pak měly různé konkrétní implementace.

- Jednotka pro sesílání kouzel – Tato jednotka bude obsahovat informace o tom, jaká kouzla jsou k dispozici, jaký je jejich aktuální stav a kolik má závodník many. Navíc bude poskytovat veřejné API pro změnu aktuálního kouzla a seslání kouzla.

- Jednotka pro zpracování vstupu – V případě hráče bude reagovat na vstup z klávesnice a myši a převádět tyto události na akce jednotky pro sesílání kouzel. V případě soupeřů pak bude obsahovat implementaci umělé inteligence, která bude opět využívat akce jednotky pro sesílání kouzel.

- Jednotka pro detekci cílů – Dle aktuálně vybavených kouzel se budou detekovat vhodné cíle v okolí závodníka.

- Jednotka pro výběr cílů – Ze všech potenciálních cílů z jednotky pro detekci cílů se vybere jeden aktuální. Pro hráče to bude ten, který je nejblíž středu obrazovky. Pro soupeře se mohou využívat různé strategie.

- Jednotka pro správu přicházejících kouzel – Tato jednotka bude mít přehled o tom, jaká kouzla na závodníka právě míří, z jakého směru a jak jsou daleko. Pro hráče se pak na základě toho mohou vykreslit indikátory na obrazovce (viz sekce 2.7).

## 3.5 Chování soupeřů

V požadavku **P22** jsme si určili, že by se měli soupeři přizpůsobovat hráči. Podobně jako v případě generování levelu by také chování soupeřů mělo vycházet ze statistik hráče (více v sekci 2.3.2). Ty by se tedy využily pro inicializaci, ale následně by se chování ještě dále měnilo dle aktuální situace v závodě. Také jsme již zmínili, že by soupeři měli dělat přesvědčivé chyby a jejich míra by závisela právě na statistikách. Nyní si tedy řekneme, jak přesně by se mohlo chování soupeřů řídit.

Ještě než se však pustíme do detailů, nejprve shrneme, co vše by měli soupeři umět a co k tomu budou potřebovat:

- prolétávat obručemi a kontrolními body až do cíle – musí vědět, kde se nachází jednotlivé prvky trati,
- sbírat bonusy – musí vědět, kde se nachází bonusy a jakého jsou typu,
- vyhýbat se překážkám – musí být schopni vnímat své okolí a předcházet kolizím,
- letět nad zemí – musí znát reprezentaci terénu v levelu, aby věděli, jak vysoký kde je,
- sesílat kouzla (tj. vybrat vhodné a dostupné kouzlo, zvolit vhodný cíl) – musí mít přiřazená kouzla ve slotech, umět určit jejich stav a umět detekovat cíle.

### 3.5.1 Statistiky a chybovost

Soupeři by tedy měli být schopní dělat různé chyby na základě statistik, které jim budou přiřazeny. Mohli bychom postupovat obecně tak, že vezmeme doplněk statistiky jako míru chybovosti, tj. něco jako pravděpodobnost, se kterou bude soupeř dělat chybu odpovídající této statistice. Ve skutečnosti se však nebude jednat o pravděpodobnost. Pro konkrétní druhy chyb se totiž bude hodnota využívat různým způsobem (např. pro vyčtení hodnoty z křivky). Zajistí se tak, aby maximální hodnota neznamenala třeba skutečně minutí úplně všech obručí na trati.

Během implementace přitom budeme postupovat iterativně. Nejprve se pokusíme vytvořit takové soupeře, kteří budou jednat víceméně bez chyb a bude jen velmi těžké je porazit. Toto experimentálně ověříme a teprve poté postupně zavedeme jednotlivé druhy chyb.

Nyní si uvedeme výčet různých druhů chyb v závislosti na hodnotě dané statistiky. Můžeme si povšimnout, že tyto chyby odpovídají zhruba kritériím, která se vyhodnocují při výpočtu hráčových statistik po dokončení závodu (viz sekce 2.6).

- *Rychlost* – Hodnota bude určovat, jak moc bude soupeř zpomalovat a vynechávat zrychlující bonusy. Zohlední se ale také poměr aktuální maximální rychlosti a maximální rychlosti dostupné ve hře (např. pokud soupeř ani nemůže letět příliš rychle, pak se míra chyby sníží).
- *Obratnost* – Hodnota bude udávat, jak často soupeř narazí do překážky nebo poletí špatným směrem. Také určí, jak dlouho bude ještě pokračovat dosavadním směrem, když se rozhodne pro změnu cílového bodu.
- *Přesnost* – Hodnota určí, jak často bude soupeř míjet své cíle (např. obruče nebo bonusy) a také narážet do překážek. Pro jednoduchost nebudeme uvažovat možnost minout kontrolní body.
- *Magie* – Hodnota ovlivní, jak často bude soupeř míjet bonusy pro manu a dobití kouzel a v jakých intervalech bude provádět rozhodování o sesílání kouzel. Také jak často nesešle kouzlo, i když bude připravené k seslání, a kolik ze svých vybavených kouzel bude skutečně používat.
- *Vytrvalost* – Tato statistika nebude mít žádný vliv na chování soupeřů.

#### Inicializace hodnot

Na začátku závodu bychom měli nějak určit úroveň schopností jednotlivých soupeřů. V sekci 2.3.2 jsme popsali, že na trati bude celkem 5 soupeřů a chtěli bychom dosáhnout toho, aby byli 2 o trochu lepší než hráč, 1 zhruba srovnatelný, 1 o trochu horší a nakonec 1 ještě horší. Měli bychom tedy vycházet ze statistik hráče, které bychom však upravili pro dosažení různých úrovní schopností.

Změna by mohla být absolutní, tj. přičítala/odečítala by se konkrétní hodnota a výsledek by se pak oříznul do povoleného rozsahu. To by ovšem nefungovalo dobře u extrémů, protože kvůli zaříznutí by pak mohlo vzniknout více soupeřů se stejnými hodnotami. Lepší by tedy bylo počítat změnu relativně. Pokud bychom chtěli statistiku navýšit, přičteme určité procento z doplňku hodnoty. Pokud ji chceme naopak snížit, odečteme určité procento z hodnoty. Díky tomu nikdy nepřesáhneme povolený rozsah.

### 3.5.2 Adaptabilita

Samotná inicializace soupeřů dle hráčových aktuálních statistik nestačí. Statistiky nemusí být dostatečně objektivní a zachycovat opravdu dobře hráčovy schopnosti. Pak by však soupeři setrvávali na stále stejné, špatně zvolené úrovni. Je tedy potřeba zajistit, aby se přizpůsobovali také tomu, jak hráč letí právě teď. Mohli bychom tak lépe řídit, kdy se mají soupeři zlepšit, nebo zhoršit, aby se drželi pořád v okolí hráče a závody tak byly napínavější.

#### Rubber banding

Tomuto konceptu se říká *rubber banding*. Můžeme si to představit tak, jako by mezi hráčem a soupeřem byla neviditelná gumička. Čím víc se hráč vzdálí od soupeře, tím víc to soupeře táhne směrem k hráči. Stejně tak v závodě, čím víc bude hráč před soupeřem, tím víc by se měl soupeř snažit hráče dohnat. A naopak

čím dál by byl hráč za soupeřem, tím spíš by měl soupeř trošičku ubrat, aby se k němu mohl hráč více přiblížit. Je však třeba vhodně odladit parametry, aby toto chování nebylo zřejmé.

Nic Melder v *A Rubber-Banding System for Gameplay and Race Management* [43] rozlišuje rubber banding dvou typů, přičemž dle jeho názoru je v praxi nejlepší využít kombinaci obou:

- *power-based* – založený na modifikaci zrychlení a maximální rychlosti soupeřů (měl by se použít, až když následující nemá žadný vliv),
- *difficulty-based* – založený na modifikaci úrovně schopností soupeřů.

Power-based rubber banding by tedy znamenal jednoduše to, že bychom uměle měnili maximální rychlost soupeřů v závislosti na jejich pozici oproti hráči. Pokud by byli hodně pozadu, dovolili bychom jim letět rychleji než obvykle, aby měli šanci hráče dohnat. Rozdíly by však neměly být příliš velké a aplikovat se až opravdu daleko od hráče, aby nebylo poznat, že podvádí.

Difficulty-based rubber banding by v našem případě odpovídal tomu, že bychom přizpůsobovali hodnoty statistik, které mají soupeři přiřazené. Tím by se upravovala míra chyb. Pokud by tedy byli daleko za hráčem, nedělali by téměř žádné chyby. V blízkosti hráče by dělali chyby odpovídající počátečním hodnotám statistik. Daleko před hráčem by dělali chyb více, aby se k nim hráč postupně přibližoval.

Podobný princip se popisuje také v článku *The Pure Advantage: Advanced Racing Game AI* [44]. Nazývá se *Dynamic Competition Balancing* a využívá tzv. *skills*, což zhruba odpovídá našim statistikám – také mají počáteční hodnoty, mění se podle aktuálního stavu závodu a ovlivňují výkon soupeřů. Navíc v článku dobře popisují, jaké by měly být splněny požadavky. Např. že by skills (tedy pro nás statistiky) měly mít dostatečně velký rozsah (od velmi dobré AI až po velmi špatnou), aby se mohly dostatečně přizpůsobit, a vliv na chování by měl být ideálně spojitý a rovnoměrně rozdělený. To také odpovídá naší zamýšlené implementaci vlivu statistik na chybovost soupeřů.

### Normalized distance raced

Pro přizpůsobování pak musíme být schopni vhodně spočítat vzdálenost soupeře od hráče. Na to bychom mohli využít tzv. *normalized distance raced*, popsanou v *Capture and Analysis of Racing-Gameplay Metrics* [45], která určuje, jak daleko v závodě agent je. V naší hře bychom se mohli podívat, kde je pro daného závodníka následující obruč, sečíst vzdálenosti všech obručí (včetně startu) až do ní a nakonec odečíst vzdálenost závodníka od této následující obruče. Jako mikrooptimalizace by bylo možné součty předpočítat do pole.

### Race scripts

V článku *The Pure Advantage: Advanced Racing Game AI* [44] také popisují tzv. *race scripts*. Jedná se o koncept, kdy se vytvoří jakýsi scénář, jak by se měl závod vyvíjet (např. jak budou během závodu soupeři rozdělení do skupin, jak se bude v průběhu závodu zhruba měnit umístění hráče apod.). V praxi se pak implementuje tak, že má každý soupeř určený bod v určité vzdálenosti před hráčem,

nebo za ním. Schopnosti soupeřů se pak přizpůsobují na základě vzdálenosti od tohoto bodu (ne od hráče samotného), kdy se ho pokoušejí dosáhnout. Daný bod (určený relativně k hráči) se navíc může dynamicky měnit během závodu, což umožní ještě větší kontrolu nad chováním soupeřů.

Přesně takový způsob parametrizace soupeřů využijeme také v naší hře. Podle toho, zda má být soupeř lepší, nebo horší než hráč, bude mít přiřazenou křivku, která bude popisovat vzdálenost relativně k hráči v průběhu celého závodu. Statistiky soupeře se pak budou měnit dle vzdálenosti k tomuto cílovému bodu.

#### Shrnutí

Soupeři tedy začnou s počátečními hodnotami statistik, které se určí relativní změnou hráčových statistik dle toho, jaký by měl soupeř obecně být (např. trochu lepší než hráč). Z těchto hodnot se odvodí rozsah, v rámci kterého bude mít soupeř povoleno přizpůsobovat se situaci v závodě. Toto přizpůsobení pak bude probíhat na základě toho, jestli je soupeř v daném okamžiku před určitým bodem relativně k hráči, nebo za ním.

Mohli bychom pak popsat čtyři různá chování, která bychom soupeřům přiřadili. Pomocí nich bychom zajistili, že by byl průběh závodu dostatečně rozmanitý a hráč by měl šanci nakonec se umístit první.

- Soupeř bude zůstávat hodně vzadu. Na začátku závodu se bude držet kolem hráče a na konci by pak měl být nějakou pevně danou vzdálenost za ním. Bude se tedy postupně vzdalovat.
- Soupeř bude zhruba uprostřed. Na začátku bude mířit kus před hráče, na konci závodu by pak měl být stejný kus za ním.
- Soupeř bude vepředu. Na začátku závodu bude mířit hodně před hráče, na konci by pak měl být zhruba kolem hráče.
- Soupeř se bude po celou dobu závodu držet poblíž hráče, aby nikdy nebyl příliš sám.

Navíc bychom přidali pár drobných úprav inspirovaných článkem *The Pure Advantage: Advanced Racing Game AI* [44]. Po nějakou dobu na začátku závodu by měli soupeři maximální hodnoty statistik, aby okamžitě vyrazili ze startu a vznikl tak napínavý souboj hned o první bonusy. Pohyblivý cíl by se měnil jen po zhruba 75–80 % závodu, aby měl hráč nakonec ještě možnost předehnat i nejlepšího soupeře. Na posledních 20 % by se postupně usazovaly statistiky soupeřů na počátečních hodnotách, aby měl možnost hráč skončit první, i kdyby udělal ještě nějakou chybu na úplném konci.

### 3.5.3 Modulární návrh

Během návrhu bychom chtěli zajistit, aby bylo možné zvolenou implementaci snadno změnit. To by umožnilo snadné experimentování a vylepšování. Vytvoříme tedy modulární návrh, ve kterém bude možné nahrazovat jednotlivé podčásti. Celková implementace se tak rozdělí do dvou velkých celků – části zodpovědné za navigaci a části zodpovědné za kouzlení.

Zcela bokem bude navíc část pro chybovost (reprezentující úroveň schopností soupeře), která bude obsahovat parametry týkající se chyb soupeřů a bude poskytovat metody pro získání pravděpodobností určitých jevů na základě hodnoty statistiky. S touto částí pak budou pracovat ostatní.

### Navigace

Část starající se o navigaci bude řešit, kam má hráč letět. Bude tedy vhodně vybírat následující cílový bod. Různé implementace pak mohou reprezentovat různé způsoby řízení nebo volby cílů. Cílem v tuto chvíli myslíme nějaký dílčí úkol v rámci závodu (např. proletět obručí, sebrat bonus). Tuto část pak rozdělíme ještě na několik podjednotek:

- Jednotka pro volbu cílů – Tato jednotka bude zodpovědná za výběr následujícího cíle na základě aktuální situace. Bude však řešit pouze cíle související s průletem trati (tj. bonusy, obruče, konrolní body, cílová čára). Navíc bude využívat část pro chybovost pro zohlednění možnosti přeskočení cíle.

- Jednotka pro vykonávání akcí ke splnění cíle – Tato část bude pracovat s již zvoleným cílem a bude volit akce pro zajištění jeho splnění (např. letět správným směrem). Pokud bude součástí cíle doletět na konkrétní pozici, pak bude využívat jednotku pro řízení letu. Dále bude brát v úvahu aplikaci souvisejících chyb (např. minutí cíle).

- Jednotka pro řízení letu k danému cíli – Tato část bude řešit pouze let na danou cílovou pozici. Současně bude uvažovat také potenciální kolize a snažit se jim vyhnout. Navíc však zohlední možnost chyb v podobě nárazu nebo dočasného udržování směru místo zatočení.

Tento přístup nám umožní vhodně kombinovat *deliberation* a *reactivity*. Tyto koncepty jsou dobře popsány v *Reactivity and Deliberation in Decision-Making Systems* [46]. Reactivity popisuje schopnost agenta reagovat na podnět z prostředí. V našem případě toto zajišťuje jednotka pro řízení letu, která zohledňuje také případné překážky na trase. Deliberation pak označuje schopnost agenta dělat rozhodnutí a následně provádět akce. Jednotka pro volbu cílů bude provádět právě taková rozhodování. Může se rozhodnout cíl změnit na základě jeho validity a rozumnosti. Také se může pokoušet v pravidelných intervalech cíl přehodnotit, aby se soupeř nějakého cíle nedržel zbytečně dlouho, pokud mezitím začal existovat nějaký lepší. Nemělo by se to ovšem dít příliš často, aby bylo chování uvěřitelné.

### Kouzlení

Kouzlení bude implementováno ve zcela separátní části v rámci modulárního návrhu. Pokud totiž pro jednoduchost pomineme případy, kdy by si mohl soupeř chtít doletět blíž k cíli, je kouzlení nezávislé na směru letu.

Již v sekci 3.4.4 jsme zmínili, že vše související s kouzlením bude rozděleno do několika podjednotek. Co se týče implementace umělé inteligence pro soupeře, ta bude obsažena ve dvou z nich:

- Jednotka pro zpracování vstupu – V této jednotce bude hlavní část logiky, která bude rozhodovat o tom, které kouzlo kdy použít. Navíc se využije část pro chybovost pro zohlednění odpovídajících druhů chyb.

- Jednotka pro výběr cílů – Ze všech potenciálních cílů pro aktuálně zvolené kouzlo je třeba zvolit jeden. Bude možné použít různé strategie, např. ten nejbližší, nebo nejbližší z těch, na které už nemíří to samé kouzlo.

### 3.5.4 Varianty implementace

Vzhledem k velkému rozsahu hry nebudeme mít možnost vymýšlet velmi komplexní umělou inteligenci nebo porovnávat několik různých variant mezi sebou. Pokusíme se tedy jít spíše jednodušší cestou, abychom získali chování, které je pochopitelnější a tím také odladěnější. Vytvoříme tak spíše jen proof-of-concept než skutečně finální implementaci, takže do budoucna zbyde ještě spoustu prostoru pro možná vylepšení.

Část pro kouzlení bychom prozatím řešili zcela jednoduše tak, že kdykoliv by soupeř měl nějaké kouzlo připravené k použití, seslal by ho, ale tato rozhodování by prováděl jen v určitých intervalech (délka by závisela na chybovosti). Do budoucna bychom mohli uvažovat o rozšíření tohoto systému, kdy by se uvažovalo, o jaké konkrétní kouzlo se jedná a v jaké situaci je nejlepší ho použít.

Zajímavější je pak část starající se o navigaci. Oproti skutečným závodním hrám je naše hra velmi jednoduchá, protože nevyužívá žádné komplikované fyzikální simulace. Sice je v ní velká svoboda pohybu, ale jen s velmi jednoduchými pravidly. Díky tomu ani není potřeba vymýšlet opravdu sofistikované postupy. Trať je jasně určena obručemi, kolem je dostatek volného prostoru. Mělo by být tedy vcelku snadné vytvořit téměř dokonalou umělou inteligenci, do které následně zavedeme různé druhy chyb.

Nejdůležitější částí umělé inteligence je ta, starající se o volbu následujícího cíle. Nyní si tedy popíšeme několik způsobů, jak by se mohla implementovat. V práci však implementujeme ten první, který je nejjednodušší, ale přitom dává dostatečně dobré výsledky. Ostatní způsoby by nám sice mohly dát pokročilejší možnosti (např. možnost přiřadit různým soupeřům různé charakteristiky), ale jejich implementace by přidávala další vrstvu složitosti k již vcelku rozsáhlé práci. Do budoucna by však bylo možné implementovat také další a navzájem je porovnat. Ohled by se mohl brát na to, jak přesvědčivá daná verze je, jak snadno je parametrizovatelná a jak snadno je rozšiřitelná.

#### Přímočará AI

Prvním způsobem je něco, co nazýváme přímočará AI. Jedná se totiž o postup, kdy se soupeř snaží v podstatě o ideální výkon. Jako další cíl uvažuje vždy následující významný bod (tj. obruč, kontrolní bod, bonus). Takto jednoduše a přímočaře prolétává tratí.

Pro jednotlivé cíle se však vyhodnocuje jejich rozumnost. Soupeř si tak např. nezvolí hned další bonus, pokud by pro let k němu musel příliš změnit směr, nebo nepoletí pro bonus pro manu, pokud už ji má zcela naplněnou. Případně se může někdy rozhodnout cíl přehodnotit, jestli se mezitím neobjevil lepší. To by mohlo umožnit řešit také nečekané situace.

Úroveň schopností pak ovlivňuje výběr a plnění cíle. Pokud má soupeř nějaký cíl, může se rozhodnout o jeho minutí (tj. soupeř poletí úmyslně do místa o kus vedle). Kdykoliv si volí další cíl, s určitou pravděpodobností se může rozhodnout o jeho

přeskočení (tj. daný cíl se vůbec neuvažuje, ale soupeř ho může omylem splnit při plnění toho skutečně zvoleného).

### Utility AI

Další možností by bylo implementovat tzv. *Utility AI* (popsaná např. v *Behavior Selection Algorithms: An Overview* [47]). Ta se hodí především pro systémy, kde děláme rozhodnutí na základě spojitých hodnot (např. jak daleko něco je), a tyto hodnoty pak mapujeme na jiné spojité (např. jak moc chceme dosáhnout konkrétního cíle). Utility AI umožňuje zkombinovat mnoho okolností k vyhodnocení preferencí potenciálních cílů. Podle nich pak jeden zvolíme. Do takového systému se snadno přidávají nová chování, ale hůře se balancuje.

V naší hře bychom pak mohli využít implementaci podobnou té ve hře *The Sims* [48]. Soupeři by měli nějaké potřeby (odpovídající různým strategiím, např. co nejpříměji do cíle, co nejvíc ovlivňovat ostatní, co nejvíc sbírat bonusy, hrát defenzivně apod.), podle kterých by se chovali. Následně by existovala nějaká ústřední část, do které by se zaregistrovaly všechny zajímavé objekty (s informacemi jako kde jsou, jakou potřebu naplňují, jak moc). Každý soupeř by pak měl pro každou potřebu různou křivku, jak je pro něj důležitá.

Když by se soupeř rozhodoval o příštím cíli, prošel by všechny zajímavé objekty v blízkém okolí. Zvážil by, co nabízejí, co je pro něj důležité, jakým směrem a jak daleko se nacházejí. Pak by si vybral z pár nejlepších náhodně (tím se přidá trochu nedeterminismu). Až by soupeř zvolenou akci splnil, znovu by se rozhodoval stejným způsobem. Důležité je však zohlednit také to, že některý cíl může přestat být validní (např. jiný závodník sebral bonus dřív).

Tato varianta by se dala určitým způsobem aplikovat také na kouzlení. Nechtěli bychom však definovat separátní potřebu pro každé kouzlo, ale místo toho pevně definovat nějaké základní potřeby, ke kterým by různá kouzla různě přispívala. Pak by s nimi mohla pracovat také v budoucnu přidaná kouzla. Potenciální cíle kouzel by se mohly hlásit ústřední části a soupeř by z nich pak vybíral na základě toho, jaká kouzla má ve slotech a jakou potřebu by pak ve skutečnosti seslání kouzla na daný cíl naplnilo. Také když se rozhoduje o tom, jaká kouzla se soupeřům vybaví do závodu, mohla by se zohlednit jejich preferovaná strategie.

### Plánování

Velmi klasickou možností by pak mohla být nějaká implementace plánování, kdy by se nerozhodovalo pouze o jednom následujícím cíli, ale vytvořil by se plán několika akcí dopředu, který by se následně postupně vykonával a občas přehodnocoval. Plánovat by se mohly jak cíle, tak akce vedoucí k aktuálnímu cíli (v tom případě by se neřešily pouze reaktivně). Dále by mohlo být možné zahrnout do plánu také kouzla, kdy by se mohlo rovnou počítat s jejich dobíjením a s automatickým doplňováním many.

Problémem by ale mohla být náhodnost, protože s určitou pravděpodobností dochází k různým chybám na základě úrovně schopností soupeře. Muselo by se tedy plánovat s ohledem na to.

### Steering behaviours

Zcela mimo námi navrženou modulární architekturu by pak bylo možné implementovat něco podobného lokálním navigačním pravidlům (tzv. *steering behaviours*), která navrhnul Craig W. Reynolds [49]. Systém by se skládal z množiny jednoduchých chování, kdy každé má nějaký názor na to, jakým směrem se pohybovat. Tato chování se následně kombinují dohromady, takže je nakonec agent schopný dosáhnout i složitějších cílů. Případně bychom mohli využít rozšíření, tzv. *context steering* (popsaný např. v *Context Steering: Behavior-Driven Steering at the Macro Scale* [50]). V něm chování nevrací pouze výsledný směr, ale celý kontext, ve kterém se rozhodují. Ten zhruba odpovídá výčtu možných směrů (vhodně kvantizovaných) s přiřazenými preferencemi. Následně se kombinují právě tyto kontexty, ze kterých se nakonec vybere výsledný směr.

Místo určování konkrétních cílů bychom tedy měli jednotlivá chování (např. průlet obručí, sebrání bonusu, vyhýbání se překážkám). Každé chování by pak přiřazovalo různým směrům různou důležitost, kdy by se navíc zohlednila také vzdálenost. Výsledky jednotlivých chování by se zkombinovaly a nakonec by se vybral nejsilnější směr. Nevyužívali bychom ale explicitně *path following* chování (tj. jedno z původních Reynoldsových chování [49]), aby byl pohyb po správné trase spíše emergentní vlastností systému.

Jelikož vstupy, které se budou používat pro pohyb, budou mít diskrétní hodnoty (3 možné hodnoty pro 3 různé osy), mohli bychom vytvořit *context map* se sloty odpovídajícími jejich kombinacím.

Tento přístup by se pak dal snadno rozšířit o další chování, např. zablokovat závodníka, který se soupeře snaží předletět, nebo vytlačit závodníka ven těsně před obručí. Musely by se pouze vhodně nastavit váhy jednotlivých chování, aby vždy převažoval pohyb správným směrem po trati.

### Závěrečné shrnutí

Pokud bychom do budoucna chtěli s implementací umělé inteligence experimentovat, všechny zmíněné přístupy vypadají dostatečně zajímavě, dávají smysl v kontextu naší hry a stály by tak za zvážení. Nejzajímavější (avšak také nejkomplikovanější) by byla Utility AI, která by nám poskytovala velkou volnost v implementaci a také značnou možnost konfigurovatelnosti. Výsledkem by navíc nebylo zcela předpřipravené chování, nýbrž chování přirozeně vzešlé ze spolupráce několika menších jednotek, což by mohlo vést k nečekaným situacím ve hře. Jako první bychom se tedy nejspíše zaměřili právě na tento přístup.

## 3.6 Záloha stavu pro rychlý závod

V sekci 2.10 jsme zmínili, že bychom chtěli do hry přidat režim rychlého závodu. Současně bychom chtěli mít možnost využít co nejvíc řešení z hlavního kariérního režimu. Na mnoha místech je ale třeba pracovat se stavem hry a snaha o oddělení by příliš komplikovala celkový návrh. Rychlý závod by však měl být zcela nezávislý na stavu hry a neměl by ho nijak měnit.

Jednodušší řešení by tedy bylo, kdybychom si na začátku rychlého závodu uložili bokem původní stav, poté bychom přepsali aktuální stav hodnotami potřebnými

pro rychlý závod a po dokončení závodu bychom vrátili stav zpět do původních hodnot.

Úplně nejjednodušší pak bude toto řešit rovnou na úrovni souborů. Když se tedy spustí rychlý závod, vytvoří se záloha uloženého stavu překopírováním potřebných souborů. Kdykoliv by se pak spouštěl kariérní režim hry, zkontrolovalo by se, jestli náhodou neexistuje záloha. Pokud ano, nejprve by se stav hry obnovil z ní a poté by se záloha smazala.

Toto řešení nám umožní během rychlého závodu měnit stav tak, jak se to děje během normálního závodu, aniž by to mělo vliv na uloženou rozehranou hru.

## 3.7 Barevná paleta

Jelikož se grafické uživatelské rozhraní bude skládat ze spousty menších částí, bude třeba důkladně zvolit barvy, aby spolu vše ladilo. S větším množstvím obrazovek pak budeme potřebovat jednu barvu využít na několika různých místech (např. pro všechna tlačítka ve hře), ale kdybychom ji nastavovali ručně, snadno může docházet k chybám, kdy při změně nenahradíme všechny její výskyty.

Pokud bychom si nějakým způsobem definovali barevnou paletu a používali barvy z ní, získali bychom celou řadu výhod. Konkrétní barva by byla definována jen na jednom místě, ale používala by se z několika jiných míst. Stačilo by tedy provést jen jednu změnu a automaticky by se projevila všude jinde. Také bychom měli všechny důležité barvy pohromadě, takže by se snáz navrhovalo, jaké barvy kde použít, aby se k sobě hodily. Navíc bychom mohli třeba vytvořit několik různých barevných palet a pak je zaměňovat pro rychlé srovnání různých vzhledů nebo v budoucnu dokonce dát hráči možnost zvolit si barevné téma.

Pokusili bychom se přitom o takovou implementaci, která by nám co nejvíce usnadnila práci. Paletu by pak mělo být možné použít dvěma různými způsoby. Mohli bychom barvu něčemu přiřadit přímo v editoru, což by se využilo převážně pro prvky UI, které mají neustále tu samou barvu (např. tlačítka, pozadí, texty). Současně bychom ale poskytli přístup k jednotlivým barvám palety také za běhu z kódu, takže by bylo možné je použít pro prvky UI, které barvu mění během svého života (např. cena položky v obchodě indikující červeně, že hráč nemá dostatek mincí).

### Reprezentace palety

Nyní se nabízí otázka, jakým způsobem bychom měli reprezentovat samotnou paletu s barvami. Mohli bychom využít `ScriptableObject`, který by měl datové položky pro konkrétní barvy a jejich hodnoty. Tak by bylo možné vytvořit několik různých palet jako assety přímo v editoru a přepínat mezi nimi. Navíc bychom přidali statickou datovou položku pro snadný přístup k právě používané instanci palety. Tak by bylo možné používat barvy z ní také odkudkoliv z kódu.

Do palety bychom však neukládali úplně všechny barvy používané ve hře, protože pak by to bylo nepřehledné. Spíše tam umístíme časté barvy, které se používají na vícero souvisejících místech, mají větší vliv na celkový vzhled a je větší šance, že se třeba změní.

### Nastavení barvy

Dále bychom potřebovali mít možnost nějakým způsobem provázat barvu nějakého prvku s konkrétní barvou z palety.

Mohli bychom vytvořit speciální komponenty pro práci s různými běžnými komponentami obsahujícími barvu (např. `Image`, `SpriteRenderer`). Pak bychom tyto komponenty přidávaly objektům, jejichž barvu chceme změnit, a zvolili bychom jednu z barev. Museli bychom však zajistit, aby se veškeré změny provedly také v editoru, tedy např. hned při přidání komponenty a nastavení barvy z palety.

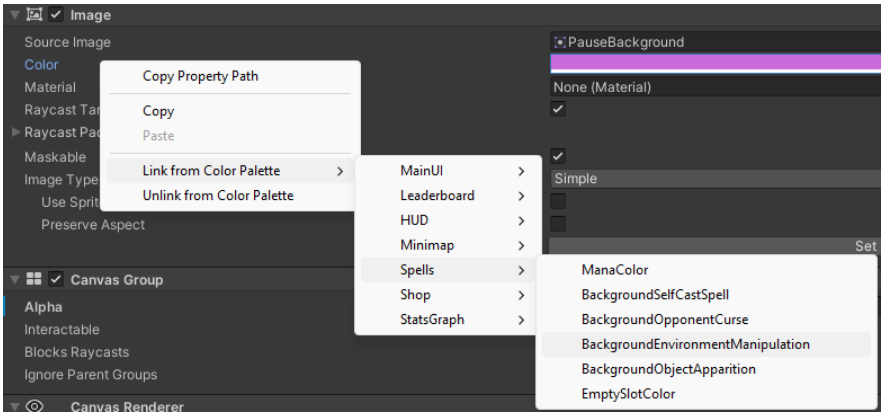
O něco jednodušší na použití by však bylo pracovat rovnou s již existujícími komponentami, které obsahují barvu, a pouze přidat možnost přímo provázat hodnotu s konkrétní barvou z palety (např. pomocí položky v kontextovém menu). Někde bokem by pak byl objekt zodpovědný za zapamatování si všech takových vazeb. Díky tomu by bylo možné propagovat libovolné změny (např. pokud se změní hodnota barvy v paletě, projdou se všechny provázané položky a změní se také hodnota v nich).

Takové možnosti použití nabízí nástroj *Colorlink* [51], jehož autorem je Eugene Radaev. Použili bychom jej jako základ implementace naší palety. Ještě více bychom ho však zjednodušili a přizpůsobili našim potřebám.

### Volba barvy

Jak tedy bylo řečeno, barvy by byly uloženy v datových položkách barevné palety. Současně bychom ale potřebovali snadno získat jejich výčet, abychom je mohli zobrazit v kontextovém menu pro provázání. Definujeme tedy navíc výčtový typ, který bude obsahovat přehledné názvy všech barev. V názvech umožníme nějaký oddělovač, pomocí kterého budeme moci zobrazovat barvy seskupené pro ještě snazší vyhledávání.

Na obrázku 3.7 pak vidíme, jak by mohlo nakonec vypadat nastavení provázání mezi barvou v komponentě `Image` a barvou z palety. Z kontextového menu zvolíme, že chceme navázat barvu z palety. Následně vybereme některou ze skupin barev (určených z názvů hodnot výčtového typu rozdělených dle oddělovače) a nakonec zvolíme požadovanou barvu.




**Obrázek 3.7** Ukázka nastavení provázání barvy v komponentě `Image` s barvou z barevné palety.

V barevné paletě by navíc byla veřejná metoda pro získání konkrétní barvy na základě odpovídající hodnoty výčtového typu, která by se pak dala použít kdekoliv z kódu.

## 3.8 Audio

V sekci 2.9 jsme uvedli, že bychom chtěli do hry přidat audio různých typů. K tomu bychom mohli využít podporu pro audio, která je k dispozici v samotném Unity. Pak by byla veškerá logika přehrávání audia přímo v kódu. Jelikož bychom však nechtěli omezovat naše možnosti do budoucna, rozhodli jsme se použít raději specializovaný audio middleware, který lze integrovat do Unity. Z krátkého průzkumu vyplynulo, že dva nejpoužívanější jsou *Wwise* [52] a *FMOD* [53]. Pro naši hru jsme zvolili FMOD, protože s ním máme již předchozí zkušenosti a jeho integrace do Unity je snadná.

Díky audio middlewaru budeme mít možnost v budoucnu snadno zavést adaptivní audio. Současně nebude muset být na straně Unity příliš velká režie a kód se tak znatelně zjednoduší. Většina logiky by totiž byla zpracována přímo v FMOD. Kód v Unity by se omezil čistě na řízení, kdy má zaznít jaká audio událost a případně s jakými parametry. Právě tato abstrakce pomocí audio událostí společně s rozhraním pro změnu parametrů pak umožňuje snadné úpravy, kdy můžeme měnit pouze audio klipy a události přímo v FMOD, ale kód zůstává stejný. Bude tedy v budoucnu možné vcelku jednoduše přejít na adaptivnější zvukové efekty.

Obecně se pokusíme co nejlépe navrhnout rozhraní jednotlivých událostí a co nejvíce redukovat počet vnějších parametrů, zatímco se celá řada parametrů zautomatizuje v FMOD. Příkladem mohou být zvuky prostředí během závodu, které závisí na oblasti, ve které se hráč nachází. Z Unity se pouze nahlásí změna oblasti, ale přímo v FMOD se pak budou volit vhodné audio klipy, měnit jejich hlasitost apod.

### Audio systém

Práci se zvuky bychom mohli zapouzdřit do jednoho objektu, který by reprezentoval audio systém, se kterým by ostatní objekty komunikovaly, pokud by chtěly nějaký zvuk přehrát. Jednalo by se o singleton přetrvávající mezi scénami (více v sekci 3.9.1). Takový objekt by mohl řešit co nejvíc situací, aby ostatní odstínil od detailů implementace, totiž že se využívá konkrétně FMOD. Pak by ale mohl být až příliš velký a všemohoucí.

Raději bychom tedy udělali tento audio systém jednodušší. Řešil by pouze to, co je potřeba úplně všude (např. přehrávání hudby na pozadí či přehrání zadané FMOD události jako jednorázového zvukového efektu). Ostatní by se pak řešilo přímo na odpovídajícím místě, mimo audio systém.

Tímto přístupem by sice byl kód pracující s FMOD roztroušený po celém projektu, ale jednotlivé případy by měly být jednoduché a intuitivní. Navíc nepředpokládáme, že bychom někdy přecházeli od FMOD k něčemu jinému (nabízí nám vše, co potřebujeme, a ještě mnohem víc), takže není nutně třeba abstrahovat vše související s FMOD.

### Reference na audio události

Audio systém by nabízel metodu pro přehrání zadané audio události jako jednorázového zvukového efektu. Abychom tedy mohli tuto metodu použít z nějakého skriptu, musíme mít přístup k dané FMOD události. Mohli bychom použít `string` obsahující cestu k události, ale to může snadno vést k chybám. Případně bychom mohli použít datovou položku přímo v daném skriptu a nastavit událost v Inspector okně. Pak by ale mezi obvyklými parametry skriptu najednou byla položka pro zvuk.

Vytvoříme si tedy `ScriptableObject` se všemi událostmi v datových položkách. Referenci na něj pak uložíme do audio systému. Kdykoliv tedy budeme chtít použít metodu pro přehrání zvukového efektu, jako parametr předáme událost z odpovídající datové položky. Díky tomu budou veškeré reference v kódu snadno udržovatelné, protože konkrétní propojení s FMOD událostmi se bude vyskytovat na jediném místě.

Nevýhodou tohoto přístupu je, že přidání nové události bude vyžadovat několik kroků. Nejprve budeme muset vytvořit danou událost v FMOD, pak pro ni vytvořit datovou položku v objektu s referencemi a nastavit ji. Nečekáme však, že bychom nové zvuky přidávali nějak často.

### Zvuky prostředí

Jak už jsme řekli, zvuky prostředí během závodu závisí na aktuální oblasti. Implementačně bychom pak mohli zvolit jeden ze dvou způsobů. Mohli bychom mít různé audio události pro různé oblasti a pak mezi nimi přepínat. To by umožnilo s jednotlivými oblastmi pracovat odděleně a stačilo by najít jeden vhodný audio klip pro každou oblast.

Lepší by ale mohlo být, kdybychom měli pro všechny oblasti k dispozici několik vrstev zvuků, ale dle aktuální oblasti bychom pak měnili jejich hlasitost či intenzitu (např. aby nezněly zvuky vody tam, kde voda není). Díky tomu bychom mohli snadno využít jednu a tu samou vrstvu ve více oblastech a také by mohly být přechody mezi oblastmi hladší. Samotné vytvoření jednotlivých vrstev pak ale bude o něco komplikovanější, protože budeme muset najít izolované složky zvuků jednotlivých oblastí a někdy je také vrstvit pro možnost navýšit intenzitu. Přesto se však pokusíme implementovat právě tento způsob.

Kromě samotné oblasti má pak na zvuky prostředí vliv také výška hráče nad zemí. Některé složky zvuků by totiž měli být hlasitější, pokud je hráč těsně nad zemí (např. zvuky vody, hmyzu). Jiné složky by pak měly být stejně hlasité bez ohledu na výšku (např. hromy). Bude tedy třeba toto správně uvažovat a ovlivňovat pouze odpovídající vrstvy. Audio událost pro zvuky prostředí pak bude mít dva parametry nastavované z Unity, a to aktuální oblast a výšku nad zemí. Dle nich se pak v FMOD budou správně nastavovat intenzity jednotlivých vrstev.

Některé zvuky pak budou znít neustále ve smyčce, ale jiné se budou vyvolávat jako jednorázové zvukové efekty v náhodných intervalech.

### Implementace různých druhů zvuků

Během implementace bychom se snažili co nejvíce využít již existující FMOD komponenty (např. `FMODStudioEventEmitter`). V případě jednorázových zvuko-

vých efektů bychom zvuky vyvolali přímo ve skriptu s ostatní logikou. V komplikovanějších případech, kdy by bylo třeba nastavovat parametry a řešit aktuální stav, bychom však vytvořili vlastní komponenty, které by se o to staraly.

Jelikož se jedná o 3D hru, budeme chtít využít také 3D prostorový zvuk (případně s Dopplerovým efektem). Některé objekty tak budou mít zvuk rovnou na sobě a samy se starat o jeho přehrávání a řízení. Může se jednat např. o zvuk letu koštěte.

Obecně bychom pak chtěli podporovat několik kategorií zvuků:

- Hudba na pozadí – Hrála by neustále, ale v určitých scénách by byla mírně potlačená (kdykoliv hráč letí na koštěti). Řešila by se přímo v centrálním audio systému.
- Zvuky prostředí – Ozývaly by se různé zvuky dle aktuální oblasti. Jednalo by se o jedinou adaptivní událost, která by se řídila z dedikovaného objektu přímo ve scéně, kde je potřeba. Např. šumění větru, bzučení včel.
- Jednorázové zvukové efekty (tzv. one-shots) – Jedná se o velmi krátké zvuky vyvolané nějakou událostí ve hře. Patří k nim vše v UI, ale také některé 2D/3D zvuky vyvolané ve 3D scéně. Řešily by se přímo na objektech, ke kterým náleží, a to pomocí FMOD komponent a/nebo nějakých našich vlastních komponent využívajících metody centrálního audio systému. Např. kliknutí, seslání kouzla.
- Dlouhotrvající zvuky (tzv. sustained) – Jedná se o zvuky, které se přehrávají po delší dobu a mohou být ovlivňovány nějakým parametrem. Řešily by se přímo na 3D objektu, ke kterému náleží, ve speciální komponentě. Např. zvuk letu koštěte (ovlivněný rychlostí letu).

### Variabilita zvuků

Při návrhu audia je důležité zařídit také určitou variabilitu zvuků, aby se nepřehrával neustále ten samý audio klip. Jednoduchým způsobem je mít rovnou několik audio klipů a pak z nich náhodně vybrat jeden, který se přehraje. Pokud však nemáme k dispozici dostatek klipů, můžeme si poradit náhodnou modulací různých parametrů, jako je např. lowpass filter, highpass filter nebo reverb. V případě, kdy používáme několik vrstev smyček, které hrají společně, můžeme randomizovat jejich počáteční offset, takže každá instance tohoto zvuku bude znít pokaždé trochu jinak.

### Audio během pozastavené hry

Když je hra pozastavená, některé zvuky ze hry by měly přestat hrát pro naznačení zastavení času. Jiné by však měly znít dál, akorát utlumeně, aby se posílil pocit, že v tu chvíli hráč není ve hře, ale je od ní vzdálen. Mohlo by pak dávat smysl následující – hudba na pozadí by pokračovala dál, ale trochu hlasitější, zvuky prostředí by se mírně utlumily, zvukové efekty ze hry by se přerušily a zvukové efekty UI by se ozývaly beze změny.

Co se týče zvuků, které by se měly přerušit, nabízely by se obecně dvě možnosti:

- Zvuky by se pozastavily a po návratu do hry by dozněly. Díky tomu by hráči neunikly žádné informace. Bylo by však těžší zařídit, že to i s krátkými zvuky bude znít dobře, aby nezačaly okamžitě po návratu znít naplno, ale současně aby ještě byly slyšet.
- Zvuky by se pouze utlumily a během pauzy by tak utlumené dozněly. Bylo by to technicky méně náročné řešení a navíc by se nejspíš nestávalo až tak často, že by se hra pozastavila zrovna uprostřed nějakého významného zvuku, který by měl doznít.

Vzhledem k tomu, že druhá možnost má zřejmou výhodu v jednoduchosti implementace a nevýhody jsou pro nás zanedbatelné, zvolíme tu.

#### Snapshoty

V určitých situacích bychom chtěli změnit vlastnosti přehrávaného zvuku, např. pokud je hra pozastavená (pak by měla být hudba na pozadí výraznější a ostatní zvuky potlačené) nebo pokud je hráč pod vodní hladinou. K tomu můžeme využít koncept tzv. *snapshotů*, který je v FMOD podporován. Stačí jen pro každou takovou situaci vytvořit jeden snapshot, v něm nastavit požadované efekty (např. lowpass filter, reverb) a pak ho v odpovídající chvíli zapnout nebo vypnout (stejně jako by to byla normální audio událost).

## 3.9 Pomocné nástroje

Kromě dříve popsaných zásadních herních systémů bychom potřebovali implementovat také celou řadu pomocných nástrojů. Některé z nich tedy popíšeme v této sekci.

### 3.9.1 Implementace singletonu

Pro různé části implementace by se nám hodilo mít nějaké speciální objekty, které spravují nějaká data a ostatní z nich pak tato data získávají, pokud je potřebují ke své práci. Případně mohou tyto objekty nabízet nějaké užitečné metody. Mohlo by se jednat např. o stav hry, detekci vstupu od hráče, načítání scén. V některých případech by možná stačila statická třída. Pokud však potřebujeme instanci, mohli bychom implementovat návrhový vzor *singleton* [54]. Ten je definován jakožto třída s právě jednou instancí, která je globálně přístupná.

V kontextu Unity to však může také znamenat, že by singleton mohla být komponenta a pak by měl být ve scéně pouze jeden objekt s danou komponentou. Někdy bychom mohli vyžadovat také to, aby takový objekt existoval po celou dobu běhu hry, tj. přetrvával mezi scénami. Vytvoříme si tedy vlastní implementaci singletonu z komponenty, která pak bude sloužit jako společný předek pro nějaké konkrétní implementace. Využijeme to např. pro různé globální systémy a pro snazší provázání částí hry bez nastavování referencí.

Existují různé přístupy implementace dle toho, jakých všech vlastností chceme dosáhnout. My se pokusíme zvolit implementaci tak, abychom udělali plně konfigurovatelný singleton, který umožňuje zvolit libovolnou kombinaci následujících chování:

- vytvoření objektu s danou komponentou, pokud již neexistuje ve scéně (pro zajištění instance),
- odstranění duplicitních objektů ve scéně, pokud již existuje instance, ale ve scéně se nachází další (pro zajištění, že existuje právě jedna instance),
- líná inicializace až při prvním přístupu k instanci,
- perzistence mezi scénami (pomocí `DontDestroyOnLoad(Object target)`).

Jednotlivé možnosti definujeme jako hodnoty výčtového typu s atributem `[Flags]`, abychom mohli hodnoty kombinovat. Konkrétní chování se pak nastaví ve statickém konstruktoru, aby se zajistilo provedení před vším ostatním a také bez nutnosti instance. Tak budeme moci přizpůsobit chování dané komponentě na míru a využívat stejnou základní třídu v různých situacích. Finální implementace je pak popsána v sekci 4.10.9.

### 3.9.2 Načítání scén

Jelikož se bude naše hra skládat z několika oddělených scén (podrobněji jsou popsané v sekcích 2.2 a 4.1.1), bude potřeba mezi nimi přecházet. Unity nabízí třídu `SceneManager` s metodami pro samotné načtení scény. Jelikož bychom však chtěli přidat také další funkcionalitu, vše zapouzdříme do vlastní komponenty na objektu, který bude přetrvávat mezi scénami a bude globálně dostupný, aby jej mohl využít kdokoliv.

Kdykoliv se pak má přejít do jiné scény, zavolá se metoda tohoto objektu s odpovídajícím parametrem. Nejprve se ztmaví obrazovka, aby nebylo vidět dění na pozadí, až poté se začne načítat scéna a po dokončení se obrazovka opět rozjasní.

Mohli bychom také řešit, co se má stát v situaci, kdy se během načítání jedné scény požádá o načtení jiné. V takovém případě bychom si mohli třeba scénu zapamatovat a začít ji načítat po dokončení. Buď bychom si mohli takto ukládat celou frontu scén a načítat jednu po druhé, což by zajistilo, že by pro každou běžel důležitý inicializační krok s případnými vedlejšími efekty (např. změna stavu, logování), nebo bychom mohli rovnou načíst až tu aktuálně poslední. Zatím toto v implementaci pomineme, protože by k takové situaci nemělo nikde docházet. Do budoucna bychom o tom však mohli uvažovat pro zobecnění implementace.

#### Předávání parametrů mezi scénami

Někdy by se mohlo hodit předávat parametry mezi scénami, např. proto, abychom mohli říct, že se má po návratu ze scény testovací trati načíst scéna přehledu hráče rovnou s otevřeným obchodem (protože právě z něj se původně přešlo do testovací trati). Připravíme pro to tedy také obecný mechanismus.

Objekt poskytující metody pro načtení scény by mohl mít dále metody pro nastavení parametrů (zadaly by se jako jméno a hodnota, přičemž by se podporovalo několik základních datových typů). Dále bychom mohli definovat abstraktní komponentu s metodami pro přijímání parametrů. Každá scéna, která by potřebovala přijímat nějaké parametry, pak může implementovat vlastní odvozenou komponentu pro reakci na parametry. Tato komponenta by se pak umístila někam

do scény. Jakmile by se dokončilo načítání nové scény a existovaly by parametry pro předání, zkusila by se najít komponenta odvozená od té základní, aby se jí mohly parametry předat.

#### Načítací obrazovka

Po ztmavení obrazovky během načítání zobrazíme také speciální načítací obrazovku, která bude mít pohyblivé prvky, podle kterých bude možné poznat, zda se skutečně něco děje a načítání jen trvá dlouho, nebo zda se hra třeba úplně nezasekla. Aby se obrazovka mohla správně aktualizovat, využijeme možnost asynchronního načítání scén, kdy se nová scéna načítá na pozadí, zatímco aktuální stále běží.

Do budoucna bychom mohli udělat načítací obrazovku zajímavější, aby např. obsahovala nějaké tipy ke hře nebo něco jiného, čím by se hráč mohl zabavit, pokud by načítání trvalo příliš dlouho.

#### Dlouho se načítající objekty

Při implementaci načítací obrazovky jsme ovšem narazili na problémy, kdy se animace v ní zasekávaly, pokud jsme načítali větší scény (např. dlouhý závod). V buildu sice byly problémy mírnější oproti editoru, ale stále dost znatelné. Zdá se totiž, že asynchronní načítání scény není dokonale asynchronní [55] a inicializace objektů ve scéně (pomocí jejich metod `Awake()` a `Start()`) se zřejmě dějí v hlavním vlákně, i když instanciace objektů běží asynchronně. Nejlepší by tedy bylo během inicializace dělat minimum práce a to nejnáročnější pak přesunout jinam, např. až do metody `Update()` (a po celou dobu nastavit `Time.timeScale = 0`, aby mezitím hra neběžela dál).

Největším problémem pak bylo načítání scény závodu, protože během její inicializace se musí vygenerovat celý level. Zkusili jsme tedy level generovat z *coroutine* a v určitých okamžicích se na frame vrátit. Problém se tím drobně zlepšil, ale je třeba dobře vyvážit, jak často se vrací řízení z coroutine. Pokud hodně často, pak trvá generování levelu o to déle. Pokud naopak málo, pak jsou animace na načítací obrazovce trhanější.

Nakonec je však nutné zařídit, že načítací obrazovka nezmizí a obrazovka se nerozjasní, dokud se nedokončí inicializace i takových dlouho se načítajících objektů. Mohli bychom tedy vytvořit obecné řešení, kdy by se takové objekty hned na začátku inicializace zaregistrovaly u objektu zodpovědného za načítání scény. Ten by pak počkal, dokud neskončí celá inicializace všech zaregistrovaných objektů, než by skryl načítací obrazovku.

### 3.9.3 Lokalizace

Jak jsme uvedli již v požadavku **P4**, chtěli bychom hru lokalizovat do více jazyků. Existuje celá řada již hotových řešení, ať už nějaká řešení třetích stran, nebo *Localization package* [56] přímo od Unity, který nabízí hezké rozhraní pro správu jazyků a frází. Současně by však nemělo být těžké implementovat si nějaké vlastní, dostatečně funkční řešení. Tím bychom si vytvořili něco, co dobře známe a můžeme to libovolně upravit dle svých potřeb (navíc je to v souladu s požadavkem **P7**). Vydáme se tedy tímto směrem.

Když už víme, že bychom se pustili do vlastní implementace, musíme také vyřešit, jakým způsobem bychom ukládali jednotlivé fráze v různých jazycích.

- Mohli bychom použít `ScriptableObject`, pro každý jazyk vytvořit jednu instanci a pak je načítat z kódu. Pak by to ale bylo součástí buildu hry a nebylo by zřejmě snadné poskytnout update ke hře s novým jazykem.
- Nebo bychom mohli použít jednoduše externí soubor. Součástí buildu hry by pak byla jen logika a soubor by k ní poskytoval data. Stačilo by tedy jen upravit soubor a spustit hru.
- Případně bychom mohli použít Google tabulky. V každém sloupci by byl jeden jazyk, v řádcích pak fráze. Soubor by se tak dobře sdílel, takže by mohli přispívat také další lidé. Pak bychom mohli napsat kód pro import dat z tabulky.

Mohli bychom nakonec využít kombinaci dvou možností. Jako základní zdroj dat použít Google tabulku, se kterou se dobře pracuje a nabízí případnou výhodu sdílení. Tu bychom pak mohli manuálně převést na soubor, který by se přiložil ke hře. Za předpokladu, že se překlady příliš nemění, nebude nutné tabulku exportovat příliš často. Tím bychom nemuseli načítat data přímo z Google tabulky, což by vyžadovalo netriviální kus kódu, ale stačilo by jen číst soubor v nějakém vhodném formátu.

Co se týče formátu souboru s daty, mohli bychom použít základní CSV, ale pak by se musel zvolit vhodný oddělovač, který by se nevyskytoval nikde v textu. Bezpečnější by tedy byl JSON. Google tabulky se sice nedají exportovat rovnou do něj, ale můžeme využít externí nástroj, např. *opensheet* [57], který vytvořil Ben Borgers.

Kdykoliv by se spustila hra, načetl by se obsah souboru a uložil do nějakého správce, který by pak poskytoval veřejné metody pro získání fráze na základě klíče. Mohlo by být dobré nemít pořád načtené v paměti úplně všechny fráze pro všechny jazyky, protože by se tím zbytečně zabíralo místo, když může být vždy aktivní pouze jeden jazyk. Na druhou stranu ale plánujeme podporovat zatím jen dva jazyky a navíc naše hra není nijak vystavená na narativu, takže v porovnání se vším ostatním a celým enginem bude spotřeba paměti zanedbatelná.

#### Textová komponenta

K hlavní implementaci lokalizace bychom navíc vytvořili vlastní komponentu, která by jako parametr dostala klíč lokalizované fráze a spolupracovala by s obvyklou komponentou pro zobrazení textu v UI. Kdykoliv by se měla zobrazit, zajistila by, že se do textové komponenty dosadí fráze ve správném jazyce. Navíc by si zaregistrovala callback na změnu jazyka, aby mohla správně aktualizovat obsah.

Do menu v editoru navíc přidáme položku, která do scény vloží rovnou objekt s přednastavenými komponentami pro lokalizovaný text.

### 3.9.4 Tooltipy

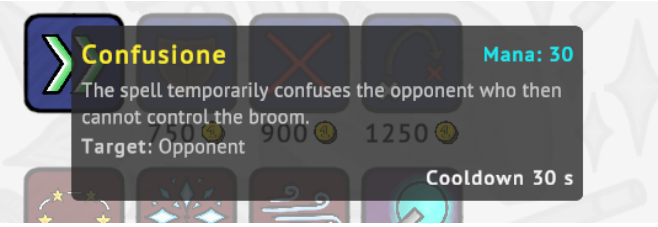
V sekci 2.10 jsme popsali některé situace, ve kterých by se hodilo zobrazit tooltipy. Nyní se zamyslíme nad jejich implementací. Chtěli bychom navrhnout

dostatečně obecnou a rozšiřitelnou infrastrukturu, která bude podporovat různé možnosti struktury obsahu, různé styly částí textu a také bude spolupracovat s lokalizací. Velikost tooltipu by se pak měla přizpůsobovat obsahu a pozice by se měla měnit na základě kurzoru myši, ale tak, aby nikdy tooltip nepřečníval přes okraje obrazovky. Omezíme se přitom pouze na tooltipy pro UI prvky a nebudeme uvažovat tooltipy u objektů ve scéně.

Obsah v tooltipu může být jednodušší (např. jediná věta), nebo více strukturovaný (např. různé části popisu kouzla). Vytvoříme tedy pevně danou, ale dostatečně členitou podobu tooltipu. Pak se budou jednotlivé části naplňovat textem, přičemž pokud některá bude prázdná, zmenší se, aby nezabírala místo.

Abychom mohli měnit vzhled tooltipu, přidáme možnost definovat styl, který by se následně použil (inspirujeme se projektem *simple-tooltip* [58], který vytvořil Norbert Staskevicius). Ve stylu bude možné definovat základní vlastnosti, jako je barva pozadí, velikost textu, barva textu a maximální šířka. Navíc ale umožníme definovat vlastní značky, které bude možné použít pro označení částí obsahu, kde by se měly základní vlastnosti přepsat jinými. Využijeme přitom možnost formátování textu pomocí balíčku TextMesh Pro, takže definované značky přeložíme na značky jím podporované.

Na obrázku 3.8 pak vidíme, jak vypadá tooltip u kouzla v obchodě. Obsah je rozdělen do několika částí, přičemž pro změnu stylu textu se využívají definované značky.




**Obrázek 3.8** Zobrazení tooltipu s informacemi o kouzlu v obchodě.

### 3.9.5 Chování kamery v cutscenes

Před začátkem závodu bychom chtěli spustit krátkou sekvenci, kdy by kamera přecházela mezi nastoupenými závodníky a následně by se zaměřila na hráče. Podobně by se měla spustit sekvence poté, co hráč proletí cílem, kdy se na něj kamera zaměří zepředu a hráč se usměje. Potřebujeme tedy nějakým způsobem animovat pohyb kamery. Jelikož ale máme levely procedurálně generované, pozice startu a cíle není pokaždé stejná, takže není možné jednoduše vytvořit animaci pozice kamery (jedině animovat nějaký relativní offset vůči hlavní pozici, která by se nastavila dle pozice startu/cíle). Vytvoříme si tedy rovnou vlastní infrastrukturu, která nám navíc umožní výběr z více různých chování.

Jako základ použijeme `Timeline`, která bude řídit přepínání mezi různými virtuálními kamerami a také ztmavení/rozjasnění obrazovky během něj. Jednotlivým kamerám pak ale přidáme komponenty, které budou řídit jejich chování. Mezi chováními jediné kamery pak můžeme přecházet, pokud je třeba. Implementujeme celkem pět různých variant:

- Statická – Kamera se hned na začátku umístí do zadané pozice se zadanou rotací.
- Statická relativně k objektu – Kamera se hned na začátku umístí do zadané pozice se zadanou rotací relativně k aktuální pozici a rotaci zadaného objektu.
- Pohyblivá – Kamera se hned na začátku umístí do zadané pozice se zadanou rotací. Následně se v metodě `Update()` bude posouvat zadaným směrem a zadanou rychlostí.
- Pohyblivá relativně k objektu – Kamera se hned na začátku umístí do zadané pozice se zadanou rotací relativně k aktuální pozici a rotaci zadaného objektu. Následně se v metodě `Update()` bude posouvat zadaným směrem a zadanou rychlostí.
- Sledující objekt – Kamera se hned na začátku i v každém `LateUpdate()` umístí do zadané pozice relativně k aktuální pozici zadaného objektu (se zohledněním jeho orientace) a otočí se směrem do zadaného bodu relativně k počátku objektu.

Kameru s touto komponentou a nastaveným chováním na sledování objektu využijeme také v tutoriálu, pokud se budeme chtít zaměřit na objekt, který hráči právě představujeme.

### 3.9.6 Tweening

Ačkoliv bychom se zaměřili na vývoj demo verze, chtěli bychom, aby byla funkcionalita v ní co nejvíce odladěná a dotažená do konce. Na mnoha místech bychom tak použili tzv. *tweening*, který má původ v animaci, ale obecně může označovat postupnou interpolaci nějaké hodnoty v čase. Obvykle přitom určujeme počáteční hodnotu, cílovou hodnotu a pomocí jaké funkce se interpoluje.

Velmi časté je použití v uživatelském rozhraní. Pokud by se například měl otevřít nějaký panel, nezobrazil by se rovnou, ale interpolovala by se jeho průhlednost, aby se objevil postupně. Změna by však probíhala jen velmi rychle (ve zlomku sekundy).

V našem projektu bychom mohli použít např. balíček DOTween [59], který umožňuje interpolaci různých vlastností na celé řadě komponent v Unity. Hodil by se pro nějaké jednoduché, méně komplexní situace.

Navíc bychom si však implementovali vlastní řešení, které by nám umožnilo jednoduše aplikovat často používaná nastavení přímo z editoru. Vytvořili bychom velmi konfigurovatelnou komponentu, kterou bychom přidali objektu. Následně bychom mohli zvolit, co přesně chceme interpolovat (např. barvu, průhlednost, velikost) a s jakými parametry (počáteční a cílová hodnota, křivka interpolace). Pak už by se za nás našla na daném objektu vhodná komponenta s odpovídající vlastností.

Dále bychom umožnili tween spouštět z kódu, nebo automaticky hned po inicializaci objektu. Pomocí `UnityEvent` bychom přidali možnost provedení nějakých akcí při zahájení a při ukončení tweenu (tak by se mohly tweeny na sebe také navázat). Nakonec bychom přidali možnost opozdit začátek tweenu o určitou dobu, přehrávat tween také pozadu a případně ve smyčce.

Díky tomu bychom pak mohli velmi jednoduše a zcela bez kódu dosáhnout také komplikovanějších chování, např. aby se zeď vyčarovaná kouzlem Materia Muri (popsáno v sekci 2.7) nejprve postupně zvětšila do plné velikosti, následně pulzovala po celou dobu životnosti a nakonec zmizela zmenšením se.

### 3.9.7 Pozastavení hry

Kdykoliv je hráč přímo ve hře, měl by mít možnost ji pozastavit. To bychom mohli udělat jednoduše nastavením `Time.timeScale = 0`. Je však třeba vyřešit několik podproblémů.

#### Zanořené pozastavení

Během tutoriálu je někdy třeba hru zapauzovat, zatímco se hráči něco vysvětluje, aby se neměnil stav hry na pozadí. Současně by však měl být hráč pořád schopný hru pozastavit také sám, aby se mu zobrazilo menu s nabídkou možností. Při návratu do hry by pak ale měla kvůli tutoriálu zůstat pozastavená.

Umožníme tedy podporu zanořeného zapauzování. Rozlišíme přitom případ, kdy se hra zapauzuje z jiného herního kódu (tedy bez zobrazení menu) a kdy o pozastavení požádá přímo hráč (a zobrazí se tedy také menu). Následně si situaci ještě zjednodušíme tak, že bude možný libovolný počet pozastavení bez menu (během toho se bude inkrementovat čítač) následovaný maximálně jedním pozastavením s menu. Tak budeme stále podporovat všechny potřebné případy, ale výrazně se nám zjednoduší implementace.

#### Zákaz pozastavení

V určitých momentech bychom potřebovali zakázat možnost pozastavení hry nebo naopak obnovení hry, např. během načítací obrazovky, ze zobrazeného nastavení při již zapauzované hře, po dokončení závodu apod. Chtěli bychom tedy vytvořit dostatečně obecný mechanismus, aby mohl kdokoliv zvenčí říct, že chce pauzu zakázat, a pak aby měl také možnost toto vzít zpět a pauzu opět povolit. Pokud by se pak hráč snažil hru pozastavit nebo obnovit, zatímco existuje alespoň jeden objekt, který si toto nepřeje, nic se nebude dít.

Nyní je třeba vymyslet co nejvhodnější způsob, jak poznamenávat, že by pauza měla být zakázaná:

- Mohli bychom použít zásobník, ale to by fungovalo pouze v případě, že jsou dvojice zakázání a povolení správně zanořené.
- Nebo bychom si mohli pamatovat nějaké identifikátory objektů, které chtějí pauzu zakázat, a podle nich párovat zákaz/povolení. Pak je však třeba skutečně zajistit, že stejný objekt pauzu zase povolí zpět. Např. když komponenta `RaceController` pauzu zakáže po dokončení závodu, před svým zničením ji musí zase povolit. Nestačilo by, že by ji jen povolila na začátku své existence ve scéně závodu, protože příště už by to byla jiná instance.
- Případně bychom si mohli poznamenat pouze typ objektu, ne konkrétní instanci, čímž by se jednoduše vyřešil případ s `RaceController` komponentou. Toto by ale nefungovalo, pokud bychom někdy skutečně potřebovali rozlišovat i mezi různými instancemi toho samého typu.

Zdá se tedy, že neexistuje nějaké opravdu obecné řešení, které by současně bylo funkční ve všech možných případech. Při důkladnějším promyšlení našeho případu jsme dospěli k závěru, že období zákazu pauzy se nikdy nepřekrývají, tj. v každém okamžiku ji chce zakázat maximálně jeden objekt. Implementujeme tedy jednodušší řešení s obyčejnou `bool` proměnnou, které pokryje všechny potřebné případy. Současně však nastavení této proměnné schováme do veřejných metod pro zakázání/povolení pauzy, aby k tomu neměl přístup kdokoliv a v budoucnu se mohlo řešení snadno nahradit nějakým lepším, pokud by bylo třeba.

#### Efekty v UI

I když se zastaví čas, nechceme zastavit úplně vše. Např. by měly pořád fungovat různé animace a efekty v UI, když je zobrazené menu pozastavené hry. Musíme tedy na odpovídajících místech správně pracovat s časem:

- V `Animator` komponentě můžeme nastavit `Update Mode` na `Unscaled Time`. Pak jsou animace nezávislé na právě nastavené rychlosti času.
- Tweeny z balíčku DOTween jsou ve výchozím nastavení závislé na rychlosti času ve hře. Při pozastavené hře by tedy neběžely. Je třeba zavolat na vzniklém objektu tweenu `SetUpdate(true)`, aby se použil neškálovaný čas.
- Kdekoliv si vytváříme vlastní efekty v kódu, místo `Time.deltaTime` bychom měli použít `Time.unscaledDeltaTime`. Pak dostaneme skutečně uběhnutý čas, nezávislý na rychlosti hry.

### 3.9.8 Cheaty

Abychom si usnadnili testování během vývoje, chtěli bychom do hry přidat také podporu pro cheaty, o kterých jsme psali již v sekci 2.10. Nyní si popíšeme způsob jejich implementace.

Běžnou praxí v Unity je použít *Immediate Mode GUI* (také *IMGUI*) pro vytvoření tlačítek na obrazovce, která slouží jako cheaty. Taková příprava UI by však byla při větším množství cheatů poměrně náročná a obrazovka by byla všemi tlačítky dost zaplněná. Navíc bychom museli řešit také zadávání parametrů. Právě proto jsme raději zvolili podobu příkazového řádku. Ten by navíc vypadal lépe i ve finální hře, kdybychom se nakonec rozhodli v ní cheaty ponechat.

Každý cheat by měl nějaký svůj jednoslovný příkaz, dále metodu pro naparsování případných parametrů a provedení a nakonec nějaký svůj popis, který by se zobrazil při vyžádání nápovědy. Dávalo by tedy smysl cheat uzavřít do nějaké třídy. Současně by však bylo zbytečně komplikované definovat speciální třídu zvlášť pro každý cheat, neboť je potřeba jen jedna jediná instance. Definujeme tedy jen jednu obecnou třídu, od které vytvoříme rovnou instance specializované na konkrétní cheat a parsovací metodu předáme jako lambda funkci.

Mimo jednotlivé cheaty pak budeme mít také objekt, který je bude všechny zastřešovat. Ten bude obsahovat `Dictionary` mapující příkaz na instanci cheatu. Kdykoliv se pak do řádku něco zadá, první slovo se bude interpretovat jako příkaz a na základě něj se získá instance, aby bylo možné naparsovat zbytek.

Dále bychom chtěli zařídit, aby byly v různých scénách k dispozici různé cheaty. Tak bychom alespoň trochu zabránili nežádoucím situacím. Mohli bychom tedy rovnou při vytváření instance vytvořit také pole s názvy scén, ve kterých je příkaz povolený či zakázaný (dle toho, který výčet je kratší). Je však nutno podotknout, že použití cheatů je na vlastní riziko, neboť i tak mohou vzniknout nestandardní situace.

Cheaty by měly být co nejvíc nezávislé, běžný herní kód by o nich vůbec neměl vědět. Měly by tedy využívat pouze veřejné API ostatních pro změnu stavu hry, registrovat callbacky a/nebo reagovat na příjem zprávy.

Během testování různých částí hry pak může být třeba často opakovat ty samé cheaty dokola (např. při každém spuštění závodu). Přidáme tedy zapamatování historie, aby bylo snadné zopakovat již dříve zadaný příkaz. Navíc ale zavedeme také možnost specifikovat seznam cheatů, které se provedou při inicializaci nebo při každém načtení konkrétní scény.

Návrh cheatů si pak usnadníme tím, že nebudeme nijak podporovat lokalizaci. Jedná se o interní nástroj pro vývojáře a testery, který by hráč neměl ideálně vůbec používat. Nemělo by tedy vadit, pokud bude pouze v angličtině.

## 3.10 Rozšíření editoru

Unity nabízí vývojářům prostředky, jak vylepšit editor pro usnadnění práce. My bychom je během vývoje rádi využili a pokusili se implementovat několik vlastních rozšíření editoru, obvykle v podobě úpravy Inspector okna pro konkrétní typy komponent. Obecně nás však vede k implementaci vlastního rozšíření několik různých důvodů:

- Pokud chceme řídit to, kdy jsou které datové položky v Inspector okně viditelné a kdy ne (např. na základě hodnoty v nějaké jiné položce). Příkladem může být nastavování parametrů jednotlivých kouzel, kdy se dle typu cíle mohou nastavovat jiné parametry.

- Pokud chceme zobrazit v Inspector okně nějaké varování nebo chybu (třeba pokud chybí některá důležitá komponenta nebo není vyplněná datová položka). Příkladem může být opět kouzlo, kdy komponenta zodpovědná za řízení efektu potřebuje komponentu obsahující skutečný funkční efekt.

- Pokud chceme zobrazit nějaké okno (třeba dialog). Příkladem může být zobrazení dialogu pro zvolení jména assetu při vytváření prefabu kouzla.

- Pokud chceme upravit, jakým způsobem se něco v Inspector okně zobrazuje. Např. pokud bychom chtěli zobrazit některé datové položky vedle sebe nebo různě odsazené.

Všechny skripty, které nějakým způsobem rozšiřují editor, pak musí být ve složce `Editor/`.

Pokud bychom tedy chtěli nějakým způsobem ovlivnit způsob zobrazení konkrétní komponenty v Inspectoru, můžeme přímo pro ni vytvořit vlastní rozšíření editoru, ve kterém bychom pak implementovali požadovaný vzhled. V kódu bychom přistupovali k jednotlivým serializovaným položkám komponenty a s nimi nějakým

způsobem pracovali. Kdybychom však místo datové položky potřebovali takto pracovat s vlastností definovanou v komponentě, musíme ve skutečnosti získat její backing field, protože právě ten obsahuje hodnotu a serializuje se pro zobrazení v Inspectoru. Jestliže u dané vlastnosti využíváme automaticky generovaný backing field, můžeme ho v kódu rozšíření editoru získat pomocí jeho automaticky vygenerovaného názvu. Např. vlastnost `Identifier` bude mít backing field pojmenovaný `<Identifier>k_BackingField`. Spoléhali bychom se tak však na implementační detail překladače, který se ale může někdy v budoucnu změnit. Bezpečnější by tak bylo k vlastnosti v komponentě přidat vlastní backing field, jehož jméno pak dobře známe a můžeme ho dle něj snadno získat.

Dále existuje několik dalších situací, ve kterých se hodí rozšířit editor nějakým jednodušším způsobem a které bychom mohli využít:

- Pokud chceme přidat nějakou položku do menu editoru, např. pro vytvoření nějakého objektu s již přednastavenými komponentami. Příkladem by pak bylo vytvoření assetu s prefabem kouzla nebo umístění objektu s lokalizovaným textem do scény.
- Pokud chceme přidat nějakou položku do kontextového menu nějaké komponenty, např. aby se zavolala nějaká metoda, něco se uložilo do souboru apod. Konkrétním příkladem by mohla být možnost uložit vygenerovanou mesh terénu jako asset (pro pozdější využití jinde).
- Pokud chceme nějak základně pracovat s editorem. Můžeme chtít třeba zobrazit nově vytvořený asset s prefabem kouzla v Project okně.

## 3.11 Úskalí práce s enginem Unity

V průběhu práce na naší hře jsme narazili na spoustu nečekaných situací. Některé z nich nyní uvedeme společně s řešením, které jsme nakonec museli aplikovat.

### Model a animace postavy

Když jsme vytvářeli model pro postavu hráče a základní idle animace, setkali jsme se s několika problémy.

Prvním problémem bylo, že některé části modelu mizely, zřejmě pod určitým úhlem kamery. Nakonec bylo třeba zaškrtnout `Update When Offscreen` u jednotlivých částí modelu v jejich `SkinnedMeshRenderer` komponentě. Pro zlepšení výkonu se totiž ve výchozím nastavení neaktualizují části, které nejsou viditelné. V našem případě zřejmě docházelo ke špatnému určení viditelnosti. Zaškrtnutím této možnosti se pak bounding volume meshe přepočítává v každém snímku.

Druhým problémem bylo, že u některých částí modelu při určitém směru pohledu trčely nějaké trojúhelníky do prostoru. Nakonec jsme to vyřešili nastavením `Quality` na `1 Bone` ve `SkinnedMeshRenderer` komponentě odpovídající části. Toto nastavení ovlivňuje maximální počet kostí, které se berou v úvahu při skinningu.

### Pořadí metod z `MonoBehaviour`

Veškeré komponenty rozšiřující `MonoBehaviour` mají několik významných metod, např. inicializační metody `Awake()`, `Start()` a `OnEnable()`. Pořadí volání těchto metod je pevně dané [60] a je důležité jej při vývoji zohlednit. Víme tedy naprosto přesně, že při inicializaci objektu se zavolá nejprve `Awake()`, následně `OnEnable()` a až nakonec `Start()`.

Toto pořadí je však zajištěno pro každý objekt zvlášť. Už se tedy nemůžeme spoléhat na to, že by bylo stejné pořadí zachováno také mezi jednotlivými objekty (tj. nejprve `Awake()` pro všechny objekty, teprve poté `OnEnable()`). Ve skutečnosti jsme pozorovali, že běží `Awake()` a `OnEnable()` pro každý objekt hned za sebou, teprve pak se přechází na další. Unity však negarantuje ani toto pořadí. Jediné dokumentované chování je to, že ve chvíli, kdy běží `Start()` již zaručeně proběhly veškeré `Awake()` a `OnEnable()` metody.

### Hierarchie komponent

Pokud závodník nemá vybavené žádné kouzlo, chtěli bychom deaktivovat komponenty související se sesíláním kouzel, aby neběžely zbytečně. Abychom je však neměli napevno v kódu, rozhodli jsme se vytvořit `List`, do kterého bychom odpovídající komponenty nastavili v Inspector okně.

Narazili jsme ovšem na problém při pokusu o deaktivaci collideru (konkrétně `SphereCollider`), který je kolem závodníka pro detekci potenciálních cílů kouzel. Není totiž potomkem `MonoBehaviour`, ale až třídy `Component`, která však nemá `enabled` pro možnost deaktivace. To totiž dodává až `Behaviour` (následník `Component`, předek `MonoBehaviour`). U collideru je pak k dispozici jen proto, že si ho definuje zvlášť.

Nakonec tedy nebylo možné deaktivovat komponenty jednotně. Museli jsme jedině využít `UnityEvent`, kde jsme postupně pro každou komponentu určili, aby se deaktivovala.

#### `AnimationEvent`

Když jsme implementovali animace obličeje pro závodníky, chtěli jsme spustit animaci úsměvu během cutscene po dokončení závodu. Využili jsme na to `AnimationEvent`, která umožňuje zavolat funkce ze skriptu objektu v určitém okamžiku animace. Během testování v editoru vše fungovalo, jak má, ale v buildu se animace úsměvu vůbec nespustila, ačkoliv obecně animace obličeje na jiných místech fungovaly.

Pomocí ladicích výpisů jsme zjistili, že se událost z animace vůbec nevyvolala. Následně jsme našli také různé příspěvky na diskuzním fóru [61], které popisovaly podobný jev. Zřejmě se tedy někdy může stát, že se událost nevyvolá, pokud je ke konci animace.

Nakonec jsme se proto pokusili místo `AnimationEvent` použít `Timeline` a signály v ní. Poté se animace spouštěla jak v editoru, tak v buildu. Zdá se tedy, že použití `Timeline` je mnohem spolehlivější variantou.

### Synchronizace s coroutines

Jednotlivé části tutoriálu se řídí z coroutines, aby bylo možné snadno pozastavit běh, dokud nebude splněna podmínka pro posun na další krok. V určitém okamžiku jsme však potřebovali během tutoriálu pozastavit hru, aby se mezitím neměnil stav na pozadí. I když jsme tedy zavolali námi implementovanou metodu pro pozastavení hry, kdy se nastaví `Time.timeScale = 0`, hra běžela dál a `Time.timeScale` bylo opět 1.

Experimentálně jsme zjistili, že pozastavení mimo coroutine (např. z metody `Update()`) však funguje správně. Pravděpodobně je tedy nějaký problém se synchronizací mezi coroutine, updatem a detekcí vstupu (protože se hra měla pozastavit, dokud hráč neklikne myší). Nakonec jsme tedy museli v coroutine pouze nastavit `bool` proměnnou, že bychom chtěli pozastavit hru, a tuto proměnnou pak kontrolovat v `Update()` a na základě ní hru skutečně pozastavit.

### `Vector3` a `Vector2`

Když pracujeme s vektory, dosazení `Vector3` do proměnné typu `Vector2` nebo předání `Vector3` jako parametr typu `Vector2` projde bez jakéhokoliv varování. Může tak velmi snadno docházet k těžko detekovatelným chybám.

### Serializace vlastností

Pokud v nějaké komponentě použijeme veřejné vlastnosti místo datových položek, nedochází automaticky k jejich serializaci a nejsou tak vidět v Inspector okně. Jestliže ale má vlastnost setter, třeba i jen privátní, můžeme využít atribut `[field:SerializeField]`, který se aplikuje na automaticky generovaný backing field vlastnosti. Díky tomu můžeme dobře naznačit záměr (např. viditelné zvenčí pomocí veřejného getteru, ale bez možnosti změny pomocí privátního setteru), zatímco máme stále možnost měnit hodnotu v Inspector okně. Pokud bychom pak definovali backing field sami, stačí přidat `[SerializeField]` atribut k němu.

Možnost změny hodnoty přímo v Inspector okně má pak několik výhod:

- Hodnota se ukládá přímo se scénou, takže je nezávislá na původní hodnotě nastavené v kódu.
- Hodnota se dá měnit bez nutnosti překladu kódu.
- Hodnota se dá měnit snadno i za běhu, což se hodí při ladění.
- Je to přístupnější pro návrháře, kteří tak vidí rovnou nastavitelné parametry a nemusí rozumět kódu.

### Volání `Destroy()` mimo komponentu

Na několika různých místech jsme potřebovali zničit všechny potomky nějakého objektu. Hodilo by se tedy vytvořit si pomocnou statickou metodu, která by se pak mohla použít opakovaně. Abychom však mohli volat metodu `Destroy(Object obj)` pro zničení nějakého herního objektu, musíme být v kódu třídy rozšiřující `MonoBehaviour`, tj. komponenty. Už však není nutné, aby byla tato komponenta na nějakém objektu umístěném ve scéně. Stačí tedy všechny takové pomocné metody umístit do třídy rozšiřující `MonoBehaviour`.

# 4 Vývojová dokumentace

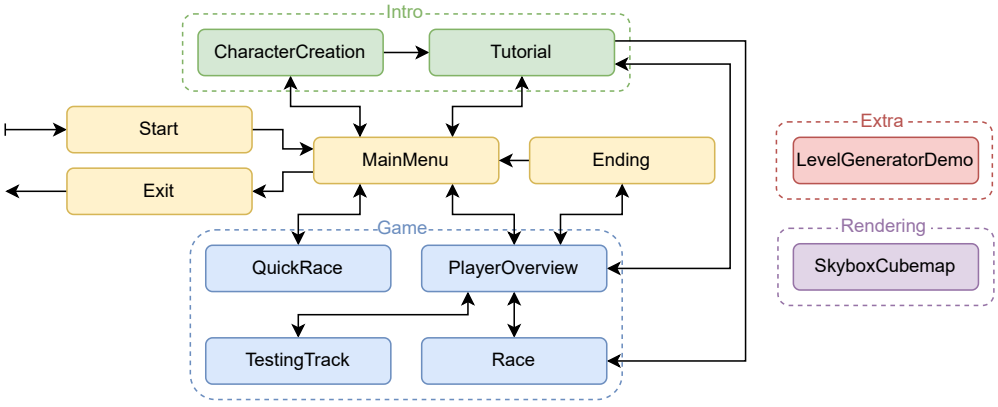
V předchozích kapitolách jsme si popsali proces, který vedl k vytvoření návrhu hry, a učiněná rozhodnutí týkající se různých technických řešení. V této kapitole si představíme již finální implementaci, která vznikla právě na základě předchozího popisu. Ta je obsažena v projektu `Brooom`, který byl vytvořen v Unity editoru verze 2021.3.23f1 (tj. nejnovější LTS verzi dostupné v době zahájení vývoje) a nachází se také v elektronické příloze této práce (ve složce `src/`).

## 4.1 Struktura projektu

Projekty vytvářené v Unity se skládají ze *scén*, do kterých se umisťují herní objekty, ke kterým se pak připojují komponenty zajišťující nějaká chování. Programátoři si mohou definovat vlastní komponenty pomocí tzv. *skriptů*. Scény i skripty společně s dalšími potřebnými zdroji (např. prefaby, modely, sprity) jsou uložené jako assety v adresáři projektu (v podadresáři `Assets/`). Rozhodli jsme se pro ně použít hierarchii adresářů na základě typu obsahu (např. `Assets/Scenes/`, `Assets/Scripts/` nebo `Assets/Prefabs/`).

### 4.1.1 Scény

Unity projekt pro naši hru tvoří celkem 12 scén, z nichž 2 jsou však pouze pomocné a nejsou dosažitelné v rámci samotné hry. Na obrázku 4.1 vidíme možné přechody mezi scénami a také jejich rozdělení do skupin, které jsou pak uložené ve společném adresáři. Nyní každou ze scén stručně popíšeme, abychom získali celkový přehled o jednotlivých částech hry, které pak probereme detailněji v následujících sekcích této kapitoly.




**Obrázek 4.1** Přechody mezi scénami.

- `Start` se načte jako první po spuštění hry a slouží pro inicializaci všeho potřebného. Především obsahuje singletony, které načítají externí data. Po dokončení inicializace se přejde do `MainMenu`.

- **MainMenu** obsahuje UI hlavního menu (popsáno v sekci 4.11). Z něj je možné spustit novou hru (načte se scéna `CharacterCreation`), pokračovat v rozehrané hře (načte se scéna `PlayerOverview`, případně `Tutorial`, pokud byla hra přerušena uprostřed úvodního tutoriálu), nebo spustit režim rychlého závodu (načte se scéna `QuickRace`). Dále umožňuje zobrazit nastavení a informace o hře, zvolit jazyk nebo úplně odejít ze hry (načte se scéna `Exit`).

- **CharacterCreation** umožňuje hráči zvolit si postavu, se kterou pak hraje ve hře (více v sekcích 4.6.1 a 4.11). Na výběr má z několika možností a může využít také náhodnou inicializaci. Ze scény se může buď vrátit zpět do `MainMenu`, nebo potvrdit volby a přejít do `Tutorial` scény.

- Ve scéně **Tutorial** se odehrává úvodní tutoriál po spuštění nové hry, ve kterém se hráč naučí létat a seznámí se se základními prvky trati. Po dokončení hráč přejde do prvního závodu ve scéně `Race`. Později v této scéně probíhá také tutoriál sesílání kouzel, ze kterého se hráč vrátí do `PlayerOverview`. Detailnější popis tutoriálu je v sekci 4.9. Po pozastavení hry během tutoriálu může hráč odejít do `MainMenu`.

- **PlayerOverview** obsahuje UI pro přehled hráče (viz sekce 4.11), který zobrazuje aktuální hodnoty statistik v grafu, globální žebříček závodníků a získaná ocenění. Dále hráč může přejít do obchodu (viz sekce 4.11), kde si může zakoupit kouzla nebo vylepšení koštěte. V přehledu i v obchodě hráč vidí svá vybavená kouzla a může je měnit. Z přehledu se může vrátit do scény `MainMenu`, nebo přejít do dalšího závodu ve scéně `Race`. Z obchodu pak může jít do testovací trati v `TestingTrack`. Pokud si hráč koupil první kouzlo, spustí se tutoriál načtením `Tutorial`, a pokud dosáhnul konce hry, chviličku po zobrazení přehledu se načte scéna `Ending`.

- Scéna **TestingTrack** (více v sekci 4.8) obsahuje pevně daný level s menší oblastí vyhrazenou ochrannou bariérou. V ní jsou umístěné některé základní prvky trati (např. obruč, bonus) a krouží v ní jeden soupeř. Hráč má možnost létat a sesílat kouzla bez omezení. Stisknutím `ESC` se pak může vrátit zpět do `PlayerOverview` s již otevřeným obchodem.

- Ve scéně **Race** se odehrávají jednotlivé závody (viz sekce 4.3). Na začátku se procedurálně vygeneruje level. Poté má hráč možnost libovolně tratí prolétávat, než se rozhodne skutečně zahájit závod. Po dokončení závodu se zobrazí výsledky a hráč bude moct přejít do `PlayerOverview`. Pokud by se hráč rozhodl závod vzdát z menu pozastavené hry, pak přejde také do `PlayerOverview`.

- Scéna **Ending** (viz sekce 4.11) se zobrazí, jakmile hráč dosáhne konce hry. Obsahuje pouze animované UI a možnost pokračovat dál ve hře (načte se `PlayerOverview`), nebo hru ukončit (načte se `MainMenu`).

- Ve scéně **QuickRace** se odehrává jeden rychlý závod (více v sekcích 4.3 a 4.11). Nejprve se procedurálně vygeneruje level a následně se rovnou zahájí závod. Po jeho absolvování se hráči zobrazí výsledky s tlačítkem

pro návrat do `MainMenu`. Z menu pozastavené hry má hráč možnost vrátit se do `MainMenu` také kdykoliv během závodu.

- **`Exit`** je úplně poslední scénou, která se načte při ukončování hry. Díky ní se zajistí, že objekty přetrvávající mezi scénami se zničí až jako poslední poté, co k nim ostatní objekty přistoupily naposledy.

- **`LevelGeneratorDemo`** je scéna dostupná pouze pomocí cheatu pro změnu scény, konkrétně `scene LevelGeneratorDemo` (viz sekce 4.13.1). Obsahuje UI pro generování levelu dle zadaných parametrů (viz sekce 4.11). Z této scény se dá pomocí tlačítka přejít do `MainMenu`.

- **`SkyboxCubemap`** je pomocná scéna obsahující skripty pro náhodné rozmístění mraků a renderování do *skyboxu* (více v sekci 4.10.10), který se pak může použít v jiných scénách.

#### Načítání scén

Funkcionalitu načítání scén (které jsme popsali již v sekci 3.9.2) zajišťuje skript `SceneLoader`, který je navěšený na stejnojmenném objektu ve scéně `Start`. Implementuje `MonoBehaviourSingleton<T>` (více popsaný v sekci 4.10.9), díky čemuž přetrvává mezi scénami a je dostupný komukoliv a odkudkoliv. `SceneLoader` poskytuje veřejné metody pro načtení scény (udává se jako hodnota výčtového typu `Scene`) a navíc umožňuje zaregistrovat callback `onSceneStartedLoading` na zahájení načítání scény a `onSceneLoaded` na dokončení načítání scény. Objekt `SceneLoader` má navíc podobjekty obsahující UI pro načítací obrazovku, která se zobrazuje ve vhodnou chvíli. Skript `LoadingSceneUI` pak řídí animace a audio.

Pokud se nějaký objekt inicializuje dlouho, je lepší oddělit náročnou část inicializace mimo metody `Awake()` a `Start()`. Pak může takový objekt rozšiřovat `MonoBehaviourLongInitialization` a vhodně implementovat jeho abstraktní metody pro rozdělení inicialize. Tím se objekt také zaregistruje u `SceneLoaderu`, takže se na něj pak čeká a načítací obrazovka se skryje teprve, když je objekt inicializovaný.

`SceneLoader` navíc poskytuje metody pro možnost předání parametrů scéně, která se bude načítat jako další. Tato scéna však musí obsahovat objekt se skriptem odvozeným od `SceneLoadingParameters`, aby bylo možné jí parametry předat. Příkladem implementace je skript `PlayerOverviewLoadingParameters` na objektu `SceneLoadingParameters` ve scéně `PlayerOverview`, který při obdržení parametru `OpenShop` nastaveného na `true` rovnou po načtení scény zobrazí obchod.

### 4.1.2 Skripty

Během vývoje jsme pro psaní zdrojového kódu v jazyce C# použili Visual Studio 2019. Z Unity editoru se tak vytvořil *solution* `Brooom.sln`, který sice není v příloze této práce, ale dá se z otevřeného Unity projektu vygenerovat. V solution se pak nachází dvě zásadní *assemblies*:

- `Assembly-CSharp` – zdrojový kód pro veškerý herní obsah, např. UI, herní mechaniky (v následujících sekcích se budeme věnovat především této části),

- `Assembly-CSharp-Editor` – zdrojový kód pro rozšíření editoru, která jsme si sami implementovali (více popsaná v sekci 4.12).

Samotné skripty jsou ve složce `Assets/Scripts/` rozdělené do několika podsložek, obvykle dle herního systému, ke kterému se vztahují. Ve skriptech pak používáme běžné zvyklosti vývoje v Unity. Mezi ně patří především využívání speciálních metod, např. `Awake()`, `Start()`, `OnCollisionEnter()`, a také využívání různých atributů ovlivňujících zobrazení datových položek v Inspector okně, např. `[SerializeField]` pro zviditelnění neveřejné datové položky, `[Header("title")]` pro rozdělení datových položek do skupin s nadpisem nebo `[Range(min, max)]` u číselné položky pro zobrazení posuvníku se zadaným rozsahem.

Po celou dobu jsme se snažili dodržovat běžné zásady dobrého vývoje softwaru. Dle toho jsme vše navrhli tak, aby to bylo co nejvíc univerzální, znovupoužitelné a parametrizované (pokud to tedy dává smysl). Současně je vše navrženo a pojmenováno tak, aby to bylo samodokumentující se. U tříd a veřejných metod jsou však doplněné také dokumentační komentáře, které mohou vysvětlit nějaké komplikovanější koncepty. Důležitou částí dokumentace jsou také tooltipy u veřejných parametrů komponent. Ty se pak zobrazují v Inspector okně a blíže vysvětlují, o co se jedná a/nebo jak se daná hodnota používá.

V některých částech kódu je možné narazit na `TODO` komentáře, případně na celé kousky kódu v komentářích. Jejich výskyt je záměrný, neboť se jedná zpravidla o něco, co bude implementováno teprve v budoucnu nad rámec demo verze. Umožňují nám ale označit přímo odpovídající místo v kódu a také naznačit možný způsob provedení.

### 4.1.3 Použité balíčky a zdroje třetích stran

Během vývoje jsme použili několik dalších zdrojů nad rámec toho, co je v Unity k dispozici ve výchozím stavu. Jedná se např. o následující balíčky:

- **TextMesh Pro** [62] pro hezčí zobrazení textu v UI a bohatší možnosti formátování.
- **Cinemachine** [63] pro možnost vytvářet virtuální kamery a plynule mezi nimi přecházet.
- **DOTween** [59] pro podporu tweenování různých vlastností několika zásadních komponent.
- **Newtonsoft JSON** [64] pro práci s JSON soubory.
- **FMOD for Unity** [65] pro integraci FMOD do Unity (zpřístupňuje API a speciální komponenty).

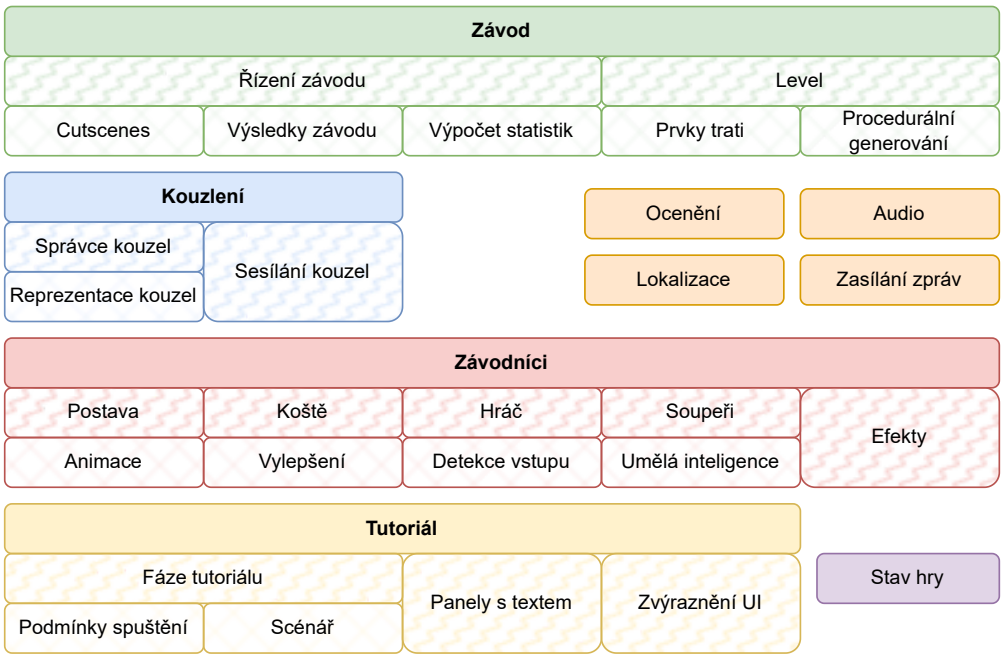
Veškeré audio ve hře je pak získáno z Freesound databáze [66], případně z `itch.io` nebo `kenney.nl`. Fonty jsme zvolili z Google fontů [67]. Až na pár výjimek, uvedených v příloze A.2, se jedná o zdroje s licencí Creative Commons 0, takže je možné je volně využít a modifikovat bez nutnosti uvedení.

## 4.2 Architektura

V sekci 4.1.1 jsme si již popsali rozdělení na jednotlivé scény. Nyní se však zaměříme na popis architektury aplikace z hlediska systémů, které ji tvoří. Některé systémy se totiž používají ve více různých scénách a některé dokonce přetrvávají mezi scénami, takže jejich životnost je určena dobou běhu hry.

Obecně můžeme rozlišit celkem tři kategorie obsahu – herní logika, uživatelské rozhraní a pomocné systémy. Tyto kategorie se však navzájem nevylučují, takže např. nějaký pomocný systém může mít přesah také do uživatelského rozhraní. Nyní se tedy podíváme postupně shora dolů, jaké systémy jsou ve hře a jaké další podsystémy pak využívají ke svému fungování.

Na obrázku 4.2 vidíme náznak architektury obsahující klíčovou herní logiku, rozdělenou do několika částí, a také některé pomocné systémy. Všechny tyto části si stručně představíme.




**Obrázek 4.2** Klíčové části herní logiky a některé pomocné systémy.

- **Závod** (viz sekce 4.3) je nejdůležitější z nich, protože zastřešuje veškerý průběh závodu od začátku až do konce. Obsahuje komponentu, která vše řídí a spouští také cutscenes. Dále využívá generátor levelu pro přípravu trati se všemi jejími prvky (viz sekce 4.4). Po dokončení závodu zobrazí výsledky a vyvolá výpočet statistik.
- **Kouzlení** (viz sekce 4.5) je další velkou částí herní logiky. Zahrnuje v sobě reprezentaci kouzel, udržuje přehled o aktuální dostupnosti kouzel a také obsahuje komponenty, které využívají závodníci pro sesílání kouzel během závodu.
- **Závodníky** (viz sekce 4.6) můžeme uvažovat jako další část logiky, která zahrnuje např. model postavy s animacemi, detekci vstupu od hráče pro umožnění

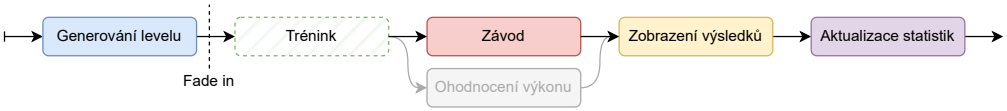
pohybu (s možností přemapování) a umělou inteligenci pro pohyb soupeřů. Každý závodník má navíc koště se specifickými vylepšeními a během závodu může být ovlivňován různými efekty z kouzel a bonusů.

- Na pozadí hry pak běží **tutoriál** (viz sekce 4.9) rozdělený do několika částí, které se vyvolávají ve vhodný okamžik. Pro zobrazení informací hráči tutoriál využívá panely s textem a zvýraznění konkrétní části uživatelského rozhraní.
- Mimo herní logiku jsou pak různé pomocné systémy, např. **správce ocenění** (viz sekce 4.10.2), který zaznamenává pokrok hráče v získaných oceněních, **audio systém** (viz sekce 4.10.6), **lokalizace** (viz sekce 4.10.3) nebo **zasílání zpráv** (viz sekce 4.10.8) pro nepřímou komunikaci mezi objekty. Tyto systémy jsou pak využívány z různých míst herní logiky či z uživatelského rozhraní.
- A nakonec je součástí hry také reprezentace jejího **stavu** (viz sekce 4.10.1), který je implementován jako globálně dostupný singleton (více v sekci 4.10.9).

Ve zbytku kapitoly se podíváme na implementaci zmíněných i dalších systémů ještě blíže. Začneme popisem zásadních částí herní logiky, následně popíšeme pomocné systémy (viz sekce 4.10), pak přejdeme k popisu uživatelského rozhraní (viz sekce 4.11) a na závěr popíšeme implementovaná rozšíření editoru (viz sekce 4.12).

## 4.3 Řízení závodu

Jednotlivé závody se odehrávají ve scéně `Race`, přičemž veškeré dění můžeme popsat jako posloupnost několika fází (na obrázku 4.3). Jakmile hráč vstoupí do dalšího závodu v rámci režimu kariéry, načte se tato scéna a okamžitě se začne generovat nový level. Teprve poté se skryje načítací obrazovka. Hráč nejprve začne ve fázi tréninku (pokud ji nemá vypnutou v nastavení, viz sekce 4.11), ze které pak může kdykoliv přejít již přímo do závodu, po jehož dokončení se mu zobrazí výsledky. Po celou dobu závodu se pak ohodnocuje hráčův výkon. Všechny tyto části jsme podrobně popsali již v sekci 2.6.




**Obrázek 4.3** Jednotlivé fáze závodu ve scéně `Race` (trénink může být přeskočen na základě zvoleného nastavení).

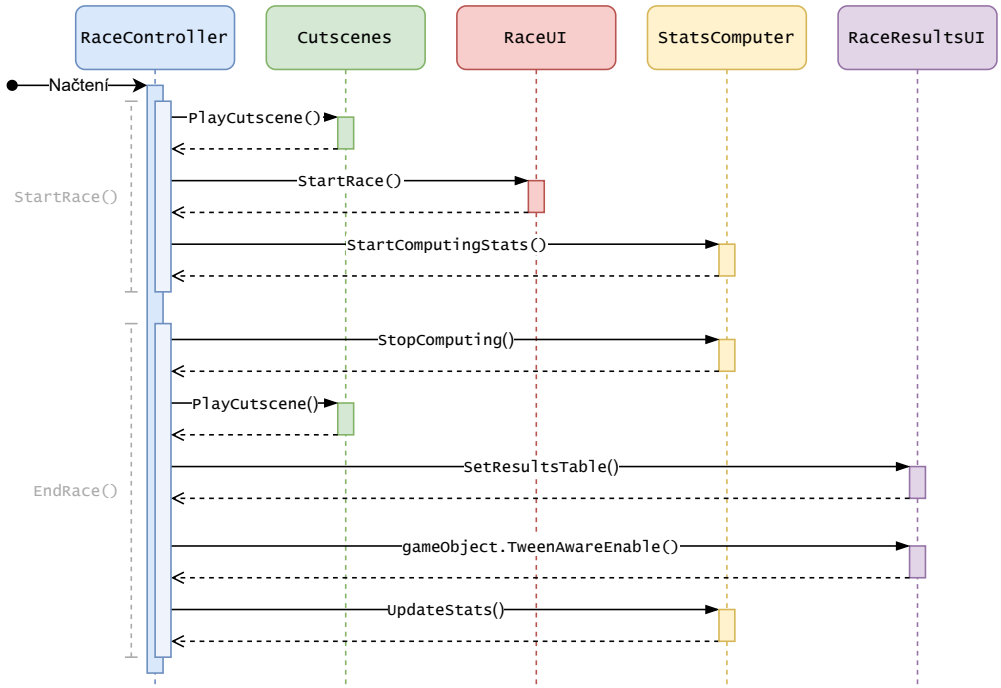
Ve scéně `Race` také najdeme všechny důležité objekty s jejich skripty. Zcela zásadní je pak objekt `RaceController`, který propojuje všechny fáze řízení závodu. Podobně také pro řízení rychlého závodu (popsaného v sekci 2.10) je ve scéně `QuickRace` objekt `RaceController`.

Oba zmíněné objekty mají komponentu odvozenou od `RaceControllerBase`, která poskytuje základ pro logiku spojenou s průběhem závodu. Jedná se o jednoduchý singleton s instancí ve statické datové položce. Udržuje si přehled o aktuálním stavu závodu (tj. rozlišení tréninku a závodu, z výčtového typu `RaceState`), má

reference na všechny závodníky (v podobě třídy `RacerRepresentation`, která sdružuje reference na několik jejich komponent), spouští cutscenes na začátku a na konci závodu a zobrazuje výsledky závodu. Současně se stará také o zvýraznění následující obruče a o vizuální indikaci minuté obruče nebo letu špatným směrem.

Ve scéně `Race` se pak používá odvozená komponenta `RaceController`, která přidává funkcionalitu navíc. Např. zajišťuje vyhodnocování výkonu během závodu a výpočet statistik po závodě, určuje finanční odměnu jednotlivých závodníků za dokončení závodu. Oproti tomu scéna `QuickRace` využívá svou vlastní implementaci `QuickRaceController`, která zajišťuje automatické přeskočení fáze tréninku.

Na obrázku 4.4 vidíme, jak `RaceController` komunikuje s ostatními komponentami. Když je ukončen trénink a zahájen závod, `RaceController` spustí úvodní cutscene pomocí komponenty `Cutscenes`, inicializuje uživatelské rozhraní v komponentě `RaceUI` a zahájí ohodnocování výkonu hráče v závodě, které počítá komponenta `StatsComputer`. Po dokončení závodu pak nejprve ukončí ohodnocování výkonu, pak spustí závěrečnou cutscene, následně zobrazí výsledky závodu pomocí `RaceResultsUI` a nakonec zajistí aktualizaci hodnot statistik.




**Obrázek 4.4** Komunikace `RaceController` komponenty s ostatními na začátku a na konci závodu.

### 4.3.1 Cutscenes

Na začátku a na konci závodu se spouští animované sekvence. Ty jsou ve scénách `Race` i `QuickRace` reprezentované objektem `Cutscenes`. Jedna konkrétní sekvence je pak určená jeho podobjektem s komponentou `PlayableDirector`, která přehrává `Timeline`, ve které se může přepínat mezi různými kamerami,

(de)aktivovat objekt apod. Všechny tyto sekvence se najdou pomocí skriptu `Cutscenes` na kořenovém objektu a pak je možné je pomocí této komponenty také spouštět podle jména.

V obou sekvencích se využívají instance prefabu `CutsceneCamera`, který reprezentuje jednu kameru s určitým přednastaveným chováním (dle rozboru v sekci 3.9.5). V její komponentě `CutsceneCamera` je pak zvoleno právě jedno z těchto chování a u jednotlivých instancí jsou nastaveny vhodné parametry.

### 4.3.2 Detekce postupu v trati

Každý závodník potřebuje nějakou reprezentaci svého stavu, zachycující, jak daleko je v trati. K tomu slouží komponenta `CharacterRaceState`, kterou pak má každý závodník svou. V ní je zaznamenáno několik údajů, např. jaká je pro daného závodníka následující obruč, kterými obručemi proletěl a které minul nebo jaký je čas dokončení závodu.

Ve skriptu `CharacterRaceState` jsou pak také různé metody pro změnu stavu. Např. pokud závodník proletí obručí, zavolá se metoda `OnHoopPassed()`, a pokud proletí cílem, zavolá se `OnFinishPassed()` (která v případě hráče zajistí ukončení celého závodu ve spolupráci s `RaceController` skriptem popsaným výše). Samotný skript se však v metodě `Update()` automaticky stará o aktualizaci některých hodnot (např. pozice závodníka v závodě, detekce letu špatným směrem, detekce minuté obruče).

Kromě toho komponenta poskytuje možnost zaregistrovat callbacky na zajímavé události. Ty využívá např. uživatelské rozhraní během závodu (více v sekci 4.11) pro aktualizaci zobrazených informací.

### 4.3.3 Výsledky závodu

Jakmile hráč doletí do cíle, závod tím končí. Jelikož v tu chvíli ale nemuseli být ještě všichni soupeři v cíli, je potřeba pro ně dopočítat výsledky, aby bylo možné je zobrazit, aniž by hráč musel čekat, než závod skutečně všichni dokončí. Toto se děje v metodě `CompleteOpponentState()` skriptu `RaceControllerBase`, který jsme popisovali už výše. V rámci doplnění je třeba určit počet minutých obručí a čas dokončení závodu. Využíváme přitom dosavadní hodnoty, které jen extrapolujeme.

- Spočteme, kolik obručí z těch již absolvovaných závodník minul. Ze zbývajících obručí pak prohlásíme stejnou část obručí za minutou.
- Z aktuálního času a počtu doposud absolvovaných obručí spočteme průměrný čas potřebný na absolvování jedné obruče. Jako čas dokončení trati pak vezmeme tento čas vynásobený celkovým počtem obručí na trati.

Pokud by se tímto výpočtem došlo k lepšímu finálnímu času, než který měl hráč, pak ještě čas uměle navýšíme o dostatečně velkou hodnotu náhodně zvolenou z určitého rozsahu.

### 4.3.4 Výpočet statistik

Nakonec musíme ohodnotit výkon hráče v závodě, abychom získali nové hodnoty statistik, na základě kterých se pak vygeneruje příští level. Na objektu `RaceController` je tak skript `StatsComputer`, který po celou dobu závodu měří všechny potřebné hodnoty (dle popisu v sekci 2.6). Jakmile pak hráč závod dokončí, z naměřených hodnot spočte aktuální ohodnocení výkonu v podobě hodnot jednotlivých statistik. Nakonec tyto nové hodnoty zkombinuje s těmi, které jsou již uložené ve stavu hráče (viz sekce 4.10.1), a uloží je. Začátek a konec závodu pak tomuto skriptu signalizuje již zmíněný `RaceController` zavoláním odpovídajících metod.

Skript `StatsComputer` obsahuje celou řadu parametrů, pomocí kterých se dá výpočet statistik ovlivnit. Příkladem může být frekvence vzorkování aktuální rychlosti, penalizace za náraz do překážky nebo míra tolerovaných chyb. V případě, že hráč závod vzdá, pak tento skript zajišťuje také penalizaci v podobě snížení hodnot statistik.

## 4.4 Level

Každý závod se odehrává v levelu, jehož podobu jsme popsali v sekci 2.4. Navíc jsme určili, že se levely budou procedurálně generovat a v sekci 3.3 jsme nastínili postup, který bychom použili. Nyní tedy popíšeme výslednou implementaci samotného procesu generování levelu společně s infrastrukturou jednotlivých prvků trati a prostředí.

### 4.4.1 Procedurální generování levelu

Kdykoliv potřebujeme vygenerovat level, využíváme prefab `LevelGenerator`, který obsahuje potřebné objekty a komponenty. Díky tomu je možné generovat levely v různých scénách a měnit parametry všech naráz. Instance prefabu se vyskytují jako objekt `LevelGenerator` ve scénách `Race`, `QuickRace` a `LevelGeneratorDemo`. Objekt má pak několik podobjektů, pod které se postupně přidávají generované objekty (např. obruče, prvky prostředí).

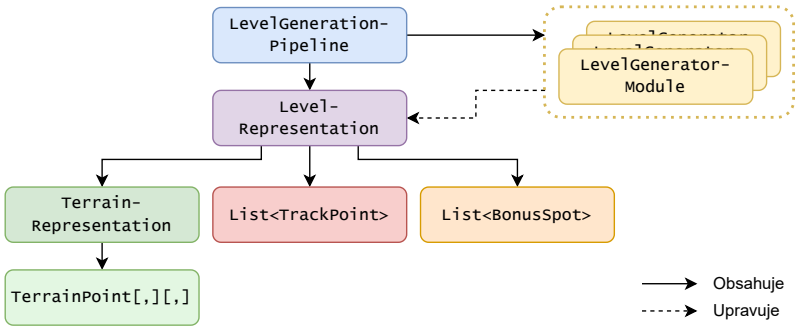
Klíčovou komponentou je `LevelGeneratorPipeline`, která implementuje rozdělení generátoru na jednotlivé moduly přesně tak, jak jsme ho popsali v sekci 3.3.1. Obsahuje tedy seznam modulů implementujících `LevelGeneratorModule`, přičemž u každého se dá zvolit, zda má být aktivní. Když je pak potřeba vygenerovat level, zavolá se metoda `GenerateLevel()`, ve které se postupně projdou všechny moduly a nakonec se vytvoří mesh terénu.

V komponentě se také nachází celá řada nastavitelných parametrů (např. rozlišení terénu, velikost bloků nebo seznam tematických oblastí). Navíc umožňuje zaregistrovat callback `onLevelGenerated` na dokončení generování levelu.

#### Reprezentace levelu

Všechny moduly pracují nad společnou reprezentací levelu pomocí instance třídy `LevelRepresentation`, kterou dostanou jako parametr ve své `Generate()` metodě. V této třídě se nachází např. popis terénu, trati a použitých tematických

oblastí. Na obrázku 4.5 pak vidíme závislosti mezi klíčovými třídami podílejícími se na generování levelu a jeho reprezentaci.




**Obrázek 4.5** Závislosti mezi klíčovými komponentami a třídami zodpovědnými za procedurální generování levelu a jeho reprezentaci.

Terén je obecně reprezentován pomocí třídy `TerrainRepresentation` a jeden konkrétní bod terénu pak popisuje typ `TerrainPoint`, který zaznamenává důležité údaje, jako je pozice nebo přiřazená oblast. Interně je terén rozdělen do několika bloků (jedná se o jednu z optimalizací popsaných v sekci 3.3.4), avšak tento implementační detail se skrývá definicí indexeru, který umožňuje přistupovat k jednotlivým bodům terénu pomocí souřadnic z celé mřížky. Kromě samotných bodů terénu pak `TerrainRepresentation` zachycuje také různé parametry terénu, např. celkové rozměry nebo počet bloků v jednotlivých osách.

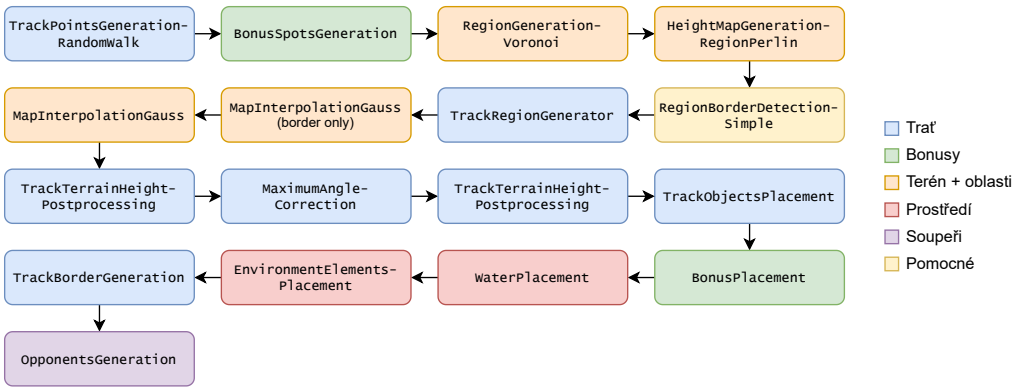
Trať je reprezentována jako pole instancí třídy `TrackPoint`. Ty představují body trati, ve kterých se nachází obruč nebo kontrolní bod, a obsahují související parametry. Podobně je v `LevelRepresentation` pole instancí `BonusSpot`, které reprezentují potenciální body, do kterých mohou být umístěny bonusy. Nakonec je v reprezentaci také uložena startovní pozice trati a reference na objekt cílové čáry.

### Použité moduly

Na obrázku 4.6 vidíme, které moduly jsme nakonec použili ve finální implementaci a v jakém pořadí běží. Můžeme si navíc všimnout, jak se moduly generující různé části levelu prolínají.

Nyní si každý z modulů stručně představíme ve stejném pořadí, v jakém se používají v generátoru. Jejich implementace je vystavená na analýze možných přístupů ze sekce 3.3 a nachází se ve složce `Assets/ProceduralGeneration/Modules`.

Jako první běží modul `TrackPointsGenerationRandomWalk`, který generuje významné body trati, do kterých se později umisťují obruče a kontrolní body. Pro generování využívá náhodnou procházku tak, jak jsme ji popsali v sekci 3.3.3, včetně prevence překřížení trati. Vygenerované body se pak vycentrují kolem počátku systému souřadnic, abychom co nejlépe využili vyšší přesnost menších desetinných čísel. V parametrech modulu se dá nastavit např. vzdálenost po sobě jdoucích bodů, počet bodů nebo vzdálenost startovní pozice od první obruče. Jelikož bychom mohli v budoucnu uvažovat o jiné implementaci, oddělili jsme různé základní metody a parametry do třídy `TrackGenerationBase`, kterou tento modul rozšiřuje a implementuje její abstraktní metodu `GenerateTrackPoints()`.




**Obrázek 4.6** Jednotlivé moduly, ze kterých se skládá generátor levelu, v pořadí, ve kterém běží. Barvy určují část levelu, kterou daný modul generuje/upravuje.

Jakmile máme body trati, můžeme mezi nimi zvolit body, do kterých se mohou potenciálně umístit bonusy, pomocí lineární interpolace (jak jsme popsali v sekci 3.3.3). O to se stará modul `BonusSpotsGeneration` a vytvořené body pak uloží do reprezentace levelu v podobě instancí třídy `BonusSpot`.

S již vygenerovanou tratí pak víme, jaké rozměry by měl mít terén pod ní, takže začneme generovat ten (dle postupu popsaného v sekci 3.3.2). Nejprve se pomocí modulu `RegionGenerationVoronoi` určí tematické oblasti terénu s využitím Voroného diagramu. Každému bodu terénu pak uloží přiřazenou oblast do jeho reprezentace `TerrainPoint`. Následně modul `HeightMapGenerationRegionPerlin` vygeneruje height map terénu. Využívá přitom různé hodnoty parametrů pro různé oblasti (např. rozsah nadmořských výšek, počet oktáv Perlinova šumu, frekvence a amplituda první oktávy).

S určenými oblastmi můžeme určit také body terénu, které se vykytují na hranicích těchto oblastí. To nám umožní s takovými body později pracovat speciálním způsobem (např. terén v nich vyhladit nebo do nich neumisťovat konkrétní prvky prostředí). O to se tedy stará modul `RegionBorderDetectionSimple`. Pro každý bod terénu se jednoduše podívá, jestli se v jeho okolí v osmi různých směrech vyskytuje bod s odlišnou oblastí. V parametrech modulu pak můžeme nastavit výchozí velikost tohoto okolí a případně ji přepsat zvlášť pro různé oblasti.

Kromě oblastí terénu musíme určit také oblasti trati, o což se stará modul `TrackRegionGenerator`. V parametrech obsahuje pro každou takovou oblast pravděpodobnost jejího výskytu, rozsah nadmořských výšek a počet bodů trati, které má pokrývat. Pak se s danou pravděpodobností zvolí úsek trati dané délky a výška bodů v něm se přizpůsobí. Současně se vyhladí rozdíly výšek bodů trati na okraji tohoto úseku. Navíc se zajistí, že pokud se nějaká oblast trati hráči nově zpřístupnila, určitě se v trati vyskytne.

Když už máme vygenerovaný terén a určené oblasti v něm, můžeme ho nyní pomocí modulu `MapInterpolationGauss` trochu vyhladit. Tento modul k tomu využívá Gaussovský kernel, přičemž si v parametrech můžeme vybrat ze 3 velikostí a také můžeme zvolit, kolikrát se má aplikovat. Modul používáme hned dvakrát za sebou. Nejprve pouze v bodech, které jsou označené jako hranice oblastí, kde využijeme více iterací, protože chceme skutečně hladké přechody. Následně v celém terénu, již pouze s jednou iterací, abychom se zbavili příliš ostrých vrcholů terénu.

Jelikož jsme generovali nejprve body trati a až poté terén, mohlo se stát, že jsou body trati zanořené v terénu. Modul `TrackTerrainHeightPostprocessing` tedy u všech bodů trati a bodů pro umístění bonusů zajistí, že budou v nějaké minimální výšce nad terénem v jejich blízkém okolí. V parametrech modulu přitom můžeme nastavit tuto minimální výšku a případně ji přepsat i pro konkrétní oblasti zvlášť.

Použitím předchozího modulu se však mohlo stát, že se porušily maximální úhly vertikální změny směru mezi dvěma po sobě jdoucími body trati, které se dodržovaly při generování trati pomocí náhodné procházky. Jako další tak použijeme modul `MaximumAngleCorrection`, který projde body trati a upraví jejich výšku ještě tak, aby byly úhly dodržené. Nakonec ale pro jistotu spustíme ještě jednou `TrackTerrainHeightPostprocessing`. Pokud by tím byl nějaký bod změněn, mohly by se sice zase porušit maximální úhly, ale rozdíly by už neměly být tak zásadní a navíc by obruče měly být dostatečně daleko od sebe, aby i větší rozdíly byly zvládnutelné.

V tuto chvíli už máme ty nejdůležitější části vygenerované a máme přesně určené body, do kterých můžeme umístit obruče, kontrolní body a bonusy. Jako další tak běží modul `TrackObjectsPlacement` pro umístění obručí, kontrolních bodů, startovní zóny a cílové čáry. Hned po něm se spouští modul `BonusPlacement`, který určí přiřazení různých typů bonusů k různých bodům pro jejich umístění (pomocí vzorů jejich výskytu, jak jsme popsali v sekci 3.3.3) a následně je skutečně vytvoří, přičemž náhodně určí počet instancí v daném bodě z rozsahu zadaného jako parametr.

Poté můžeme dokončit prostředí kolem trati. Modul `WaterPlacement` se stará o umístění vodní hladiny, která je však tvořena pouze jednoduchou poloprůhlednou rovinou, která se umístí doprostřed levelu a roztáhne tak, aby ho celý pokryla. Následně se pomocí modulu `EnvironmentElementsPlacement` do prostředí umístí různé prvky postupem popsaným v sekci 3.3.2. Každá oblast má v parametrech modulu svou vlastní nabídku prvků s různými parametry (např. pravděpodobnost, různé varianty, rozsah možných velikostí). V modulu se pak určují body pro umístění prvků (v mřížce zadaných parametrů) a pak se vybere jeden konkrétní prvek podle oblasti a jejích parametrů, případně se rozhodne o tom, že se neumístí žádný. Současně se však zajistí, že v okolí startovní pozice trati nebudou umístěny žádné vysoké prvky, aby hráči nezakrývaly výhled hned ze začátku.

Na závěr se ještě pomocí modulu `TrackBorderGeneration` umístí ochranná bariéra kolem trati. Každá dvojice po sobě jdoucích bodů trati se ohraničí ze dvou stran a následně se ještě uzavřou oba konce trati. Nakonec se využije modul `OpponentsGeneration`, aby se vytvořili jednotliví soupeři a umístili se kolem hráče na startovní pozici. V parametrech modulu jsou pak nastavené požadované úrovně soupeřů (viz sekce 3.5.1) a barvy, které se jim přiřadí.

Teprvé poté, co doběhnou všechny moduly, se vytvoří také fyzický terén přímo ze samotné `LevelGenerationPipeline` komponenty. Z reprezentace levelu získává data jednotlivých bloků terénu, pro které pak vytváří instance prefabu `TerrainBlock`. Každá taková instance obsahuje mesh, kterou generuje v metodě `GenerateTerrainMesh()` skriptu `TerrainBlock` ze zadané reprezentace levelu a pro zadané indexy bloku.

#### Nastavení parametrů

Abychom mohli vygenerovat level odpovídající hráčovým statistikám (dle požadavku P19), musíme ještě před zahájením generování vhodně nastavit parametry generátoru a jeho modulů, aby ty samotné byly nezávislé na stavu hráče (jak jsme řekli již v sekci 3.3.1).

Pro závod v rámci kariérního režimu se o to stará skript `RaceGeneration` na objektu `RaceController` ve scéně `Race`. Ten obsahuje velké množství datových položek pro nastavení parametrů (např. minimální a maximální počet kontrolních bodů, podmínky zpřístupnění jednotlivých tematických oblastí, počet soupeřů). Během své inicializace pak nejprve nastaví požadované parametry v závislosti na aktuálním stavu hry (více v sekci 4.10.1) a následně spustí generování levelu. Jelikož toto generování trvá netriviálně dlouho, skript současně rozšiřuje `MonoBehaviourLongInitialization` (popsaný v sekci 4.1.1).

Generování levelu v rámci rychlého závodu se však řídí trochu jinými pravidly, takže objekt `RaceController` ve scéně `QuickRace` obsahuje speciální skript `QuickRaceGeneration`. Ten rozšiřuje `RaceGeneration` z normálního závodu, ale přepisuje způsob volby tematických oblastí. V normálním závodě se vždy volí maximálně jedna doposud nenavštívená oblast (a současně se vždy zvolí, pokud je k dispozici), zatímco v rychlém závodě se volí zcela náhodně ze všech dostupných. Dále se liší dostupnost oblasti *Kouzelný les*, která je v kariérním režimu zpřístupněná po dosažení konkrétní fáze tutoriálu, ale v rychlém závodu by toto nedávalo smysl, takže je zpřístupněná hodnotou statistiky *vytrvalosti*.

### 4.4.2 Trať

V sekci 2.4.1 věnované návrhu trati jsme popsali, jaké všechny prvky se na ní nachází. Jelikož se na trati obvykle nacházejí ve větším počtu a navíc jsou umisťovány procedurálně (viz sekce 4.4.1), vytvořili jsme pro ně prefaby, ze kterých pak stačí vytvářet instance. Všechny tyto prefaby jsou ve složkách `Assets/Prefabs/Track` (např. obruč, cílová čára) a `Assets/Prefabs/Bonus` (všechny podporované typy bonusů).

#### Základní prvky

Prefab `HoopBase` tvoří základ pro obruče a kontrolní body. Obsahuje tak společnou logiku ve skriptu `Hoop`, který detekuje závodníky pomocí *trigger colliderů*. Jelikož v Unity neexistuje pro obruč vhodný tvar collideru, použili jsme průnik kvádru a koule. Pokud se tedy bude závodník nacházet v obou současně, na jeho `CharacterRaceState` komponentě (popsané v sekci 4.3.2) se zavolá metoda `OnHoopPassed()`, aby se mohl aktualizovat stav. Skript `Hoop` dále obsahuje metodu pro zvýraznění dané obruče jako té následující. Využívá se instance prefabu `HighlightArrow`, který obsahuje model šipky, jehož pozice se pak animuje, aby se šipka otáčela a pohybovala nahoru a dolů.

Od prefabu `HoopBase` jsou vytvořeny dvě prefab varianty – `Hoop` pro obyčejnou obruč a `CheckpointHoop` pro kontrolní bod. Ke společnému základu tak přidávají svůj specifický model a také ikonku pro zobrazení na minimapě.

Cílová čára je reprezentovaná prefabem `FinishLine`. Ten obsahuje trigger collider pro detekci závodníků, která se pak ošetřuje ve skriptu `FinishLine`.

V takovém případě se na `CharacterRaceState` komponentě daného závodníka zavolá metoda `OnFinishPassed()`. Vykreslení cílové čáry je zajištěno pomocí shaderu `ScrollingTextureShaderDistance`, který zařizuje, že se po povrchu posouvá textura a její průhlednost závisí na vzdálenosti od pozorovatele (čím dál, tím průhlednější). Navíc můžeme nastavit různé parametry tohoto shaderu (např. rychlost a směr posouvání, barva, otočení textury).

Ochranná bariéra kolem trati je tvořena instancemi prefabu `TrackBorder`. Ten obsahuje obyčejnou krychli, která se škáluje dle potřeby. Tato krychle je pak vykreslena pomocí shaderu `TrackBorderShader`, který definuje pohyblivý náhodný vzor na povrchu a navíc v případě kolize s hráčem umožňuje zvýraznit bod nárazu. Kolize se detekují a ošetřují v komponentě `TrackBorder`.

Aby mohl hráč signalizovat, že je připraven přejít z fáze tréninku do fáze závodu, na začátek trati se umisťuje instance prefabu `StartingZone`. Ten pomocí trigger collideru dokáže detekovat, že hráč vletěl do zóny. Z jeho skriptu `StartingZone` se pak začne odpočítávat čas, po který hráč v zóně setrvává, a také se zobrazí odpočet na obrazovce (aktivací již připravených podobjektů a pravidelnou aktualizací zobrazené hodnoty).

### Bonusy

Dále se přímo na trati vyskytují bonusy různých typů (jejich výčet je uveden v sekci 2.4.1). Pro ně je vytvořen společný prefab `Bonus`, od kterého pak existuje jedna prefab varianta pro jeden typ bonusu. Každý bonus má podobu koule vykreslené pomocí shaderu `BonusShader`, který jí dodává vzhled pulzující svítivé bubliny. Každý typ bonusu pak má svou specifickou barvu.

Skript `Bonus` obsahuje základní chování. Pokud detekuje kolizi se závodníkem, pak pro něj zajistí vyvolání odpovídajícího efektu. Po sebrání bonus zmizí a znovu se objeví až po uplynutí pevně dané doby (z parametru `reactivationTime`). Jelikož se však stejný prefab používá také pro bonusy vyčarované kouzlem (více v sekci 2.7), které se po sebrání zcela zničí, skript `Bonus` umí rozlišit i takovou situaci pomocí parametru `shouldReactivate`. Bonus se objevuje a mizí pomocí tweenu z komponenty `GenericTween` (více v sekci 4.10.5).

Aby bylo možné aplikovat efekt bonusu na závodníka, musí mít bonus komponentu odvozenou od `BonusEffect`. Každý typ bonusu si tak definuje svou vlastní implementaci, kterou pak přidává ve své vlastní prefab variantě. Kromě samotného efektu v metodě `ApplyBonusEffect()` navíc implementují metodu `IsAvailable()`, která na základě aktuálního stavu hry říká, zda by se měl bonus vyskytovat v trati nebo zda může být vyčarován kouzlem.

- Prefab `SpeedBonus` obsahuje skript `SpeedBonusEffect`, který závodníkovi přidá bonusovou rychlost po určenou dobu. Zatímco je pak závodník tímto efektem ovlivněn, kolem něj se zobrazuje vizuální efekt (více v sekci 4.6.5).

- Prefab `TrajectoryBonus` má navíc skript `NavigationBonusEffect`, ve kterém se při aplikaci efektu vytvoří instance prefabu `HighlightTrajectory`. Ten se pak pomocí svého skriptu pohybuje po cestě přes několik následujících významných bodů trati a za sebou nechává viditelnou stopu. Tento bonus není dostupný ve scénách `TestingTrack` a `Tutorial`, protože v nich není skutečná trať pro zvýraznění.

- Prefab `ManaBonus` pomocí svého skriptu `ManaBonusEffect` přidává závodníkovi manu navíc (prostřednictvím jeho komponenty `SpellController`, viz sekce 4.5.2). Bonus je dostupný pouze tehdy, pokud má hráč odemčené alespoň jedno kouzlo.
- Prefab `RechargeSpellsBonus` ze skriptu `RechargeSpellsBonusEffect` závodníkovi okamžitě dobíjí všechna jeho kouzla zavoláním metody na komponentě `SpellController` (viz sekce 4.5.2). Tento bonus je také dostupný pouze tehdy, pokud má hráč odemčené alespoň jedno kouzlo.

### 4.4.3 Prostředí

V sekci 3.3.2 jsme popsali, jakým způsobem bychom reprezentovali tematické oblasti. Každý modul generátoru má své vlastní parametry, takže určité aspekty popisu oblasti jsou takto rozdělené. Jelikož však existují i některá základní data, vytvořili jsme pro reprezentaci oblasti navíc třídu `LevelRegion`. Jedná se o `ScriptableObject`, který pro každou oblast obsahuje její název, přiřazenou barvu (používá se jako barva terénu) a také obrázek, který se používá, pokud je třeba hráče upozornit na nově zpřístupněnou oblast. Všechny instance jsme umístili do složky `Assets/ScriptableObjects/Regions/`.

V levelu se pak vyskytují také různé prvky prostředí. Jakmile je umístíme, už není třeba s nimi jakkoliv interagovat. Nemají tedy žádnou vlastní reprezentaci a pracujeme čistě s jejich modely, které jsme importovali do složky `Assets/Models/Environment/`.

## 4.5 Kouzlení

Velmi zásadní herní mechanikou ve hře je kouzlení. Musíme tedy být schopni reprezentovat různá kouzla, implementovat jejich efekty a závodníkům dát možnost tato kouzla sesílat na vhodné cíle. V následujících podsekcích popíšeme, jakým způsobem jsme toho dosáhli. Vycházeli jsme přitom z rozboru ze sekce 3.4.

### 4.5.1 Správce kouzel a jejich reprezentace

V sekci 3.4.1 jsme uvedli možné způsoby reprezentace kouzel. Nakonec jsme došli k závěru, že by každé kouzlo bylo reprezentováno prefabem. Ty jsme tedy vytvořili a nachází se ve složce `Assets/Resources/Spells/`. Abychom je pak ve hře mohli používat, potřebujeme mít někde uložený jejich seznam. Jelikož do hry plánujeme v budoucnu přidávat nová kouzla (viz sekce 2.7), usnadníme si práci tím, že budeme kouzla načítat až za běhu.

O načtení se stará skript `SpellManager`, který je na objektu `Spells` ve scéně `Start`. Využívá přitom třídu `Resources` (zmíněnou v sekci 3.2) a její metodu `LoadAll<T>()`. Navíc poskytuje různé pomocné metody, např. pro získání konkrétního kouzla na základě jeho identifikátoru nebo pro ověření, zda kouzlo se zadaným identifikátorem existuje. Jedná se o implementaci singletonu (více v sekci 4.10.9), takže k němu má kdokoliv snadný přístup.

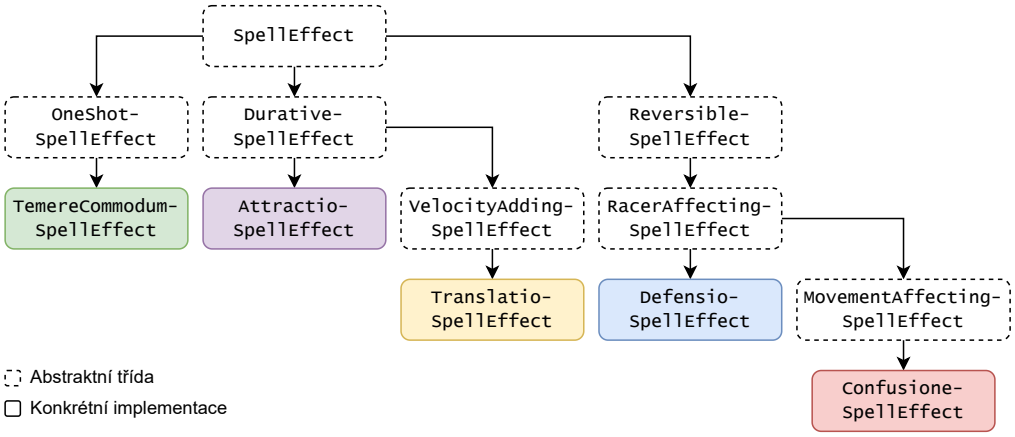
### Prefaby kouzel

Každý prefab kouzla pak má čtyři zásadní komponenty – `Spell` obsahující důležitá data, komponentu odvozenou od `SpellEffect` implementující funkční efekt kouzla, `SpellEffectController` řídící průběh seslání kouzla a nakonec komponentu `SpellEffectColorInitializer`, která se může použít pro nastavení barvy různým vizuálním efektům. Prefaby jsou ale ve skutečnosti prefab varianty. Ve složce `Assets/Prefabs/Spells/` se nachází základní prefab `SpellPrefab`, který obsahuje právě jen zmíněné komponenty. Od něj jsou pak odvozené prefab varianty `SpellSelfCastPrefab` pro kouzla sesílaná na sebe sama (obsahuje navíc vizuální efekt seslání) a `SpellWithTrajectoryPrefab` pro kouzla sesílaná směrem pryč (obsahuje už připravený základ pro vizuální efekt seslání kouzla a zasažení). Teprve od těchto prefab variant jsou pak odvozené prefaby pro konkrétní kouzla.

Kromě základních informací o kouzlu, jako je jméno, přiřazená barva, ikonka apod., skript `Spell` obsahuje také metodu `CastSpell()`, pomocí které se pak kouzlo sešle (viz sekce 4.5.2).

### Efekty kouzel

Již v sekci 3.4.4 jsme naznačili, že bychom mohli vytvořit hierarchii dědičnosti pro efekty kouzel, abychom tak abstrahovali co nejvíce režie do společného předka. Na obrázku 4.7 pak vidíme výslednou hierarchii (společně s pár konkrétními příklady efektů kouzel), kterou nyní popíšeme podrobněji.




**Obrázek 4.7** Hierarchie dědičnosti komponent pro funkční efekt kouzel s několika příklady konkrétních implementací.

Úplným základem je `SpellEffect`, který především definuje rozhraní, pomocí kterého se pak může s konkrétními implementacemi pracovat. Nejdůležitější je metoda `ApplySpellEffect()` pro vyvolání efektu, které se předávají parametry seslání kouzla v podobě `SpellCastParameters` (obsahuje např. cílový objekt či pozici).

Dále máme tři různé odvozené třídy. Jednou je `OneShotSpellEffect` reprezentující jednorázový efekt, který se provede v jediném snímku a okamžitě skončí. Příkladem takového efektu je vyčarování náhodného bonusu pomocí kouzla *Temere Commodum* (viz sekce 2.7). To je implementované v `TemereCommodumSpellEffect`.

Dalším odvozeným typem je `DurativeSpellEffect` pro efekty, které mají nějakou délku trvání. Takový efekt se pak zahájí, pravidelně se aktualizuje a po uplynutí doby se ukončí. Příkladem je `AttractioSpellEffect`, ve kterém se postupně po určitou dobu přesouvá bonus blíž k sesílajícímu závodníkovi. Speciálním typem efektu s délkou trvání je pak `VelocityAddingSpellEffect`, který se používá pro kouzla, která po dobu působení přidávají nějakou dodatečnou rychlost cílovému závodníkovi pomocí jeho komponenty `CharacterMovementController` (více v sekci 4.6.3). Příkladem je `TranslatioSpellEffect`, který sesílajícího závodníka rychle přemístí dopředu, čehož je dosaženo tím, že se po krátkou dobu přidává velká rychlost.

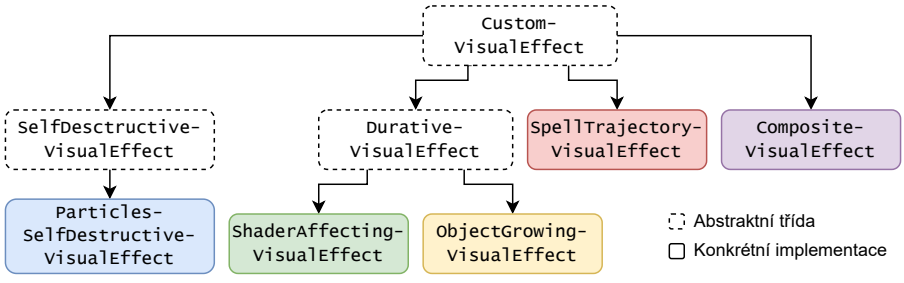
Posledním odvozeným typem je pak `ReversibleSpellEffect`, který se může použít pro efekty, které se začnou aplikovat a po uplynutí určité doby se zruší. Žádné z aktuálně implementovaných kouzel nerozšiřuje tento typ přímo. Místo toho je od něj odvozen nejprve typ `RacerAffectingSpellEffect`, který poskytuje základ pro kouzla ovlivňující závodníka po určitou dobu. Když efekt začíná působit, u cílového závodníka se poznamená (více v sekci 4.6.5) a na konci působení se pak odebere. Během působení se navíc kolem závodníka zobrazuje vizuální efekt (více informací je v následující sekci). Tento typ pak implementuje např. `DefensioSpellEffect` pro kouzlo *Defensio*, které může závodník použít sám na sebe pro udělení štítu. Kromě přidání efektu a zobrazení vizuálního efektu působení kouzla se tak kolem závodníka vyčaruje štít a na konci působení efektu zase zmizí.

Od `RacerAffectingSpellEffect` dědí i `MovementAffectingSpellEffect`, který navíc ovlivňuje pohyb cílového závodníka prostřednictvím jeho komponenty `CharacterMovementController` (více v sekci 4.6.3). Konkrétní implementací je např. `ConfusioneSpellEffect`, který zakáže pohybové akce cílového závodníka s tím, že se ale závodník ihned nezastaví, nýbrž pokračuje setrvačností dál.

Složka `Assets/Scripts/Gameplay/Spells/Effect/` obsahuje všechny konkrétní implementace efektů.

### Vizuální efekty

V sekci 3.4.2 jsme uvedli různé vizuální efekty spojené se sesláním kouzla. Navíc jsme nastínili myšlenku, že bychom pro ně mohli vytvořit společný návrh, abychom s nimi mohli pracovat jednotně, a navíc definovat hierarchii jednoduchých a znovupoužitelných chování. Na obrázku 4.8 vidíme výslednou implementaci. Jednotlivé skripty se nachází ve složce `Assets/Scripts/Gameplay/Spells/VisualEffect/`.




**Obrázek 4.8** Hierarchie dědičnosti komponent pro řízení vizuálních efektů.

Nejprve jsme zavedli abstraktní komponentu `CustomVisualEffect`, která definuje základní rozhraní s metodami pro spuštění efektu, pravidelnou aktualizaci a ukončení efektu. Od ní jsou pak odvozené dvě další abstraktní komponenty. Jednou je `SelfDestructiveVisualEffect` reprezentující efekt, který se po ukončení současně sám zničí. Druhou je pak `DurativeVisualEffect` pro efekt, který má nějakou délku trvání. Efekt se tedy spustí, poté se neustále aktualizuje a po uplynutí dané doby se sám ukončí.

Následně máme několik konkrétních implementací výše zmíněných abstraktních typů. V sekci 3.4.2 jsme také popsali, že bychom pro konkrétní vizuální efekty vytvořili prefaby s těmito komponentami pro řízení efektu. Prefaby se tedy nachází ve složce `Assets/Prefabs/Spells/`.

`ParticlesSelfDestructiveVisualEffect` je určen pro použití s komponentou `ParticleSystem`. Po spuštění tak spustí také particles a pokud mají pevně nastavenou délku trvání, tento vizuální efekt to zohlední a po uplynutí dané doby se ukončí a zničí. Pokud však particles nemají určenou délku trvání, pak je třeba vizuální efekt zastavit manuálně zavoláním odpovídající metody. Tato komponenta se využívá např. v prefabu `SpellInfluencePrefab`, který slouží jako základ pro vizuální efekty působení kouzla na nějakého závodníka.

`ShaderAffectingVisualEffect` představuje vizuální efekt s délkou trvání, během které interpoluje zadané parametry shaderu. Podporuje parametry několika základních typů. Pro každý parametr pak má počáteční a cílovou hodnotu a mezi nimi postupně přechází. Využívá se např. v prefabu `DefensioInfluenceEffect` pro efekt rozpadu štítu vyčarovaného kolem závodníka, když skončí jeho působnost.

`ObjectGrowingVisualEffect` je dalším efektem s délkou trvání. Má referenci na nějaký herní objekt (resp. jeho `Transform` komponentu) a po zadanou dobu mění jeho škálu mezi zadanými hodnotami dle zadané animační křivky. Navíc umožňuje daný objekt při spuštění efektu aktivovat a při ukončení naopak deaktivovat. Tento efekt se používá např. v prefabu `SelfCastSpellHitEffect` pro efekt seslání kouzla na sebe sama, kdy se kolem závodníka postupně zvětšuje poloprůhledná koule.

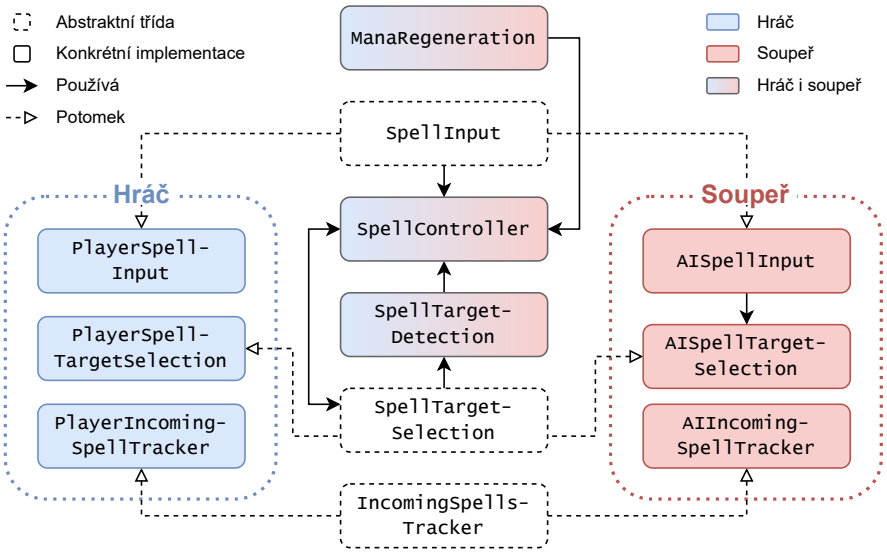
Dalším typem vizuálního efektu je `CompositeVisualEffect`, který jednoduše obsahuje seznam jiných vizuálních efektů a ovládá je všechny naráz. Po spuštění přitom pokračuje v přehrávání, dokud alespoň jeden podřízený efekt ještě běží. Pokud však všechny již skončily, skončí také tento nadřazený efekt. Používá se stejně jako předchozí např. v prefabu `SelfCastSpellHitEffect`, kdy se navíc mění parametry shaderu zobrazené koule, aby se postupně rozpadala.

Posledním vizuálním efektem je `SpellTrajectoryVisualEffect`, který řídí přímo samotné seslání kouzla do nějaké cílové pozice. Sesílané kouzlo je reprezentováno objektem, jehož pozice se mění tímto vizuálním efektem. Objekt kouzla se však nepohybuje přímo, ale po určité trajektorii, kterou udává konkrétní implementace `SpellTrajectoryComputer` (dle popisu v sekci 3.4.3). Navíc se se spuštěním efektu spustí také případný `ParticleSystem` a `TrailRenderer` (pro stopu zanechanou letícím kouzlem). Tento efekt se používá přímo v prefabech kouzel, která se sesílají určitým směrem (na jejich `SpellEffect` podobjektu).

V sekci 3.4.2 jsme také uvedli, že by se prefaby efektů dosadily do komponenty řídící průběh seslání kouzla, která by je pak přehrávala ve vhodné okamžiky. Tou komponentou je `SpellEffectController`, který se nachází v prefabech jednotlivých kouzel.

### 4.5.2 Sesílání kouzel

Jak jsme popsali již v sekci 3.4.4, veškerou logiku sesílání kouzel bychom rozdělili do několika jednotek, z nichž některé by byly společné pro hráče i soupeře a jiné by pak měly různé konkrétní implementace. Tímto návrhem jsme se řídili při implementaci a výslednou hierarchii vidíme na obrázku 4.9. Nyní stručně popíšeme mapování mezi finálními komponentami a jednotkami ze sekce 3.4.4. Všechny tyto komponenty jsou pak umístěny v prefabech reprezentujících závodníky, které více popisujeme v sekci 4.6.




**Obrázek 4.9** Zásadní komponenty pro sesílání kouzel a závislosti mezi nimi.

Komponenta `SpellController` představuje jednotku pro sesílání kouzel umožňující volit aktuální kouzlo, kontrolovat aktuální stav kouzel a many a sesílat právě zvolené kouzlo. Navíc poskytuje možnost zaregistrovat callbacky na významné události, jako je např. změna množství many či seslání kouzla. Komponenta `SpellTargetDetection` je pak reprezentací jednotky pro detekci cílů, která neustále udržuje seznamy vhodných cílů pro všechna kouzla, která má závodník vybavená. Cíle přitom detekuje pomocí trigger collideru kolem závodníka.

Abstraktní komponenta `SpellTargetSelection` reprezentuje jednotku pro výběr cílů. Zohledňuje právě zvolené kouzlo a umožňuje získat pro něj cíl (objekt, nebo pozici) z těch zaznamenaných ve `SpellTargetDetection`. Pro hráče je tato funkcionalita implementovaná v komponentě `PlayerSpellTargetSelection`, kdy se jako cílový objekt vybírá ten, který je nejblíž středu obrazovky, a jako cílová pozice se vybírá pozice v určité vzdálenosti ve směru pohledu. Navíc se však komponenta stará o správné zobrazení zaměřovače a/nebo zvýraznění cíle.

Soupeř pak má svou implementaci `AISpellTargetSelection`. Pokud je cílem právě zvoleného kouzla závodník, zvolí se náhodně, přičemž pravděpodobnost každého je určená jeho vzdáleností (čím blíže, tím větší šance ho zvolit). Pokud by už ale na zvoleného závodníka mířilo to samé kouzlo, zkusí se vybrat jiný. V případě, kdy je cílem jiný objekt, zvolí se náhodně jeden ze dvou nejbližších. Pro kouzla sesílaná určitým směrem se pak rozhoduje dle typu účinku. Pro kouzla s pozitivním účinkem se volí směr pohybu. Pro ta s negativním účinkem se zvolí

náhodně buď směr dozadu nebo drobně vychýlený směr pohybu, aby do efektu kouzla sesílající rovnou nevletěl. Pravděpodobnostní rozdělení je určené aktuálním umístěním v závodě, aby např. závodník na posledním místě zbytečně neposílal kouzlo za sebe.

Další abstraktní komponentou je `IncomingSpellsTracker`, která představuje jednotku pro správu přicházejících kouzel. Ta udržuje seznam kouzel (v podobě seznamu instancí `IncomingSpellInfo`), která na daného závodníka právě letí, a poskytuje metody pro přidání či odebrání takového kouzla. Navíc umožňuje zaregistrovat callback na přidání nového kouzla, který pak využívá implementace `PlayerIncomingSpellTracker`, aby mohla hráči zobrazit na obrazovce indikátory blížících se kouzel (více v sekci 4.11). Navíc zajistí zatřesení kamery, pokud je hráč kouzlem zasažen. Soupeř má svou vlastní implementaci `AIIncomingSpellTracker`, která je však prozatím prázdná, protože není potřeba žádná další funkcionalita navíc.

Poslední abstraktní komponenta `SpellInput` reprezentuje jednotku pro zpracování vstupu. Základní komponenta poskytuje pouze metody pro povolení a zakázání sesílání kouzel (čímž se povolí/zakážou odpovídající komponenty) a nabízí možnost registrovat na tyto události callbacky. Pro hráče se další funkcionalita přidává v komponentě `PlayerSpellInput`, která detekuje vstup a na základě něj komunikuje se `SpellController` a vyvolává jeho akce. Soupeři pak mají vlastní implementaci v komponentě `AISpellInput`, přičemž se jedná o součást implementace umělé inteligence, kterou popíšeme více v sekci 4.7.2.

Navíc jsme vytvořili komponentu `ManaRegeneration`, která se stará o průběžné doplňování many daného závodníka, jak jsme jej popsali v sekci 2.7.

#### Reprezentace kouzel v závodě

Kdykoliv pracujeme s kouzly, obvykle k nim přistupujeme skrz jejich `Spell` komponentu, která obsahuje základní informace. Během závodu však ke každému kouzlu potřebujeme poznamenat také jeho aktuální stav, tj. zda je dobité nebo kolik času zbývá do dobití. Definujeme proto třídu `SpellInRace`, která obsahuje referenci na `Spell`, ale současně přidává funkcionalitu navíc. Kromě aktuálního stavu obsahuje také metodu `CastSpell()`. Kdykoliv tedy v závodě sesíláme kouzlo pomocí komponenty `SpellController`, ve skutečnosti pracujeme se `SpellInRace`, voláme jeho metodu a teprve z ní se pak volání deleguje na komponentu `Spell`, ve které se vytvoří instance prefabu kouzla, která pak reprezentuje přímo seslané kouzlo.

Nakonec převezme řízení komponenta `SpellEffectController` a zajistí celý průběh seslání kouzla. Nejprve tedy přesouvá objekt kouzla až do cílového bodu, následně vyvolá efekt kouzla a po jeho skončení instanci kouzla zničí. Během toho v odpovídajících okamžicích spouští vizuální efekty, na které má reference.

## 4.6 Závodníci

Další významnou částí hry jsou závodníci, kteří obsahují celou řadu komponent, aby se chovali tak, jak by měli. V případě hráče je třeba reagovat na vstup a na základě něj provádět akce. Soupeři jsou pak řízení umělou inteligencí. Kromě samotného chování jsou však třeba také další součásti, jako je model závodníka

s různými možnostmi nastavení vzhledu a koště, které je možné vylepšit. V této sekci tedy popíšeme implementaci všech zmíněných částí (kromě umělé inteligence, pro kterou jsme vytvořili samostatnou sekci 4.7).

### 4.6.1 Postava

Ještě než se pustíme do detailů, nejprve se podíváme na reprezentaci závodníka obecně. Jádrem je prefab `RacerBase` (ve složce `Assets/Prefabs/Racers/`), který obsahuje společný základ pro všechny závodníky, tj. postavu, koště, collidery a obecné komponenty. Tento prefab má pak dvě prefab varianty – `Player` (přidávající kamery a komponenty specifické pro hráče) a `Opponent` (od které jsou vytvořeny ještě další varianty pro různá chování). Instance zmíněných prefabů pak používáme, kdekoliv jsou potřeba (např. ve scéně `Race` nebo `TestingTrack`).

O vizuální podobu postavy se stará prefab `Character`, jehož instance se používá v prefabu závodníka. Obsahuje modely jednotlivých částí postavy (např. boty, vlasy, oblečení) v několika různých variantách. Abychom umožnili vzhled měnit (dle popisu v sekci 2.3), na objektu je komponenta `CharacterAppearance`, která udržuje přehled o aktuálně zvolených možnostech reprezentovaných jako `CustomizationVariantData`. Navíc poskytuje metody pro randomizaci zvolených možností nebo pro aplikaci konkrétní možnosti vzhledu na postavu. Ve složce `Assets/ScriptableObjects/` se pak nachází `CharacterCustomizationOptions`, což je typ obsahující všechny dostupné varianty vzhledu pro každou část postavy. Reference na tuto instanci je nastavená ve stavu hráče (viz sekce 4.10.1), aby k ní byl snadný přístup.

Jednu upravitelnou část reprezentujeme abstraktní třídou `Customization`, která může obsahovat seznam dostupných variant a umí vracet variantu na zadaném indexu, nebo náhodnou. Pak máme konkrétní implementace dle toho, jaký aspekt vzhledu se mění. Některé možnosti popisují pouze změnu barvy (např. odstín pleti, barva vlasů), pro ně máme typ `MaterialColorCustomization` se seznamem dostupných barev. Jiné mění celou mesh společně s materiály, takové jsou popsány typem `MeshAndMaterialCustomization`, obsahujícím seznam variant jako instance `MeshAndMaterialVariant`.

Animace postavy řešíme dvěma různými způsoby. Jedním je obyčejná `Animator` komponenta, pomocí které spouštíme skeletální animace celé postavy. Navíc jsme ale implementovali vlastní skript `TextureSwapAnimationPlayer`, který se využívá pro přehrávání animací obličeje výměnou textury. Jedna animace je reprezentována typem `TextureSwapAnimation` a vytváří se přímo v Inspector okně. Obsahuje např. identifikátor animace, pod kterým se pak spouští, a seznam klíčových snímků jako `TextureSwapAnimationKeyframe` (textura s délkou trvání). Takto jsme definovali animace mrkání a úsměvu.

Každá postava má kromě svého vzhledu přiřazené také jméno. Když si hráč vytváří svou postavu, má možnost nechat si pro ni zvolit náhodné jméno. Kdykoliv se navíc ve hře objeví nějaký soupeř (např. v závodě nebo v žebříčku závodníků), měl by mít nějaké jméno. K tomu slouží třída `NamesManagement`, která načítá seznam možných jmen ze souboru `Assets/StreamingAssets/names.txt` a poskytuje metodu pro získání náhodného jména ze seznamu.

### 4.6.2 Koště

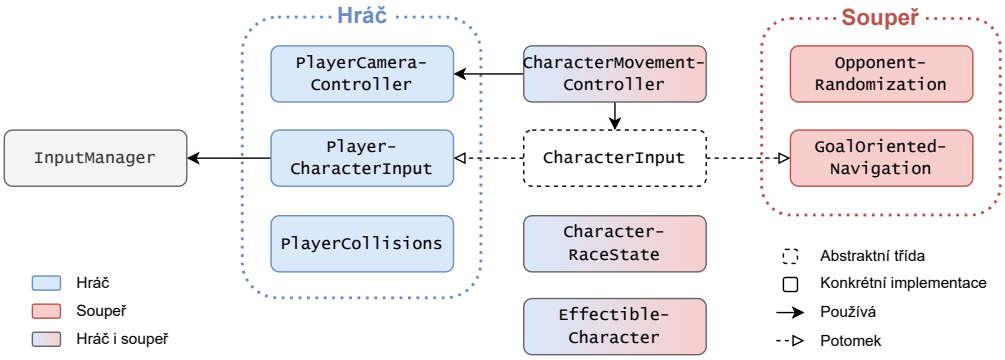
Prefaby závodníků dále obsahují instanci prefabu `Broom`, který obsahuje vše související s koštětem. Nachází se v něm model koštěte se všemi možnými variantami vzhledu (mění se v závislosti na aktuální úrovni vylepšení koštěte, viz sekce 2.5). Navíc obsahuje seznam všech možných vylepšení koštěte, kdy je jedno dostupné vylepšení reprezentováno jako jedna instance komponenty `BroomUpgrade` na podobjektu `Upgrades`. V ní jsou všechny podstatné informace (např. jméno vylepšení, pomocí kterého se pak identifikuje, cena jednotlivých úrovní, maximální úroveň), ale především také efekty jednotlivých úrovní v podobě `UnityEvent`. Přímo v Inspector okně pak nastavujeme, co se má při navýšení úrovně stát.

Jelikož ale některé efekty mají přesah mimo samotné koště (tj. nemění pouze jeho vzhled), nastavují se finálně až o úroveň výš v prefabu `RacerBase`, odkud máme přístup také k dalším komponentám. Můžeme tak měnit parametry komponenty `CharacterMovementController`, která je více popsaná v následující sekci.

Vše pak zastřešuje skript `Broom`, který si získává seznam dostupných vylepšení a udržuje aktuální stav (tj. aktuální úrovně jednotlivých vylepšení). Současně obsahuje metody pro navýšení úrovně konkrétního vylepšení nebo pro náhodnou inicializaci úrovní. Navíc se umí inicializovat z uloženého stavu hráče (viz sekce 4.10.1). Kdykoliv se pak mění úroveň nějakého vylepšení (také při inicializaci), vyvolávají se také odpovídající efekty.

### 4.6.3 Komponenty

Nyní si představíme zásadní komponenty, ze kterých je závodník složen (kromě těch zodpovědných za sesílání kouzel, které jsme popsali již v sekci 4.5.2) a které se starají především o jeho pohyb. Přehled těchto komponent vidíme na obrázku 4.10. Začneme těmi, které jsou společné pro hráče i soupeře, a poté stručně popíšeme také ty, které se pak liší.




**Obrázek 4.10** Zásadní komponenty tvořící závodníky.

Nejdůležitější komponentou je `CharacterMovementController`, která zajišťuje veškerý pohyb závodníka postupnou změnou `velocity` na komponentě `Rigidbody`. Současně na závodníka aplikuje rotaci dle toho, kterým směrem letí. Také omezuje maximální výšku, do které lze stoupat. Kromě zmíněné základní funkcionality pak poskytuje další metody, které umožňují např. nastavit bonusovou rychlost (využívá se pro zrychlující bonus a kouzlo), která se postupně tweenuje, nebo

zcela zakázat pohybové akce (s možností zvolit, zda přitom má závodník okamžitě zastavit, nebo má pokračovat setrvačností dál).

Požadovaný směr pohybu se získává jako `CharacterMovementValues` z abstraktní komponenty `CharacterInput`. Hráč i soupeř přitom poskytují své vlastní implementace. V případě hráče se jedná o komponentu `PlayerCharacterInput`, která určuje směr na základě vstupu. Soupeři pak využívají umělou inteligenci (popsanou v sekci 4.7.2) v rámci komponenty `GoalOrientedNavigation`.

Nakonec jak hráči, tak soupeři, obsahují komponentu `CharacterRaceState`, kterou jsme představili již v sekci 4.3.2. Tato komponenta zachycuje jejich aktuální pokrok v závodě.

Hráč pak obsahuje dvě speciální komponenty. Jednou je `PlayerCollisions`, která je velmi jednoduchá a pouze detekuje kolize a případně je hlásí zainteresovaným stranám pomocí zprávy (více v sekci 4.10.8). Druhou komponentou je pak `PlayerCameraController`, která zajišťuje vše týkající se kamery z pohledu hráče. Otáčí ji dle pohybu myši, přepíná do pohledu dozadu, pokud je vyžádán, a navíc poskytuje metody pro přiblížení (využívá se pro posílení dojmu zrychlení), zatřesení (používá se při zasažení kouzlem) a reset. V komponentě je dále připravená funkcionalita pro možnost přepínání mezi několika kamerami, které však aktuálně není ve hře podporováno (jak jsme vysvětlili v sekci 2.5).

Soupeř má oproti hráči jednu komponentu navíc, a to `OpponentRandomization`. Pokud totiž umisťujeme soupeře v závodě, musíme je inicializovat nějak náhodně a tato komponenta pro to poskytuje rozhraní. Spolupracuje přitom s již dříve zmíněnými komponentami. Pomocí `CharacterAppearance` se náhodně zvolí jméno a vzhled, komponenta `Broom` pak vybere náhodně úrovně vylepšení koštěte a nakonec komponenta `SpellController` vybaví soupeře nějakými kouzly (dle návrhu v sekci 2.3.2).

### 4.6.4 Ovládání

Jak již bylo řečeno, komponenta `PlayerCharacterInput` se stará o převedení vstupu hráče na požadovaný směr pohybu, který pak využívá komponenta `CharacterMovementController`. Za tím vším se však skrývá komponenta `InputManager`. Jedná se o perzistentní singleton (více v sekci 4.10.9), který je umístěný ve scéně `Start` a umožňuje nám snadno pracovat se vstupem pomocí Unity balíčku *Input System* [68]. V každém snímku projde všechny možné vstupní akce (z assetu `Controls` ve složce `Assets/Inputs/`) a poznamená si jejich aktuální hodnoty. Pak poskytuje metody pro jejich získání.

Kromě samotné detekce vstupu však umožňuje také perzistentní ukládání aktuálního mapování akcí. Díky tomu si může hráč v nastavení výchozí mapování změnit (viz sekce 4.11), přičemž se uloží a příště opět načte. Navíc poskytuje pomocné metody pro získání hezkého lokalizovaného textu pro možnost zobrazit aktuální mapování v UI.

### 4.6.5 Efekty působící na závodníka

Všichni závodníci mají také komponentu `EffectibleCharacter`. Ta se využívá pro zaznamenání, jaké efekty, ať už z bonusu či kouzla, na daného závodníka právě působí. Kdykoliv tedy má být závodník něčím ovlivněn, přidá se nový efekt

v podobě instance třídy `CharacterEffect`, která obsahuje důležité informace (např. délku trvání, zda je pozitivní) a navíc umožňuje registrovat callback na zahájení a ukončení tohoto efektu. K efektu se navíc může přibalit také vizuální efekt. Jakmile efekt skončí, odebere se ze seznamu. S využitím této komponenty je pak možné zobrazit hráči informaci o právě působících efektech v UI.

## 4.7 Umělá inteligence soupeřů

V předchozích sekcích jsme se věnovali jednotlivým komponentám, které dohromady tvoří chování závodníků. Popsali jsme sice implementaci pro hráče, ale ještě nám zbývá probrat umělou inteligenci soupeřů, která je implementována právě v těchto komponentách. Celkový návrh pak odráží přesně ten popsaný v sekci 3.5.3, kdy máme tři různé části – jednu zodpovědnou za navigaci na trati, druhou zodpovědnou za kouzlení a nakonec třetí reprezentující úroveň schopností soupeře. Nyní tedy popíšeme podrobně implementaci těchto částí.

Ve složce `Assets/Prebafs/Racers/OpponentTypes/` se pak nachází prefab `BasicGoalOrientedOpponent`, který obsahuje níže popsané komponenty a skutečně se využívá v závodech. Komponenty související s implementací umělé inteligence se pak nachází na objektu `AI`, případně jeho podobjektech.

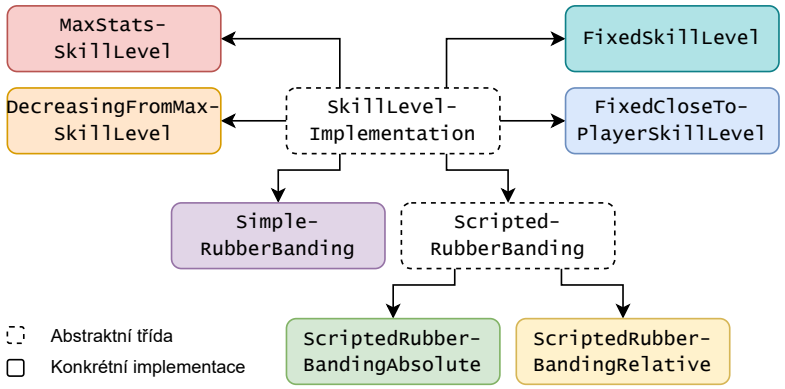
### 4.7.1 Úroveň schopností

V sekci 3.5.1 jsme popsali různé druhy chyb, které by měli soupeři dělat na základě svých přiřazených statistik reprezentujících jejich úroveň schopností. Asset `AIMistakesParameters` ve složce `Assets/ScriptableObjects/` umožňuje zapnout konkrétní druhy chyb a nastavit pro ně animační křivky, pomocí kterých se mapuje pravděpodobnost dané chyby na nějakou hodnotu parametru použitého v provedení této chyby. Např. pro chybu, kdy soupeři plně nevyužívají maximální rychlost, se mapuje pravděpodobnost na část maximální rychlosti, která se smí použít.

Současně jsme ovšem v sekci 3.5.2 řekli, že bychom měli hodnoty statistik soupeřů také přizpůsobovat aktuální situaci v závodě. Definovali jsme proto abstraktní komponentu `SkillLevelImplementation`, která určuje jak počáteční hodnoty statistik, tak aktuální hodnoty statistik v libovolném okamžiku. Pak už je na konkrétní implementaci, jak přesně se s tím vypořádá. Vytvořili jsme přitom hned několik implementací, jejichž přehled je na obrázku 4.11.

Parametry chyb a konkrétní implementace přizpůsobení hodnot statistik se pak propojují v komponentě `AISkillLevel`, která se inicializuje s konkrétní úrovní schopností soupeře relativně k hráči (z výčtového typu `SkillType`) a následně umožňuje pomocí veřejných metod získávat rovnou pravděpodobnosti chyb založených na konkrétní statistice. Ty se pak využívají přímo v částech pro navigaci a pro sesílání kouzel.

Nyní si představíme konkrétní implementace `SkillLevelImplementation`, některé jsou však čistě experimentální. V `MaxStatsSkillLevel` mají statistiky po celou dobu maximální hodnotu. V `DecreasingFromMaxSkillLevel` sice začínají na maximálních hodnotách, ale pokud je soupeř před hráčem, mohou se hodnoty snížit (tím víc, čím dál před hráčem soupeř je). V implementaci `FixedSkillLevel` se statistiky inicializují na pevně dané hodnoty odpovídající




**Obrázek 4.11** Přehled různých implementací přiřazení hodnot statistik na základě soupeřovy úrovně schopností.

konkrétní hodnotě `SkillType`. Implementace `FixedCloseToPlayerSkillLevel` pak hodnoty přiděluje pořád na základě `SkillType`, ale pomocí relativní modifikace hráčových hodnot statistik. Implementace `SimpleRubberBanding` je pak ještě větším zobecněním, kdy se hodnoty statistik navíc mění v průběhu závodu v závislosti na pozici vůči hráči (před hráčem se hodnoty snižují, za hráčem zvyšují).

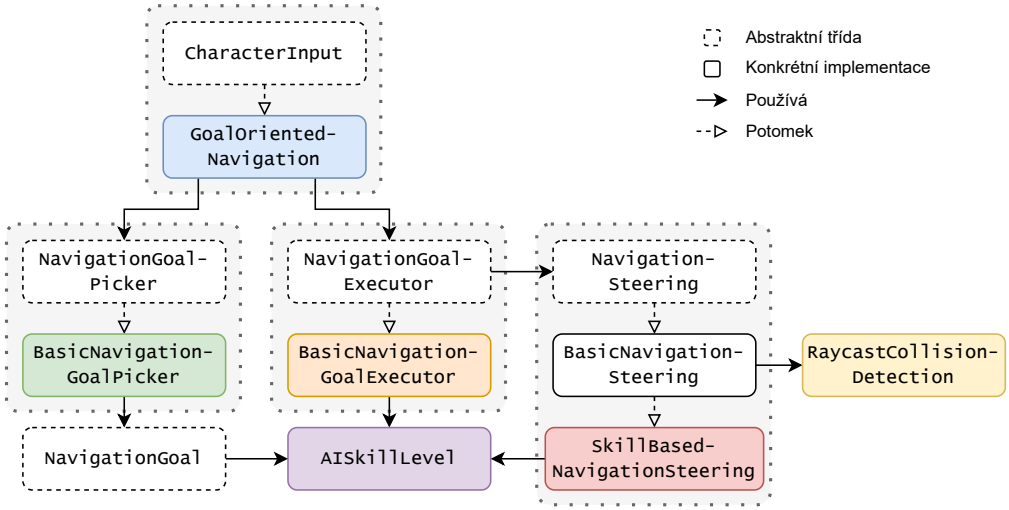
Ještě stále abstraktní `ScriptedRubberBanding` pak do rubber bandingu zavádí navíc tzv. race scripts, které jsme představili v sekci 3.5.2. Pomocí křivky se zachycuje, v jaké vzdálenosti od hráče by se měl soupeř snažit držet v průběhu celého závodu. Pak se mu upravují hodnoty statistik vůči tomuto bodu a ne přímo hráči. Navíc hodnoty statistik závisí také na tom, jak daleko v závodě už jsme. V implementaci `ScriptedRubberBandingAbsolute` jsou počáteční hodnoty statistik pevně dané, zatímco v implementaci `ScriptedRubberBandingRelative`, kterou také nakonec skutečně používáme, se statistiky inicializují relativní změnou hráčových statistik (dle konkrétní hodnoty `SkillType`).

### 4.7.2 Pohyb v trati

Již tedy máme zavedený způsob, jak reprezentovat chybovost soupeřů a jak inicializovat a dále přizpůsobovat úroveň jejich schopností. Nyní se podíváme, jak to zapadá do části zodpovědné za navigaci po trati. V sekci 3.5.3 jsme popsali, že se bude skládat ze tří jednotek – jedné pro volbu cílů, další pro vykonávání akcí ke splnění cíle a nakonec poslední pro řízení letu k danému cílovému bodu. Na obrázku 4.12 vidíme výsledný návrh souvisejících komponent a tříd. Nyní si je postupně projdeme.

Abstraktní komponenta `NavigationGoalPicker` definuje základní rozhraní pro jednotku zodpovědnou za volbu dalšího cíle. To pak implementuje komponenta `BasicNavigationGoalPicker`, která je ve skutečnosti implementací přímočaré AI popsané v sekci 3.5.4. Kdykoliv se rozhoduje o dalším cíli, zváží následující obruč (či kontrolní bod) a následující bonus. Pak vybere jako cíl ten, který je lepší (na základě vzdálenosti a směru). Současně však uvažuje možnost, že se cíl může přeskočit jako výsledek chybovosti AI.

Dále máme abstraktní komponentu `NavigationGoalExecutor`, která předsta-




**Obrázek 4.12** Přehled komponent a tříd podílejících se na umělé inteligenci pro navigaci po trati. Rámečkem jsou zvýrazněny řetězce dědičnosti. Barevné komponenty jsou ty skutečně použité ve finální verzi hry.

vuje jednotku pro vykonávání akcí vedoucích ke splnění zvoleného cíle. Na základě aktuálního cíle tak zvolí bod, do kterého je třeba se navigovat. Konkrétní implementací je pak komponenta `BasicNavigationGoalExecutor`, která navíc zohledňuje chybovost. Např. se tedy může zvolit odchýlený cílový bod nebo se může navigovat k nově zvolené cílové pozici až po chvilce a mezitím pokračovat původním směrem.

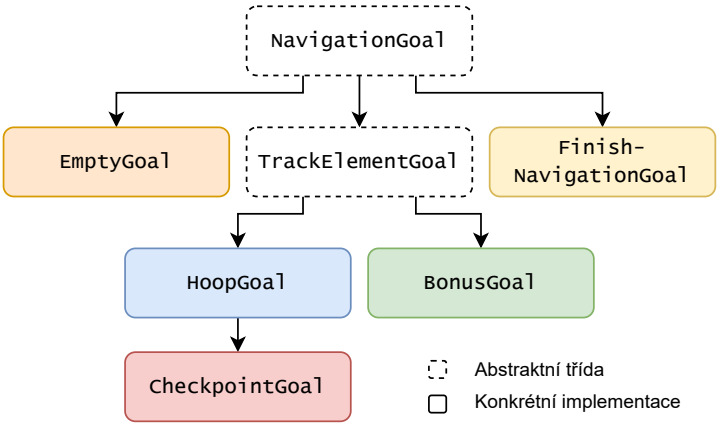
Pro samotnou navigaci do cílového bodu se pak využívá abstraktní komponenta `NavigationSteering`, resp. některá z jejích konkrétních implementací. Ta na základě cílového bodu vrací `CharacterMovementValues`, které se mohou použít pro řízení pohybu. Implementace `BasicNavigationSteering` dělá právě to, ale navíc pracuje s komponentou `RaycastCollisionDetection` pro detekci okolních objektů (pomocí raycastů) a možnost se jim vyhnout. Rozšíření `SkillBasedNavigationSteering` pak ještě do prevence kolizí zavádí chyby v závislosti na úrovni schopností. Pak je možné, že se vůbec nebude brát ohled na to, že konkrétním směrem je překážka, a poletí se dál.

Všechny výše zmíněné části pak zastřešuje `GoalOrientedNavigation`, což je komponenta implementující `CharacterInput` (tj. jednu ze základních komponent závodníka, viz sekce 4.6.3). Využívá `NavigationGoalPicker` pro výběr dalšího cíle a `NavigationGoalExecutor` pro získání požadovaného směru pohybu k aktuálnímu cíli. Navíc ale umožňuje také cíl v pravidelných intervalech přehodnocovat a současně kontroluje, zda je ještě validní či zda nebyl třeba již splněn.

### Reprezentace cíle

Pro reprezentaci jednoho cíle (tj. nějakého dílčího úkolu v rámci závodu) máme abstraktní třídu `NavigationGoal`, která obsahuje např. typ cíle (jako hodnotu výčtového typu `NavigationGoalType`), cílovou pozici a reference na komponenty potřebné pro implementaci logiky spojené s daným cílem. Tato třída navíc definuje abstraktní metody, pomocí kterých je možné určit, zda byl cíl splněn, zda je stále validní (jinak by se zvolil jiný) a jak moc je rozumný (případně by se mohl zkusit

zvážit jiný). Kromě nich má ještě abstraktní metody spojené s chybovostí (zda se má cíl přeskočit nebo zda má plnění cíle selhat). Od tohoto abstraktního typu je pak odvozených několik konkrétních. Celou hierarchii vidíme na obrázku 4.13. Příkladem může být `HoopGoal` pro průlet obručí, `BonusGoal` pro sebrání bonusu nebo `FinishNavigationGoal` pro průlet cílovou čarou.




**Obrázek 4.13** Hierarchie dědičnosti tříd reprezentujících možné cíle v umělé inteligenci řídící navigaci po trati.

### 4.7.3 Kouzlení

Implementace části pro kouzlení je v porovnání s tou pro navigaci mnohem jednodušší. V sekci 4.5.2 jsme již popsali, z jakých komponent se skládá a popsali jsme rovnou implementaci komponenty `AISpellTargetSelection`. Nyní se tedy zaměříme pouze na zbývající `AISpellInput`, která obsahuje ústřední logiku a vyvolává akce na komponentě `SpellController`. Současně implementuje přesně takové chování, které jsme popsali v sekci 3.5.4.

V delších intervalech komponenta `AISpellInput` kontroluje aktuální stav vybavených kouzel. Pokud je alespoň jedno kouzlo připraveno k použití (tj. je nabité, existuje pro něj vhodný cíl a závodník má dostatek many pro jeho seslání), náhodně se zvolí jedno ze všech takových, pomocí `SpellController` se vybere a následně sešle.

Navíc se však zohledňuje také chybovost na základě úrovně schopností. Komponenta `AISpellInput` má tedy referenci na `AISkillLevel`, ze kterého získává pravděpodobnost chyb souvisejících s *magií*. Ta se pak používá hned na několika místech. V první řadě určuje délku intervalů, ve kterých se rozhoduje o seslání kouzla (čím větší pravděpodobnost chyby, tím delší interval). Navíc se z pravděpodobnosti chyby určí pravděpodobnost, se kterou se vůbec žádné kouzlo nesešle, i když je některé připraveno. Nakonec tato pravděpodobnost také omezuje počet kouzel, ze všech vybavených, která se berou v potaz. Ostatní se zcela ignorují, jako by byly sloty prázdné.

## 4.8 Testovací trať

Testovací trať ve scéně `TestingTrack` se od obvyklého závodu (ze scény `Race`) liší v tom, že level je v ní pevně daný. Terén je tedy předem určený, stejně tak důležité objekty jsou ručně umístěné. Ve scéně je ale navíc objekt `TestingTrackManager` se stejnojmenným skriptem. Ten zajišťuje speciální podmínky, které jsme popsali již v sekci 2.8. Kdykoliv tedy hráč sešle kouzlo, okamžitě se zase dobije a navíc se kompletně naplní mana. To je zajištěné pomocí callbacků zaregistrovaných v hráčově komponentě `SpellController` na události seslání kouzla a změny množství many.

V levelu se také vyskytují všechny možné cíle kouzel, aby měl hráč možnost si je libovolně vyzkoušet. Jedním z cílů je tedy také soupeř. Místo toho obvyklého jsme ovšem použili prefab `TestingTrackOpponent`, který obsahuje také speciální chování ve skriptu `LoopPathFollowNavigation` (nahrazujícím `GoalOrientedNavigation` popsaný v sekci 4.7.2). Jako parametr dostane seznam bodů a následně soupeře řídí tak, aby postupně těmito body prolétával ve smyčce. Tím jsme zařídili, že soupeř v testovací trati neustále létá dokola.

## 4.9 Tutoriál

V sekci 2.10 jsme uvedli, že bychom do hry chtěli přidat tutoriál (pro naplnění cíle C3). Navíc jsme nastínili, z jakých částí by se skládal a kdy by se dané části spustily. Nyní si popíšeme, jak přesně je řízení tutoriálu zajištěno a jaké další pomocné nástroje se při jeho průběhu využívají.

### 4.9.1 Fáze tutoriálu a jejich řízení

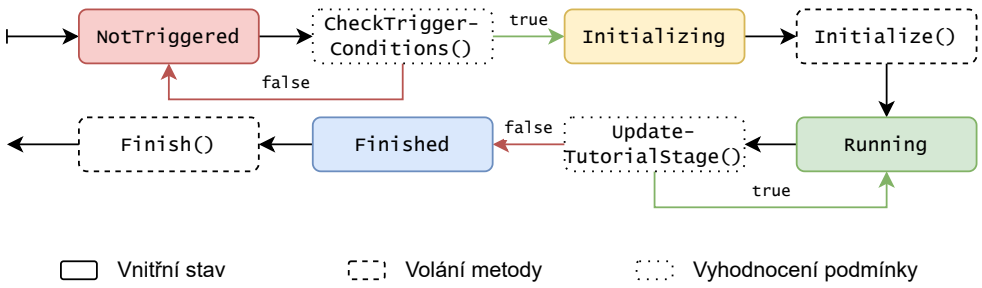
Základem všeho je prefab `Tutorial`, jehož instance se nachází ve scéně `Start`. Obsahuje skript `Tutorial`, který rozšiřuje `MonoBehaviourSingleton<Tutorial>` pro zajištění funkcionality singletonu perzistentního mezi scénami (viz sekce 4.10.9). Především pak ale spravuje veškerý běh tutoriálu. Poznamenává si aktuální fázi tutoriálu (jako hodnotu výčtového typu `TutorialStage`) a zajišťuje, že se mezi jednotlivými fázemi správně přechází. Navíc obsahuje reference na pomocné objekty a poskytuje metody např. pro uložení či načtení stavu tutoriálu (více v sekci 4.10.1) nebo pro přeskočení aktuální fáze tutoriálu.

#### Reprezentace fáze tutoriálu

Jedna fáze tutoriálu je reprezentována abstraktní třídou `TutorialStageBase`, jejíž konkrétní implementace pak představují již konkrétní fáze ve hře. Tato abstraktní třída poskytuje společný základ, jako je např. rozlišení vnitřního stavu fáze (`TutorialStageState`) a aktualizace tohoto stavu nebo možnost pozastavit hru. Navíc však definuje celou řadu abstraktních metod, které pak jednotlivé fáze implementují.

Na obrázku 4.14 vidíme zakreslení přechodů mezi jednotlivými stavy fáze. Každá fáze začíná ve stavu `NotTriggered`, ve kterém se opakovaně ověřují podmínky jejího spuštění metodou `CheckTriggerConditions()`. Teprve když jsou

splněny, fáze přejde do stavu `Initializing`, kdy se skutečně zahájí prostřednictvím metody `Initialize()`. Po jejím skončení se přesune do stavu `Running`, ve kterém se neustále provádí aktualizace pomocí `UpdateTutorialStage()`. Tím se zařídí samotný průběh scénáře tutoriálu, až dokud se daná fáze nedokončí. Pak přejde do stavu `Finished` a proběhne finalizace v metodě `Finish()`. Jakmile jedna fáze skončí, komponenta `Tutorial` může přejít do další, pokud je k dispozici.




**Obrázek 4.14** Vnitřní stavy jedné fáze tutoriálu s přechody mezi nimi.

Jednotlivé fáze se pak skládají z několika po sobě jdoucích kroků. V jednom kroku mohou např. zobrazit hráči nějakou informaci nebo po něm chtít provést nějakou akci. V takovém případě mohou poskytnout vlastní implementaci abstraktní třídy `TutorialStepProgressTracker`, která definuje metody pro sledování postupu jednoho konkrétního kroku. Příkladem implementace může být třída `BonusProgress<T>`, která reprezentuje krok, ve kterém se čeká, dokud hráč nesebere konkrétní počet bonusů typu `T`. `TutorialStageBase` pak poskytuje metodu, které se předá taková reprezentace kroku a následně se čeká, dokud se nesplní. Konkrétní fáze ji tak mohou využít ve svých scénářích.

#### Konkrétní fáze

Ve hře je implementovaných celkem 7 fází tutoriálu, jejichž implementace se nachází ve složce `Assets/Scripts/Tutorial/Scenarios/`. Svým obsahem pak odpovídají přesně popisu v sekci 2.10.

Úvodní tutoriál (ze třídy `IntroductionTutorial`) a tutoriál sesílání kouzel (ve třídě `CastSpellsTutorial`) se odehrávají ve speciální scéně `Tutorial`. V ní jsou již připraveny všechny potřebné objekty. Jedním z nich je také instance prefabu `TutorialTriggerZone` (se stejnojmenným skriptem), který představuje speciální zónu, do které musí hráč v určitých okamžicích doletět, aby mohl pokračovat do dalšího kroku. Tak zajistíme, že má hráč dostatek času vyzkoušet si nové koncepty.

Tutoriál prvního závodu (`FirstRaceTutorial`) se pak odehrává přímo ve scéně `Race`. Zbylé fáze hráči představují různé obrazovky a prvky uživatelského rozhraní v přehledu hráče a v obchodě. Probíhají tak v odpovídající scéně `PlayerOverview`.

### 4.9.2 Kamera

Abychom se mohli během tutoriálu zaměřit na nějaký objekt ve scéně, který chceme hráči představit, vytvořili jsme prefab `TutorialCamera`. Ten ve skutečnosti využívá instanci `CutsceneCamera` prefabu (viz sekce 4.3.1), ke které ze skriptu

`TutorialCamera` přidává další užitečné metody (např. zaměření kamery na konkrétní objekt, deaktivace kamery). Instance tohoto prefabu se pak používají, kdekoliv je potřeba, tedy ve scénách `Race` (pro možnost ukázat startovní zónu) a `Tutorial` (pro možnost ukázat jednotlivé prvky trati).

### 4.9.3 Pomocné nástroje tutoriálu

V průběhu tutoriálu obvykle potřebujeme různé pomocné funkce, např. přesunout hráče nebo deaktivovat jeho akce. Skript `TutorialBasicManager` tedy přesně takové funkce poskytuje. Jedná se o jednoduchou implementaci singletonu (dosazením instance do statické položky během inicializace), díky čemuž je snadno dostupný. Skript je pak navěšený na objektu `TutorialSceneManager` ve scénách `Tutorial` a `Race`.

Dále ovšem potřebujeme také získat reference na důležité objekty ve scéně, např. abychom je mohli hráči ukázat nebo je mohli vhodně (de)aktivovat. Takovou funkci plní několik skriptů, z nichž každý je zaměřený na jednu konkrétní scénu. V každém se do datových položek nastaví důležité objekty ze scény, pro které pak navíc poskytuje užitečné metody (např. (de)aktivace, reset). Jedná se o skript `TutorialPlayerOverviewReferences` pro scénu `PlayerOverview` a skript `TutorialSceneManager` pro scénu `Tutorial`. Tyto skripty se pak stejně jako předchozí nachází přímo na objektu v odpovídající scéně.

Kromě toho však tutoriál potřebuje také pracovat s uživatelským rozhraním, aby mohl na obrazovce zobrazit nějaký panel s textem nebo mohl zvýraznit konkrétní prvek UI, pokud jej hráči vysvětluje. Prefab `Tutorial` tak navíc obsahuje různé podobjekty zodpovědné právě za tuto funkcionalitu, se kterými pak pracuje ze svého `Tutorial` skriptu. Blíže je však popíšeme v sekci 4.11, která se věnuje právě popisu uživatelského rozhraní.

## 4.10 Pomocné systémy a nástroje

V předchozích sekcích jsme popsali implementaci klíčových částí hry. Kromě nich se však využívá také celá řada různých pomocných systémů či nástrojů. Popisu jejich implementace se tedy budeme věnovat v této sekci. Většinou však nebudeme zacházet do přílišných detailů a spolehneme se na dostatečnou dokumentaci odpovídajících zdrojových kódů.

### 4.10.1 Stav hry

Už na několika různých místech jsme zmínili stav hry nebo stav hráče. Nyní si tedy řekneme, co přesně obsahuje a jak se ukládá.

Stavem hry rozumíme cokoliv, co je třeba někde uchovávat mezi jednotlivými scénami a někdy dokonce mezi sezeními. Takové informace, které jsou spojené přímo s pokrokem ve hře, sdružujeme v typu `PlayerState`, který je implementací singletonu (více v sekci 4.10.9). Nachází se na stejnojmenném objektu ve scéně `Start`. Uchovává pak konkrétně následující informace:

- Jméno a vzhled hráčovy postavy, množství mincí, zda již hru dohrál do konce (tj. umístil se první v žebříčku závodníků).

- Aktuální hodnoty statistik hráče (a také ty předchozí, aby je bylo možné zakreslit do grafu, viz sekce 4.11).
- Maximální možné množství many v závodě, kouzla dosazená do slotů, zakoupená kouzla a doposud seslaná kouzla (ta je třeba znát pro výpočet statistiky *magie*, viz sekce 2.6).
- Zakoupené úrovně vylepšení koštěte a maximální výška, do které může vystoupat.
- Doposud navštívené tematické oblasti (jsou potřeba pro rozhodnutí, zda zobrazit informaci o nové oblasti, a také pro generování levelu, kdy se přednostně použijí nenavštívené) a zpřístupněné oblasti.
- Jména závodníků, kteří se již někdy objevili v žebříčku (aby se při příštím zobrazení mohla použít ta samá jména na stejných příčkách).

`PlayerState` má navíc referenci na všechny možné varianty vzhledu postavy (viz sekce 4.6.1). Dále také poskytuje metody pro změnu stavu a navíc umožňuje zaregistrovat callbacky na některé zajímavé události (např. změna počtu mincí nebo vybavených kouzel).

Uzavřenější systémy, jako např. ocenění (v sekci 4.10.2) či tutoriál (v sekci 4.9), si pak uchovávají stav přímo u sebe.

#### Save systém

Některé části stavu hry se mohou pokaždé znovu přepočítat (např. zpřístupněné tematické oblasti), ale jiné je třeba perzistentně ukládat, aby se mohly při příštím spuštění hry zase načíst (dle požadavku **P3**). Kdykoliv se tedy stav změní, vyvolá se jeho uložení. V sekci 3.2 jsme uvedli, že bychom implementovali vlastní systém pro ukládání dat a stav hry bychom rozdělili do několika různých souborů. To jsme tedy také udělali.

Hlavní funkcionalitu poskytuje třída `SaveSystem` prostřednictvím svých statických metod. V parametru metod se předají data k uložení a systém je pak uloží do správného souboru. Nebo se naopak požadovaná data načtou ze správného souboru a vrátí z metody jako návratová hodnota. Veškerá ukládaná data jsou pak rozdělena do několika různých typů (např. `RegionsSaveData` či `SpellsSaveData`), z nichž každý reprezentuje data co nejjednodušším způsobem. Pak se celé instance takových typů serializují do souborů. Veškerá implementace související s načítáním a ukládáním stavu se nachází ve složce `Assets/Scripts/SaveSystem/`.

Ostatní herní logika pak s tímto systémem spolupracuje a ve svých metodách pro načtení nebo uložení stavu ho používá.

### 4.10.2 Ocenění

Dalším z pomocných systémů, které jsme implementovali, jsou ocenění, která může hráč získávat za různé menší cíle v rámci hry. Základní myšlenku jsme popsali v sekci 2.10. Z hlediska implementace jsou pak klíčové dva typy – `Achievement` a `AchievementManager`. Získaná ocenění se zobrazují v přehledu hráče (více v sekci 4.11).

Jedno možné ocenění dosažitelné ve hře je reprezentováno pomocí typu `Achievement`. Jedná se o `ScriptableObject` obsahující základní informace, jako je ikonka ocenění a typ (z výčtového typu `AchievementType`). Každé ocenění navíc může mít několik různých úrovní podle nějaké hodnoty, které je třeba dosáhnout. V typu je tedy také seznam takových hodnot. Pro všechna možná ocenění jsou vytvořeny instance ve složce `Assets/Resources/Achievements/`. Kompletní seznam je také uveden v sekci 6.2.

Komponenta `AchievementManager` se pak stará o přehled všech dostupných ocenění a hráčův pokrok v jejich získávání. Jedná se o perzistentní singleton (viz sekce 4.10.9) umístěný ve scéně `Start` na objektu `Achievements`. Na začátku hry načte všechna dostupná ocenění a uložený pokrok a při každé změně pokrok zase naopak ukládá.

Pokrok v rámci jednoho ocenění se reprezentuje typem `AchievementProgress`, kde se mimo jiné poznamenává, zda je ocenění nově získáno. Následně jsme definovali abstraktní třídu `AchievementData`, která má na starosti měření a ukládání dat potřebných pro určení úrovně ocenění. Různá ocenění jsme pak rozdělili do několika skupin, přičemž pro každou takovou skupinu existuje jedna konkrétní implementace (např. `SpellsData` pro data ocenění týkajících se kouzlení nebo `CoinsData` pro data ocenění týkajících se získaných mincí).

### 4.10.3 Lokalizace

Jedním z našich požadavků na hru byla také lokalizace veškerého obsahu (požadavek **P4**). Té jsme se věnovali již v sekci 3.9.3, kdy jsme mimo jiné zvolili způsob reprezentace frází v různých jazycích (Google tabulka převedená na JSON soubor) a naznačili návrh řešení. Všechny popsané části jsme tak skutečně implementovali a nyní je popíšeme.

V centru dění je `LocalizationManager`, což je singleton přetrvávající mezi scénami (viz sekce 4.10.9). Vytvořili jsme prefab s tímto skriptem a jeho instanci pak umístili do scény `Start`. Jedná se přitom o správce, který po spuštění hry načte všechny fráze ze souboru `translations.json` ve složce `Assets/Resources/`. Pak poskytuje metody pro změnu aktuálního jazyka či pro získání lokalizované fráze uložené pod konkrétním klíčem. Navíc umožňuje zaregistrovat callback na změnu jazyka.

Pro samotné použití lokalizovaného textu v uživatelském rozhraní jsme vytvořili speciální komponentu `LocalizedTextMeshProUI`. Tato komponenta musí být na objektu, na kterém je také `TextMeshProUGUI`, protože právě tu využívá pro zobrazení textu. Následně v komponentě nastavíme klíč fráze a kdykoliv je třeba text zobrazit, podle klíče se získá fráze ve správném jazyce a nastaví do `TextMeshProUGUI`.

### 4.10.4 Tooltipy

V sekci 2.10 jsme zmínili, že by bylo vhodné na různých místech využít tooltipy pro možnost zobrazit hráči více informací. Následně jsme v sekci 3.9.4 popsali blíže požadované chování a také možnou inspiraci pro implementaci. V této sekci popíšeme jednotlivé části výsledné implementace.

Klíčovou roli hraje komponenta `TooltipController`, která poskytuje metody

pro nastavení obsahu toooltipu a jeho zobrazení či skrytí. Jedná se navíc o implementaci singletonu, který přetrvává mezi scénami. Vytvořili jsme tedy prefab `Tooltip` obsahující tento skript a do scény `Start` umístili jeho instanci, která se pak využívá, kdykoliv je třeba zobrazit tooltip.

V prefabu `Tooltip` jsou navíc připravené potřebné prvky UI. Panel s textem jsme rozdělili do tří sekcí, z nichž každá obsahuje ještě několik částí, takže můžeme zobrazit různě strukturovaný text. Komponenta `TooltipPanel` se pak stará o správné nastavení obsahu a současně řídí umístění tooltipu a jeho velikost.

Kdykoliv pak chceme zobrazit tooltip, na související objekt přidáme komponentu odvozenou od `TooltipBase`. V ní nastavíme obsah tooltipu (podporuje se také lokalizace) a další parametry (např. po jak dlouhé době se má tooltip zobrazit). Kdykoliv pak na daný objekt najede kurzor myši, pomocí `TooltipController` (a následně `TooltipPanel`) se nastaví obsah a tooltip se zobrazí. Poskytujeme přitom dvě konkrétní implementace – `SimpleTooltip` pro zobrazení jediné sekce textu a `Tooltip` pro zobrazení textu rozděleného do několika sekcí (pomocí `TooltipSectionsTexts`).

Abychom mohli použít různá formátování pro různé části textu, definovali jsme typ `TooltipStyle`. Jedná se o `ScriptableObject`, takže se jeho instance ukládají jako assety v projektu. Jedna instance pak reprezentuje jeden možný styl popisující např. barvu pozadí, maximální šířku panelu či velikost fontu. Navíc však umožňuje definovat různé značky, kterými můžeme označit konkrétní části textu pro změnu jejich formátování (využívají se také v lokalizovaných frázích). `TooltipController` má referenci na výchozí styl a poskytuje pomocné metody pro převod vlastních značek na formátovací značky TextMesh Pro. Samotný `TooltipPanel` pak umí změnit svůj vzhled dle zadaného stylu.

### 4.10.5 Tweening

V sekci 3.9.6 jsme poměrně podrobně popsali, jak bychom mohli implementovat vlastní řešení tweeningu, které bychom pak mohli použít jak na 3D objekty, tak na prvky uživatelského rozhraní. V této sekci už se tedy zaměříme spíše jen na technické provedení. Popsanou funkcionalitu jsme implementovali v komponentě `GenericTween`, kterou pak můžeme přidat objektu, jehož vlastnosti chceme ovlivňovat.

Komponenta `GenericTween` umožňuje zvolit, které vlastnosti se mají tweenovat a s jakými parametry. Když se pak spustí tween pomocí metody `DoTween()`, zařídí průběžnou aktualizaci hodnot zvolených vlastností. Navíc umí ošetřovat různé pokročilé možnosti, jako je cyklení tweenu, pozdržení začátku nebo vyvolání nějaké akce po dokončení.

Pro reprezentaci tweenu jedné konkrétní vlastnosti pak máme abstraktní třídu `TweenProperty<TValue>`, kde `TValue` je typ vlastnosti (např. `float` nebo `Vector3`). V ní se pak implementuje metoda pro změnu hodnoty odpovídající vlastnosti na základě normalizovaného času v rámci tweenu. Konkrétní implementace jsou např. `TweenPropertyPosition` pro změnu pozice objektu skrz jeho `Transform` komponentu nebo `TweenPropertyAlpha` pro změnu průhlednosti, přičemž umí pracovat s komponentami `CanvasGroup`, `SpriteRenderer` nebo `Image` (pokusí se na cílovém objektu najít některou z nich).

Nakonec máme ještě abstraktní třídu `TweenPropertyValues<TValue>`, která

reprezentuje jen samotný tween na typu `TValue`. Obsahuje parametry pro počáteční a koncovou hodnotu a pro křivku, podle které se hodnota mění v čase (v typu `TweenVectorCurves`). Pokud má typ více složek, máme navíc možnost zadat křivku pro každou složku zvlášť. Konkrétní implementace pak definují metodu pro získání hodnoty v konkrétním normalizovaném čase, která se využívá z jednotlivých implementací `TweenProperty<TValue>`, aby bylo možné dosadit správnou hodnotu do vlastnosti.

### 4.10.6 Audio

V požadavku **P5** si klademe za cíl přidat do hry také audio zahrnující hudbu na pozadí, zvuky prostředí v levelu a různorodé zvukové efekty ve hře i v UI. V sekci 2.9 jsme pak popsali konkrétněji, o jaké zvuky a efekty by se jednalo. A nakonec v sekci 3.8 jsme na to dále navázali a popsali, jak bychom přistoupili k řešení konkrétních problémů. Nyní tedy popíšeme finální implementaci vystavenou na informacích ve zmíněných sekcích.

#### Audio systém

Centrální částí je komponenta `AudioManager`, která je implementací singletonu perzistentního mezi scénami. Používá se pak v prefabu `AudioManager`, jehož instance je ve scéně `Start`. Tato komponenta nabízí ostatním užitečné metody, jako je např. změna hlasitosti celé skupiny zvuků nebo přehrání jednorázového zvukového efektu (může být také jako 3D zvuk s pozicí ve světě).

Jelikož samotný objekt přetrvává mezi scénami, stará se také o přehrávání hudby na pozadí (k tomu využívá FMOD komponentu `FMODStudioEventEmitter`). Skript `AudioManager` pak nastavuje globální parametr `Scene`, definovaný v našem FMOD projektu (ve složce `src/Audio/` elektronické přílohy), který mění hlasitost hudby dle aktuální scény (v herních scénách je potlačená).

Abychom mohli přehrát jednorázový zvukový efekt, musíme metodě předat jako parametr referenci na odpovídající FMOD událost. `AudioManager` tak obsahuje instanci `FMODEvents`, což je ve skutečnosti `ScriptableObject` ze složky `Assets/ScriptableObjects/`. V jeho datových položkách jsou pak nastavené reference na události, takže je můžeme z kódu snadno získat.

#### Dlouhotrvající zvuky

Jednoduché zvukové efekty ve hře obvykle řešíme pomocí `AudioManageru`, nebo rovnou FMOD komponent. Některé zvuky jsou ale složitější – znějí po delší dobu a musí se řídit nastavováním parametrů. Pro takové jsme vytvořili vlastní komponenty, které pak celé toto řízení zapouzdřují, aby neznepřehledňovalo kód s herní logikou.

Příkladem mohou být zvuky prostředí, kdykoliv se hráč nachází v nějakém levelu. Odpovídající FMOD událost má parametr pro aktuální oblast a dále pro výšku nad zemí, která může ovlivňovat hlasitost určitých zvuků. Logika je pak implementována v komponentě `RegionAmbienceAudio` v prefabu `Ambience`, který je umístěn ve scénách `Race` a `QuickRace`. Jeho zjednodušená podoba se používá ve scénách `Tutorial` a `TestingTrack`, kdy je komponenta nahrazena

komponentou `SimplifiedRegionAmbience`, jelikož level v těchto scénách je pevně daný.

Druhým příkladem je pak zvuk během letu, který se přizpůsobuje aktuální rychlosti. Pro něj existuje skript `BroomFlyingAudio` spolupracující s komponentou `CharacterMovementController` (viz sekce 4.6.3) pro získání rychlosti. Nachází se pak přímo na objektu závodníka (již v prefabu `RacerBase`).

#### Efekty

V určitých situacích bychom měli na veškeré audio aplikovat nějaké efekty. Jednou z nich je pozastavená hra, kdy se hudba na pozadí zvýrazní a okolní zvuky potlačí. V FMOD projektu jsme tak vytvořili snapshot, který se pak spouští či ukončuje pomocí metod centrálního audio systému (`AudioManager`). Druhou situací je, pokud se hráč vyskytuje pod vodní hladinou. Tato situace se řeší opět pomocí snapshotu, ale tentokrát v objektu `Ambience`, který se stará o celkové zvuky prostředí v závislosti na aktuální oblasti.

#### Pomocné komponenty

Pro přidání zvuků prvkům uživatelského rozhraní jsme pak museli vytvořit dvě vlastní pomocné komponenty, abychom mohli ošetřit složitější případy, které se nedají snadno řešit jen pomocí základních FMOD komponent.

Skript `ButtonPointerDownAudio` tak umožňuje přehrát zvuk při stisknutí tlačítka, přičemž rozlišuje, zda je tlačítko interaktivní, či nikoliv. Pak může mít pro každou ze situací přiřazenou jinou audio událost.

A nakonec skript `UIOnValueChangedAudio` přidává zvuk prvkům, jejichž hodnoty se mění. Může se jednat např. o `Slider` nebo `Scrollbar`. Kdyby zvukový efekt zazněl při každé změně, mohl by se přehrávat příliš často a znít nelibozvučně. Místo toho tedy definujeme interval, ve kterém se zvuk přehrává, pokud se během toho hodnota mění.

### 4.10.7 Herní analytiky

Pro účely sběru dat během experimentů prováděných v rámci cíle C4 jsme potřebovali implementovat herní analytiky, tj. nějaký systém, pomocí kterého bychom sbírali informace o důležitých událostech ve hře. Vytvořili jsme proto `Analytics` skript, který se pak nachází na stejnojmenném objektu ve scéně `Start`. Jedná se o singleton, který přetrvává mezi scénami a je tak neustále k dispozici.

Zmíněný skript se stará o celkovou správu analytik – poskytuje metodu `LogEvent()` pro nahlášení nějaké události a veškeré tyto události ukládá do výstupního souboru. Navíc zajišťuje, že velikost souboru nikdy nepřekročí nějaký limit (pro případ, že by hráči hráli hru po delší dobu i mimo experiment popsaný v kapitole 5). Metoda `LogEvent()` se může použít z libovolného jiného skriptu, kde dochází k zajímavým událostem, ale některé události obstarává `Analytics` skript sám. Aby je mohl detekovat, registruje se k odběru odpovídajících zpráv (viz sekce 4.10.8) nebo registruje callbacky u jiných skriptů.

Abychom umožnili snadné strojové zpracování výstupního souboru, zvolili jsme konkrétní formát výpisu události. Každá událost má přiřazenou nějakou kategorii (z výčtového typu `AnalyticsCategory`). Při výpisu události se pak

vypíše nejprve aktuální datum a čas, následně aktuální scéna a poté kategorie události následovaná nějakou hláškou specifickou pro danou událost. Jednotlivé části se pak nachází na stejném řádku oddělené symbolem `|`.

Skript `Analytics` navíc umožňuje výpis analytik úplně vypnout. Pro usnadnění experimentů pak také poskytuje možnost po ukončení hry překopírovat výstupní soubor do konkrétní složky na ploše. Když pak mají účastníci experimentů tento soubor odevzdat společně s dotazníkem, nemusí ho zdlouhavě hledat.

Detekovaných událostí je celá řada a jejich kompletní výčet by tak byl příliš dlouhý. Uvedeme tedy jen pár příkladů. Vypisujeme např. každou změnu scény, průlet obručí, změnu hodnot statistik či umístění v žebříčku, seslání kouzla, posunutí do další fáze tutoriálu, změnu mapování kláves a mnohé další. Díky tomu máme jasnou představu o tom, jak hráč hrou procházel a co v ní dělal.

### 4.10.8 Messaging

V sekci 3.1.2 jsme popsali různé možnosti komunikace mezi objekty a závěrem jsme uvedli potřebu implementovat si svůj vlastní systém pro zasílání zpráv. Objekty by se v něm mohly zaregistrovat k odběru konkrétní zprávy, takže když by někdo takovou zprávu poslal, byly by na to upozorněny.

Popsaný mechanismus zasílání zpráv jsme tedy implementovali ve statické třídě `Messaging`. Ta poskytuje statické metody pro zaregistrování se ke konkrétní zprávě (identifikátorem zprávy je `string`), odregistrování se nebo pro zaslání konkrétní zprávy. Při registraci ke zprávě se pak předává delegát, který bude vyvolán jako forma notifikace na zprávu. Jedná se tak vlastně o zaregistrování callbacku přes prostředníka, takže nemusíme přesně znát cílový objekt.

Kdykoliv se někdo přihlásí k odběru zprávy, systém si ho poznamená. Pokud pak někdo posílá zprávu, systém projde seznam všech delegátů zaregistrovaných k této zprávě a postupně je vyvolá. Ke zprávě je navíc možné přiložit nějaký parametr. Prozatím však podporujeme pouze zprávy bez parametrů a zprávy s jediným parametrem jednoho z několika základních typů (`string`, `int`, `float`, `bool` a `GameObject`).

Zasílaných zpráv je celá řada. Obvykle jsou to takové, u kterých se neočekává konkrétní příjemce nebo které jsou užitečné pro analytiky (viz sekce 4.10.7) či ocenění (viz sekce 4.10.2). Jedná se např. o zahájení hry, náraz do překážky nebo zpřístupnění nové oblasti.

### 4.10.9 Singleton

Již několikrát jsme v této kapitole zmínili implementaci singletonu. V sekci 3.9.1 jsme navíc popsali, jaká chování bychom chtěli podporovat, aby byl singleton co nejuniverzálnější a dal se použít v různých situacích. Nyní tedy popíšeme výslednou implementaci.

Jedná se o třídu `MonoBehaviourSingleton<T>`, kterou pak konkrétní příklady využití rozšiřují. Typový parametr `T` označuje typ instance, tedy ten, který singleton nakonec implementuje. Ve třídě se nachází statická vlastnost pro získání instance a zajištění její inicializace. Dále je pomocí `SingletonOptions` zvolená kombinace požadovaných chování. V různých metodách se pak zajišťují naplnění těchto chování. Pokud chceme změnit výchozí nastavení chování, je třeba to provést

ve statickém konstruktoru, protože tak změna zajisté proběhne ještě před prvním přístupem k instanci a líná inicializace může fungovat, jak má.

Od typu rozšiřujícího tento singleton navíc požadujeme, aby implementoval rozhraní `ISingleton`. Díky tomu máme záruku, že na typu budou existovat inicializační metody (nahrazující obvyklé `Awake()` a `Start()`), které pak můžeme ve vhodný okamžik použít pro inicializaci instance. Pokud navíc musíme tyto metody explicitně implementovat, sníží se tím riziko toho, že bychom omylem přepsali zděděné metody `Awake()` a `Start()`, které však musí zůstat beze změny pro zajištění správného fungování.

Konkrétními příklady implementace singletonu jsou např. `SceneLoader` (více v sekci 4.1.1), `PlayerState` (viz sekce 4.10.1) nebo `AudioManager` (popsaný v sekci 4.10.6). Obvykle se pak jedná o objekty perzistentní mezi scénami, umístěné ve scéně `Start`. Typicky jsme z nich navíc vytvořili prefaby, aby bylo možné je během testování snadno umisťovat také do jiných scén, pokud chceme dočasně spouštět hru odtamtud a přitom zajistit existenci těchto objektů.

### 4.10.10 Renderování skyboxu

Posledním příkladem pomocného nástroje je možnost renderovat scénu do *cubemap*, která se pak může použít jako skybox v ostatních scénách. Využili jsme to k tomu, abychom mohli už přímo do skyboxu vykreslit mraky a nebe tak nebylo příliš prázdné, i kdybychom skutečných fyzických modelů neumisťovali tolik. Slouží k tomu skripty `PlaceClouds` a `RenderToCubemap` umístěné na objektu `SceneSetup` v pomocné scéně `SkyboxCubemap`.

`PlaceClouds` se stará o náhodné rozmístění mraků na ploše horní polokoule se středem v počátku soustavy souřadnic. Můžeme přitom nastavovat různé parametry, např. poloměr koule, celkový počet umisťovaných mraků nebo rozsah velikostí (mraky se také náhodně škálují a otáčí). `RenderToCubemap` pak zajišťuje samotné renderování do cubemap pomocí kamery umístěné ve scéně. Jako parametr můžeme nastavit název výsledného assetu ve složce `Assets/Sprites/`, do kterého se cubemap uloží.

Jelikož však scéna není nijak přístupná přímo ve hře, je možné toto řešení použít pouze při spuštění scény v editoru. Metody obou skriptů jsou navíc pomocí atributu `[ContextMenu("name")]` přidány do jejich kontextového menu v Inspectoru, takže můžeme snadno po spuštění vše měnit a přegenerovávat, dokud nejsme s výsledkem spokojení.

## 4.11 Uživatelské rozhraní

V předchozích sekcích této kapitoly jsme popisovali implementaci čistě z hlediska herní logiky. V této sekci k tomu tedy připojíme také popis uživatelského rozhraní, přičemž budeme postupovat zhruba po jednotlivých scénách. Nebudeme však zacházet příliš do detailů, místo toho se zaměříme spíše na prvky se zajímavějším chováním. Rozhodně tedy nebudeme uvádět konkrétní využití běžných Unity komponent pro tvorbu uživatelského rozhraní, jako je např. `Button`, `Image` či `HorizontalLayoutGroup`. Naopak ale zmíníme použití prefabů nebo vlastních skriptů.

Pro často se opakující části uživatelského rozhraní jsme vytvořili prefaby, abychom tak sjednotili vzhled a mohli ho snadno upravovat u všech výskytů naráz. Tyto prefaby se nachází v adresáři `Assets/Prefabs/UI/General/` a jedná se o různá tlačítka a texty s přednastavenou velikostí písma a případně s lokalizací. Dále se tam nachází prefab `Canvas`, který představuje kořenový objekt libovolného UI ve scéně a obsahuje již důležité komponenty s vhodným nastavením. Příkladem jsou komponenty `CanvasGroup` a `GenericTween` pro možnost postupného zobrazení nebo `CanvasScaler` pro přizpůsobení se různým velikostem zobrazení (dle požadavku P6).

## Hlavní menu

Scéna `MainMenu` s hlavním menu, které vidíme na obrázku 4.15, obsahuje zpravidla jen základní použití obvyklých komponent pro tvorbu UI. Všechny je pak propojuje pomocí skriptu `MainMenuUI`, který se stará o postupné zobrazení správných možností (na základě toho, jestli už existuje rozehraná hra, či nikoliv) a současně obsahuje metody, které se vyvolají po stisknutí jednotlivých tlačítek.


**Obrázek 4.15** Uživatelské rozhraní v hlavním menu.

Jedinou implementačně zajímavou částí jsou pak tlačítka 1 pro změnu jazyka. Když se scéna inicializuje, z `LocalizationManageru` (viz sekce 4.10.3) se získají všechny dostupné jazyky a pro každý z nich se pak vytvoří jedna instance prefabu `LanguageToggleUI`. Obrázky vlajek se přitom získávají ze složky `Assets/Resources/Flags/`.

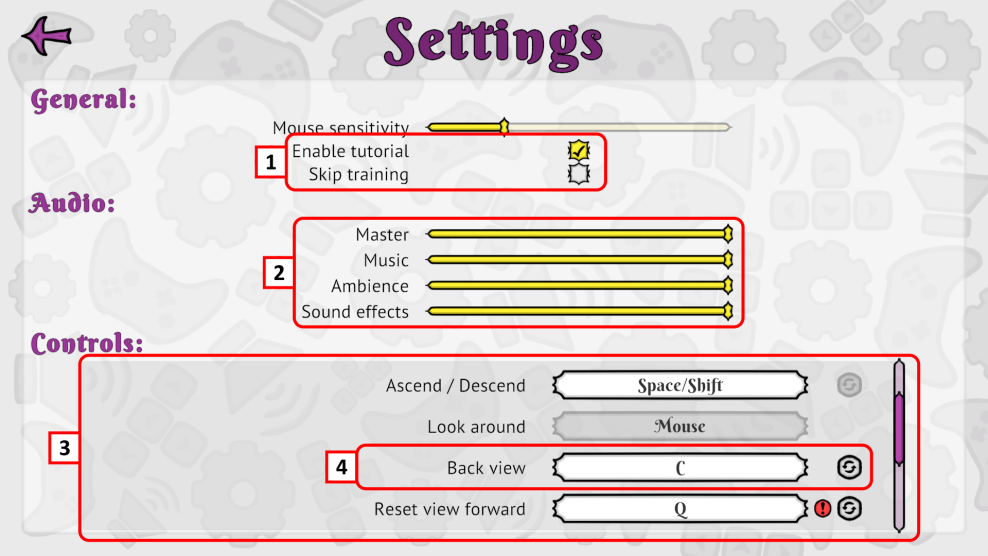
Abychom mohli do menu umístit pohybující se postavu, vložili jsme do scény instanci prefabu `Player` (s deaktivováním nepotřebných komponent), ale současně jsme pro ni vytvořili animaci, která se spouští pomocí přidané `Animator` komponenty. Pozadí za postavou je pak také součástí 3D scény, a to jako textura na obdélníkové ploše. Díky tomu může postava na pozadí vrhat stíny, ale současně šetříme výpočetní výkon tím, že se nejedná o skutečný level se spoustou objektů.

Jelikož ale pozadí není součástí UI, nemůžeme u něj snadno zajistit přizpůsobení se různým velikostem zobrazení. Na příliš širokých monitorech by byl po stranách vidět prostor za pozadím včetně postavy, zatímco zbylé prvky UI

by se přizpůsobily celému prostoru. Veškeré UI se tak nachází pod objektem pevných rozměrů, aby nemohlo přesahovat mimo pozadí, a k okrajům jsme přidali černé obdélníky, které zakryjí postavu mimo hlavní oblast. Nakonec jsme vytvořili skript `MainMenuContentHeight`, který mění výšku obsahu tak, že prvky neustále zůstávají zarovnané k rohům obrazovky.

## Nastavení

Z hlavního menu se pak může přejít do nastavení. Jelikož je však možné jej zobrazit také z menu pozastavené hry, vytvořili jsme z něj prefab `SettingsUI`, z něhož se pak vytváří instance v odpovídajících scénách. Navíc má skript `SettingsUI`, který zajišťuje funkcionalitu některých pasivnějších prvků a také perzistentní ukládání a načítání hodnot. Finální podobu vidíme na obrázku 4.16, přičemž některé významnější části si nyní popíšeme.




**Obrázek 4.16** Uživatelské rozhraní v nastavení.

Zaškrtávací políčka 1 umožňují zakázat tutoriál a trénink před závodem. Každé z nich je instancí prefabu `SettingsToggleUI` sdružujícího políčko s popisem. Prefab pak obsahuje skript `SettingsCheckbox`, který se však stará pouze o změnu barvy pozadí, pokud je políčko zaškrtnuté.

V další části jsou posuvníky 2 pro změnu hlasitosti konkrétních skupin audia. Ty jsou instancemi prefabu `SettingsSliderUI` obsahujícího posuvník s popiskem. Dále má základní komponentu `SettingsSliderUI`, která umožňuje zobrazit aktuální hodnotu vedle posuvníku (avšak v nastavení tuto možnost nevyužíváme), a navíc komponentu `AudioVolumeSlider`, která pak při každé změně hodnoty zajistí přímo změnu hlasitosti odpovídající skupiny pomocí `AudioManageru` (viz sekce 4.10.6).

Úplně dole se pak zobrazují všechna právě nastavená mapování vstupu na akce ve hře 3. Všechny související části jsou obsažené v prefabu `RebindingUI`, jehož instance je pak umístěná ve scéně. Prefab má skript `KeyRebindingUI`, který zajišťuje naplnění seznamu aktuálními daty, kdykoliv se zobrazuje nastavení. Navíc v něm můžeme říct, které akce se mají ze seznamu vynechat (např. protože

ještě nejsou plně implementované) nebo které akce jsou read-only (tj. hráč nemá možnost změnit jejich mapování).

Jeden řádek seznamu 4 je pak instancí prefabu `KeyBindingUI` (se stejnojmenným skriptem) a představuje jednu konkrétní akci. Pomocí tlačítka zobrazujícího aktuální mapování je pak možné mapování změnit. Napravo od něj je tlačítko pro reset mapování do původní hodnoty a případně upozornění na duplicitní mapování (tzn. použití stejné klávesy u dvou a více různých akcí).

## Vytvoření postavy

Pokud spustíme novou hru, dostaneme se do scény `CharacterCreation`, ve které máme možnost si vytvořit postavu. Odpovídající obrazovku jsme popsali již v sekci 2.3 společně s obrázkem návrhu uživatelského rozhraní. Na obrázku 4.17 pak vidíme, jak tato obrazovka vypadá skutečně ve hře.


**Obrázek 4.17** Uživatelské rozhraní ve scéně pro vytvoření postavy.

V uživatelském rozhraní jsou již připravené popisky pro všechny upravitelné části, ke kterým se však dynamicky za běhu doplňují jednotlivé možnosti ze skriptu `CharacterCreatorUI` (získává je z `CharacterCustomizationOptions` ve stavu hráče, viz sekce 4.10.1). Pro jednotlivé skupiny možných nastavení vzhledu 1 se používají instance prefabu `CustomizationOptionsGroupUI`, ve kterých je pak několik instancí prefabu `CustomizationOptionUI` se skriptem stejného jména pro reprezentaci individuálních možností a jejich funkcionality.

Třída `CustomizationVariantData` sdružuje všechna potřebná data pro zobrazení jedné možnosti vzhledu. Když pak hráč potvrdí své volby, v metodě skriptu `CharacterCreatorUI` se vytvoří instance třídy `CharacterCustomizationData` reprezentující jeden celkový vzhled a uloží se do stavu hráče (viz sekce 4.10.1).

Abychom mohli v UI zobrazit náhled postavy 2 se zvolenými možnostmi, ve scéně je umístěná instance prefabu `Character`, která se renderuje do textury a následně vykresluje pomocí komponenty `RawImage`. Kdykoliv se pak zvolí některá možnost, pomocí skriptu `CharacterAppearance` na objektu postavy se aplikuje. Postava má navíc skript `RotateObjectOnMouse`, který umožňuje postavou otáčet tažením myši, aby bylo možné si zvolený vzhled prohlédnout z různých stran.

## Závod (HUD)

Během závodu ve scéně `Race` i `QuickRace` je třeba hráči předávat poměrně velké množství informací. V sekci 2.6 jsme již popsali, z jakých důležitých částí by se uživatelské rozhraní skládalo, a poskytli jsme také nákres. Na obrázku 4.18 pak vidíme, jak odpovídající obrazovka skutečně vypadá ve hře.


**Obrázek 4.18** Uživatelské rozhraní během závodu.

Pro veškeré UI zobrazené během závodu jsme vytvořili prefab `RaceHUD`, který obsahuje jednotlivé části přehledně rozdělené, z nichž některé jsou pak separátní prefaby, aby bylo možné je snadno použít i jednotlivě (např. v testovací trati, kde se nezobrazují všechny části). Pomocí skriptu `RaceUI` se řídí zobrazení a skrytí relevantních částí a také se z něj aktualizují některé pasivní části.

Vlevo dole se nachází minimapa (dle požadavku P17) a informace o postupu v závodě 1. Vše je společně v prefabu `BottomLeft-Minimap`. Abychom mohli zobrazit minimapu s okolím hráče, využíváme separátní kameru na objektu `MinimapCamera`. Skript `MinimapCameraController` ji pak neustále udržuje ve fixní výšce nad hráčem. Výsledek se renderuje do `Minimap` textury, která se následně pomocí `RawImage` komponenty zobrazuje na obrazovce. Kamera je nastavena tak, aby viděla pouze vrstvy s terénem a s ikonkami, které mají jednotlivé objekty přímo u sebe (např. obruč v prefabu `Hoop`, závodník v prefabu `RacerBase`).

Vpravo dole jsou pak základní informace o aktuální rychlosti a výšce 2, které jsou instancí prefabu `BottomRight-SpeedAltitude`. Zobrazené hodnoty se získávají z `CharacterMovementController` komponenty hráče.

Vpravo po straně je část věnovaná všemu souvisejícímu s kouzly 3. Jedná se o instanci prefabu `Right-Spells` a zobrazuje se pouze tehdy, pokud má hráč alespoň jedno vybavené kouzlo. Obsahuje ukazatel aktuálního množství many a jednotlivé sloty s dosazenými kouzly, které jsou instancemi prebafu `RaceSpellSlotUI`. Ten ve skutečnosti obsahuje instanci prefabu `SpellSlotUI` (se stejnojmenným skriptem pro zobrazení slotu s ikonkou kouzla a tooltipem s dalšími informacemi), ke kterému přidává zobrazení ceny seslání kouzla a vizuální indikaci aktuálního stavu, který je řízený ze skriptu `RaceSpellSlotUI`. Pokud hráč nemá dostatek many pro seslání, cena zčervená, a pokud se kouzlo dobíjí, postupně mu nabývá vý-

plň. Jestliže kouzlo není připravené k použití, jeho slot je zmenšený a zprůhledněný. Právě zvolené kouzlo se zvýrazňuje pomocí rámečku.

Nakonec vpravo nahoře se pomocí prefabu `TopRight-Effects` zobrazují efekty právě působící na hráče 4. Ze skriptu `EffectsUI` se reaguje na událost přidání nového efektu do hráčovy `EffectibleCharacter` komponenty. V takovém případě se vytvoří nová instance prefabu `EffectSlotUI`, který pomocí stejně pojmenovaného skriptu zobrazuje ikonku efektu se zbývajícím časem. Jakmile efekt skončí, slot se sám zničí.

Hlavní skript `RaceUI` navíc zajišťuje zobrazení dalších důležitých informací, jako je odpočet před závodem nebo vizuální upozornění na let špatným směrem či minutí obruče.

Poslední částí uživatelského rozhraní v závodě, která však není na obrázku vidět, je indikace blížícího se kouzla (popsaná v sekci 4.5.2). Jestliže se na hráče blíží kouzlo, z jeho `PlayerIncomingSpellTracker` komponenty se na obrazovce vytvoří instance prefabu `IncomingSpellIndicator` řízená stejnojmenným skriptem.

## Výsledky závodu

Po dokončení závodu se hráči zobrazí výsledky, jejichž podobu jsme popsali již v sekci 2.6 společně s nákresem. Obrázek 4.19 pak zachycuje skutečnou podobu ve hře.


**Obrázek 4.19** Uživatelské rozhraní pro zobrazení výsledků závodu.

Zobrazení zajišťuje prefab `RaceResults`, který obsahuje veškeré potřebné prvky UI a řídicí logiku ve skriptu `RaceResultsUI`. Pomocí jedné z jeho metod se předají konkrétní hodnoty pro jednotlivé závodníky a skript se poté postará o jejich zobrazení. Pro každý jeden záznam 1 vytváří instanci prefabu `ResultsRow`, kdy pomocí jeho skriptu `RaceResultRowUI` předá požadovaný obsah. Následně z coroutine zařídí postupné zobrazení těchto řádků.

Jeden řádek zobrazuje umístění, jméno, čas včetně penalizace a případnou odměnu. V případě soupeřů se přidává navíc reprezentace jejich přiřazené barvy, která se používá také pro ikonky na minimapě. Řádek s výsledky hráče je přitom barevně a velikostně odlišen.

## Upozornění na novou oblast

Pokud se ve vygenerovaném levelu vyskytuje tematická oblast, se kterou se hráč doposud nesetkal, zobrazí se mu o tom informace (na obrázku 4.20) prostřednictvím objektu `NewRegionInfo` se skriptem `NewRegionInformation`. Pro oblast se tak vypíše její název a popis na pozadí tvořeném obrázkem oblasti. Všechny tyto informace se přitom získávají z její `Region` reprezentace.


**Obrázek 4.20** Uživatelské rozhraní upozornění na novou oblast v levelu.

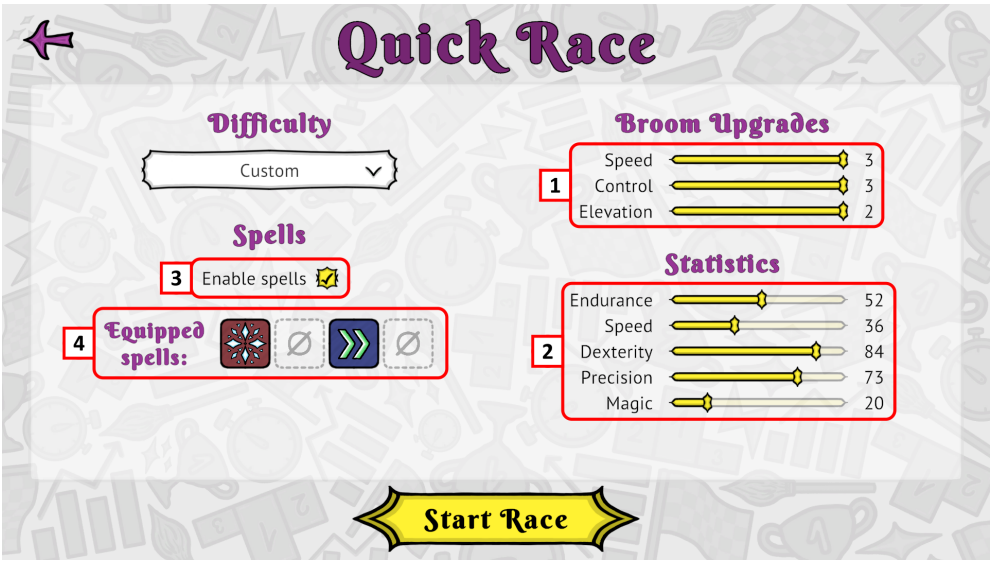
Jestliže je nových oblastí v levelu více, informace se pro ně zobrazují postupně. Teprve po zavření jedné se začne zobrazovat další.

## Rychlý závod

Uživatelské rozhraní během rychlého závodu ve scéně `QuickRace` je naprosto totožné jako to během normálního závodu ve scéně `Race`, které jsme popsali výše. Když ovšem chceme rychlý závod spustit z hlavního menu, nejprve je třeba zvolit některá nastavení. Ve scéně `MainMenu` je reprezentuje objekt `QuickRace`. Ten obsahuje veškeré potřebné prvky, jejichž podobu vidíme na obrázku 4.21.

Objekt v menu navíc obsahuje skript `QuickRaceSettingsUI`, který zajišťuje správné zobrazení prvků, poskytuje jim funkcionalitu a navíc se stará o perzistentní ukládání naposledy použitých hodnot, které se pak načítají při zobrazení rozhraní. Současně řídí změnu stavu hry dle zvolených hodnot (způsob odlišení od stavu kariérního režimu jsme popsali v sekci 3.6).

Pokud si hráč zvolí vlastní obtížnost, zobrazí se posuvníky pro volbu úrovní vylepšení koštěte 1 a hodnot statistik 2, ze kterých se pak vychází při generování levelu. Základem obou je prefab `SettingsSliderUI` (popsaný výše v sekci věnované nastavení). Pro vylepšení koštěte je zabalen v prefabu `BroomUpgradeSettings` a pro hodnotu statistiky pak v prefabu `StatSettings`. Prostřednictvím jejich skriptů se mu tak dodává funkcionalita navíc. Hlavní skript `QuickRaceSettingsUI` pak získává seznam ve hře dostupných vylepšení a statistik a dynamicky vytváří instance odpovídajících prefabů.




**Obrázek 4.21** Uživatelské rozhraní s nastaveními rychlého závodu.

Bez ohledu na zvolenou obtížnost si pak hráč může v levé části zvolit, zda mají být povolená kouzla 3. Používá se již dříve zmíněný prefab `SettingsToggleUI`. Pokud kouzla povolí, zobrazí se navíc jednotlivé sloty 4, do kterých si kouzla může dosazovat. Všechny sloty dohromady reprezentuje prefab `EquippedSpells` se skriptem `EquippedSpellsUI`. Z něj se na začátku za běhu vytvoří a inicializuje správný počet instancí prefabu `EquippedSpellSlot` pro jednotlivé sloty. Tyto prefaby obalují základní `SpellSlotUI`, ale přidávají funkcionalitu navíc ze skriptu `EquippedSpellSlotUI`.

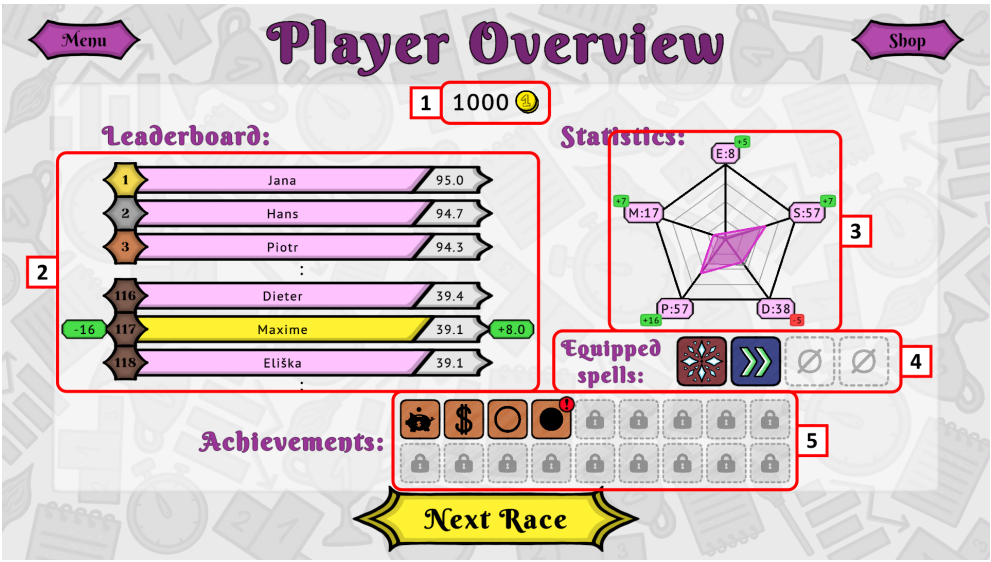
## Přehled hráče

Mezi závody se hráč dostává do scény `PlayerOverview`, kde se nachází uživatelské rozhraní zobrazující přehled všech důležitých informací. Tuto obrazovku jsme popsali již v sekci 2.3.1 společně s nákresem, jak by mohlo rozhraní vypadat. Na obrázku 4.22 pak vidíme skutečnou podobu.

Úplně nahoře se zobrazuje aktuální počet mincí 1 pomocí prefabu `CoinsUI` (se stejnojmenným skriptem pro změnu zobrazených hodnot).

Vlevo je pak globální žebříček závodníků 2, který se při každém zobrazení vytváří dynamicky tak, aby obsahoval první tři místa a poté závodníky v těsném okolí umístění hráče. Slouží k tomu skript `LeaderboardUI`, ve kterém se nastavují parametry žebříčku – celkový počet závodníků, rozsah průměrů statistik a křivka popisující vývoj statistik napříč všemi umístěními. Z těchto parametrů a ze stavu hráče (viz sekce 4.10.1) se pak pomocí prefabů `LeaderboardRow` (pro jeden řádek tabulky) a `LeaderboardGap` (pro mezeru vyjadřující vynechané řádky) sestaví tabulka. Řádky mají skript `LeaderboardRowUI` pro nastavení hodnot a zobrazení změn od minula v případě hráče.

Vpravo nahoře se vykresluje graf hráčových statistik 3. Využívá k tomu komponentu `RadarGraphUI`, která umí tvořit radarový graf zadaných parametrů (např. poloměr, počet os, maximální hodnota jedné osy). Pro zakreslení os se využívá komponenta `UIPathRenderer` poskytující metodu pro zakreslení cesty ze seznamu bodů. Jeden polygon vyznačující hodnoty v grafu je reprezentován




**Obrázek 4.22** Uživatelské rozhraní v přehledu hráče.

prefabem `GraphPolygonUI` se skriptem `RadarGraphPolygonUI`, který ve svých metodách vykresluje hrany polygonu pomocí `UIPathRenderer` komponenty a výplň pomocí `UIGraphPolygonRenderer`. Popisky u os grafu jsou pak instance prefabu `RadarGraphLabel`, v jehož skriptu `RadarGraphLabelUI` se kromě nastavení obsahu také vybírá vhodný roh pro umístění informace o změně hodnoty od minula.

Pod grafem je přehled slotů s vybavenými kouzly 4. Jedná se o instanci prefabu `EquippedSpells`, který jsme popisovali již u nastavení rychlého závodu.

Nakonec úplně dole se nachází přehled dosažených a doposud neznámých ocenění 5. O jejich zobrazení se stará skript `AchievementsUI`, který s pomocí `AchievementManageru` (viz sekce 4.10.2) získává data o aktuálním pokroku v získávání ocenění a poté pro každé ocenění dostupné ve hře vytvoří instanci prefabu `AchievementSlot`, který ve svém skriptu `AchievementSlotUI` správně inicializuje svůj vzhled na základě předaných dat (tj. barvu pozadí dle získaného stupně ocenění, ikonku a případnou notifikaci nově získaného ocenění). Ocenění jsou pak seřazená sestupně dle nejvyšší dosažené úrovně, takže všechna neznámá jsou až na konci.

## Obchod

Z přehledu hráče se dá přejít do obchodu, který se zobrazí aktivací objektu `Shop` ve scéně `PlayerOverview`. Tento objekt pak obsahuje skript `ShopUI`, pomocí kterého se inicializují prvky uživatelského rozhraní. V sekci 2.8 jsme popsali návrh této obrazovky včetně jejího nákresu. Nyní popíšeme výslednou implementaci, jejíž podoba je na obrázku 4.23.

V horní části se stejně jako v přehledu hráče zobrazuje aktuální počet mincí 1 pomocí instance prefabu `CoinsUI`.

Vlevo je pak nabídka kouzel, jejíž obsah se vytváří dynamicky při zobrazení. Ze `SpellManager` singletonu (viz sekce 4.5.1) se získá seznam všech kouzel dostupných ve hře a na základě stavu hráče (viz sekce 4.10.1) se pak určí, která z nich má hráč již zakoupená (taková jsou vizuálně odlišená od těch nezakoupených). Jedno


**Obrázek 4.23** Uživatelské rozhraní v obchodě.

kouzlo v nabídce 2 je instancí prefabu `ShopSpellSlot`, který obaluje základní `SpellSlotUI` a navíc k němu přidává prvky (např. cenu) a funkcionalitu (v rámci skriptu `ShopSpellSlotUI`). Současně se jedná o tlačítko, jehož kliknutím se dané kouzlo zakoupí.

Pod nabídkou kouzel se zobrazují aktuálně vybavená kouzla 3 pomocí instance prefabu `EquippedSpells` (stejně jako v přehledu hráče a v nastavení rychlého závodu popsaných výše).

Vpravo je pak přehled vylepšení koštěte. Jeho obsah je opět vytvořen dynamicky při zobrazení ze skriptu `ShopUI`. Jeden řádek s možností vylepšení koštěte 4 je instancí prefabu `BroomUpgradeRow`, který se ze svého skriptu `BroomUpgradeRowUI` stará o vizualizaci aktuální a maximální úrovně instancací prefabu `BroomUpgradeLevel`. Při maximální úrovni se pak skryje tlačítko pro zakoupení.

Pod výčtem možných vylepšení je umístěn náhled koštěte 5, abychom si mohli prohlédnout změny vzhledu pro jednotlivé úrovně. Přímo ve scéně se tak nachází instance prefabu `Broom`, která se renderuje do textury a zobrazuje v UI. Současně se právě pomocí jejího skriptu `Broom` získává seznam možných vylepšení pro zobrazení. Kdykoliv se pak některé vylepšení zakoupí, změny se aplikují přímo na tuto instanci metodou v odpovídající `BroomUpgrade` komponentě.

Pokud hráč na zakoupení něčeho nemá dostatek mincí, odpovídající cena se vykreslí červeně a při pokusu o zakoupení se nestane nic, pouze se ozve zamítavý zvuk.

## Výběr kouzel

Pokud ve kterékoliv obrazovce, ve které se zobrazuje přehled vybavených kouzel pomocí prefabu `EquippedSpells`, klikneme na jeden ze slotů (instanci prefabu `EquippedSpellSlot`), zobrazí se nabídka kouzel (na obrázku 4.24) reprezentovaná prefabem `SpellSelection`, který je umístěný ve stejné scéně. V nabídce je jako první vždy prázdná možnost, která umožňuje nic do zvoleného slotu nedosadit. Poté následuje dynamicky vygenerovaný seznam všech kouzel, která má hráč

k dispozici a současně nejsou dosazená v žádném jiném slotu (nechceme totiž umožnit dosazení stejného kouzla do více slotů naráz). Tato inicializace se provádí ve skriptu `SpellSelectionUI`.


**Obrázek 4.24** Uživatelské rozhraní nabídky kouzel pro dosazení do slotu.

Jedno takové kouzlo v nabídce 1 je instancí prefabu `SpellSelectionSlot`. Ten stejně jako libovolné jiné výskyty kouzel v UI využívá základní prefab `SpellSlotUI`. Navíc k němu však přidává v komponentě `SpellSelectionSlotUI` další funkcionalitu pro zajištění, že se kliknutím na daný slot dosadí kouzlo do odpovídajícího slotu a změna se propíše také do stavu hráče (viz sekce 4.10.1).

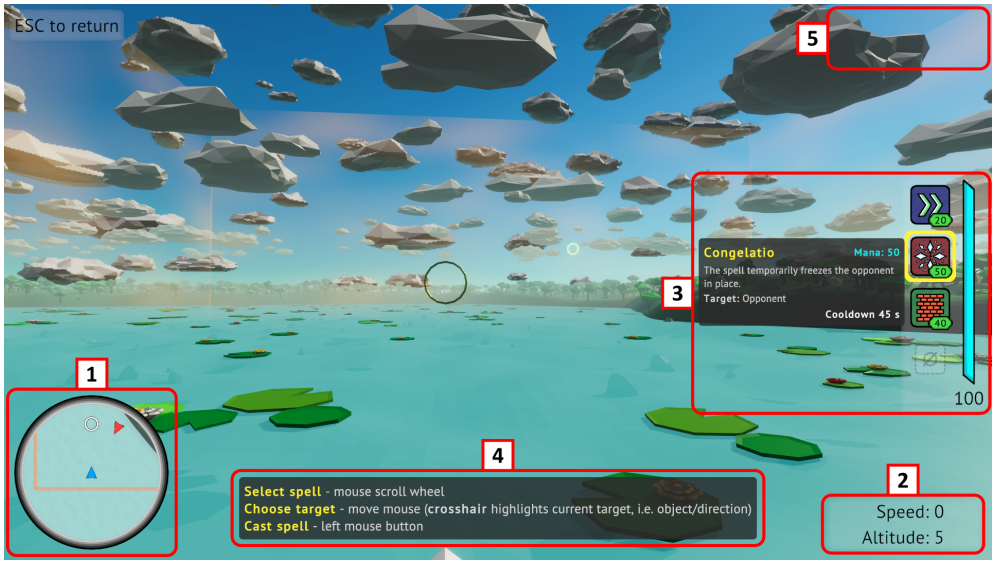
## Testovací trať

Z obchodu se dá dále přejít do testovací trati ve scéně `TestingTrack`. V sekci 2.8 jsme popsali možnou podobu a přidali také nákres, který však odrážel spíše budoucí plány. Prozatím implementovanou podobu tak vidíme na obrázku 4.25. Uživatelské rozhraní je nyní velmi podobné tomu v závodě, využívají se také většinou ty samé prefaby, ovšem s několika rozdíly. Ústředním skriptem je pak `TestingTrackUI`, který podobně jako `RaceUI` řídí zobrazování jednotlivých částí a aktualizaci některých hodnot.

Vlevo dole se nachází minimapa 1, která je však oproti závodu zjednodušená, protože kolem ní chybí doplňující informace o pokroku v závodě. Využívá se pro ni prefab `BottomLeft-MinimapSimplified` a samotné fungování je pak stejné jako v závodě.

Vpravo dole jsou pomocí prefabu `BottomRight-SpeedAltitude` vypsané údaje o aktuální rychlosti a výšce 2. Jsou přitom naprosto totožné s těmi v závodě.

Vpravo se stejně jako v závodě nachází vše související s kouzly 3 a využívá se k tomu ten samý prefab (`Right-Spells`). Narozdíl od závodu se však pro právě zvolené kouzlo zobrazuje panel s jeho popisem. Skript `TestingTrackUI` má zaregistrovaný callback na změnu kouzla (v hráčově `SpellController` komponentě) a na ni pak reaguje změnou obsahu panelu. Samotný panel se ve skutečnosti vykresluje jako statický tooltip (více v sekci 4.10.4) s komponentou `TooltipPanel`.




**Obrázek 4.25** Uživatelské rozhraní v testovací trati.

Pokud má hráč alespoň jedno kouzlo přiřazené ve slotu, po celou dobu se ve spodní části obrazovky zobrazují základní instrukce k sesílání kouzel 4. Opět se pro ně využívá panel tooltipu nastavený jako statický.

V pravém horním rohu se pak stejně jako během závodu zobrazují efekty působící na hráče 5 pomocí prefabu `TopRight-Effects`. Na obrázku však není vidět ani jeden efekt, protože v dobu pořízení zrovna žádný na hráče nepůsobil.

Nakonec v levém horním rohu se zobrazuje informace o možnosti návratu zpět pomocí klávesy `ESC`.

## Konec hry

Jakmile hráč dokončí hru tím, že se umístí v globálním žebříčku na prvním místě, načte se scéna `Ending`. Její podoba (na obrázku 4.26) je vcelku jednoduchá. Přímo uprostřed je umístěn obrázek poháru, který se postupně objeví a pak se neustále naklání ze strany na stranu pomocí komponenty `GenericTween` (více v sekci 4.10.5).

Zpoza poháru pak v náhodných intervalech vylétávají barevné hvězdičky 1. Ve scéně je jich připravených celkem osm, přičemž všechny jsou instance prefabu `Star`. Každá má pak `Animator` komponentu (pro animaci postupného zobrazení a zmizení) a skript `EndingStarUI`. Tento skript obsahuje různé užitečné metody, např. pro změnu barvy či spuštění animace.

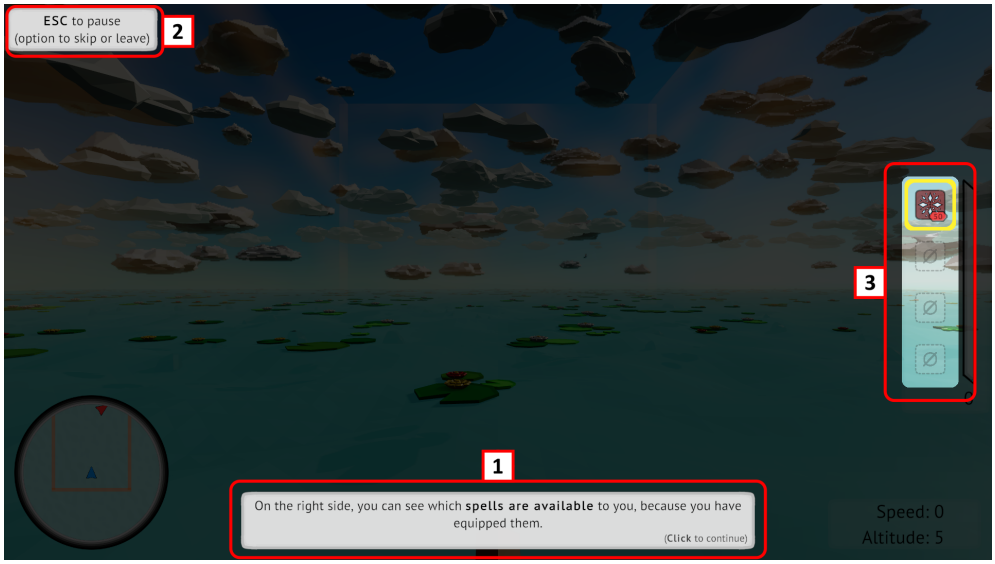
Všechny hvězdičky pak zastřešuje skript `EndingUI`, který je v určitých intervalech vypouští. Pokaždé se podívá, které všechny jsou právě v nečinném stavu (tj. nejsou vůbec zobrazené), náhodnou z nich vybere a spustí.

S krátkým časovým odstupem po zobrazení scény se v pravé části obrazovky objeví dvě tlačítka – jedno pro pokračování ve hře a druhé pro návrat do hlavního menu.


**Obrázek 4.26** Uživatelské rozhraní ve scéně po dokončení hry.

## Tutoriál

Když se ve hře spustí tutoriál, potřebujeme umět pracovat s uživatelským rozhraním dvěma různými způsoby – zobrazit nějaký panel s textem a/nebo zvýraznit nějaký konkrétní prvek, abychom na něj hráče upozornili. Příklad vidíme na obrázku 4.27. Navíc by se hodilo umět ztmavit obrazovku, pokud chceme, aby hráč zatím nevnímal nic na pozadí a pouze si četl zobrazený text. Prefab `Tutorial` (popsaný již v sekci 4.9.1) tak obsahuje podobjekty, které zařizují přesně uvedenou funkcionalitu. Ztmavení obrazovky se pak řeší z jeho skriptu `Tutorial`.




**Obrázek 4.27** Uživatelské rozhraní během tutoriálu.

Podobjekt `Panels` obsahuje několik předpřipravených panelů pro zobrazení textu. Poté ve skriptu `TutorialPanels` poskytuje metody pro jejich zobrazení a skrytí. Navíc má speciální metodu `ShowTutorialPanelAndWaitForClick`, kdy

v panelu rovnou zobrazí informaci, že pro pokračování má hráč kliknout, a následně čeká, dokud neklikne, než panel skryje.

Pro samotné panely pak máme dvě různé komponenty. `SimpleTutorialPanel` slouží pro zobrazení jen jednoduchého textu. `TutorialPanelWithAlignment` ho pak rozšiřuje o možnost určit konkrétní umístění na obrazovce (hodnotou z výčtového typu `TutorialPanelAlignment`) a zobrazit informaci o nutnosti kliknout pro pokračování. Když se pak pomocí komponenty `TutorialPanels` zobrazuje panel, můžeme zvolit také umístění, čímž zvolíme, který z předpřipravených panelů **1** se zobrazí. Panel v levém horním rohu **2** pak zobrazuje informace speciálně o možnosti použití `ESC` pro pozastavení hry nebo přeskočení aktuální fáze tutoriálu.

Dále má prefab `Tutorial` podobjekt `Highlight`, který umožňuje zvýraznění vybrané části obrazovky **3**. Jeho skript `TutorialUIHighlight` nabízí metody pro zvýraznění určitého obdélníku (`Rect`) či konkrétního prvku (`RectTransform`). Můžeme také říct, jestli má být možné skrz zvýrazněnou oblast klikat, nebo ne. Samotné zvýraznění se pak provádí vymaskováním konkrétní obdélníkové oblasti z `Image` komponenty ztmavující celou obrazovku. Abychom navíc mohli blokovat klikání mimo zvýrazněnou oblast, rozmístíme kolem vyříznutého obdélníku čtyři průhledné objekty (v hierarchii pod objektem `RaycastBlocks`).

Speciálně v tutoriálu během prvního závodu pak navíc chceme hráči zobrazit panel s možností zvolit, zda chce zachovat možnost tréninku před závodem (viz sekce 2.6), nebo ji chce vypnout. K tomu slouží podobjekt `SkipTrainingPanel`, ke kterému se pak implementace dané fáze tutoriálu dostane přes referenci ve skriptu `Tutorial`.

Pokud se tutoriál odehrává ve scéně `Tutorial`, zobrazují se na obrazovce podobné prvky uživatelského rozhraní jako během závodu a v testovací trati. Využívají se tedy prefaby, které jsme popsali již v předchozích sekcích. Jedná se o minimapu (ve zjednodušené podobě jako v testovací trati), aktuální rychlost a výšku, vybavená kouzla s ukazatelem many a efekty působící na hráče. Vše pak řídí komponenta `SimplifiedHUD`, která je předkem komponenty `TestingTrackUI` použité v testovací trati.

## Pozastavení hry

Pro možnost pozastavení hry ve scénách, kde se hráč přímo pohybuje na koštěti (tj. `Race`, `QuickRace`, a `Tutorial`), slouží prefab `GamePause`, jehož instance je pak umístěná v odpovídající scéně. Veškerou logiku řídí stejnojmenný skript, který tak nejen poskytuje metody pro pozastavení či obnovení hry (s podporou funkcí popsaných v sekci 3.9.7), ale navíc zajišťuje správné zobrazení UI.

V různých scénách se možnosti v zobrazeném menu liší. Během tutoriálu má hráč možnost přeskočit danou část, nebo úplně odejít do hlavního menu, zatímco během závodu je k dispozici možnost vzdát závod. Menu pozastavené hry tak má rovnou připravené všechny tyto možnosti jako podobjekty, které se pak jen správně (de)aktivují z metod skriptu `GamePause`.

### Generátor levelů

Ve scéně `LevelGeneratorDemo` se nachází oddělený generátor levelů (instance prefabu `LevelGenerator`), o kterém jsme psali již v sekci 2.10 (a který jsme si vytyčili v požadavku **P30**). Díky němu pak máme možnost si volně prohlížet levely, které jsme schopni ve hře generovat. Abychom však mohli různě upravovat parametry, je třeba vytvořit uživatelské rozhraní. Jelikož to však není přímo součástí hry, použili jsme jen zcela základní UI s výchozím vzhledem. V jeho implementaci se tedy nenachází nic zvláštního či překvapivého.

O propojení hodnot z prvků uživatelského rozhraní se samotným generátorem se stará skript `LevelGeneratorDemo`. Navíc umožňuje stisknutím mezerníku veškeré UI skrýt (případně znovu zobrazit). To je užitečné především při pořizování snímků obrazovky, pro které jsme také přidali podporu. Stisknutím kombinace `Ctrl+C` se tak pořídí snímek a uloží do podsložky `Application.persistentDataPath`.

## 4.12 Rozšíření editoru

Kromě samotné herní logiky a uživatelského rozhraní jsme během vývoje implementovali také několik skriptů, které nám umožňují snazší práci v editoru. Touto problematikou jsme se zabývali již v sekci 3.10, kde jsme také uvedli několik obecných důvodů pro implementaci vlastních rozšíření editoru. Nyní se tedy podíváme na konkrétní příklady. Všechny důležité skripty se pak nachází ve složce `Assets/Scripts/Editor/`.

### 4.12.1 Barevná paleta

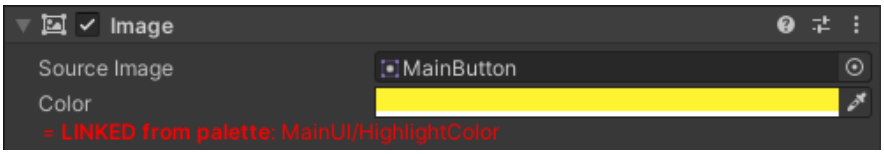
V sekci 3.7 jsme se věnovali barevné paletě a velmi podrobně jsme popsali jak motivaci pro její použití, tak zamýšlený návrh včetně použitého zdroje. Nyní už tedy jen velmi stručně zmíníme konkrétní třídy, které se na její implementaci podílí. Jedná se o rozšíření editoru, jelikož se paleta používá z Inspector okna, do kterého přidáváme nové možnosti.

Paletu reprezentuje třída `ColorPalette`, která ve skutečnosti implementuje `ScriptableObject`, abychom mohli instance vytvářet snadno v editoru a palety tak byly součástí projektu jako assety. Díky tomu s nimi pak můžeme snadno manipulovat (např. mezi nimi přecházet). V samotné paletě jsou pak datové položky pro nastavení jednotlivých barev a také metoda pro jejich získání z kódu (na základě hodnoty z výčtového typu `ColorFromPalette`).

Paleta navíc funguje jako singleton, aby byla snadno dostupná, přičemž se do statické položky pro instanci dosadí vždy ta, která se jmenuje `ColorPalette` a nachází se ve složce `Assets/Resources/`. Musíme tedy zajistit, že instance palety, kterou chceme nakonec ve hře používat, toto splňuje. Jelikož je ale důležité pracovat se správnou instancí také v editoru, nejen za běhu, pomocí skriptu `CustomAssetModificationProcessor` se detekují změny takových assetů (např. přesunutí, vytvoření, smazání). V takovém případě se instance singletonu resetuje, takže budeme pracovat opět s tou správnou.

Pokud pak chceme pro něco použít barvu z palety, buď to můžeme provést z kódu pomocí metody v `ColorPalette`, nebo nastavit provázání již v editoru (v závislosti na situaci). Pomocí skriptu `ColorPropertyEditor` přidáváme k libo-

volné `Color` vlastnosti zobrazené v Inspectoru do kontextového menu možnosti pro provázání s barvou z palety. Zvolením některé z možností se barva ihned aplikuje. A pokud je konkrétní vlastnost již provázaná s barvou z palety, pak se v Inspectoru zobrazí upozornění (na obrázku 4.28).




**Obrázek 4.28** Vizuální indikace propojení `Color` vlastnosti s barvou z palety.

Takto vytvořená provázání se pak ukládají pomocí `ColorPaletteLinks`. Jedná se o rozšíření `ScriptableSingleton<ColorPaletteLinks>`, což zajistí chování singletonu navíc s automatickým uložením do souboru ve složce `ProjectSettings/` v adresáři projektu. Pro každou barvu z palety pak vytváříme instanci `ColorGroup`, do které se ukládají všechny propojené vlastnosti. Díky tomu můžeme libovolnou změnu barvy aplikovat na všech odpovídajících místech. `ColorPaletteLinks` navíc poskytuje metody pro správu těchto propojení a pro různé dotazy nad nimi.

Nakonec jsme vytvořili také skript `ColorPaletteEditor`, který do Inspectoru pro `ColorPalette` přidává tlačítko pro možnost aplikovat barvy na všech místech dle `ColorPaletteLinks`. Pokud bychom pak tedy změnili barvu v paletě, můžeme jediným stisknutím tlačítka zajistit, že se barva změní také u všech provázaných objektů. Využívá se k tomu metoda `ApplyColors()` v `ColorPaletteLinks`. Současně takto můžeme aplikovat barvy také z palety, která není nastavená jako ta výchozí (svým umístěním a pojmenováním), což může být užitečné např. pro porovnání různých vzhledů. Navíc u palety v Inspectoru zobrazujeme upozornění, že po každé změně je třeba barvy aplikovat.

Barvy z námi definované palety pak využíváme na mnoha místech napříč projektem. Obvykle se jedná o prvky uživatelského rozhraní.

### 4.12.2 Ostatní rozšíření

Mimo barevnou paletu jsme implementovali také několik dalších, spíše menších, rozšíření. Můžeme je přitom rozdělit do několika skupin podle jejich účelu.

#### Skrytí nerelevantních položek

Některá rozšíření upravují zobrazení datových položek komponent v Inspectoru tak, aby se zobrazovaly pouze položky relevantní pro použitou kombinaci hodnot parametrů. Rozlišují se přitom rozšíření dvou druhů, což určuje také to, jaký typ se rozšiřuje:

- Pokud chceme implementovat jiné zobrazení pro celou komponentu (tj. potomka `MonoBehaviour`), rozšíření vytvoříme pomocí třídy, která je potomkem `Editor` a má atribut `[CustomEditor(...)]` pro určení typu komponenty, kterou upravuje.
- Pokud chceme upravit zobrazení obyčejné třídy, která je ovšem označená atributem `[System.Serializable]`, takže se zobrazuje v Inspectoru, pokud

se používá jako parametr v nějaké komponentě, pak vytvoříme třídu, která je potomkem `PropertyDrawer` a má atribut `[CustomPropertyDrawer(...)]` pro určení typu, se kterým má pracovat.

V naší hře se pak jedná například o následující konkrétní implementace:

- `CutsceneCameraEditor` – Zobrazuje různé datové položky pro parametry chování kamery (v komponentě `CutsceneCamera`) na základě toho, které chování je zvoleno.
- `SpellEditor` – V komponentě `Spell` zobrazuje pouze ty datové položky, které jsou relevantní pro právě zvolený typ cíle kouzla. Např. položka `spellTargetTag` je viditelná pouze tehdy, pokud je cílem kouzla nějaký objekt, který pak můžeme specifikovat pomocí tagu.
- `TooltipPanelEditor` – Do zobrazení komponenty `TooltipPanel` přidává položku pro nastavení reference na `RectTransform` komponentu pouze tehdy, pokud se nejedná o statický panel a je tedy třeba upravovat jeho velikost.
- `SpellColorInitializerParameterPropertyDrawer` – Upravuje zobrazení třídy `SpellColorInitializerParameter`, která se používá v komponentě `SpellEffectColorInitializer` pro nastavení barvy kouzla různým vizuálním efektům. Položka pro uvedení konkrétní hodnoty barevné složky se zobrazí pouze tehdy, pokud je zaškrtnutá možnost přepisu dané složky.
- `ConditionalHidePropertyDrawer` – Umožňuje zobrazit či skrýt položku označenou atributem `[ConditionalHideAttribute(...)]` na základě `bool` hodnoty v nějaké jiné položce, jejíž jméno je uvedeno v parametru atributu.

#### Zobrazení chyby nebo varování

Další skupinou rozšíření jsou taková, která pro konkrétní komponentu v Inspectoru zobrazují navíc nějaké chyby či varování v případě, kdy nemají všechny datové položky požadované hodnoty. Jedná se pak o následující:

- `SpellEditor` – Zobrazuje chybu, pokud některé důležité položky komponenty `Spell` nemají přiřazenou hodnotu. Např. pokud chybí reference na `SpellEffectController` komponentu, která je nezbytně nutná pro seslání kouzla.
- `SpellEffectControllerEditor` – V komponentě `SpellEffectController` musí být nastavená reference na komponentu odvozenou od `SpellEffect`, která reprezentuje skutečný funkční efekt kouzla. Pokud toto neplatí, zobrazí se chybová hláška.
- `RacerAffectingSpellEffectEditor` – Jestliže kouzlo ovlivňuje závodníky a má proto komponentu `RacerAffectingSpellEffect`, bylo by vhodné v ní nastavit také vizuální efekt přehrávající se během působení. Pokud chybí, zobrazí se upozornění.

- `SpellEffectColorInitializerEditor` – Zobrazuje chybovou hlášku v případě, že v komponentě `SpellEffectColorInitializer` není vyplněný identifikátor kouzla, který je ale potřeba pro získání a nastavení barvy vizuálních efektů.
- `SpellTrajectoryVisualEffectEditor` – Pokud zapomeneme v komponentě `SpellTrajectoryVisualEffect` pro vizuální efekt během seslání kouzla nastavit komponentu odvozenou od `SpellTrajectoryComputer`, která počítá trajektorii, zobrazí se chyba. Podobně se objeví varování, jestliže není nastavena ani jedna složka vizuálního efektu, takže by kouzlo nebylo vůbec vidět.

### Vytváření předpřipravených objektů

Dále jsme ve třídě `GameObjectMenuItems` implementovali několik statických metod, které se vyvolávají z nově přidaných položek v menu editoru. Pomocí nich se vytvářejí nové objekty s již předpřipravenými komponentami, konkrétně:

- `GameObject/Localization/Localized TextMeshPro` – Umístí do právě otevřené scény nový objekt, který reprezentuje lokalizovaný text v uživatelském rozhraní. Rovnou tak obsahuje komponenty `TextMeshProUGUI` (pro samotné zobrazení textu) a `LocalizedTextMeshProUI` (pro zajištění lokalizace).
- `Assets/Create/Spell System/Spell` – Vytvoří prefab variantu od prefabu `SpellPrefab` pro zavedení nového kouzla. Pomocí `AssetNameDialogEditor` zobrazí dialog pro zadání jména assetu, následně ho uloží a rovnou zobrazí v Project okně.
- `Assets/Create/Spell System/Self-cast Spell` – Funguje stejně jako předchozí, ale základem je prefab `SpellSelfCastPrefab` pro kouzlo sesílané na sebe sama, takže obsahuje další podobjekty a komponenty.
- `Assets/Create/Spell System/Spell with Trajectory` – Poskytuje podobné chování jako předchozí, ale využívá `SpellWithTrajectoryPrefab` jako základní prefab pro vytvoření kouzla, které se sesílá směrem pryč.

### Spouštění metod za běhu v editoru

Pokud chceme umožnit volání nějakých metod v komponentách přímo za běhu hry v editoru, můžeme k takové metodě přidat atribut `[ContextMenu(...)]`, díky kterému se pak přidá položka pro volání této metody do kontextového menu dané komponenty. Využili jsme to na několika různých místech, např.:

- Prefab `LevelGenerator` pro generování levelů má na svém podobjektu `Terrain` navěšený skript `TerrainSaver`. V jeho kontextovém menu se pak nachází položky pro dvě různé metody. Pomocí jedné z nich se dá vygenerovaný level uložit jako prefab, což jsme použili pro vytvoření pevně daného levelu pro scény `TestingTrack` a `Tutorial`.
- V sekci 4.10.10 jsme popsali využití tohoto atributu pro možnost rozmístit mraky a vygenerovat skybox z editoru.

#### Spuštění hry z libovolné scény

Posledním implementovaným rozšířením je možnost spuštění hry v editoru tak, aby začínala v počáteční scéně `Start`, ať už se ve chvíli spuštění nacházíme v kterékoliv scéně. Obvyklé tlačítko pro spuštění totiž spouští hru z právě otevřené scény, ale ve scéně `Start` se nacházejí všechny důležité singletony. Současně je však nepraktické muset neustále přecházet mezi scénami, pokud pracujeme na jedné a pak chceme hru spustit a otestovat.

Vytvořili jsme tedy třídu `SceneAutoLoader` se dvěma metodami. Jedna z nich spustí hru ze scény `Start` a druhá v editoru otevře naposledy editovanou scénu (ukládá se do `EditorPrefs`), takže ji můžeme použít pro návrat do pracovní scény po ukončení hry. Obě metody jsme pak přidali do menu v editoru (v části `Play`) a navíc jim přiřadili klávesové zkratky `Ctrl+H` a `Ctrl+G` pro snazší přístup.

Toto řešení je silně inspirováno dvěma zdroji. Jedním z nich je kód od uživatele *hexdump* zveřejněný jako odpověď na Unity Discussions fóru [69]. Druhým je pak nástroj `SceneAutoLoader` [70], který vytvořil Pablo Costa.

## 4.13 Možnosti testování

Zatímco jsme hru vyvíjeli, bylo důležité najít efektivní způsob, jak otestovat různé stavy, aniž bychom museli hrou skutečně projít až do daného okamžiku. Implementovali jsme proto možnost používat cheaty za běhu hry. Navíc je možné také upravovat obsah souborů, do kterých se ukládá stav hry. Obě tyto možnosti v této sekci popíšeme detailněji.

### 4.13.1 Cheaty

V sekci 2.10 jsme popsali, jakým způsobem by se měly cheaty ve hře používat (tj. jako zadávání příkazů do konzole), a uvedli jsme také pár konkrétních příkladů cheatů. V sekci 3.9.8 jsme pak návrh cheatů rozvedli do větších detailů a popsali také principy jejich chování a funkcionalitu, kterou bychom chtěli, aby splňovaly.

Klíčový je prefab `Cheats`, jehož instance je umístěná ve scéně `Start`. Pomocí skriptu `Cheats` se mu dodává veškeré potřebné chování, přičemž se jedná o singleton přetrvávající mezi scénami (viz sekce 4.10.9).

Jeden konkrétní cheat je reprezentován třídou `CheatCommand`, která obsahuje všechna důležitá data (např. jednoslovný příkaz spojený s cheatem, funkci pro parsování a provedení cheatu, text nápovědy). Konkrétní cheaty jsou pak jednotlivé instance třídy a vytvářejí se přímo ve skriptu `Cheats`. Rovnou se také uloží do slovníku, aby bylo možné snadno získat instanci na základě jednoslovného příkazu, který se při použití cheatu zapisuje vždy jako první.

Abychom navíc mohli zajistit, že se některé cheaty mohou použít jen v určitých scénách, každý cheat má seznam scén, ve kterých je povolen, případně seznam scén, ve kterých je zakázán. Pokud je pak aktuální scéna v seznamu zakázaných, nebo pokud je seznam povolených neprázdný a aktuální scéna v něm není, cheat je zakázaný, jinak je povolený.

Skript `Cheats` má navíc seznam cheatů, které se mají provést při inicializaci, a také těch, které se mají provést po načtení konkrétní scény. Díky tomu můžeme snadno opakovaně testovat konkrétní situaci, aniž bychom cheaty museli pokaždé

manuálně zadávat. Stačí je sepsat do parametrů komponenty v Inspectoru. Dále je také možné tam úplně vypnout podporu cheatů nebo zvolit délku historie (tj. počet posledních zadaných cheatů, ke kterým se můžeme vracet).

Ve zbytku sekce nyní uvedeme kompletní výčet dostupných cheatů.

**help**
Zobrazí základní nápovědu k používání cheatů a také seznam všech dostupných v aktuální scéně.

**help <command>**
Zobrazí nápovědu k zadanému příkazu společně s příklady jeho použití.

**scene <sceneName>**
Přejde do uvedené scény. Lze použít pro přechod do scény `LevelGeneratorDemo`, která není přístupná žádným jiným způsobem.

**coins <amount>**
Změní aktuální množství mincí o zadanou hodnotu, která může být i záporná.

**stats all=<0-100>**
Nastaví všechny hráčovy statistiky na zadanou hodnotu.
**stats (<statLetter>=<0-100>)+**
Nastaví specifikované statistiky (`statLetter` je první písmeno jejich anglického názvu) na zadané hodnoty, přičemž nespecifikovaným statistikám zůstane jejich původní hodnota.
Příklad: `stats m=45 p=21` (nastaví magii na 45 a přesnost na 21)

**completion set**
Nastaví flag, že hra byla již někdy dokončena (takže se už neobjeví obrazovka konce hry).
**completion reset**
Zruší flag, že byla hra již dokončena (takže se obrazovka konce hry může objevit znovu).

**upgrade all**
Nastaví úroveň všech vylepšení koštěte na maximum.
**upgrade <upgradeIdentifier>**
Navýší úroveň tohoto konkrétního vylepšení koštěte.
Příklad: `upgrade Control` (vylepší ovladatelnost o 1 úroveň)

**downgrade all**
Nastaví úroveň všech vylepšení koštěte na 0.
**downgrade <upgradeIdentifier>**
Sníží úroveň konkrétního vylepšení koštěte.

**spell all**
Odemkne všechna ve hře dostupná kouzla.
**spell <spellIdentifier>**
Odemkne kouzlo se zadaným identifikátorem.
Příklad: `spell MateriaMuri` (odemkne kouzlo Materia Muri)
**spell reset all**
Zamkne všechna ve hře dostupná kouzla.
**spell reset <spellIdentifier>**
Zamkne kouzlo se zadaným identifikátorem.

**start**
Okamžitě ukončí trénink a spustí závod (bez nutnosti letět do startovní zóny).

**finish (time=<timeInSeconds)? (missed=<numberOfHoopsMissed>)? (place=<placeToFinishAt>)?**
Okamžitě ukončí závod se zadanými parametry (které mohou být v libovolném pořadí). Můžeme specifikovat výsledný čas, počet minutých obručí či konečné umístění.
Příklad: `finish time=82 missed=3` (hráč dokončí závod s časem 82 s a 3 minutými obručemi)

**speed <value>**
Nastaví maximální možnou rychlost na zadanou (nezápornou) hodnotu.

**mana <amount>**
Změní aktuální množství many na zadanou hodnotu.
**mana max**
Doplní aktuální množství many do maxima.
**mana off**
Zapne neomezené množství many, tj. sesílání kouzel ji nebude spotřebovávat.
**mana on**
Zapne obvyklý režim many, kdy se sesíláním kouzel spotřebovává.

**recharge all**
Okamžitě dobije všechna vybavená kouzla.
**recharge off**
Vypne režim dobíjení kouzel, takže mají nulovou dobíjecí dobu a po seslání jsou stále plně nabitá.
**recharge on**
Zapne obvyklou dobíjecí dobu kouzel, kdy je po seslání není možné určitou dobu použít znovu.

### 4.13.2 Úprava uloženého stavu

Druhou možností změny stavu hry je přímo modifikace souborů, které tento stav obsahují. Je třeba to provést ještě před spuštěním hry, než se data načtou, ale je možné tak testovat také situace, které pomocí aktuálně dostupných cheatů nejsou možné (např. změna fáze tutoriálu či pokroku v získávání ocenění).

Do souborů se většinou ukládají přímo reprezentace objektů serializovaných do formátu JSON, takže např. pojmenování položek odpovídá přesně struktuře daných tříd. Nebudeme proto formát nijak zvlášť vysvětlovat, protože by měl být vcelku zřejmý. Uvedeme pouze výčet jednotlivých souborů s příklady dat, která se v nich nachází. Všechny tyto soubory se pak vytváří v podsložce `Application.persistentDataPath`. V systému Windows je to obvykle `%userprofile%/AppData/LocalLow/Monopodiki/Brooom/Saves/`.

- `character_customization.json` – Obsahuje jméno a vzhled postavy, které si hráč zvolil na začátku hry. Serializace `CharacterCustomizationSaveData`.
- `player_state.json` – Obsahuje základní stav hráče, např. aktuální hodnoty statistik nebo počet mincí. Serializace `PlayerStateSaveData`.
- `broom_upgrades.json` – Obsahuje informace o vylepšeních koštěte, tj. název, aktuální úroveň a maximální úroveň. Serializace `BroomUpgradesSaveData`.
- `spells.json` – Obsahuje informace související s kouzly, např. dostupnost kouzel, vybavená kouzla, použití kouzel (zda byla již někdy seslána). Serializace `SpellsSaveData`.
- `regions.json` – Obsahuje seznam již navštívených tematických oblastí. Serializace `RegionsSaveData`.
- `achievements_<group>.json` – Soubory obsahující data potřebná pro poznamenání pokroku v získávání ocenění, sdružená do skupin (např. `broom`, `spells`). Serializace odpovídajícího potomka `AchievementData`.
- `quick_race.json` – Obsahuje naposledy použitá nastavení rychlého závodu, např. obtížnost, vybavená kouzla. Serializace `QuickRaceSaveData`.
- `tutorial.json` – Obsahuje informaci o aktuální fázi tutoriálu s jejím případným podstavem (obvykle je prázdný). Serializace `TutorialSaveData`.
- `settings.json` – Obsahuje používané hodnoty nastavení, např. hlasitost audia, zvolený jazyk. Serializace `SettingsSaveData`.
- `key_bindings.json` – Obsahuje aktuální mapování vstupu na akce ve hře. Pro serializaci se využívá přímo metoda `SaveBindingOverridesAsJson()` třídy `InputActionAsset`.

Na závěr je však třeba ještě zmínit, že se nijak neověřuje, zda je obsah souborů validní (mluvili jsme o tom již v sekci 3.2). Není tedy zaručené, že hra bude v konzistentním stavu, pokud se nedodrží správný formát a správné hodnoty.

# 5 Experimentální otestování

V rámci cíle C4 jsme si určili, že bychom chtěli provést experimenty, pomocí kterých bychom mohli vyhodnotit různé aspekty hry. Pomohlo by nám to identifikovat problematická místa návrhu, odladit různé parametry a také odhadnout šance na úspěch finální hry. Je přitom klíčové, abychom v tu chvíli měli již dostatečně bohatou verzi hry, která obsahuje všechny zásadní herní mechaniky a rozumně nastavené parametry, aby byla plně testovatelná. Nejprve jsme proto provedli interní playtesting, ve kterém jsme ověřili základní požadavky, a teprve na základě něj jsme pak vytvořili verzi šířenou mezi širší skupinu lidí. V této kapitole popíšeme jak průběh, tak výsledky provedených experimentů.

Během vývoje jsme se věnovali klasickému Quality Assurance (QA), který se zabývá technickým testováním hry, hledáním chyb v herním kódu a testováním stability hry. Nyní se však budeme soustředit na tzv. Game User Research (GUR), který se zaměřuje na herní zážitek, hratelnost a použitelnost hry. Důležitou částí je vyhodnocení herního návrhu a to, jak hráči interagují s hrou.

## 5.1 Interní testování

Jako první jsme tedy provedli interní playtesting, přičemž v první části se na něm podílela pouze autorka této práce. Hru hrála po delší dobu a cílem přitom bylo experimentálně ověřit několik základních požadavků:

- Jednotlivé složky statistik, ze kterých se počítají (popsali jsme je v sekci 2.6), mají rozumné váhy a jejich podíl na výsledné hodnotě odpovídá jejich důležitosti.
- Jednotlivé hodnoty statistik narůstají rozumně a nedochází k žádným náhlým skokům.
- Je možné dosáhnout konce hry (tj. určité minimální hodnoty váženého průměru statistik), a to v rozumném čase.
- Je možné dosáhnout maximální hodnoty některé statistiky pro získání odpovídajícího ocenění (viz sekce 6.2).

Pozorovali jsme, že u některých statistik bylo možné dosáhnout vysoké hodnoty vcelku brzy. Zvýšili jsme tedy váhu předchozí hodnoty, když se kombinuje nová se starou, abychom nárůst trochu zmírnili, i kdyby hráč konzistentně dosahoval vysokých hodnot v rámci ohodnocení výkonu v závodě.

Dále jsme snížili váhu složky s úspěšností sbírání bonusů pro výpočet statistiky *přesnosti*. Zpočátku je totiž její hodnota velmi snadno vysoká, ale později je to těžší, když se odemknou kouzla, tratě jsou náročnější a soupeři lepší. Takto není hráč příliš trestán, pokud stihne soupeř sebrat bonus těsně před ním. U statistiky *magie* pak docházelo k velmi náhlým výkyvům hodnoty. Nakonec se však ukázalo, že to bylo chybou ve výpočtu.

Během testování jsme ověřili, že konce hry sice je možné dosáhnout, ale přesto jsme upravili minimální potřebný průměr statistik z původních 97,2 na 95. Mělo by tak být možné se nad danou hodnotou udržet konzistentně, což sice

není pro dosažení konce hry potřeba, ale dávalo by to hráči více pocit, že si to zasloužil a neskončil na prvním místě náhodně. Díky této změně v kombinaci se snížením váhy sbírání bonusů u statistiky *přesnosti* věříme, že by měl být konec hry dosažitelný v rozumně dlouhé době. Autorka práce by ho snadno dosáhla za 2,5 hodiny.

Co se týče maximální hodnoty statistiky pro získání ocenění, pro statistiku *rychlosti* by to mělo být velmi snadné (s vylepšením koštěte, sbíráním bonusů a sesíláním zrychlujícího kouzla). Statistika *vytrvalosti* by pak měla také umožňovat vcelku předvídatelné dosažení maximální hodnoty, protože neustále roste, pokud se hráč v závodě konzistentně umisťuje v první polovině.

Ve druhé části interního testování si pak hru zahrál jeden další tester. Hraní ovšem neprobíhalo dle žádného scénáře, hrál zcela nezávisle. Výsledkem pak bylo několik podnětů, které jsme také zapracovali do hry:

- Původně plánovaná maximální délka závodu měla být zhruba mezi 5 až 10 minutami. Nakonec jsme ji však zkrátili na zhruba 5 minut, protože již závod trvající 3 minuty připadal testerovi vcelku dlouhý.

- Při výběru kouzel pro soupeře (viz sekce 2.3.2) jsme původně umožňovali jako kouzlo navíc zvolit i takové, které se sesílá na ostatní závodníky. Na základě zpětné vazby jsme však taková zakázali. Pokud si totiž hráč odemknul své první kouzlo a chtěl se ho teprve naučit pořádně používat, soupeři na něj už sesílali úplně jiné. Hráč tak byl zahlcen a neměl možnost se bránit nebo to oplácet. Nakonec tedy sice stále umožňujeme kouzlo navíc, ale nesmí se sesílat na sebe (aby hráč mohl pozorovat efekt), ani na ostatní (aby hráč nemohl být cílem).

- V původním návrhu se vůbec neumožňovaly duplicity v mapování kláves. Pokud bychom něco chtěli změnit v cyklu, museli bychom nejprve cyklus rozbít dosazením něčeho dočasného. Ukázalo se, že uživatelsky přívětivější by bylo duplicity povolit během mapování, ale pak třeba zrušit mapování té druhé akce. Nakonec jsme se rozhodli i toto druhé mapování zachovat a hráče pouze vizuálně upozornit, že se jedná o duplicitní mapování. Pokud by jej hráč nechal beze změny, pak by se tedy jednou klávesou vyvolaly obě přiřazené akce naráz.

- Pokud byl před cílovou čarou kopec, soupeři se o něj někdy zasekli a do cíle nedoletěli. Upravili jsme tedy způsob určení bodu, do kterého míří, aby k takovým situacím nedocházelo.

Díky tomuto předběžnému testování jsme navíc rovnou vyzkoušeli hru na dalším zařízení a mohli tak ověřit mimo jiné úspěšnost optimalizací (viz sekce 3.3.4).

## 5.2 Uzavřené testování

Když byla verze hry z naší strany dostatečně dokončená a odladěná, provedli jsme experimenty s širším okruhem lidí. Nejednalo se však o plně veřejné testování, pouze o předem sestavené experimenty v rámci malé skupiny důvěryhodných účastníků.

Celý průběh jsme rozdělili do dvou hlavních částí:

1. vyhodnocení srozumitelnosti ikonek kouzel (více v sekci 5.2.2),
2. obecný playtesting s vyhodnocením herního prožitku (více v sekcích 5.2.3 a 5.2.4).

Ještě před samotným provedením však bylo třeba zvážit, jakým způsobem by měly experimenty probíhat, a to především druhá část zahrnující obecný playtesting. Existuje celá řada GUR metod – jejich přehled je např. v knize *Game User Research* [71]. Pro náš konkrétní případ by stály za zvážení především dvě:

- Mohli bychom provádět experiment pro každého účastníka zvlášť, u hraní hry ho pozorovat a mluvit s ním o tom, co dělá a jak se u toho cítí. Tento způsob by byl pro obě strany dost časově náročný a riskovali bychom, že by se účastníci chovali jinak, když by byli pozorováni (tzv. *observer bias* [72]). Na druhou stranu bychom byli schopni posbírat více relevantních výsledků a skutečně porozuměli tomu, proč se účastníci chovali tak, jak se chovali.
- Nebo bychom mohli předem připravit instrukce, které by účastníci obdrželi a zcela nezávisle by je prováděli kdykoliv ve svém volném čase. Spoléhali bychom se pak na data čistě z herních analytik a dotazníků. Tento způsob by byl časově méně náročný a účastníci by hru hráli ve svém přirozeném prostředí, ale posbírali bychom tak mnohem méně dat, která by nám navíc neříkala, jaká za nimi byla motivace.

Jelikož samotná účast v experimentu byla na zcela dobrovolné bázi a byli jsme odkázáni čistě na ochotu zúčastněných, rozhodli jsme se vydat druhým směrem a účastníkům alespoň umožnit přívětivější podmínky. Vytvořili jsme tak dva Google formuláře (pro každou část jeden), které obsahovaly detailní instrukce provádějící účastníky celým procesem. Během hraní hry se pak vytvářely soubory se záznamy aktivity, které účastníci přiložili do formuláře, abychom je mohli analyzovat. Experiment jsme navíc sestavili dvojjazyčně (česky a anglicky).

### 5.2.1 Účastníci experimentu

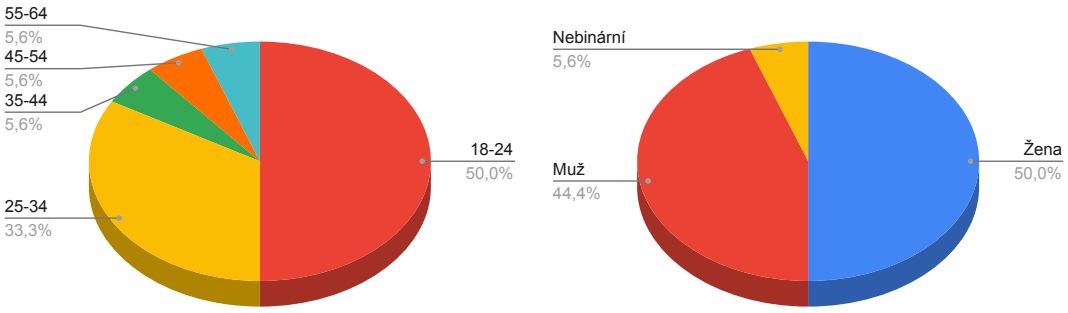
Pro účast v experimentech se nám podařilo získat poměrně různorodou skupinu 18 účastníků. Při jejich náboru jsme využili také prostředníky, díky čemuž téměř polovinu účastníků (8 z 18) autorka práce osobně neznala.

Úplně na začátku první části jsme účastníkům pomocí dotazníku položili několik základních otázek, abychom porozuměli složení skupiny. Na obrázku 5.1 vidíme rozdělení účastníků na základě věku (5.1a) a pohlaví (5.1b).

Jednotliví účastníci pak mají různé zkušenosti s hraním her obecně. Na obrázku 5.2 je graf znázorňující četnost různých platforem pro hraní her (5.2a), přičemž každý účastník měl zvolit všechny používané, a také průměrný počet hodin, který týdně stráví hraním her (5.2b).

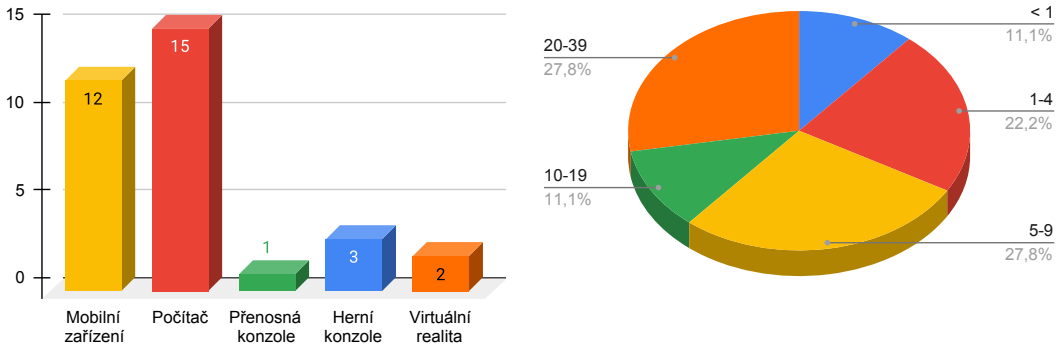
Kromě toho účastníci uvedli velkou rozmanitost herních žánrů, které hrají, avšak nikdo nezmínil závodní hry. Z odpovědí na otázku, jak moc jsou obeznámeni s hraním závodních her, pak skutečně vyplývá, že s nimi nemají příliš velké zkušenosti (viz graf na obrázku 5.3). Je tedy nutno poznamenat, že se tak jedná spíše o náhodně zvolený vzorek populace než přímo o cílovou skupinu naší hry.

Účastníky jsme také požádali, aby si zvolili nějakou přezdívku, pod kterou pak budou vystupovat v obou částech experimentu, abychom byli schopni spárovat



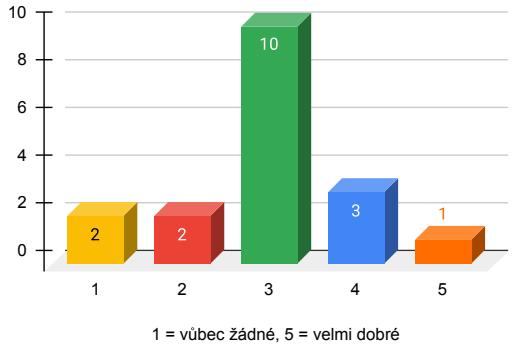

**(a)** Věkové rozdělení **(b)** Rozdělení dle pohlaví

**Obrázek 5.1** Rozdělení účastníků experimentu dle věku a dle pohlaví.




**(a)** Používané platformy pro hraní her **(b)** Průměrný počet hodin v týdnu strávený hraním her

**Obrázek 5.2** Četnost jednotlivých herních platforem mezi účastníky a rozdělení účastníků dle míry hraní her.




**Obrázek 5.3** Četnost odpovědí ohledně míry obeznámení s hraním závodních her.

odpovědi stejné osoby. Data, která jsme přiložili k této práci v elektronické příloze (podadresář `experimenty/data/`), jsme však zcela anonymizovali, aby nebylo možné jakkoliv identifikovat osoby s nimi spojené.

### 5.2.2 Srozumitelnost ikonek

V rámci první části experimentu bychom chtěli ověřit, jak srozumitelné jsou ikonky kouzel a jak dobře zachycují jejich efekty. V obchodě je sice možné si zobrazit popis kouzla včetně efektu a použití, ale během závodu už hráč vidí pouze ikonky. Ty by tedy měly jasně vyjadřovat, co dělají, aby si hráč nemusel pamatovat příliš mnoho informací nazpaměť.

V naší hře nebude tolik různých kouzel a odpovídajících ikonek jako v některých hrách žánru RPG. Navíc si hráč bude moci vybavit do závodu vždy maximálně 4 kouzla, takže spíše nezapomene jejich efekty. Obecně je však dobré uvažovat nad tím, jak moc srozumitelné ikonky jsou a jak moc hráči usnadňují hraní.

Při testování jsme vyšli z konceptu tzv. *Form Follows Function*, který popsala Celia Hodent ve své přednášce na Game Developers Conference v roce 2015 [73]. Ten říká, že by vše mělo fungovat tak, jak to vidíme. Hráči by neměli muset nejprve odvodit, co vlastně ikonka znamená, a pak si to ještě zapamatovat. Ikonka musí mluvit sama za sebe. Kromě samotného vysvětlení pak Celia Hodent poskytla také příklad konkrétního způsobu testování, který jsme převzali, ale navíc jsme přidali další dvě části. Experiment tak probíhal následovně:

1. Účastníci se seznámili s kontextem tak, že si nejprve zahráli speciálně upravenou verzi hry zcela bez kouzel. Měli projít celým úvodním tutoriálem, absolvovat svůj první závod a v rámci tutoriálu se seznámit s obrazovkou přehledu hráče.

2. V první části (převzaté z odkazovaného zdroje) jsme účastníkům postupně ukázali v náhodném pořadí jednotlivé ikonky kouzel, přičemž měli za úkol popsat jejich formu (tj. jako co ikonka vypadá, co na ní vidí, co znázorňuje) a funkci (tj. co podle nich kouzlo s takovou ikonkou dělá). Tím můžeme ověřit, jestli jsou dosavadní ikonky dostatečně dobré.

3. Ve druhé části jsme toto obrátili. Účastníkům jsme postupně zobrazili popis nějakého kouzla ve hře a jejich úkolem bylo napsat, jak by mohla vypadat ikonka takového kouzla, co by měla obsahovat. Tím můžeme získat náměty pro ikonky, u kterých bychom z první části vyhodnotili, že potřebují změnu.

4. Třetí část pak spojuje dohromady dvě předchozí. Účastníkům jsme zobrazili naráz všechny ikonky a všechny popisy kouzel. Jejich úkolem bylo pospojovat dvojice, které si myslí, že patří k sobě. Tím jsme simulovali situaci, která zhruba odpovídá té ve hře. Hráči si mohou matně pamatovat, jaká kouzla jsou ve hře, ale pak by měli být schopní toto namapovat na správnou ikonku. Současně nám to umožňuje identifikovat často zaměňované dvojice, a tím nejednoznačnosti ikonek.

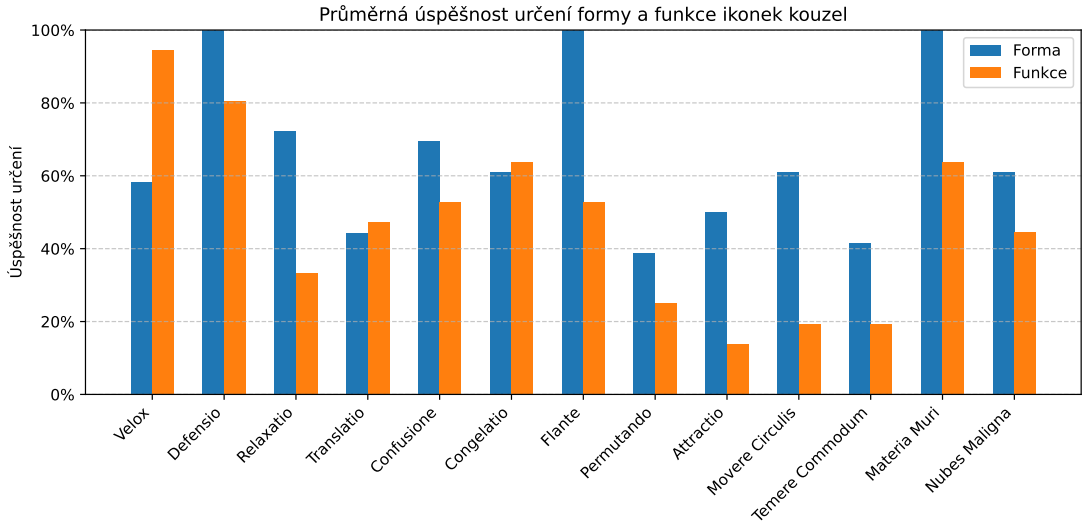
Během experimentu jsme tak použili build hry bez kouzel a poté separátní aplikaci se všemi zbývajícími částmi experimentu. Obojí se nachází v elektronické příloze ve složce `experimenty/build/`.

#### Vyhodnocení

Posbíraná data a jejich vyhodnocení najdeme v elektronické příloze v souboru `Srozumitelnost_ikonek.xlsx` (ve složce `experimenty/vyhodnoceni/`).

V první části experimentu jsme pro každou odpověď účastníka vyhodnotili, zda je správná, částečně správná, nebo nesprávná. Následně jsme pro každou ikonku určili procentuální úspěšnost zvlášť pro formu a zvlášť pro funkci (výsledky jsou pak na obrázku 5.4). Z toho můžeme vyčíst, jaké změny je třeba provést:

- Pokud již forma není jednoznačná, bylo by dobré vylepšit ikonku, jelikož vede i na špatné určení funkce.
- Pokud je forma zřejmá, ale funkce je určená velmi nesprávně, pak je třeba ikonku nahradit zcela novou.
- Pokud je funkce ikonky určená nesprávně, ale při spojování dvojic (ve třetí části) ji účastníci přiřadili ihned správně, pak není změna ikonky tak kritická, jako kdyby u ní často chybovali také u spojování.




**Obrázek 5.4** Průměrná úspěšnost určení formy a funkce jednotlivých ikonek kouzel.

Bohužel se z výsledků ukázalo, že poměrně dost účastníků mělo problém správně pochopit, co přesně znamená forma a co by k ní měli napsat. Velmi často tak popisovali spíše abstraktní koncepty již na hranici funkce. Navíc jsme byli při vyhodnocování správnosti poměrně přísní a kladli důraz na přesnější popis. Čím obecněji totiž účastníci ikonky popsali, tím větší byla šance, že se víceméně trefili, oproti těm, kteří se snažili funkci popsat opravdu pečlivě.

Uvedeme nyní příklad několika závěrů, které jsme učinili na základě výsledků první části:

- *Velox* (obr. 5.5a) – U určení funkce je vysoká úspěšnost, ale forma má úspěšnost nižší. V tomto případě by však nemuselo být třeba ikonku zcela nahrazovat, protože je problém spíše v nepochopení, co přesně by účastníci měli u formy psát (často uváděli obecně „rychlost“ či „zrychlení“, což jsme vyhodnotili pouze jako částečně správné).
- *Relaxatio* (obr. 5.5b) – Forma nebyla určena příliš špatně, ale u funkce jsme zaznamenali jen velmi nízkou úspěšnost. Velmi pravděpodobně je to způsobené nejasnou ikonkou, u které není zřejmé, co reprezentuje. Nejlepší by tedy bylo navrhnout zcela novou ikonku, která lépe zachytí funkci.

- *Flante* (obr. 5.5c) – Určení formy je naprosto bezchybné, avšak funkce už stoprocentní není. Obvykle ji účastníci spojili s nějakým vanutím větru, ale přiřazují mu různé efekty. Nejčastějším omylem je domněnka, že se tím zrychlí samotný hráč. Ikonka by se tedy mohla upravit, aby bylo zřejmé, že kouzlo ovlivňuje druhé, ale není to tak nutné (asociace s větrem a odvanutím funguje dobře a pak už je to jen o detailech).

- *Permutando* (obr. 5.5d) – Již určení formy dělalo účastníkům problém, ale opět se jednalo spíše o případ nepochopení toho, co by se vlastně mělo psát (odpovědi dávaly smysl, ale opět na hranici funkce, např. „restart“ či „opakování“). U určení funkce pak nebyli příliš úspěšní, často ji zaměnili s nějakým točením postavy nebo resetem/opakováním. Bylo by tedy vhodné ikonku upravit, aby se jasně zachytilo, že se jedná o prohazující se dva různé závodníky.

- *Attractio* (obr. 5.5e) – U formy velká část účastníků nepoznala, že se jedná o bonus, což pak způsobilo velkou míru chyb také u určení funkce. Může to být volbou modré barvy, kterou doposud u bonusů neviděli. Mohli bychom alespoň zkusit změnit barvu na zelenou.

**(a)** Velox **(b)** Relaxatio **(c)** Flante **(d)** Permutando **(e)** Attractio

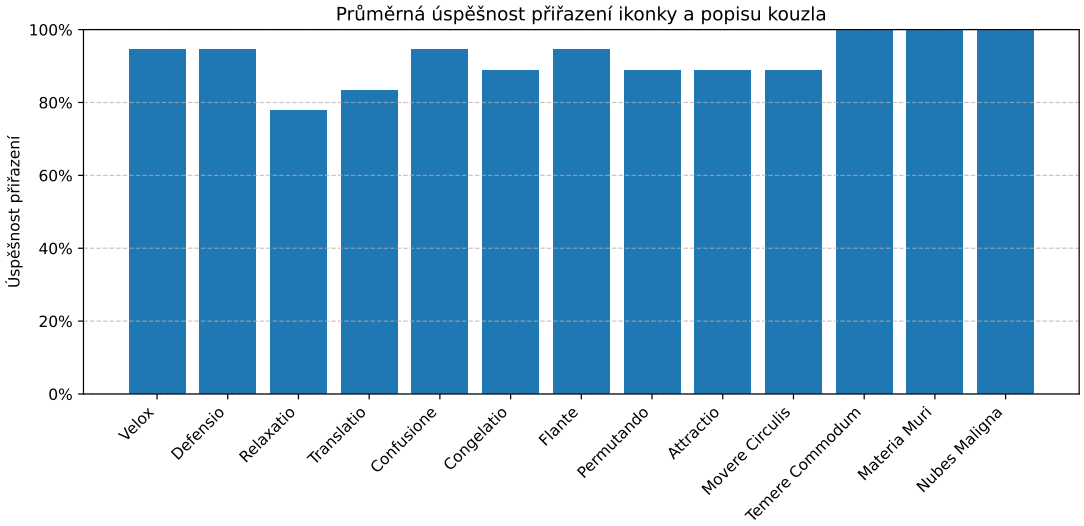
**Obrázek 5.5** Ikonky některých kouzel tak, jak byly použity v experimentu.

Současně jsme pro každou ikonku změřili, kolik času průměrně účastníci strávili odpovědí. Ukázalo se, že obvykle nejvíc času strávili u ikonek, u kterých byla také nižší úspěšnost určení funkce. Odpovídající ikonky by tedy mohlo být vhodné vylepšit, aby bylo určení formy snazší a funkce z nich mnohem lépe vyplývala. Je však třeba brát v úvahu také to, že účastníci hráli hru jen velmi krátce a nemuseli tak být dostatečně dobře obeznámeni s kontextem hry. Tím mohou být výsledky dosti zkreslené, protože může být o to těžší odvodit funkci kouzel.

Druhá část nám pak poskytuje lepší vhled do uvažování jednotlivých hráčů. Pokud jsme z první části vyhodnotili, že je třeba některou ikonku změnit, z odpovědi ve druhé části můžeme odvodit formu, kterou by mohla lépe chápat větší množina lidí. Příkladem by mohlo být doplnění siluety závodníků pro ujasnění funkce kouzel *Permutando* (obr. 5.5d) a *Translatio* (obr. 5.7b) nebo použití magnetu místo šipky pro naznačení přitažení objektu u kouzel *Attractio* (obr. 5.5e) a *Movere Circulis* (obr. 5.7c).

Ačkoliv formu a funkci se často nedařilo správně určit, ve třetí části měla většina účastníků 100% úspěšnost. Dá se tedy říct, že ačkoliv ikonky samy o sobě nejsou dostatečně zřejmé, v kontextu hry by mohly fungovat dobře. Na obrázku 5.6 pak vidíme úspěšnost zvlášť pro jednotlivá kouzla.

Uvedeme nyní příklady některých záměn, kterých se účastníci dopustili:




**Obrázek 5.6** Průměrná úspěšnost přiřazení ikonky k popisu kouzla.

- K ikonce kouzla *Flante* (obr. 5.5c) přiřadili popis efektu *Velox*. To je pravděpodobně výsledkem domněnky, že *Flante* působí na samotného sesílajícího, jak jsme popsali již výše u výsledků první části.
- Ikonku kouzla *Defensio* (obr. 5.7a) spojili s efektem *Relaxatio*. Obě kouzla mají podobný efekt, ale jedno se používá preventivně, druhé až ex post. Ikonky by tedy měly tuto skutečnost jasně rozlišit. Měla by pomoct už samotná změna *Relaxatio*, která vyplynula již z první části.
- Docházelo k časté záměně *Translatio* (obr. 5.7b) a *Permutando* (obr. 5.5d). Mohlo by tak být vhodné pomocí ikonek rozlišit, že se *Translatio* aplikuje pouze na sesílajícího, zatímco *Permutando* vyloženě prohazuje s jiným závodníkem.

**(a)** Defensio **(b)** Translatio **(c)** Movere Circulis

**Obrázek 5.7** Ikonky některých kouzel tak, jak byly použity v experimentu.

### Závěr

Prozatím jsme ikonky nijak neupravovali a v přiložené verzi hry jsou tak přesně ty, se kterými se účastníci setkali v experimentu (včetně několika ikonek pro kouzla, jejichž implementace se teprve plánuje). Výše sepsaná pozorování bychom však rádi v budoucnu aplikovali. Poté bychom mohli na provedené experimenty navázat další iterací již se změněnými ikonkami a novými účastníky. Měli bychom jim však dát více času na seznámení se s hrou a jejími prvky a poté důkladněji vysvětlit,

co je myšleno formou ikonky. Navíc by bylo možné provést A/B testy, abychom porovnali, zda dávají nové ikonky skutečně lepší výsledky oproti těm starým. Dále bychom se mohli inspirovat některými funkcemi, které účastníci chybně uvedli v první části, a implementovat je jako nová kouzla ve hře.

### 5.2.3 Herní prožitek

Ve druhé části experimentu jsme se zaměřili na obecný playtesting, který jsme chtěli rovnou spojit s vyhodnocením herního prožitku. Rozhodli jsme se použít tzv. *GUESS* dotazník, který byl popsán v článku *The Development and Validation of the Game User Experience Satisfaction Scale* [74]. Tento dotazník poskytuje seznam otázek, které umožňují ohodnotit zážitek ze hry v rámci devíti různých faktorů. Rozhodli jsme se však dva z nich vynechat, jelikož nejsou pro naši hru relevantní a v odkazovaném článku se přímo uvádí možnost je vynechat. Jedná se o Narratives (jelikož hra není založena na narativu) a Social Connectivity (jelikož hra v současné verzi neumožňuje hru více hráčů).

Znění některých otázek jsme však poupravili, aby více zachycovaly jednorázovost tohoto experimentu (např. „I spent more time playing the game than I have planned“ místo „I tend to spend more time playing the game than I have planned“. K převzatým otázkám jsme navíc vytvořili český překlad. Výsledná podoba všech otázek je pak v souboru `dotaznik-otazky.md` v elektronické příloze této práce (ve složce `experimenty/`).

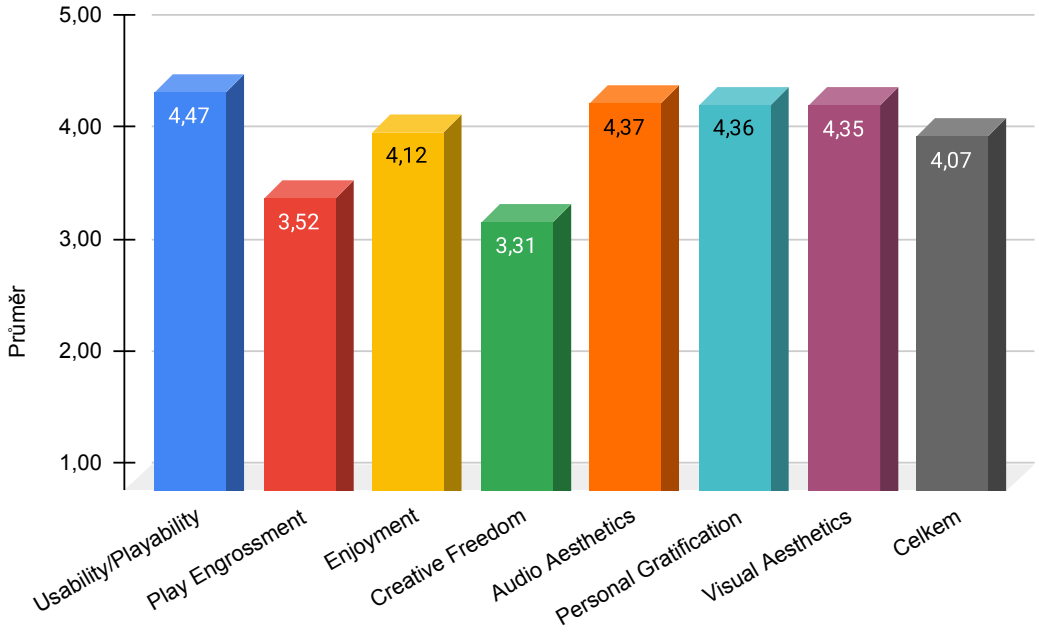
Účastníci si nejprve měli zahrát hru (použitý build se nachází ve složce `experimenty/build/` elektronické přílohy) po dobu alespoň 30 minut a následně odpovědět na všechny otázky. Vyjadřovali v nich míru svého souhlasu s určitým tvrzením pomocí Likertovy škály [75] s 5 možnostmi (tj. naprosto nesouhlasím, nesouhlasím, neutrální postoj, souhlasím, naprosto souhlasím). Posbíraná data jsme následně vyhodnotili postupem doporučeným v článku, tj. spočítali jsme průměr položek v jednotlivých faktorech. Navíc jsme však určili také směrodatnou odchylku a rozptyl. Výslednou tabulku najdeme ve složce `experimenty/vyhodnoceni/` v elektronické příloze práce. Na obrázku 5.8 pak vidíme zachycené průměry jednotlivých faktorů.

Museli jsme však vyškrtnout výsledky jednoho z účastníků (označeného „Účastník 0“), protože po celou dobu hrál verzi hry z první části experimentu, tedy zcela bez kouzel, čímž by mohly být odpovědi zkreslené.

Nyní si krátce okomentujeme výsledné hodnoty jednotlivých faktorů:

- *Creative Freedom* – V tomto faktoru se dosáhlo úplně nejhoršího výsledku. Na druhou stranu to není nic nečekaného, jelikož naše hra není nijak zvlášť zaměřená na kreativitu. Spíše je tedy překvapivé, že je výsledek stále v pozitivní části škály.

- *Play Engrossment* – Tento faktor by měl popisovat, jak dobře dokáže hra udržet hráčovu pozornost. Z výsledku se tedy zdá, že v tom naše hra není příliš úspěšná. Musíme však brát na vědomí, že jak jsme určili již v sekci 5.2.1, účastníci experimentu netvoří skutečnou cílovou skupinu a z prvotního průzkumu je zřejmé, že zpravidla nehrají závodní hry. Dva účastníci dokonce explicitně uvedli, že nejsou fanoušky závodních her. Mohli bychom tedy zvážit, zda experiment nezopakovat s lidmi, kteří mají žánru blíže. Teprve




**Obrázek 5.8** Průměrné hodnocení jednotlivých faktorů GUESS.

poté promyslet nutné úpravy, pokud by se ukázaly jako potřebné. Ani současné skóre však není celkově špatné.

- *Enjoyment* – Tento faktor má o poznání vyšší hodnocení než předchozí. Zdá se tedy, že ačkoliv nejsou hráči zcela ponoření do hry, celkově si ji užívají. To je velmi dobrý výsledek, zohledníme-li náhodnost skupiny účastníků.

- *Personal Gratification* – Tento faktor je úzce svázaný s předchozími dvěma a zachycuje pocit dosažených cílů, hráčovu motivaci pokračovat ve hře a touhu uspět. Z vysokého skóre je opět patrné, že účastníci byli motivováni ve hře, dávala jim pocit osobního uspokojení.

- *Audio Aesthetics* a *Visual Aesthetics* – Tyto faktory vyhodnocují, jak dobře hra zní a vypadá. V obou se přitom podařilo již nyní získat poměrně vysoké hodnocení. Do budoucna bychom chtěli obojí ještě dále vylepšit. Zdá se tedy, že pokud se budeme držet dosavadního uměleckého směru, neměli bychom nic pokazit.

- *Usability/Playability* – Tento faktor zachycuje celkovou uživatelskou přívětivost z hlediska rozhraní, ovládání, postupného seznámení se se vším a jasně definovaných cílů ve hře. Dosáhli jsme v něm přitom vysokého hodnocení, díky čemuž víme, že byli účastníci schopní užít si hru víceméně bez zábran. To je pak velmi důležitý základ pro všechny ostatní aspekty.

Průměrné hodnocení všech kategorií je pak 4,07. Na konci dotazníku jsme navíc přidali doplňující otázku, kolika hvězdičkami od 1 do 5 by hru ohodnotili, a výsledný průměr byl 4,0. Obě hodnoty se tedy téměř shodují a můžeme je tak považovat za celkové hodnocení hry, což je velmi dobré, vezmeme-li v úvahu, že se stále jedná pouze o prototyp.

### Závěr

Ačkoliv se tedy účastníkům pravděpodobně nepodařilo dosáhnout přímo stavu flow (viz sekce 2.1), který by odpovídal faktoru *Play Engrossment*, hru si zřejmě vcelku užívají (dle faktoru *Enjoyment*) a dostatečně je motivuje (na základě faktoru *Personal Gratification*). Když k tomu přidáme dobré hodnocení také ve faktorech týkajících se audiovizuální stránky a použitelnosti, můžeme říci, že celkově měla hra vcelku úspěch a mohlo by tedy dávat dobrý smysl v jejím vývoji pokračovat a nadále ji vylepšovat.

Hodnocení konkrétních faktorů nám pak dává lepší představu o tom, na které části hry se zaměřit jako první. Mohli bychom se pokusit dát hráčům větší prostor pro kreativitu. Např. bychom mohli rozšířit nabídku vzhledu postavy a přidat možnost postupně si vzhled vylepšovat, třeba také pomocí doplňků. Případně bychom mohli umožnit měnit vzhled koštěte nebo kouzelnické hůlky (pokud by ji v budoucnu závodníci měli). Dále bychom se měli zaměřit na udržení hráčovy pozornosti během hraní. Mohlo by pomoci vylepšit přizpůsobování schopností soupeřů, aby hráč neměl možnost příliš polevit a závody tak byly napínavější. Především bychom měli zvětšit rozsah jejich dovedností, aby mohli být jak extrémně špatní, tak extrémně dobří. Také přidání dalších zvukových efektů by mohlo pomoci hráče více vtáhnout do hry.

Pomocí dotazníku se nám navíc podařilo experimentálně ověřit dva z našich požadavků kladených na hru. Nyní víme, že dle účastníků audio dostatečně odpovídá žánru a upevňuje zážitek ze hry (požadavek **P5**) a že uživatelské rozhraní je přehledné a intuitivní (**P6**).

### Rozdíly v provedení

V původním článku [74] věnovaném vytvoření a validaci škály GUESS se také uvádí doporučení pro její použití. My jsme však v některých případech postupovali odlišně. Uvedeme tedy a především zdůvodníme tyto rozdíly v provedení.

Škála by údajně měla být užitečná pro srovnání různých her stejného žánru, her ze stejné série nebo různých verzí té samé hry. My jsme ji však využili spíše jen jako zdroj dobrých otázek, které pokryjí různé aspekty herního zážitku a umožní nám tak co nejúplněji ohodnotit naši hru. Naší motivací byla především možnost identifikovat její silné a slabé stránky.

V článku se dále uvádí, že je možné vynechat faktory *Narratives* a *Social Connectivity*, ale je třeba provést další výkum pro vyhodnocení validity takto upravené škály. Konkrétně pro naši hru byly tyto faktory irelevantní, ale bohužel se nám nepodařilo zjistit, zda byl skutečně proveden navazující výzkum. Použili jsme tak redukovanou škálu i s rizikem, že nemusí být validována. Jelikož ji však stejně nepoužíváme doporučeným způsobem, validace by pro nás neměla být tak zásadní.

Dále je doporučeno vypisovat jednotlivé položky v náhodném pořadí a vždy po pěti na jedné stránce, aby je účastníci viděli všechny současně a nemuseli mezi nimi přecházet. Položky z různých faktorů by tak měly být namíchané, což by pravděpodobně umožnilo detekovat konzistentnost odpovědí. Pro tvorbu dotazníku jsme však zamýšleli použít Google Forms či Microsoft Forms, ve kterých není možné takový průběh zajistit. Současně jsme nechtěli vytvářet zcela vlastní aplikaci (podobně jako pro ověření srozumitelnosti ikonek), protože by to bylo již příliš

časově náročné. Nechali jsme tedy otázky uspořádané do skupin dle faktoru, ale v rámci skupiny jsme otázky zobrazovali v náhodném pořadí.

Nakonec jsme také použili Likertovu škálu s 5 možnostmi, jež jsme zmínili již v úvodu této sekce, ačkoliv v článku popisují možností 7. Byli jsme totiž limitováni šířkou viditelné oblasti Google formuláře a nechtěli jsme, aby se museli účastníci neustále posouvat do stran. Tím by mimo jiné hrozilo, že by si ani nevšimli skrytých možností. Raději jsme se proto rozhodli pro méně detailní škálu.

### 5.2.4 Herní analytiky

Během obou částí experimentu, zatímco účastníci hráli hru (ve druhé části minimálně po dobu 30 minut), jsme navíc sbírali data o událostech ve hře formou herních analytik. To nám totiž umožňuje analyzovat jak chování účastníků, tak hry samotné. Pomáhají nám získat odpovědi na různé otázky a také detekovat problémy ve hře.

#### Sběr a zpracování dat

Ještě než jsme však analytiky přidali do hry, museli jsme se rozhodnout pro jeden konkrétní způsob řešení. Existuje totiž celá řada již hotových nástrojů, příkladem mohou být Unity Gaming Services Analytics [76] nebo GameAnalytics [77]. Obvykle se formou SDK přidají k projektu a pak poskytují nějaké rozhraní pro zaznamenání události s případnými parametry. Jejich zprovoznění však někdy nemusí být úplně přímočaré.

Nakonec jsme se rozhodli pro zcela vlastní řešení, jehož implementaci jsme popsali již v sekci 4.10.7. Díky tomu jsme nad sběrem dat měli úplnou kontrolu, získali jsme skutečně lineární záznam aktivit jednotlivých účastníků, ale především jsme mohli pro každého účastníka snadno spárovat herní data s odpověďmi z dotazníků, které pak mohou doplnit další kontext. Kdybychom se však v budoucnu rozhodli využít jiné řešení, stačilo by snadno změnit implementaci třídy `Analytics` (konkrétně její `LogEvent()` metody).

Po ukončení hry se pak potřebné soubory s daty překopírovaly do složky na ploše, ze které je mohli účastníci snadno nahrát do Google formuláře s instrukcemi k průběhu experimentu. Veškerá posbíraná data se nachází ve složce `experimenty/data/` v elektronické příloze práce, konkrétně soubory pojmenované `UcastnikXX-Events-Part2.data`, které jsme použili pro tuto část. Jedná se o obyčejný textový formát obsahující jednotlivé události na separátních řádcích. Každá událost má pak specifický formát, který umožňuje snadné strojové zpracování:

```
Datum a čas | Aktuální scéna | Kategorie události | Data události
```

#### Vyhodnocení

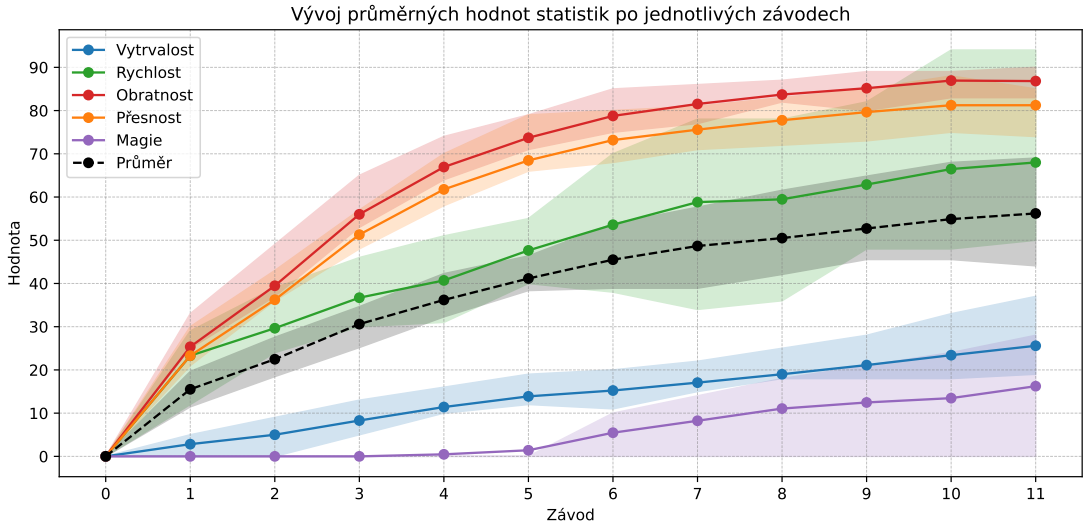
Ještě před zahájením experimentu jsme si promysleli otázky, na které bychom chtěli získat odpovědi. Na základě toho jsme pak vypisovali konkrétní relevantní události. Po skončení experimentu jsme data ze souborů předzpracovali a následně z nich vytvořili grafy (zdrojový kód i výsledné soubory se nachází v příloze ve složce `experimenty/zpracovani/`). Stejně jako v případě vyhodnocení herního prožitku

jsme však museli vynechat data od jednoho z účastníků (označeného „Účastník 0“), protože pocházela z nesprávné verze hry (bez kouzel).

V následujících sekcích uvedeme příklady zpracovaných dat a otázek, na které jsme pomocí nich hledali odpovědi. Jelikož hráči během hry ve smyčce absolvují závod a pak se dostanou do obrazovky s přehledem, je herní čas přirozeně rozdělen na části. Pokud jsme tedy chtěli zaznamenat do grafu průběh něčeho v čase, obvykle jsme to provedli v diskrétních krocích dle jednotlivých závodů. V případě, kdy jsme takto zaznamenávali data agregovaná přes všechny účastníky, uvažovali jsme pouze nejvyšší počet závodů, které absolvovali úplně všichni, tj. 11. Navíc jsme při výpočtech obecně používali pouze data z poslední rozehrané hry, i když někteří účastníci zahájili novou hru vícekrát.

### Statistiky

Velmi důležitou otázkou je, jak se vyvíjejí hodnoty jednotlivých statistik v průběhu hry, tj. především, jak rychle rostou a jestli nedochází k náhlým výkyvům. Na obrázku 5.9 vidíme graf zachycující průměrné hodnoty statistik po jednotlivých závodech. Spočítali jsme je jako průměr přes všechny účastníky a navíc jsme zobrazili také 1.–3. kvartil.



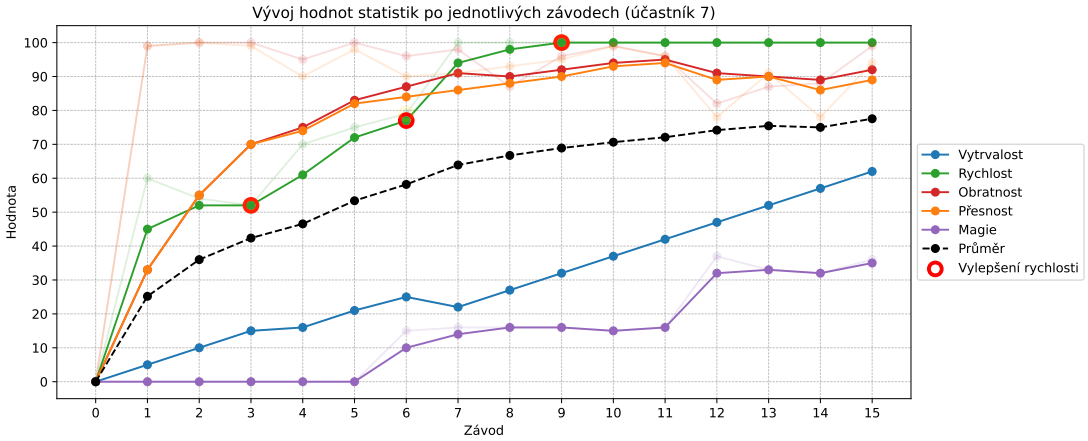

**Obrázek 5.9** Vývoj průměrných hodnot statistik po jednotlivých závodech.

Z grafu je patrné, že statistiky *obratnosti*, *přesnosti* a *vytrvalosti* se příliš neliší napříč všemi účastníky. Je to zřejmě dané tím, že nezávisí na žádných vnějších vlivech, pouze na samotných schopnostech. V případě *rychlosti* jsou pak rozdíly znatelnější, protože je ovlivněná také úrovní vylepšení koštěte. Záleží tedy, zda se účastník rozhodl si dané vylepšení zakoupit a také kdy. Podobně zřetelný rozdíl je u *magie*, jelikož si různí účastníci mohli pořídit kouzla v různých okamžicích hry (někteří se dokonce za celou dobu hraní nedostali ani k jednomu).

Můžeme však říci, že obecně mají statistiky tendenci se postupně vylepšovat, stejně tak jejich průměr. *Obratnost* a *přesnost* velmi brzy narostou do vysokých hodnot a v nich se pak drží. *Rychlost* narůstá také vcelku rychle, obzvlášť když může být podpořena vylepšením. *Vytrvalost* roste pomalu, což je dané způsobem jejího výpočtu (viz sekce 2.6), aby omezovala celkový průměr statistik a určovala

tak jistý minimální počet závodů, které musí i nejlepší hráč absolvovat před dokončením hry. Jakmile si účastník zpřístupní kouzla, zdá se, že *Magie* začne růst vcelku rychle. Celkově se tedy zdá, že by mělo být možné jednou dosáhnout konce hry.

Na obrázku 5.10 pak vidíme vývoj hodnot statistik pro jednoho konkrétního účastníka. Zachycuje jak celkové hodnoty (tj. ty, které účastník vidí v přehledu hráče), tak poloprůhledně také ty okamžité (tj. ty, které se počítají na základě výkonu v závodě a teprve poté se kombinují s předchozími hodnotami pomocí váženého průměru, viz sekce 2.6).

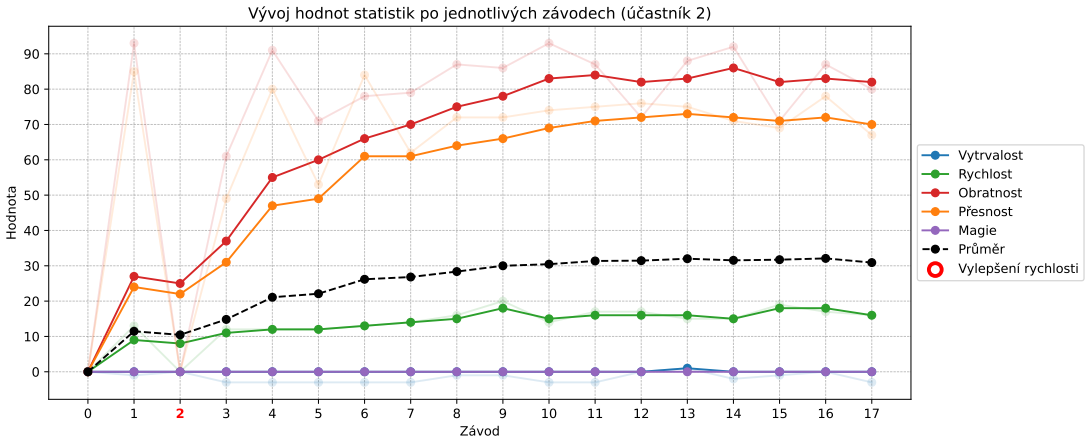



**Obrázek 5.10** Vývoj hodnot statistik po jednotlivých závodech (účastník 7).

Ačkoliv je možné dosáhnout velmi snadno vysokých hodnot *obratnosti* a *přesnosti*, vážený průměr zmírňuje celkový nárůst. U *rychlosti* pak vidíme vliv jednotlivých vylepšení koštěte, přičemž nakonec dokázal účastník dosáhnout maximální hodnoty. Na statistice *magie* se projevují zakoupení kouzel. Po pátém závodě si účastník koupil své první, proto mohla hodnota statistiky začít růst. Po jedenáctém závodě pak došlo k dalšímu zlomu, protože si účastník pořídil rovnou dvě další kouzla a začal je také používat v závodech. A jelikož se nakonec účastník umisťoval konzistentně na prvním místě, *vytrvalost* rostla lineárně.

Nakonec se podíváme ještě na jednoho dalšího účastníka. Na obrázku 5.11 vidíme zakreslený vývoj jeho statistik. V tomto případě se účastníkovi ve hře příliš nedařilo – umisťoval se zpravidla na posledním místě, takže se *vytrvalost* vůbec neměnila, neodemknul si ani jedno kouzlo a *magie* tedy zůstala na nule. V *rychlosti* pak dosáhl určitého maxima a nebyl schopný se posunout dál (navíc si koště ani jednou nevylepšil, protože nedostával odměny ze závodů). Naopak v *obratnosti* a *přesnosti* se mu poměrně dařilo. Účastník tedy zřejmě létal pomalu, možná se nedokázal sžít s ovládáním, ale tratě absolvoval vcelku úspěšně. Druhý závod přitom vzdal, proto je viditelný mírný pokles všech statistik.

Na základě těchto pozorování se tedy zdá, že pro běžného hráče by měl být konec hry dosažitelný a že se hodnoty statistik nemění s žádnými velkými skoky a obvykle celkově rostou, tudíž se hráč ve hře zlepšuje. Na druhou stranu by však mohlo být vhodné rozšířit rozsah obtížnosti hry tak, aby byla schopná se ještě více přizpůsobit také výrazně slabším hráčům a výsledkem nebylo to, že se zaseknou na místě, nýbrž že se budou jen velmi pomalu zlepšovat. Měli by být stále limitováni tím, že nebudou schopní dosáhnout hodnot statistik pro dokončení

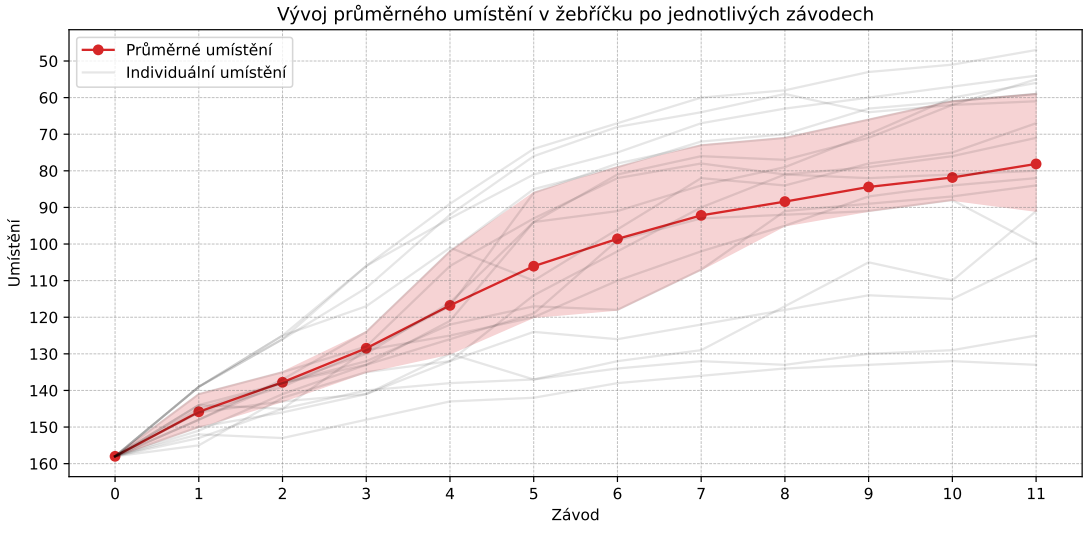



**Obrázek 5.11** Vývoj hodnot statistik po jednotlivých závodech (účastník 2).

hry, pokud se dostatečně nezlepší. Současně by však měli mít možnost se umístit na lepším místě, aby získali mince a mohli si něco zakoupit v obchodě, což by mohlo také lépe udržet jejich pozornost.

### Umístění v žebříčku

S vývojem statistik popsaným v předchozí sekci úzce souvisí také umisťování se v globálním žebříčku závodníků, které je závislé právě na aktuálním průměru statistik. Již jsme viděli, že statistiky mají obecně tendenci růst, tedy také umístění by se mělo vylepšovat. Pro úplnost však vidíme na obrázku 5.12 průměrné umístění v žebříčku po jednotlivých závodech. Šedě je pak zakreslen vývoj umístění zvlášť pro jednotlivé účastníky. Zpravidla je znatelný celkový trend zlepšení, i když se někdy umístění dočasně zhorší.




**Obrázek 5.12** Vývoj průměrného umístění v žebříčku po jednotlivých závodech.

### Množství many

Jako další bychom chtěli zodpovědět otázky týkající se sesílání kouzel v průběhu závodu. Nejprve by nás zajímalo, jak je to s množstvím many, jestli jí nemají hráči příliš mnoho, nebo naopak nedostatek. Pro jednotlivé účastníky jsme tak zakreslili do grafu, po jak dlouhou dobu vzhledem k celkovému trvání závodu měli zcela naplněnou manu.

Jelikož ze začátku mají jen jedno kouzlo, dá se předpokládat, že bude snáz docházet k naplnění many, protože ji nestihnou spotřebovávat. To však ničemu nevadí, protože alespoň nejsou hráči hned příliš omezováni a mohou se lépe obeznámit s kouzly. S více vybavenými kouzly by pak neměla být mana naplněná tak často. Z grafů se celkově zdá, že se tyto předpoklady naplnily a k situaci s maximálním množstvím many nedochází příliš často, obzvlášť v pozdějších závodech ne. Současně však takové situace nastávají, takže zřejmě není many ani úplný nedostatek. Podíváme se nyní na dva konkrétní příklady.

Na obrázku 5.13 vidíme výsledný graf jednoho z účastníků. Do pátého závodu nastoupil s jedním kouzlem, do šestého přidal druhé a nakonec od desátého dál měl naplněné všechny čtyři sloty. V posledních závodech nedosahoval úplného naplnění many, takže by se mohlo zdát, že many mohl být nedostatek. Z dalších zaznamenaných dat však vyplývá, že se běžně dostával na vyšší hodnoty a spíše jen dostatečně často sesílal kouzla, takže nedošlo k úplnému naplnění.


**Obrázek 5.13** Délka trvání období s maximálním množstvím many během jednotlivých závodů (účastník 11).

Další příklad je na obrázku 5.14. Vidíme, že daný účastník měl zcela naplněnou manu poměrně velkou část závodů. Z dalších dat jsme však zjistili, že totiž až v posledním závodě poprvé seslal kouzlo. Do té doby se vždy pouze naplnila mana a pak zůstala plná až do konce závodu. Poslední závod byl přitom výrazně delší než ostatní a účastník během něj seslal kouzlo jen dvakrát, proto je část závodu s maximální manou stále poměrně velká.

### Sesílání kouzel v závodě

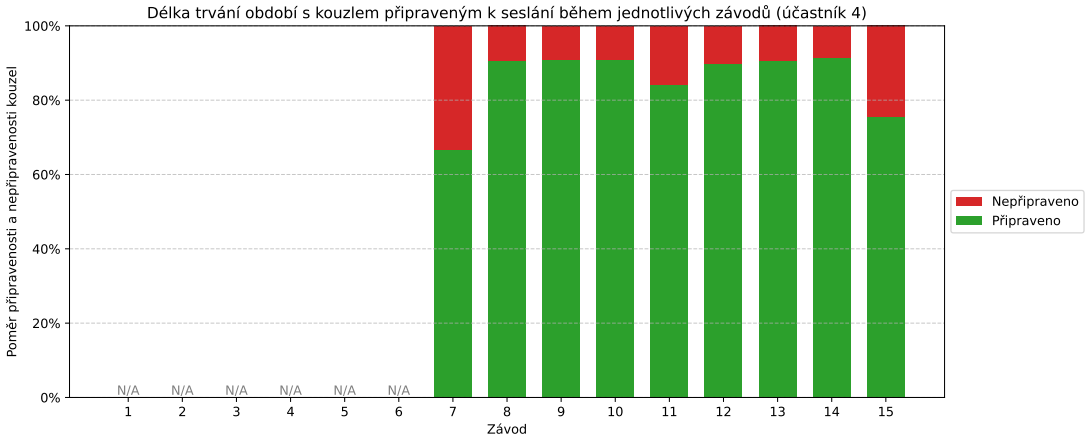
Abychom porozuměli sesílání kouzel ještě více, zaznamenali jsme navíc do grafu, po jak dlouhou dobu závodu existovalo nějaké kouzlo připravené k seslání (tj. nabité


**Obrázek 5.14** Délka trvání období s maximálním množstvím many během jednotlivých závodů (účastník 3).

a s dostatkem many). To by nám napovědělo, zda účastníci skutečně zvládali sesílat kouzla a používali je, kdykoliv mohli.

Překvapivým pozorováním bylo, že situace s připraveným kouzlem nastává po vcelku velkou část závodu pro většinu účastníků. Je však třeba uvědomit si, že taková situace pokaždé nutně neznamená, že účastník nezvládl kouzla sesílat. Někdy má třeba více vybavených kouzel a čeká na nějaké konkrétní. Nebo právě pro připravené kouzlo nemá využití, např. protože pro něj neexistuje vhodný cíl nebo má kouzlo velmi specifické použití.

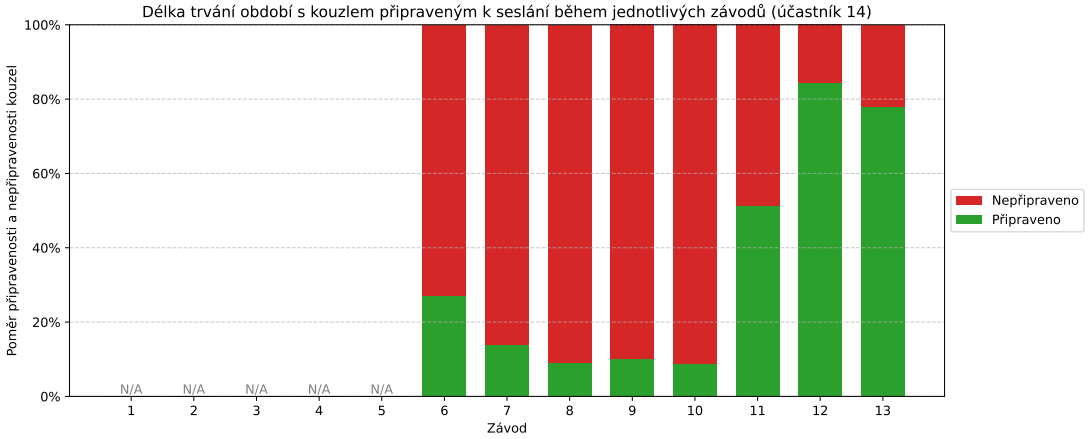
Podíváme se opět na konkrétní příklady. Na obrázku 5.15 vidíme hodnoty jednoho z účastníků. V každém závodě měl po většinu doby alespoň jedno připravené kouzlo. Je to však dané tím, že jako první měl kouzlo *Relaxatio*, které ukončuje negativní efekty. Není pro něj tedy využití, dokud nemá účastník také nějaké kouzlo ovlivňující soupeře, protože teprve pak ho také oni mohou ovlivňovat nazpět. Do devátého závodu si pak pořídil kouzlo *Velox*, které nevyužíval příliš často a navíc bylo neustále připravené také *Relaxatio*, proto se hodnoty příliš nezlepšily.




**Obrázek 5.15** Délka trvání období s kouzlem připraveným k seslání během jednotlivých závodů (účastník 4).

Další příklad je na obrázku 5.16. Účastník měl tentokrát užitečnější kouzla, ale

dostatečně je nevyužíval. Nejprve měl pouze *Velox*, které sesílal vcelku úspěšně. V posledních třech závodech však přidal další dvě kouzla a oproti prvnímu je nesesílal příliš často, takže byla po velkou část závodu stále připravena. Problém by mohl být v tom, že účastník nebyl dostatečně motivovaný používat také jiná kouzla, protože *Velox* je jednoduché na použití, levné a stoprocentně spolehlivé. Měli bychom tedy do budoucna zvážit, jak kouzla vyrovnat, aby byla srovnatelně žádoucí.




**Obrázek 5.16** Délka trvání období s kouzlem připraveným k seslání během jednotlivých závodů (účastník 14).

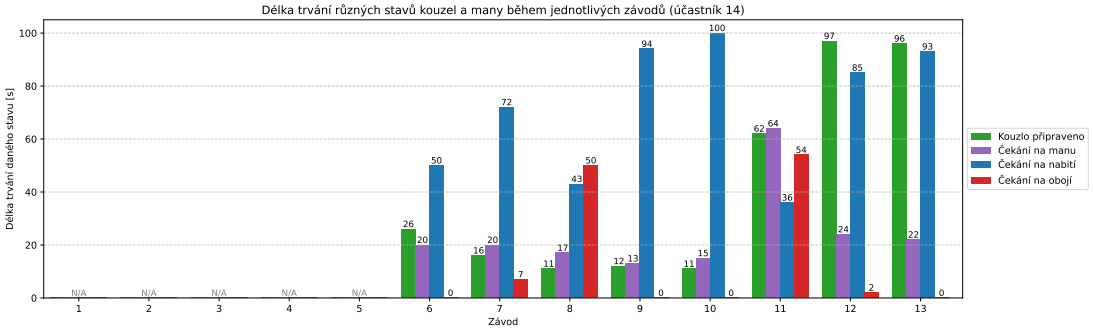
Mohli bychom jít ještě více do hloubky a podívat se podrobněji na délku trvání konkrétních stavů kouzel a many. Jednalo by se o následující:

- účastník má k dispozici kouzlo připravené k seslání,
- účastník má kouzlo, které je nabité, ale pro jeho seslání nemá dostatek many,
- účastník má kouzlo, na jehož seslání má dostatek many, ale kouzlo není nabité,
- účastník má kouzlo, které není nabité a současně pro jeho seslání nemá dostatek many.

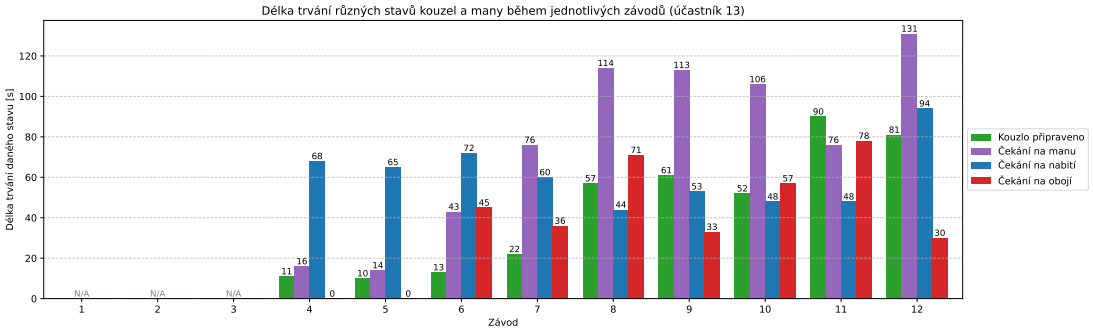
Důležité je si uvědomit, že všechny tyto stavy se navzájem nevylučují. Účastník může mít vybavených více kouzel naráz, ale pro platnost jednoho z uvedených stavů stačí existence jednoho takového kouzla, které ho splňuje.

Na obrázku 5.17 pak opět vidíme jeden konkrétní příklad. Jedná se o stejného účastníka jako na obrázku 5.16. Vidíme tedy mnohem lépe popisovaný případ, kdy ze začátku sesílal kouzlo často a nikdy ho nenechal připravené k použití příliš dlouho. Do závodu 11 pak přidal další dvě kouzla, která tolik nevyužíval, takže zůstávala připravená, ale současně často používal to původní a proto se tak dlouhou dobu čekalo na nabití, zatímco many byl většinu času dostatek, protože ji jediným kouzlem nestíhal spotřebovávat.

Typičtější případ pak vidíme na obrázku 5.18. Dokud měl účastník jen jedno kouzlo, převážnou dobu se čekalo na dobití kouzla. S více vybavenými kouzly má však více příležitostí k seslání, čímž také narůstá spotřeba many, která je tak v pozdějších závodech hlavním problémem.




**Obrázek 5.17** Délka trvání různých stavů kouzel a many během jednotlivých závodů (účastník 14).




**Obrázek 5.18** Délka trvání různých stavů kouzel a many během jednotlivých závodů (účastník 13).

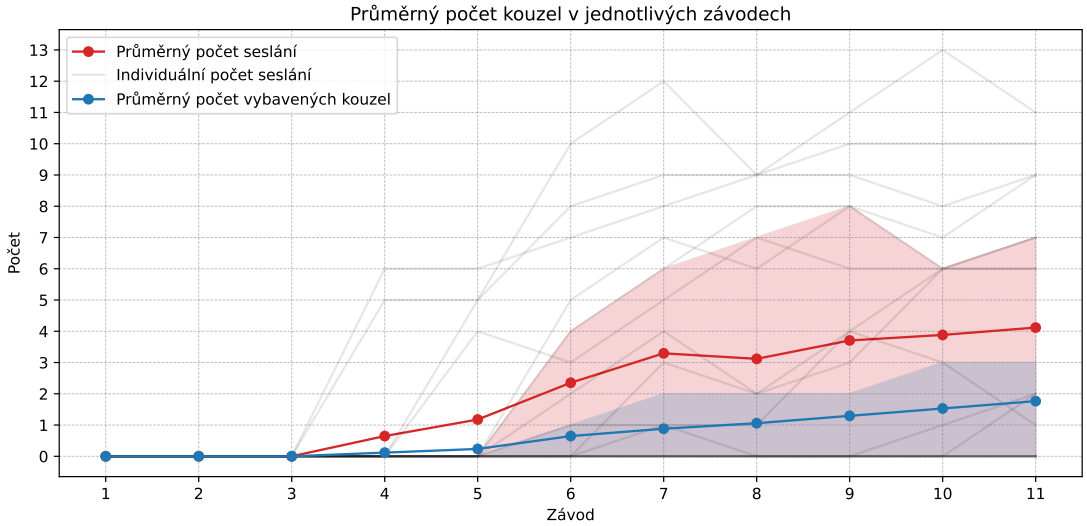
Toto platí obecně, že obzvlášť v pozdějších závodech je častější, že chybí mana, než že není kouzlo nabité. Mohli bychom se pokusit kouzla více vyvážit a např. snížit cenu seslání a naopak navýšit dobíjecí dobu. Nechceme ale hráče příliš omezovat již ze začátku, když má pouze jedno kouzlo a závody jsou krátké. Změna parametrů by tedy sice stála za zvážení, ale měla by se provést velmi opatrně.

Na závěr ještě na obrázku 5.19 vidíme graf znázorňující průměrný počet vybavených kouzel a průměrný celkový počet seslání kouzel v jednotlivých závodech. Šedě jsou navíc zakresleny počty seslání pro jednotlivé účastníky. Z grafu jsou zřejmé poměrně velké rozdíly hodnot mezi účastníky. Je to dané především tím, že pouze 10 účastníků z celkových 17 si během znázorněných 11 závodů zakoupilo alespoň jedno kouzlo. Obecně se však zdá, že počet seslaných kouzel zpravidla roste, což koresponduje s tím, že si postupně kupují a vybavují více kouzel a tratě se postupně prodlužují kvůli rostoucí statistice *vytrvalosti*.

### Využití kouzel

Doposud jsme sice získali vhled do samotného sesílání kouzel během závodu, avšak mohlo by být zajímavé také zjistit, která kouzla hráči skutečně používají a která si třeba jen vyzkouší, ale používat přestanou. Pak bychom věděli, jestli se některá kouzla nezdají zbytečná nebo jestli by nebylo vhodné je upravit tak, aby byla pro hráče lákavější.

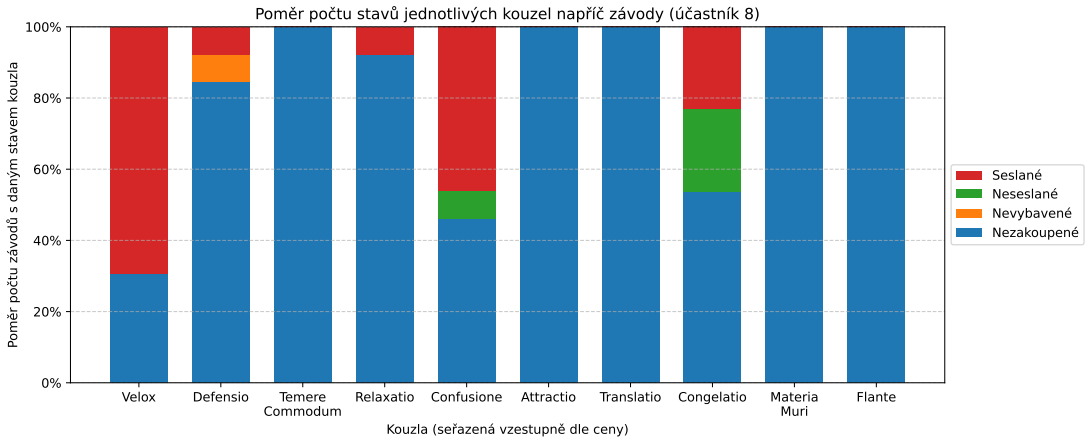
Pro každého účastníka jsme tak vytvořili graf, ve kterém jsme pro každé kouzlo zachytili poměr počtu závodů, kdy bylo kouzlo v jednom ze čtyř různých stavů:




**Obrázek 5.19** Průměrný počet kouzel v jednotlivých závodech.

- nezakoupené,
- zakoupené, ale nevybavené,
- zakoupené a vybavené, ale neseslané,
- zakoupené, vybavené a také seslané.

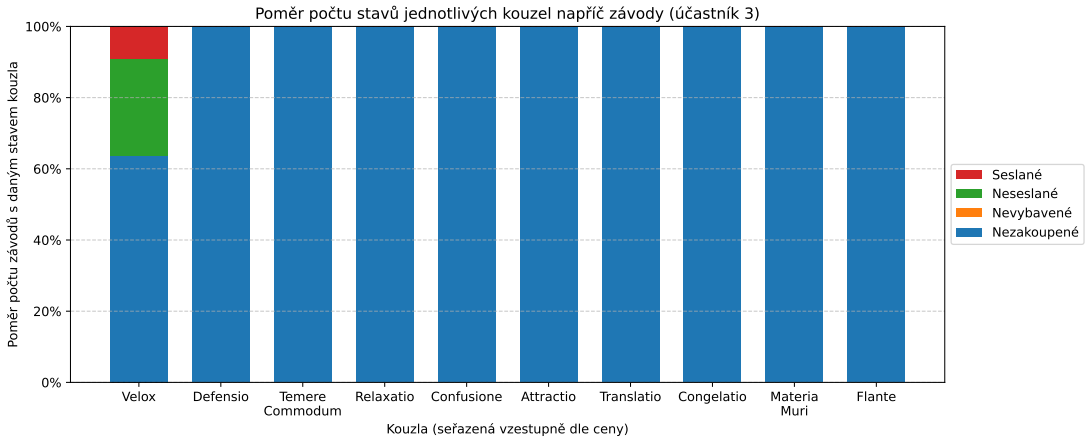
Na obrázku 5.20 vidíme výsledný graf pro jednoho z účastníků. Je z něj zřejmé, že účastník si zakoupil celkem 5 různých kouzel a zpravidla je také v závodech používal. Kouzlo *Defensio* nebylo po nějakou dobu vybavené, protože ho nahradilo kouzlo *Relaxatio* (hráči si mohou vybavit maximálně 4 kouzla a tato dvě mají velmi podobné efekty). *Confusione* a *Congelatio* pak v některých závodech nebyla seslána, což může být způsobeno tím, že pro ně musí existovat vhodný cíl (soupeř v dostatečně blízkém a viditelném okolí) a že mají podobné využití a rozdílnou cenu, takže mohl daný účastník v určitém okamžiku preferovat jedno z nich.




**Obrázek 5.20** Poměr stavů jednotlivých kouzel napříč všemi závody (účastník 8).

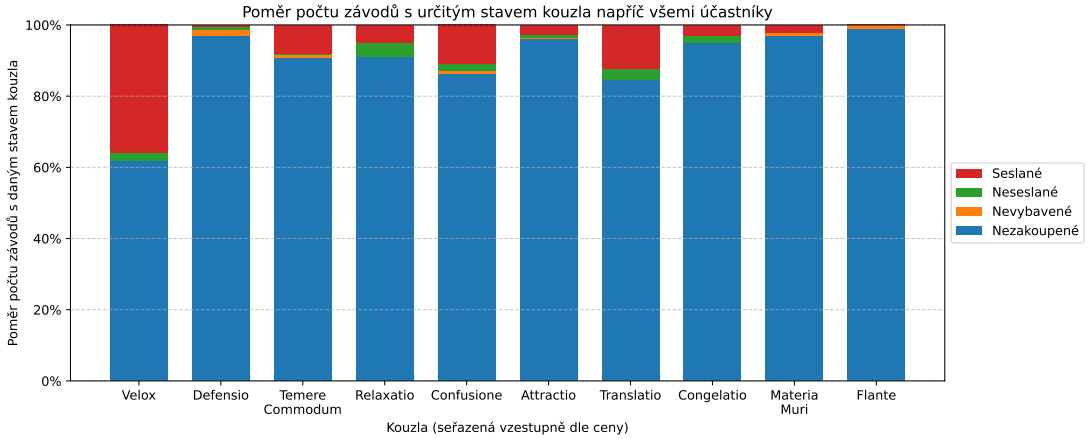
Dále můžeme z grafu vyčíst, podobně jako v předchozí sekci, jak úspěšní byli účastníci v sesílání kouzel. Např. na obrázku 5.21 vidíme účastníka, který měl pouze

jediné kouzlo *Velox*, ale ve velké části závodů ho ani jednou neseslal. Ten samý účastník však v dotazníku vyjádřil zmatení ohledně fungování kouzel sesílaných na sebe sama (mezi která patří také *Velox*). I když tedy prošel tutoriálem, kde ho musel alespoň jednou seslat, aby pokročil dál, nebyl si zřejmě jistý jeho používáním v následujících závodech. Mohli bychom se proto pokusit v rámci tutoriálu nějak zdůraznit, že taková kouzla lze seslat libovolně bez zamíření.




**Obrázek 5.21** Poměr stavů jednotlivých kouzel napříč všemi závody (účastník 3).

Z grafů jednotlivých účastníků můžeme získat informace o osobních preferencích jednotlivců. Obecně se dá říci, že když už si účastníci kouzlo koupí, obvykle si ho také dosadí do slotu a pak velmi často alespoň jednou v každém závodě sešlou. Navíc jsme však všechny výsledky agregovali a vytvořili tak graf na obrázku 5.22, který zachycuje opět poměr počtu závodů s určitým stavem kouzla, ale tentokrát pro všechny závody všech účastníků dohromady.

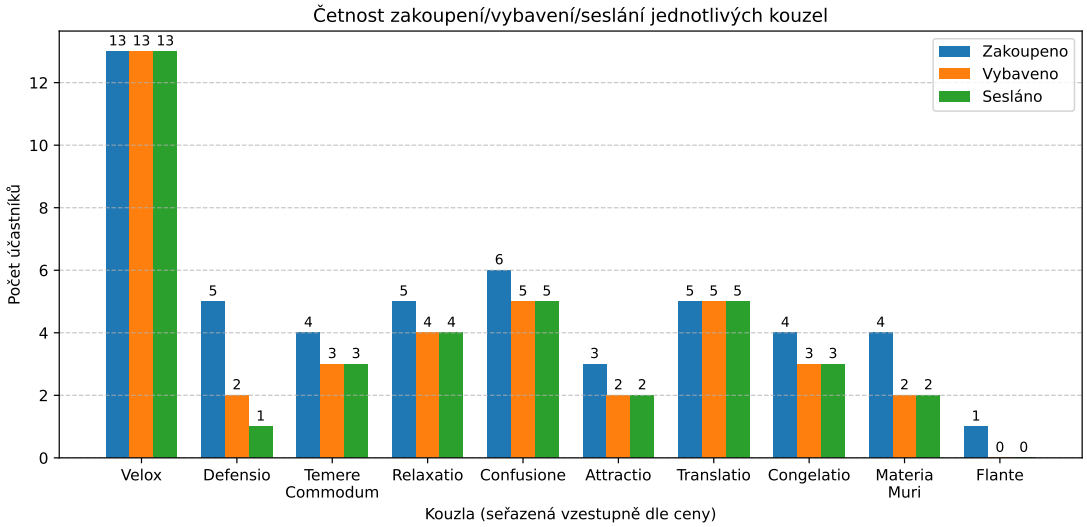



**Obrázek 5.22** Poměr počtu závodů s určitým stavem kouzla napříč všemi účastníky.

*Defensio* je ze všech kouzel nejčastěji nevybavováno. To může souviset se způsobem jeho použití, kdy se vyplatí tehdy, když na hráče soupeři sesílají kouzla. Navíc je třeba jej využívat spíše preventivně. Podobně *Relaxatio* je poměrně často nesesláno. Opět však záleží na situaci, protože toto kouzlo má smysl seslat pouze tehdy, pokud na hráče právě v tu chvíli působí nějaký negativní efekt. Může se stát, že taková situace za celý závod nenastane. Kromě kouzel s omezeným

využitím je pak neseslání patrné také u kouzel, jejichž seslání stojí více many. Je tedy možné, že účastníci v tu chvíli upřednostňovali spíše nějaké levnější.

Pro obecné určení popularity jednotlivých kouzel jsme pak vytvořili graf zachycující pro každé kouzlo, kolik účastníků si jej zakoupilo, kolik jej alespoň jednou vybavilo a nakonec kolik jej alespoň jednou seslalo. Výsledek vidíme na obrázku 5.23.




**Obrázek 5.23** Četnost zakoupení/vybavení/seslání jednotlivých kouzel.

Zcela nepřekvapivě má *Velox* výraznou převahu. Jedná se totiž o nejlevnější kouzlo, které je navíc v nabídce obchodu hned jako první. Jeho použití je pak extrémně jednoduché a dává okamžitou a spolehlivou výhodu. Popularita *Confusione* se dá vysvětlit tím, že je to nejlevnější kouzlo, kterým se dají přímo ovlivňovat soupeři. *Flante* je zdaleka nejmíň časté, ale jeho pořizovací cena je už dost vysoká a účastníci hráli jen poměrně krátkou dobu, takže je přirozené, že budou upřednostňovat spíše levnější kouzlo, které mohou mít hned.

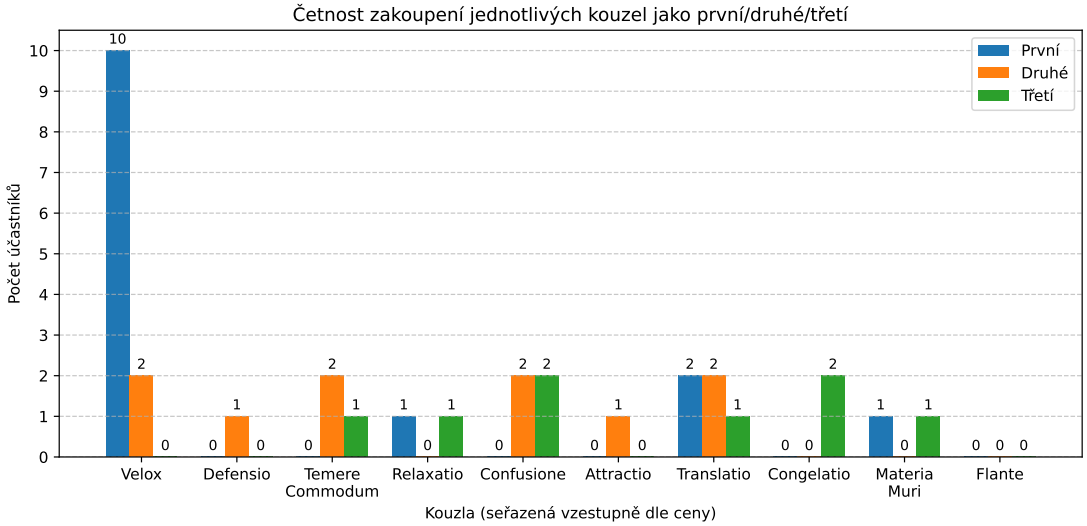
Zajímavé je pozorování, že vcelku často je kouzlo vícekrát zakoupeno než vybaveno. Z velké části je za to zodpovědný jeden z účastníků, který chtěl zřejmě dosáhnout úplnosti a hru ukončil až ve chvíli, kdy měl zakoupená všechna kouzla i vylepšení koštěte. Další případy jsou pak zřejmě způsobeny tím, že si účastníci dané kouzlo pořídili jako poslední a pak hned hru ukončili.

Abychom však z hlediska využití kouzel mohli pozorovat nějaké významnější výsledky, museli bychom nechat účastníky hrát hru po výrazně delší dobu. Jedině tak by měli všichni možnost zakoupit si více kouzel, vyzkoušet si je a pak se mezi nimi také rozhodovat.

### Zakoupení kouzel

Kromě samotného využití kouzel jsme se pak zaměřili také na jejich nakupování. Zajímalo nás totiž, která kouzla bývají zakoupena mezi prvními. Na obrázku 5.24 tedy vidíme pro každé kouzlo počet účastníků, kteří si jej zakoupili jako první, druhé a třetí.

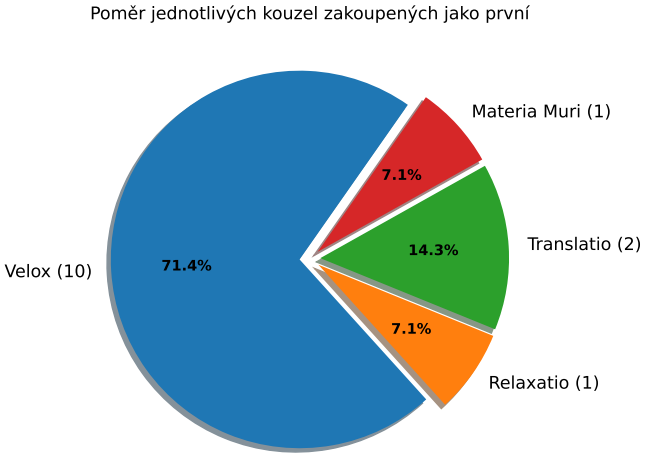
Jak už bylo řečeno také výše, *Velox* je velmi častou první volbou zřejmě kvůli své nízké pořizovací ceně. Dále se zdá, že jako druhá volba jsou populární




**Obrázek 5.24** Četnost zakoupení jednotlivých kouzel mezi prvními třemi.

především kouzla s přímočarými efekty (např. *Temere Commodum* pro vyčarování náhodného bonusu, *Translatio* pro rychlé přesunutí dopředu) a *Confusione*, jakožto nejlevnější kouzlo ovlivňující soupeře. Čím dražší pak kouzlo je, tím spíše není ani jednou ze tří prvních voleb.

Na obrázku 5.25 pak vidíme přehledně čistě první volbu. Zajímavé je, že se jeden z účastníků rozhodl zakoupit *Relaxatio*, i když se do té doby ve hře žádné negativní efekty nevyskytovaly (začínají až se zakoupením prvního kouzla sesílaného na soupeře). Není nám však jasné proč. Takový typ informací bychom mohli získat, pokud bychom prováděli experiment formou sledování účastníka a pokládání dotazů.



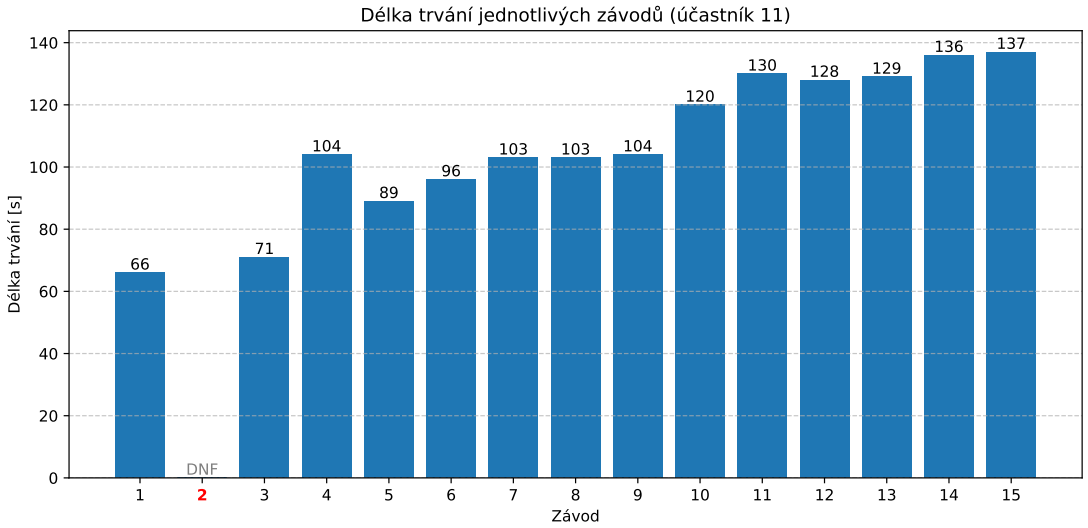

**Obrázek 5.25** Poměr jednotlivých kouzel zakoupených jako první.

### Délka trvání závodu

Další částí hry, kterou jsme chtěli analyzovat, je samotný průběh závodů. Jako první jsme tedy pro každého účastníka zjistili, jak dlouho trvaly jednotlivé závody.

Díky tomu pak můžeme zkoumat, zda délka závodů roste rozumně a zda by se mohlo stát, že by nakonec závody byly až příliš dlouhé. Pokud uvažujeme čistě délku trati, má na ni vliv hodnota *rychlosti* a *vytrvalosti*. Obecně délka trvání závodu se pak ale může zkrátit vylepšením rychlosti koštěte.

U některých účastníků byl znatelný postupný nárůst délky závodu. Příklad jednoho z nich je na obrázku 5.26. Závod 4 pak výrazně vyčnívá z řady, ale to je dáno tím, že během něj účastník letěl neobvykle pomalu, takže se mu po něm také snížila statistika *rychlosti*.



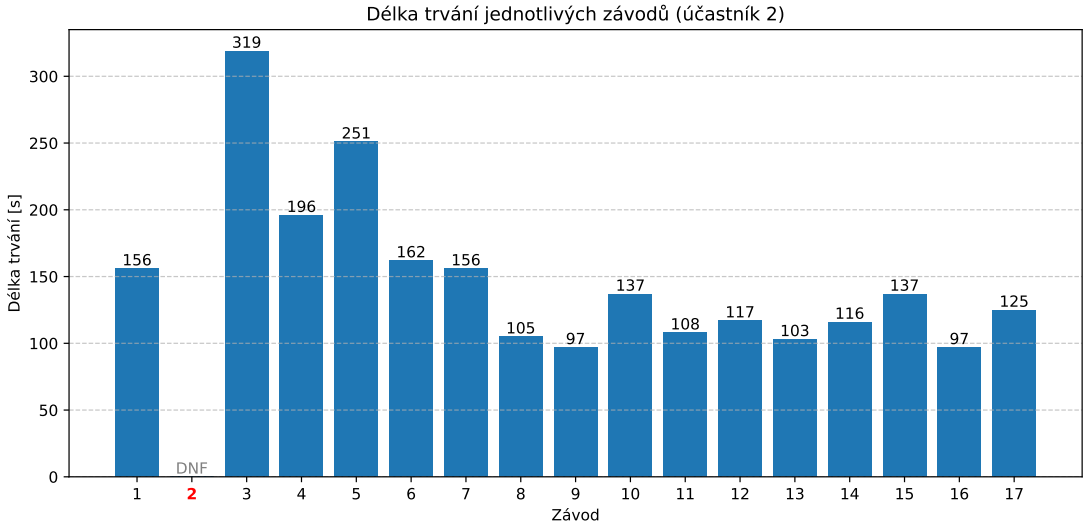

**Obrázek 5.26** Délka trvání jednotlivých závodů (účastník 11).

Zaznamenali jsme však i případ účastníka, u kterého délka trvání závodů naopak postupně klesala. Odpovídající graf vidíme na obrázku 5.27. Po bližším prozkoumání se totiž ukázalo, že nebyl ve hře příliš dobrý a hodnoty statistik se mu nijak zvlášť neměnily, takže i tratě zůstávaly stále stejně dlouhé. Jak se však účastník postupně zlepšoval v letu, dokázal trať absolvovat rychleji, ale ještě stále to nestačilo na dobré umístění, aby se tak navýšila *vytrvalost*. Současně je nutno poznamenat, že teprve třetí závod v grafu je prvním závodem v rámci druhé části experimentu. Účastník tuto část zahájil až s odstupem 3 dní, takže si zřejmě již nepamatoval ovládání, ale současně pokračoval v dříve rozehrané hře.

Obecně jsme pak zaznamenali, že pokud se účastníkům zvyšovala hodnota *vytrvalosti*, rostla také délka trvání závodů, a to rychlostí, jakou bychom zhruba očekávali pro dosažení maximální délky 5 minut. A naopak pokud některý účastník zůstával na velmi nízké hodnotě *vytrvalosti*, délky trvání závodů byly mnohem více nepředvídatelné (zřejmě kvůli kolísání výkonu účastníka). Nejspíš bychom tedy mohli hru lépe přizpůsobit, aby uměla být ze začátku ještě lehčí. Tak by i slabší hráči měli možnost se neustále posouvat dál a nezůstali by zaseklí, dokud by se nezlepšili natolik, že by dohnali počáteční obtížnost hry.

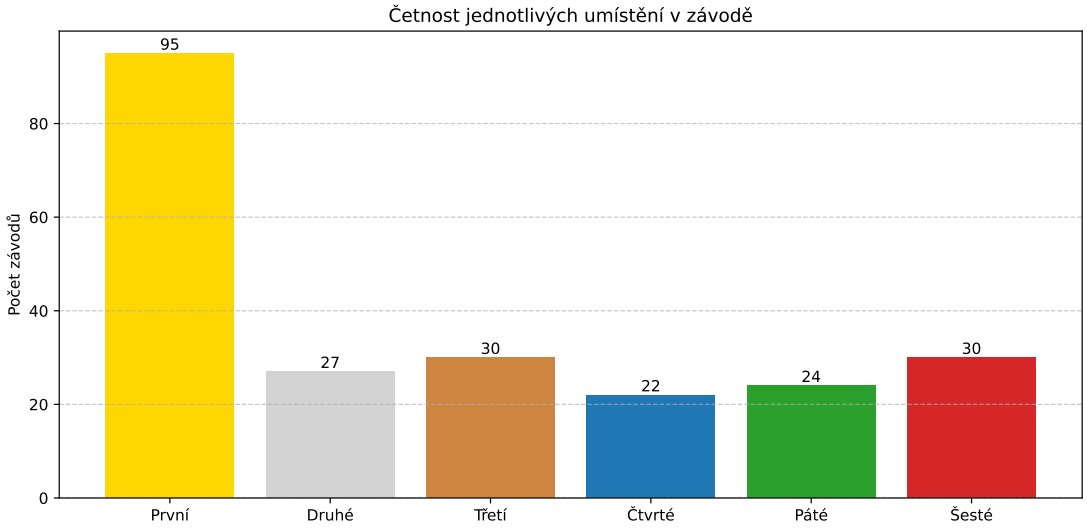
### Umístění v závodech

Dále jsme se podívali, na kolikátém místě se účastníci běžně umisťují, abychom mohli odvodit, zda pro ně nejsou soupeři příliš dobří, či špatní, a jak těžké je




**Obrázek 5.27** Délka trvání jednotlivých závodů (účastník 2).

umístit se první. Na obrázku 5.28 tak vidíme graf s četností jednotlivých umístění napříč všemi závody všech účastníků.

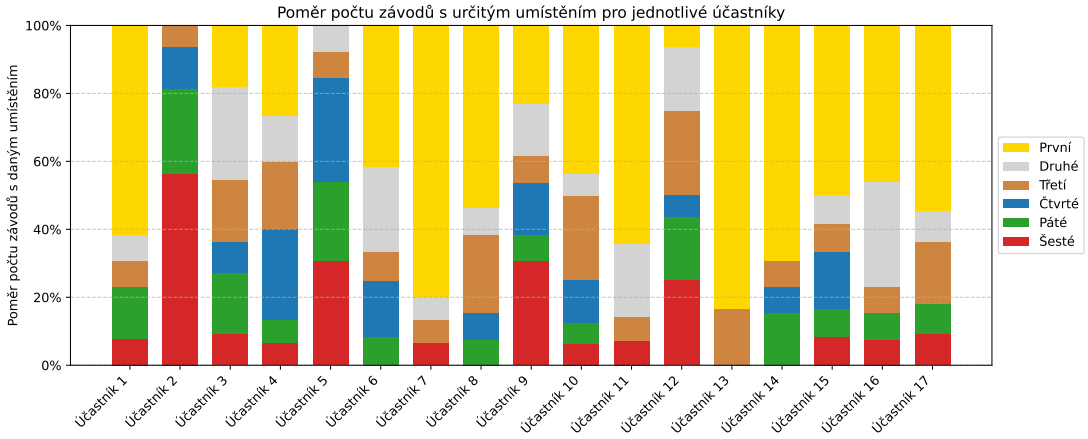



**Obrázek 5.28** Četnost jednotlivých umístění v závodě napříč účastníky.

Zdá se, že první místo je překvapivě velmi časté, naopak ostatní se vyskytují podobně často. Vypadá to tedy, jako by měl hráč velkou šanci umístit se první, ale zbylá umístění jsou zcela náhodná. Mohli bychom se tedy zaměřit na vybalancování schopností soupeřů tak, aby mezi nimi byly trochu větší rozdíly, a tím byla pravděpodobnost umístění lépe rozložena.

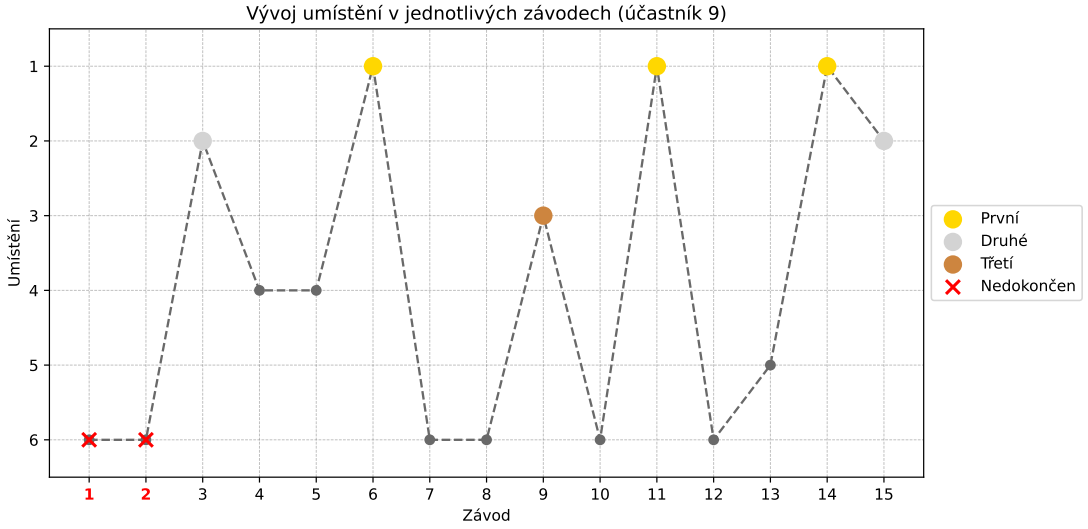
Ještě jsme se však podívali také na individuální výsledky účastníků, jestli i tam dochází k takové náhodnosti umístění. Na obrázku 5.29 tak vidíme pro všechny účastníky naráz poměr jednotlivých umístění. Existují účastníci, kteří se umisťují konzistentně dobře, a stejně tak existují účastníci, kteří se umisťují konzistentně špatně. Je ale stále znatelný rozdíl v počtu prvních míst oproti ostatním. Možná je tedy umístění se na prvním místě až příliš snadno dosažitelné.

Nakonec jsme se zaměřili na samotný vývoj umístění v čase pro každého




**Obrázek 5.29** Poměr jednotlivých umístění v závodě pro každého účastníka zvlášť.

účastníka zvlášť. Příklad takového grafu vidíme na obrázku 5.30. Obecně nebyl znatelný žádný trend. Některým účastníkům se obzvlášť dařilo, jiným obzvlášť nedařilo, ale často byla umístění dosti náhodná. Obvykle se ani nezdálo, že by se umístění průběžně zlepšovalo. Hráč by tak ale mohl mít pocit, že je umístění zcela náhodné a nezáleží skutečně na tom, jak se mu v závodě daří.

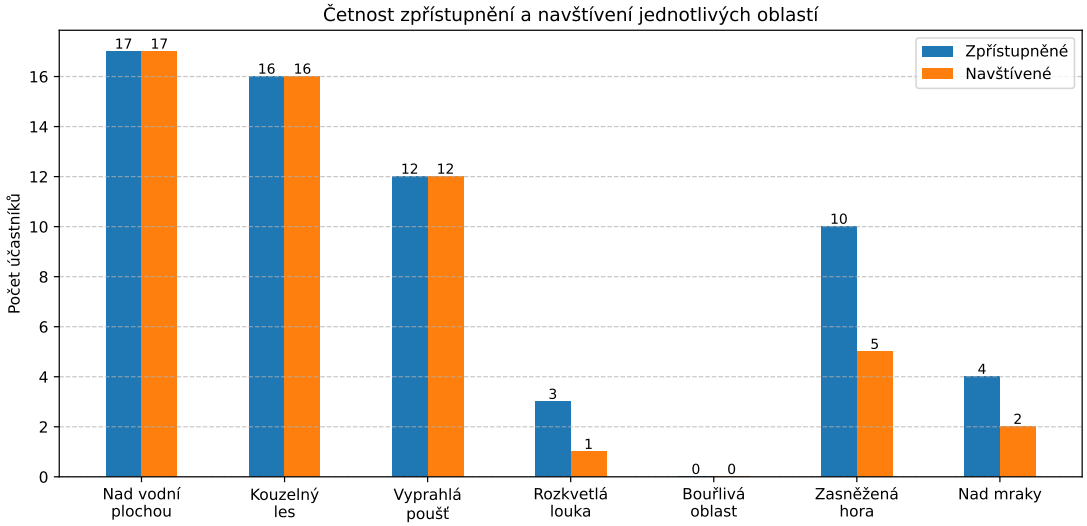



**Obrázek 5.30** Vývoj umístění v jednotlivých závodech (účastník 9).

Měli bychom se tedy více soustředit na chování soupeřů a vylepšit jejich přizpůsobování se hráčovým schopnostem. Cílem by bylo zmenšit rozdíly v počtu jednotlivých umístění a současně je udělat trochu předvídatelnější. Nechtěli bychom však četnost prvního místa úplně potlačit, protože jedině dobrým umístěním je možné postupovat ve hře dál.

### Oblasti

Se závody souvisí také tematické oblasti. Pomocí posbíraných dat jsme pro každou oblast spočítali, kolik účastníků ji zpřístupnilo a navštívilo. Výsledek pak vidíme na obrázku 5.31.




**Obrázek 5.31** Počet účastníků, kteří zpřístupnili a navštívili jednotlivé oblasti.

S výjimkou jednoho účastníka se všichni dostali dostatečně daleko v tutoriálu, aby se jim zpřístupnil *Kouzelný les*. Velký počet z nich pak dosáhl také požadované *vytrvalosti* pro *Vyprahlou poušť*. Vzhledem k omezené době hraní není překvapivé, že *Rozkvetlé louky* dosáhli jen 3 účastníci a *Bouřlivé oblasti* pak žádný.

Vcelku pozitivní je výsledek, že 10 účastníků zpřístupnilo *Zasněženou horu* a 4 z nich také oblast *Nad mraky*. Ty jsou totiž vázány na vylepšení stoupání koštěte. Jedná se přitom o pravděpodobně nejméně lákavou položku z nabídky obchodu, protože je vcelku drahá a nepřináší žádné skutečné výhody, pouze zpřístupňuje nové oblasti. Dalo by se tedy říct, že 4 účastníci byli motivováni prozkoumávat obsah dostupný ve hře.

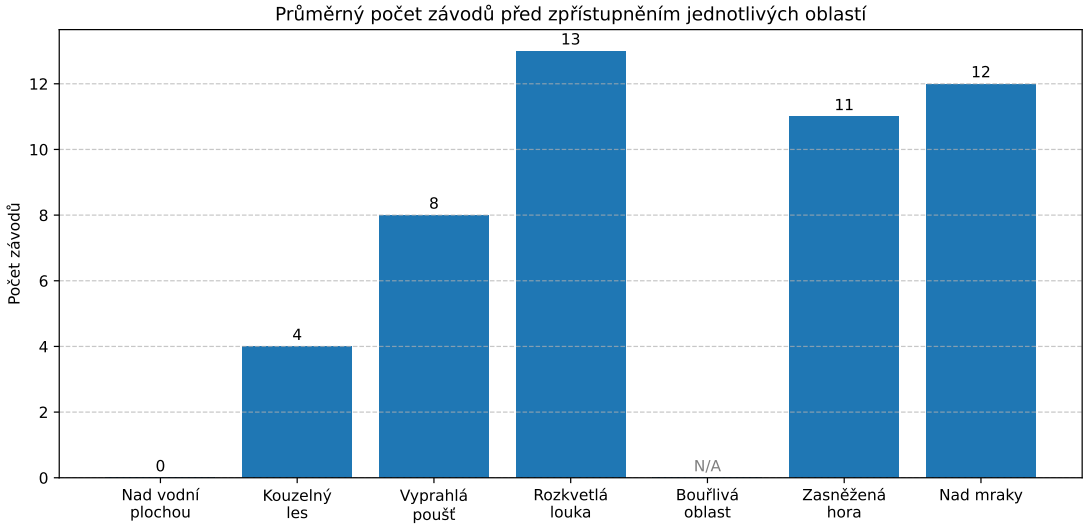
Současně se zdá, že když se oblast zpřístupní, zpravidla se také navštíví (až na pár výjimek, které vznikly zpřístupněním ke konci sezení). Zdá se tedy, že algoritmus pro generování levelů funguje a správně vynucuje zahrnutí doposud nenavštívené oblasti.

Další důležitou metrikou je, kolik závodů museli účastníci absolvovat, než se jim zpřístupnila určitá oblast. Chtěli bychom především zajistit, že jsou zpřístupnění dostatečně rozprostřená v čase. Na obrázku 5.32 pak vidíme průměrný počet závodů potřebný pro zpřístupnění jednotlivých oblastí.

Zdá se, že zpřístupňování na základě *vytrvalosti* (případně tutoriálu) funguje dobře a oblasti se odemykají postupně v rozumných intervalech. Zpřístupnění oblasti *Nad mraky*, která je vázána na vylepšení stoupání koštěte, bychom však mohli trochu pozdržet. Takto se odemykají tři oblasti vcelku brzy po sobě. Pokud bychom tedy druhou úroveň stoupání zdražili, mohli bychom dosáhnout toho, že by si ji účastníci pořídili až později, čímž by se oblast *Nad mraky* mohla zpřístupnit až po *Rozkvetlé louce*, ale současně ještě před *Bouřlivou oblastí*. Pokud bychom současně zajistili, že budou mít hráči možnost v rozumném čase získat požadovaný počet mincí, nic bychom touto úpravou neměli pokazit.

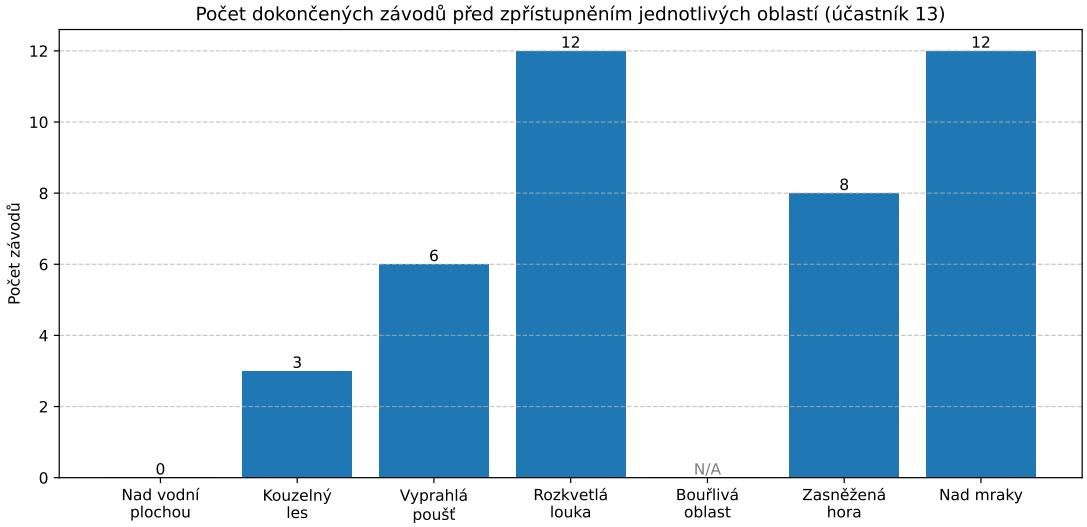
Nakonec se podíváme ještě na počty závodů zvlášť pro některé účastníky, abychom si ověřili, že změny navrhované na základě zprůměrovaných dat jsou relevantní.

Když byli účastníci úspěšnější a podařilo se jim zpřístupnit více oblastí, obvykle



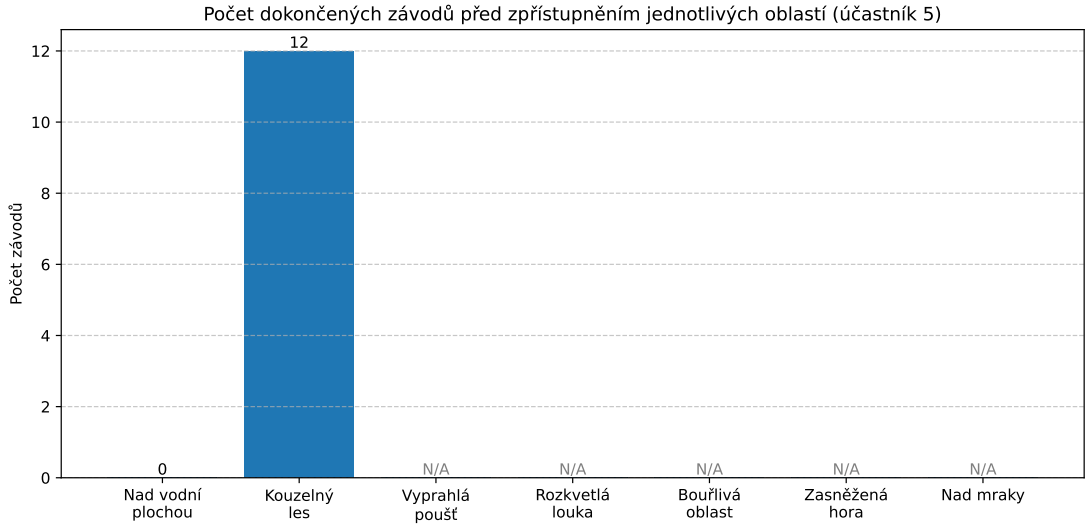

**Obrázek 5.32** Průměrný počet závodů před zpřístupněním jednotlivých oblastí.

skutečně docházelo k odemčení dvou naráz nebo velmi brzy po sobě (např. účastník na obrázku 5.33). Pokud bychom tedy upravili cenu vylepšení stoupání, mohli bychom tento problém vyřešit. Průchod každého hráče hrou se však samozřejmě velmi liší a není tak možné odladit zcela ideální zážitek pro každého.




**Obrázek 5.33** Počet dokončených závodů před zpřístupněním jednotlivých oblastí (účastník 13).

Když byl ale účastník horší, *Kouzelný les* se zpřístupnil až vcelku pozdě (v případě účastníka na obrázku 5.34 dokonce až po 12 závodech). Samotná oblast *Nad vodní plochou* však není příliš zajímavá a mohlo by tedy být lepší zpřístupnění *Kouzelného lesa* urychlit. V tom případě by mohly pomoci změny obtížnosti hry, které jsme navrhli v závěru části o vývoji hodnot statistik. Díky nim by měli i slabší hráči možnost umisťovat se lépe v závodech, takže by si dříve vydělali na nějakou položku v obchodě, čímž by se dříve odemknul *Kouzelný les* (vázaný na dosažení tutoriálu pro zakoupení něčeho).

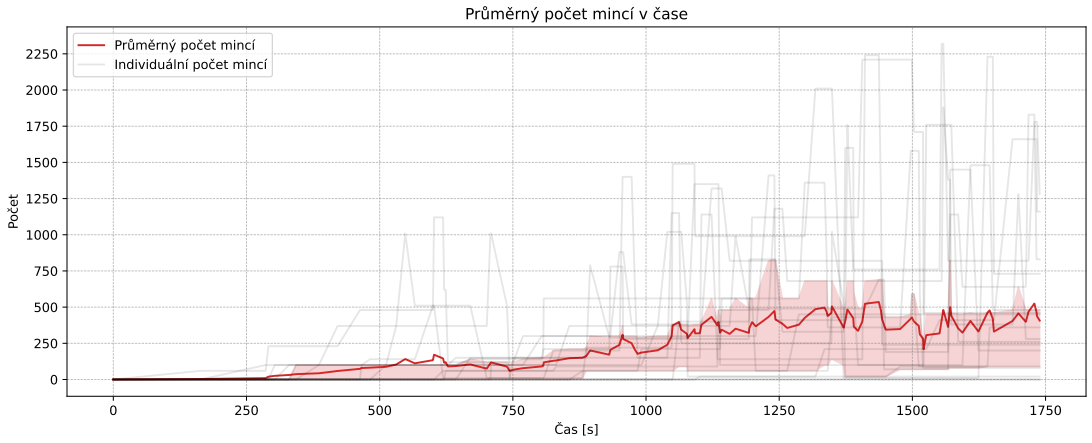



**Obrázek 5.34** Počet dokončených závodů před zpřístupněním jednotlivých oblastí (účastník 5).

## Mince

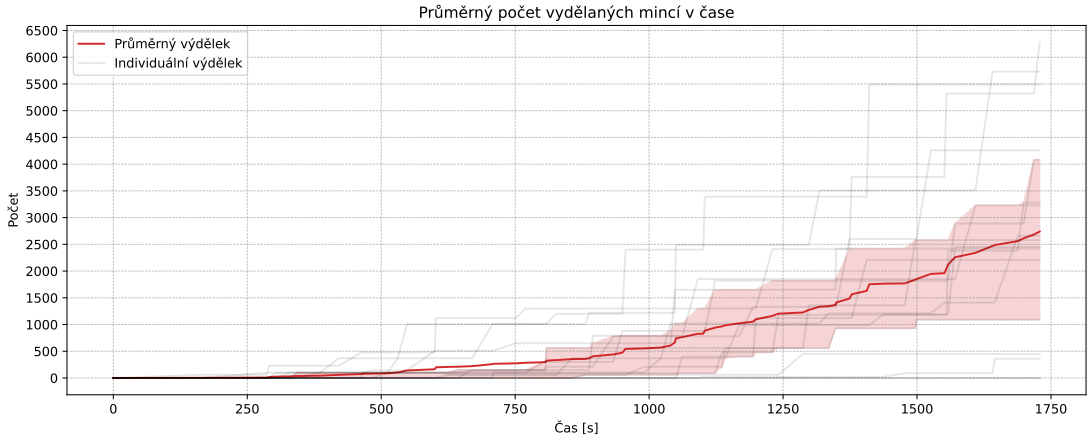
V předchozích částech už jsme prozkoumali, jak dobře účastníci postupovali ve hře a jak často dosahovali jednotlivých umístění v závodě. První tři místa jsou navíc spojena s finanční odměnou. Měli bychom však zjistit, jakým způsobem se vyvíjí množství mincí, jestli jich třeba účastníci neměli přebytek a mohli si tak kupovat až příliš často nové položky v obchodě.

Obrázek 5.35 znázorňuje, jak se vyvíjel průměrný počet mincí v čase (spočtený přes všechny účastníky). Šedě je pak zakresleno množství pro jednotlivé účastníky zvlášť. Z grafu je patrné, že mezi účastníky jsou velké rozdíly. Především ale vidíme, že mince se příliš nehromadí a účastníci je obvykle velmi brzy utratí.




**Obrázek 5.35** Průměrný počet mincí v čase.

Můžeme se pak podívat speciálně na průměrné výdělky v čase. Znázorňuje je graf na obrázku 5.36. Opět je z něj zřejmé, že se průchody jednotlivých účastníků velmi liší – někdo si za znázorněnou dobu nevydělal vůbec nic, zatímco jiný si vydělal přes 6000. Průměrně si však za necelých 30 minut hry vydělali kolem 2700 mincí (všechny položky v obchodě pak pro srovnání stojí dohromady 21350).

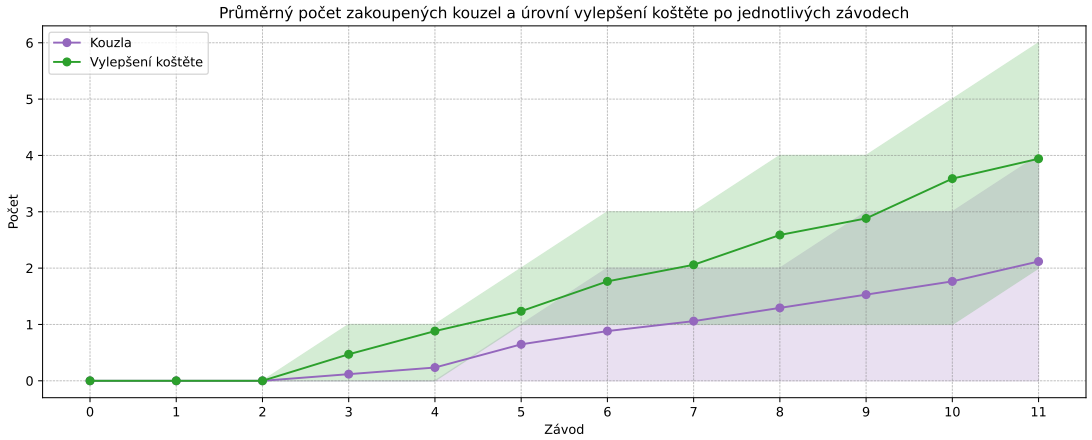



**Obrázek 5.36** Průměrný počet vydělaných mincí v čase.

Výše odměny však závisí nejen na umístění v závodě, ale také na hodnotě statistik. Postupem času by tedy hráči získávali čím dál tím víc mincí. Mohli bychom tedy nejspíš odměny o něco snížit, obzvlášť když z jedné z předchozích sekcí víme, že se účastníci velmi často umisťovali na prvním místě. Kromě snížení maximální odměny za závod bychom pak mohli také mírně navýšit odměny za druhé a třetí místo. Tímto způsobem bychom navíc zajistili, že bychom omezením maximální odměny příliš netrestali slabší hráče, kteří se občas ocitnou na lepším místě. Měli by totiž také mít šanci si v rozumné době něco pořídit v obchodě, aby pro ně hra zůstávala zajímavá.

### Nakupování

Jelikož si za vydělané mince mohou hráči nakupovat kouzla a vylepšení koštěte v obchodě, podívali jsme se rovnou také na to, jak tyto nákupy probíhaly. Nejprve jsme spočítali, jak se vyvíjel průměrný počet zakoupených položek. Výsledek pak vidíme na obrázku 5.37.



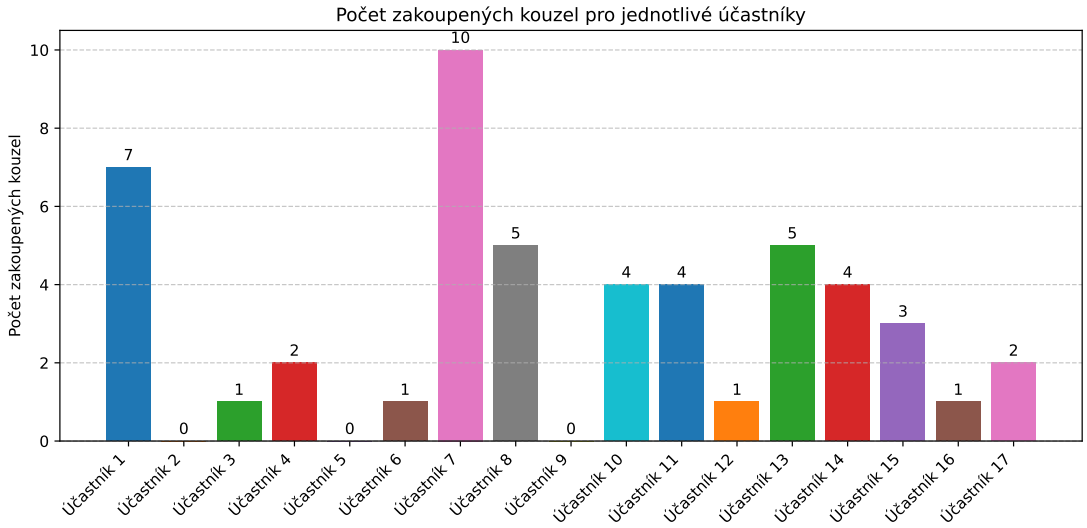

**Obrázek 5.37** Průměrný počet zakoupených kouzel a úrovní vylepšení koštěte po jednotlivých závodech.

Z grafu můžeme vypozorovat, že si účastníci mnohem více kupovali vylepšení koštěte než kouzla, i když nejsou nijak výrazně lepší a ani levnější. Mohlo by

to být však tím, že jsou přístupnější, protože jejich efekt je automatický a trvá navždy. Nepřidává se tak žádná nová funkcionalita, pouze se vylepšuje to, co už hráč má.

Současně je však vidět, že počet zakoupených kouzel i vylepšení celkově roste. Zdá se tedy, že když už byli účastníci schopní si ze závodů vydělat dostatek mincí, kupovali si něco v obchodě poměrně pravidelně a mezi jednotlivými nákupy pravděpodobně nebyly příliš velké prodlevy. Nabídka obchodu je tedy nejspíš dostatečně lákavá a účastníci byli motivováni nakupovat.

Když se navíc podíváme na obrázek 5.38, někteří účastníci byli schopní za poměrně krátkou dobu zakoupit vcelku dost kouzel. Tři z nich si však naopak nezakoupili jedno jediné. To bylo způsobeno především tím, že se jim ve hře příliš nedařilo a nedokázali si tak na žádné vydělat mince.



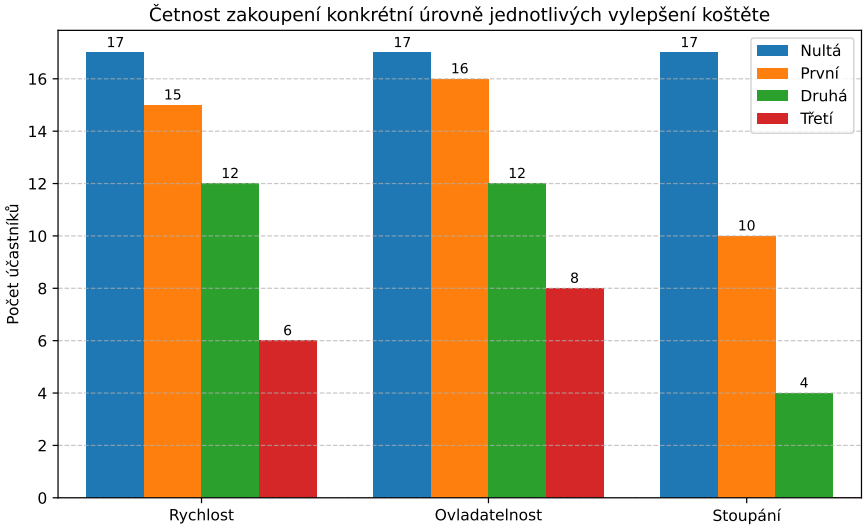

**Obrázek 5.38** Počet zakoupených kouzel pro jednotlivé účastníky.

Nakonec jsme se podívali ještě na to, kolik účastníků si zakoupilo jednotlivé úrovně vylepšení koštěte, abychom věděli, jak moc si dokázali koště vylepšit a která vylepšení byla nejpopulárnější. Výsledky pak vidíme na obrázku 5.39.

Nejčastější volbou je *ovladatelnost*, a to na všech úrovních. Zdá se tedy, že účastníci měli nejspíš problém s ovládáním koštěte. *Rychlost* však nezůstává příliš pozadu, což není nečekané, jelikož poskytuje značnou výhodu. *Stoupání* je pak pochopitelně nejméně časté nejspíš proto, že je už o něco dražší než ostatní a navíc nepřináší žádnou výhodu z hlediska výkonu v závodě, pouze zpřístupňuje další oblasti. Přesto je však překvapivé, že si první úroveň zakoupilo dokonce 10 účastníků, obzvlášť když si pouze 7 účastníků zakoupilo alespoň 4 kouzla, takže ostatní měli stále volné sloty.

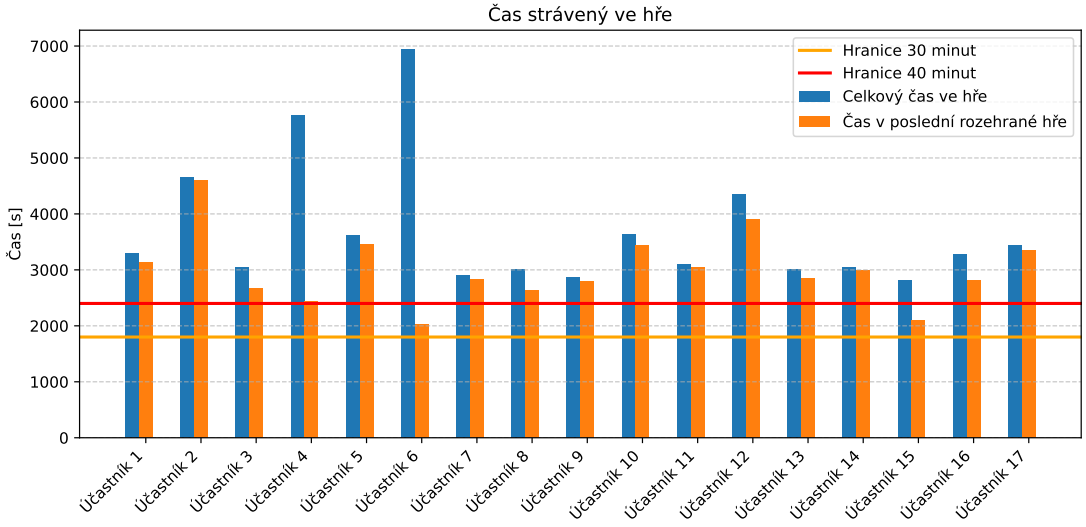
### Obecné

Na závěr jsme se ještě pro zajímavost podívali na některá obecná data k hraní hry. Vytvořili jsme tak graf na obrázku 5.40, který zachycuje, kolik času strávili jednotliví účastníci celkově ve hře a kolik času strávili přímo v poslední rozehrané hře (neuvažují se tedy dříve založené hry a ani čas strávený v hlavním menu). Jelikož zahrnujeme obě části experimentu, v grafu jsme zvýraznili jak hranici




**Obrázek 5.39** Četnost zakoupení konkrétní úrovně jednotlivých vylepšení koštěte.

30 minut (minimální požadovaná doba ve druhé části), tak 40 minut (s alokací zhruba 10 minut pro první část experimentu).



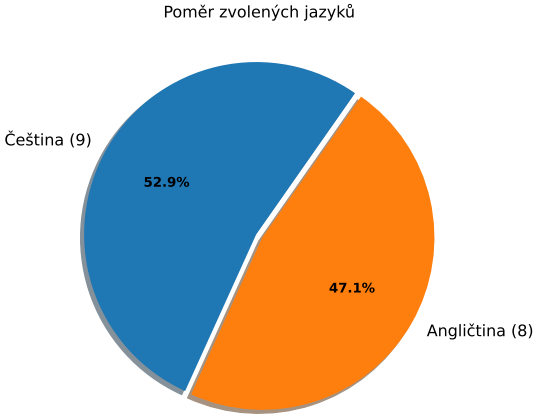

**Obrázek 5.40** Kolik času strávili jednotliví účastníci celkově ve hře a v poslední započaté hře.

Jak je vidět, dost účastníků strávilo ve hře více času, než bylo námi požadované minimum. U některých je to jen velmi těsné, takže jim možná jen první část experimentu zabrala více než odhadnutých 10 minut. Někteří však strávili hrou podstatně více času. To považujeme za dobrý výsledek, protože to znamená, že ve hře pokračovali nejen kvůli experimentu. To je také v souladu s odpověďmi na doplňující otázku v dotazníku, zda by hru ukončili dříve, či později, pokud by ji nehráli v rámci experimentu. Průměrná hodnota byla 3,47, kde 1 znamená „Ukončil/a bych ji dříve“ a 5 znamená „Ukončil/a bych ji později“.

Účastníci 4 a 6 mají výrazně více času stráveného celkově ve hře, protože rozehráli několik nových her za sebou. Podobně účastník 15, ale tam není rozdíl tak výrazný. Do času stráveného přímo ve hře (a stejně tak do všech výsledků

uvedených v této kapitole) totiž započítáváme pouze poslední započatou hru.

Dále jsme se podívali, kolik účastníků si ve hře zvolilo který jazyk. Výsledný graf vidíme na obrázku 5.41. Jako výchozí jazyk je ve hře nastavena angličtina. Ačkoliv bychom očekávali, že většina účastníků bude umět obstojně anglicky, více než polovina z nich si jazyk změnila na češtinu. To nás utvrzuje v tom, že bylo dobré rozhodnutí hru lokalizovat.




**Obrázek 5.41** Srovnání zvolených jazyků dle počtu účastníků.

Nakonec jsme ještě chtěli zjistit, zda alespoň někdo z účastníků využil změnu mapování kláves a případně co použil, abychom věděli, jak užitečné dané nastavení je a také jaké ovládání se účastníkům zdá lepší. Ukázalo se, že celkem dva účastníci se rozhodli akce přemapovat. Jeden si změnil zatáčení do stran, stoupání a klesání na ovládání pomocí šipek, zrychlení a zpomalení pak na `Q` a `A`. Nevíme však, jak přesně se mu dařilo používat také kouzla, protože jejich ovládání zůstává pomocí myši. Druhý účastník si pak změnil pohled dozadu na klávesu `C`. Takové mapování jsme původně také zvažovali, ale rozhodli jsme se použít raději klávesu `B`, která je sice dál, ale mohla by být lépe zapamatovatelná (B jako počáteční písmeno „back view“).

## 5.3 Zpětná vazba

V sekci 5.1 jsme uvedli několik změn, které jsme provedli na základě zpětné vazby z interního testování.

Poté jsme v sekci 5.2.4 navrhli možné úpravy hry dle dat posbíraných z herních analytik. Jedná se však zpravidla o problematiku balancování hry. Takové změny je třeba dělat opatrně a následně je opakovaně testovat s větší skupinou lidí (např. jako A/B testy pro srovnání různých hodnot parametrů). Proto jsme je prozatím odložili do budoucna. Konkrétně snížení maximální možné odměny za závod a navýšení odměn za druhé a třetí místo jsme provedli dokonce již v době, kdy experiment teprve probíhal. Tušili jsme totiž, že by to mohlo být problematické. Výsledky experimentu nás pak utvrdili v tom, že bylo naše rozhodnutí správné. Je však třeba experimentálně znovu ověřit vhodnost zvolených hodnot.

Na konec dotazníků jsme přidali také volitelné otevřené otázky, kde měli účastníci možnost k čemukoliv se vyjádřit (všechny posbírané odpovědi se nachází

v souboru `zpetna_vazba.md` ve složce `experimenty/` elektronické přílohy). Díky tomu se povedlo posbírat velmi cennou zpětnou vazbu. Uvedeme nyní několik příkladů.

### Provedené změny

Několik účastníků si všimlo, že dostali ocenění za zakoupení všech kouzel, i když si koupili teprve první. K problému docházelo proto, že si spustili nejprve první část bez kouzel, ve které se inicializoval stav, ale pak při spuštění druhé části již s kouzly se nepřepsal, takže zakoupení kouzla vyvolalo aktualizaci, kdy se detekovalo, že si účastník zakoupil 1 kouzlo z 0 dostupných. Ačkoliv je tedy chyba specifická pro experiment, opravili jsme ji, protože díky tomu bude hra fungovat správně v případě, kdybychom do ní přidali nová kouzla.

Někteří účastníci pak vyjádřili zmatení ohledně úvodního tutoriálu seznamujícího je se základním pohybem. V některých krocích totiž musí představenou akci provést dvakrát až třikrát, aby mohli postoupit dál, ale účastníci akci provedli třeba jen jednou a pak se zasekli a kroky jim připadaly dlouhé. Explicitně jsme tedy v odpovídajících krocích zmínili, aby si dané akce párkrát vyzkoušeli. Dva účastníci pak dokonce navrhli, abychom přidali např. speciální zónu, do které by doletěli, až by byli připravení. To by ovšem dle našeho názoru tutoriál jedině prodloužilo, protože kromě vyzkoušení akcí by museli ještě doletět na definované místo.

Pár účastníků si pak postěžovalo, že se hra drobně zasekávala během tutoriálu. Pro levely v závodech jsme totiž sice implementovali optimalizaci rozdělení terénu do bloků (viz sekce 3.3.4), ale ve scénách `Tutorial` a `TestingTrack` jsme zachovali původní neoptimalizovaný terén, jelikož jsou levely v nich poměrně malé a současně pevně dané. Pokud bychom tedy chtěli optimalizace zavést, museli bychom vygenerovat nový level, uložit, použít ve scéně a pak mu vše přizpůsobit. Na základě této zpětné vazby jsme však přesně toto provedli a věříme, že by to mělo pomoci problém redukovat.

Nakonec jeden účastník popsal problém zaseknutí ve zdi levelu během úvodního tutoriálu. Popsaný problém jsme velmi extenzivně testovali, nalétávali do ochranné bariéry trati v různých místech, z různých směrů, s různou rychlostí (dokonce i vyšší než maximální možnou ve hře), ale nepodařilo se nám jej reprodukovat. Detekce kolizí byla správně nastavená na spojitou. Více jsme bohužel nemohli udělat a problém prozatím necháváme otevřený.

### Výsledná verze hry

Po zapracování zmíněné zpětné vazby a řady dalších vylepšení pak vznikla finální demo verze hry, kterou jsme navíc v souladu s naším cílem **C5** zveřejnili na `https://michelle2.itch.io/brooom`.

Pomocí experimentů jsme navíc ověřili požadavek **P2**, ve kterém pro naši hru specifikujeme cílovou platformu. Nikdo z účastníků nenarazil na žádné problémy při spouštění hry či jejím běhu. Také se tím hra otestovala na rozličných zařízeních, od notebooků s integrovanou grafickou kartou až po výkonnější stolní počítače, přičemž se zdá, že tímto testem prošla vcelku obstojně.

Na úplném konci dotazníku pak bylo ještě několik doplňujících otázek. V jedné z nich jsme se ptali, zda by si rádi zahráli tuto hru s dalšími hráči. Z výsledků

pak vyplynulo, že režim více hráčů (viz sekce 2.11), který je sice plánovaný, ale zatím není v demu implementovaný, by mohl být velkým přínosem.

### Plánované změny

Kromě již implementovaných změn máme poznamenáno také několik námětů, které plánujeme zapracovat v blízké budoucnosti.

Jeden z účastníků v dotazníku uvedl, že mu připadalo neintuitivní, že se pozitivní změna umístění v žebříčku zobrazuje se znaménkem minus. Ačkoliv je to matematicky správně, jedná se spíše o psychologické vnímání znaménka plus jako něčeho pozitivního. Plánujeme tedy místo znamének použít raději zelenou šipku nahoru, či červenou šipku dolů, a to jak u umístění v žebříčku, tak u statistik. Díky tomu by mohla být interpretace jednoznačná.

Dále bychom chtěli omezit dosah kouzel, která se sesílají určitým směrem, na základě okolního levelu. Jeden z účastníků totiž zmínil, že mu kouzla létala také mimo mapu, čímž bylo zřejmě myšleno, že létala za ochrannou bariéru trati. Pomocí raycastu bychom tedy ve směru seslání mohli detekovat bariéru či terén a nedovolit kouzlu cestovat za ně.

Již v sekci 2.5 jsme uvedli plán podporovat dva různé režimy pohledu dozadu, přičemž hráč by si mohl v nastavení zvolit ten, který preferuje. Dva účastníci pak v dotazníku napsali, že by jim připadalo lepší, kdyby byl pohled dozadu aktivní, dokud drží klávesu, aby ji nemuseli mačkat podruhé pro ukončení. Odpovídá to tedy zamýšlenému druhému režimu. Jeho implementace se tak zdá opravdu důležitá.

Dalším vylepšením by byla úprava událostí na začátku závodu. Dva účastníci v dotazníku uvedli, že se mohli před začátkem každého závodu ještě chvilku pohybovat, i když měli aktivní přeskakování tréninku. Aktuálně je totiž před cutscene krátká prodleva, aby byla možnost zobrazit informace o nových oblastech. Jelikož je to zřejmě pro hráče matoucí, měli bychom se pokusit toto změnit. Bude to však vyžadovat úzké propojení několika částí (pokud v levelu není nová oblast, závod začne okamžitě, ale pokud je, zobrazí se tabulka s informacemi a hned po zavření se zahájí závod).

A nakonec jednomu z účastníků chyběla možnost navigovat se v uživatelském rozhraní zpět do předchozí obrazovky pomocí klávesy `ESC`. Nijak jsme nad tím neuvažovali, protože hráči ve hře stejně musí používat myš pro sesílání kouzel. Na základě zpětné vazby bychom však mohli podporu pro to přidat.

### Zvažované změny

Na závěr uvedeme několik příkladů zpětné vazby, která většinou není vyloženě negativní a nemá tak vysokou prioritu, ale nad kterou bychom přesto mohli uvažovat.

Jeden z účastníků měl problém s ovládáním hry, kdy nebyl schopný pohyb koštěte ovládat pouze jednou rukou. Z celé odpovědi se pak zdá, že by se problém dal vyřešit pouhým přemapováním kláves, ale účastník zřejmě nepřišel na to, že taková možnost ve hře je. Mohli bychom na ni tedy více upozornit, třeba v rámci tutoriálu. Účastník navíc uvedl, že je zvyklý používat herní ovladač. Pokud bychom tedy přidali podporu pro něj (což je v našich budoucích plánech), mohlo by to pomoci ještě více.

Dále bychom mohli lépe vizualizovat výsledky závodu ovlivněné penalizacemi. Jednomu účastníkovi se totiž zdálo matoucí, že v závodu se zobrazuje jedno umístění, ale ve výsledné tabulce pak jiné, protože se započtou penalizace. Chtěli bychom se tedy pokusit nejprve zobrazit tabulku s výsledky před aplikováním penalizací, následně pomocí tweenu přičíst penalizace a nakonec opět tweenem prohodit řádky do výsledného pořadí.

Již nezávisle na výsledcích experimentů jsme zvažovali vylepšení indikace následující obruče. Momentálně se nad ní objeví pohyblivá šipka a její ikonka se zvýrazní na minimapě. Chtěli bychom však přidat nějakou formu indikace přímo na obrazovce, aby měl hráč lepší přehled o tom, kde se obruč nachází relativně k jeho pozici a natočení. Díky tomu by mohly být obruče snáz navigovatelné i v případě, kdy jsou skryté mezi prvky prostředí. Jeden účastník navíc uvedl, že se občas ztratil a nevěděl, kde je další obruč. Pomocí minimapy se mu pak podařilo se zorientovat, ale je podle něj dost daleko od středu obrazovky a připadala mu malá. Doufáme tedy, že námi navržené vylepšení by mohlo této konkrétní situaci pomoci.

Poslední modifikací by mohla být možnost více interakcí se soupeři. Jeden z účastníků napsal, že má pocit, že když letí těsně vedle soupeře (např. ze startu či u nějaké obruče), nemá šanci ho nijak odtlačit. Prý by mohlo být zajímavé takovou možnost mít, pokud by se tím nerozbilo něco jiného. Přemýšleli jsme nad tím a napadlo nás přidat speciální akci kopnutí (inspirovala nás hra *Harry Potter for Kinect* [78], která se ovládá pohybem a je v ní možné strkat do soupeře). Měla by svou vlastní dobíjecí dobu, aby se nemohla používat neustále, ale její použití by nic nestálo a byla by k dispozici již od samotného začátku hry (oproti kouzlům). Akce by mohla přidat více napětí na začátek závodu, kdy se závodníci v hloučku přetahují o vedení, ale obzvlášť přínosná by byla pro plánovaný režim hry více hráčů (viz sekce 2.11).

# 6 Uživatelská dokumentace

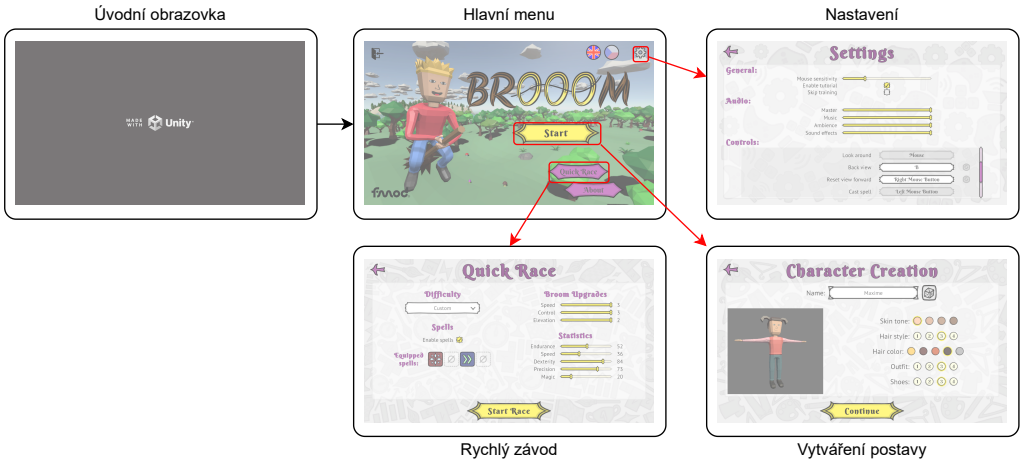
V této kapitole si nejprve popíšeme, jak hru spustit. Následně si představíme některé důležité obrazovky (se zbylými se hráč seznámí prostřednictvím tutoriálu přímo ve hře). Nakonec uvedeme referenční tabulky s přehledem ovládání a dostupných bonusů, kouzel a ocenění.

## 6.1 Spuštění hry

Spustitelnou verzi hry najdeme v elektronické příloze této práce ve složce `build/`, případně ji můžeme stáhnout na adrese `https://michelle2.itch.io/brooom`. Obdržíme tak archiv `Brooom.zip` obsahující build hry pro systém Windows. Po jeho extrakci získáme přístup k několika zásadním souborům:

- `Brooom.exe` – Samotný spustitelný soubor hry.
- `GoToSaveLocation` – Zástupce, který nás přenese do složky s uloženými daty hry (např. jejím stavem poté, co ji poprvé spustíme).
- `DeleteSaveFiles.bat` – Dávkový soubor systému Windows, který odstraní složku s daty hry (přístupnou pomocí zástupce `GoToSaveLocation`). Nahrazuje odinstalátor, jelikož se hra skutečně neinstaluje.
- `README.txt` – Soubor obsahující popis obsahu archivu.

Hru tedy zapneme spuštěním souboru `Brooom.exe`. Po spuštění nás uvítá úvodní obrazovka s nápisem „Made with Unity“, po které se za okamžik načte hlavní menu (viz sekce 6.1.1). Přehled zásadních obrazovek vidíme na obrázku 6.1 a v následujících podsekcích si je podrobněji popíšeme.




**Obrázek 6.1** Zásadní obrazovky po spuštění hry.

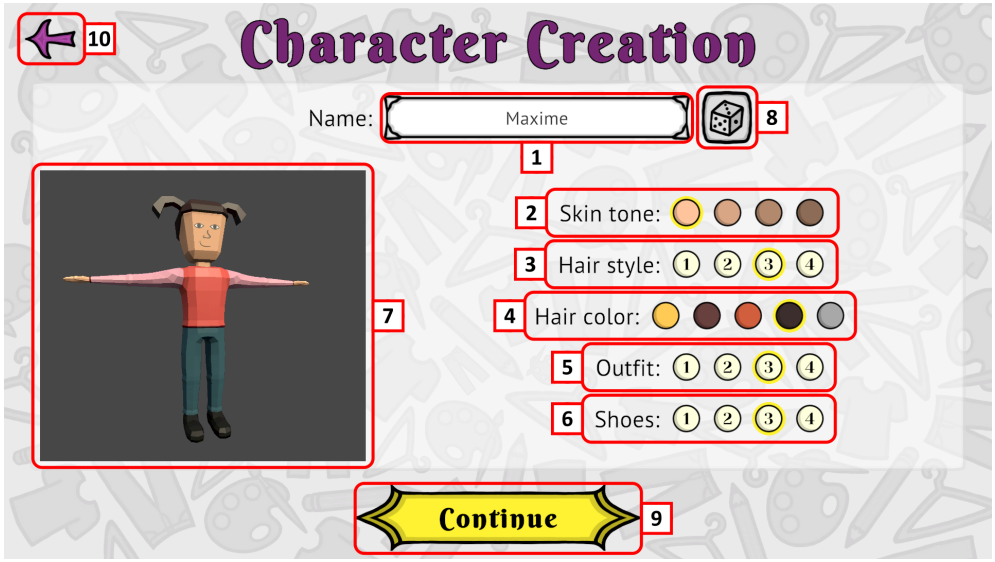
### 6.1.1 Hlavní menu

Podoba hlavního menu je znázorněna na obrázku 6.2. Pomocí tlačítek 1 vpravo nahoře si můžeme zvolit jazyk hry. Novou hru pak můžeme spustit tlačítkem 2, které nás přenese do obrazovky pro vytvoření postavy (viz sekce 6.1.2). Kromě základního kariérního režimu pak máme prostřednictvím tlačítka 3 k dispozici režim rychlého závodu (viz sekce 6.1.4). Tlačítkem 4 v pravém horním rohu si můžeme zobrazit nastavení (viz sekce 6.1.3) a tlačítkem 5 v levém horním rohu pak můžeme hru úplně ukončit.


**Obrázek 6.2** Podoba hlavního menu.

### 6.1.2 Vytváření postavy

Jestliže zahájíme novou hru z hlavního menu, zobrazí se nám možnosti pro volbu vzhledu postavy, se kterou budeme hrát. Podobu obrazovky vidíme na obrázku 6.3.




**Obrázek 6.3** Podoba obrazovky pro vytvoření postavy.

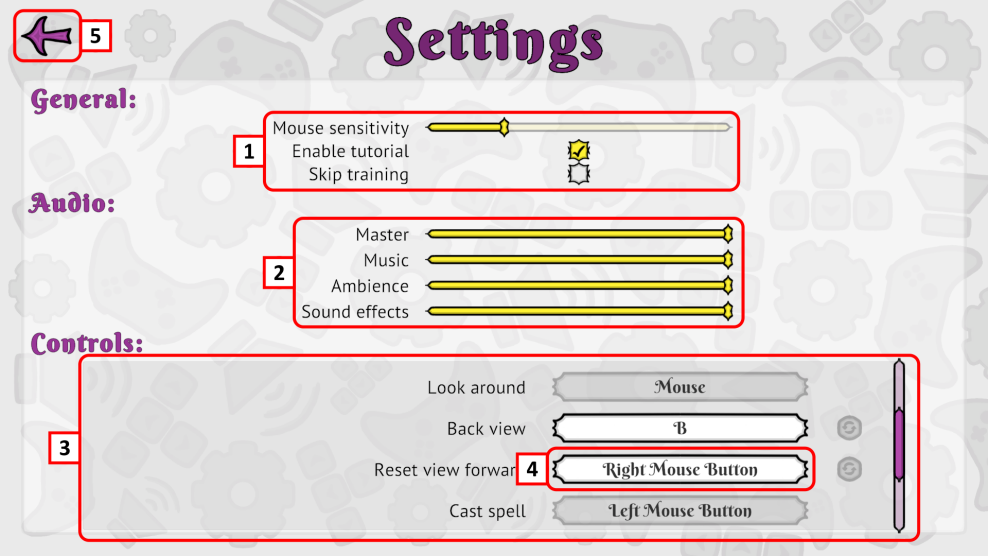
Ve vstupním poli 1 si zvolíme jméno postavy, následně můžeme vybrat odstín pleti 2, účes 3, barvu vlasů 4, barvu oblečení 5 a barvu bot 6. Provedené změny si můžeme prohlédnout v náhledu 7, přičemž stisknutím levého tlačítka myši a tažením můžeme postavou otáčet. Pomocí tlačítka 8 je pak možné zvolit zcela náhodné nastavení jména a vzhledu.

Jakmile jsme s volbami spokojení, tlačítkem 9 je můžeme potvrdit a skutečně zahájit hru. Dále nás bude hrou provázet tutoriál, pokud je zapnutý v nastavení hry (viz sekce 6.1.3).

Obrazovku můžeme kdykoliv opustit šipkou 10 vlevo nahoře.

### 6.1.3 Nastavení

Z hlavního menu nebo z menu pozastavené hry (vyvolaného pomocí klávesy ESC) můžeme zobrazit nastavení hry, jehož podoba je na obrázku 6.4.




**Obrázek 6.4** Podoba obrazovky s nastavením.

V horní části 1 můžeme zvolit citlivost myši. Dále můžeme určit, zda chceme povolit tutoriál a přeskakování tréninkové fáze před zahájením závodu. Hráčům, kteří hrají hru poprvé, vřele doporučujeme zachovat výchozí možnosti těchto dvou voleb (tj. zapnout tutoriál a naopak vypnout přeskočení tréninku).

Uprostřed pak můžeme nastavit hlasitost různých skupin audia 2. A nakonec dole se nachází přehled ovládání hry 3, které můžeme upravit stisknutím tlačítka 4 zobrazujícího aktuální mapování některé akce, čímž se zahájí interaktivní proces přemapování.

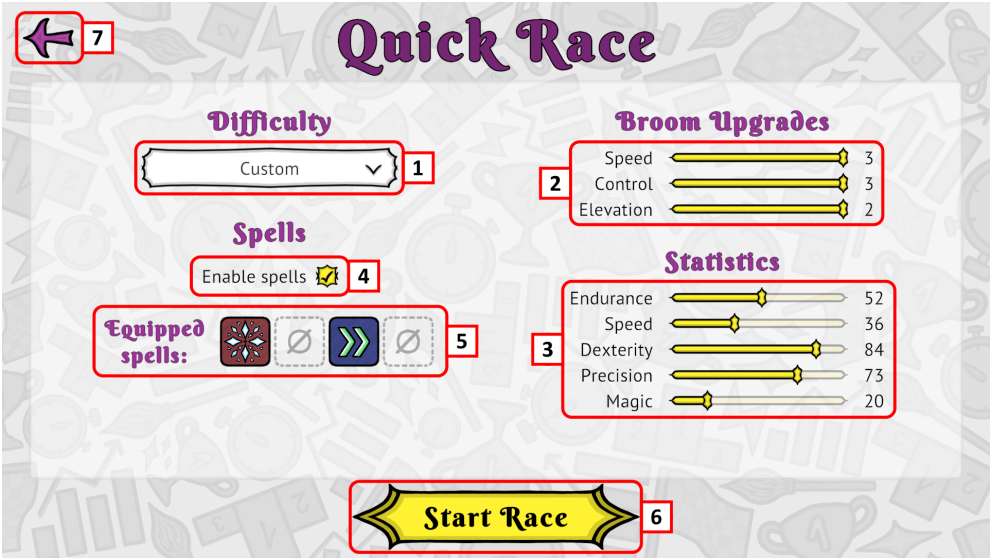
Z nastavení lze kdykoliv odejít zpět pomocí tlačítka 5 v levém horním rohu.

### 6.1.4 Rychlý závod

Kromě kariérního režimu můžeme z hlavního menu spustit také režim rychlého závodu. Ten se oproti normální hře liší tím, že si můžeme zvolit zcela libovolně počáteční parametry a současně se neukládá žádný pokrok ve hře. Jedná se tak pouze o jednorázový závod vhodný pro rychlé prozkoumání obsahu hry. Tento režim

doporučujeme hráčům až po dostatečném seznámení se se základními herními mechanikami v rámci tutoriálů v kariérním režimu.

Po zvolení rychlého závodu v hlavním menu se nám zobrazí možnosti nastavení závodu, které vidíme na obrázku 6.5.




**Obrázek 6.5** Podoba obrazovky s nastavením rychlého závodu.

V první řadě můžeme zvolit obtížnost 1, která pak ovlivňuje výslednou trať, chování soupeřů a také jejich i naše úrovně vylepšení koštěte. Pokud bychom však zvolili poslední možnost, tedy vlastní obtížnost, pak se vpravo zobrazí posuvníky pro manuální nastavení úrovní vylepšení 2 a hodnot statistik 3. Vybíráme tak úrovně vylepšení pro sebe, přičemž soupeři budou mít velmi podobné. Různé statistiky jsou pak zodpovědné za různé aspekty trati a navíc z jejich hodnot vychází také chování soupeřů.

Pomocí zaškrtávacího políčka 4 se pak můžeme rozhodnout, zda chceme povolit kouzla, či nikoliv. Pokud je povolíme, budou je smět používat také soupeři (zvolí se jim nějaká náhodná). Navíc se zobrazí sloty 5, do kterých si můžeme dosadit kouzla, která chceme využívat my sami. Na výběr přitom máme ze všech kouzel dostupných ve hře (viz tabulka 6.3 v následující sekci).

Pokud jsme s volbami spokojení, můžeme zahájit závod tlačítkem 6, případně se můžeme kdykoliv vrátit z této obrazovky zpět do hlavního menu pomocí šipky 7 vlevo nahoře.

## 6.2 Referenční tabulky

V této sekci uvádíme tabulky s informacemi, které mohou být pro hráče užitečné během hraní hry. Jedná se o přehled ovládání hry a všech dostupných bonusů, kouzel a ocenění.

### Ovládání

Tabulka 6.1 obsahuje přehled výchozího ovládání hry, které je hráči představeno také v rámci tutoriálu. V nastavení hry (viz sekce 6.1.3) je však možné mapování

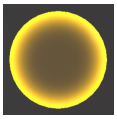
kláves změnit.

| Akce | Mapování |
|---|---|
| Pohyb vpřed | `W` |
| Zpomalení | `S` |
| Zatočení doleva/doprava | `A/D` |
| Stoupání | `Mezerník` |
| Klesání | `Levý Shift` |
| Změna kouzla | `Kolečko myši` |
| Seslání kouzla | `Levé tlačítko myši` |
| Rozhlížení se a zamíření | `Pohyb myši` |
| Pohled dozadu | `B` |
| Reset pohledu | `Pravé tlačítko myši` |
| Restart během tréninku | `Backspace` |
| Pozastavení hry | `ESC` |

**Tabulka 6.1** Výchozí ovládání letu na koštěti a sesílání kouzel v průběhu závodu.

## Bonusy

V tabulce 6.2 vidíme přehled všech bonusů, které se mohou vyskytovat na trati, včetně popisu jejich efektu.

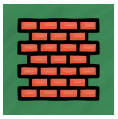
| Vzhled | Efekt po sebrání |
|---|---|
|  | Dočasně navýší maximální možnou rychlost závodníka. V trati se vyskytuje již od začátku hry. |
|  | Dočasně se zvýrazní přímá trajektorie ke třem následujícím bodům trati (tj. obručím a kontrolním bodům). V trati se vyskytuje již od začátku hry. |
|  | Okamžitě závodníkovi dobije 30 many. Zpřístupní se až po zakoupení prvního kouzla. |
|  | Závodníkovi se okamžitě dobijí všechna kouzla. Zpřístupní se až po zakoupení prvního kouzla. |

**Tabulka 6.2** Všechny dostupné bonusy ve hře.

## Kouzla

V demo verzi hry je dostupných celkem 10 různých kouzel, která si může hráč zakoupit v obchodě a následně je používat během závodů. Jejich přehled se všemi

důležitými informacemi vidíme v tabulce 6.3. Barva pozadí ikonky kouzla odpovídá kategorii, do které náleží (tj. kouzla sesílaná na sebe sama, kouzla ovlivňující soupeře, kouzla pro manipulaci s prostředím a kouzla pro vyčarování objektu).

| Ikonka | Název | Parametry | Efekt |
|---|---|---|---|
|  | Velox | Cena: 500 mincí<br>Cena seslání: 20 many<br>Dobíjecí doba: 15 s | Cíl: sesílající závodník<br>Kouzlo závodníka dočasně zrychlí, podobně jako zrychlující bonus. |
|  | Defensio | Cena: 750 mincí<br>Cena seslání: 50 many<br>Dobíjecí doba: 20 s | Cíl: sesílající závodník<br>Po krátkou dobu kouzlo závodníka ochrání proti všem blížícím se negativním kouzlům. |
|  | Relaxatio | Cena: 900 mincí<br>Cena seslání: 10 many<br>Dobíjecí doba: 25 s | Cíl: sesílající závodník<br>Kouzlo ihned odstraní veškeré negativní efekty působící na závodníka. |
|  | Translatio | Cena: 1250 mincí<br>Cena seslání: 80 many<br>Dobíjecí doba: 45 s | Cíl: sesílající závodník<br>Kouzlo závodníka rychle přemístí kus ve směru seslání. |
|  | Confusione | Cena: 1000 mincí<br>Cena seslání: 30 many<br>Dobíjecí doba: 30 s | Cíl: soupeř<br>Kouzlo na krátkou dobu omráčí soupeře, který tak nebude moci vůbec ovládat koště. |
|  | Congelatio | Cena: 1500 mincí<br>Cena seslání: 50 many<br>Dobíjecí doba: 45 s | Cíl: soupeř<br>Kouzlo na krátkou dobu zmrazí soupeře na místě. |
|  | Flante | Cena: 2000 mincí<br>Cena seslání: 70 many<br>Dobíjecí doba: 60 s | Cíl: soupeř + směr<br>Kouzlo odvane soupeře kus ve směru seslání. |
|  | Attractio | Cena: 1200 mincí<br>Cena seslání: 25 many<br>Dobíjecí doba: 60 s | Cíl: bonus<br>Kouzlo přitáhne bonus přímo k sesílajícímu. |
|  | Temere Commodum | Cena: 750 mincí<br>Cena seslání: 20 many<br>Dobíjecí doba: 20 s | Cíl: pozice/směr<br>Kouzlo vyčaruje ve směru seslání náhodný bonus. |
|  | Materia Muri | Cena: 1500 mincí<br>Cena seslání: 40 many<br>Dobíjecí doba: 20 s | Cíl: pozice/směr<br>Kouzlo vyčaruje zeď na pozici ve směru seslání. |

**Tabulka 6.3** Všechna dostupná kouzla ve hře.

## Ocenění

Za dosažení různých menších cílů ve hře může hráč získávat ocenění. Některá jsou jen jednorázová, ale jiná mají několik stupňů. Přehled všech ocenění je zachycen v tabulkách 6.4 a 6.5, přičemž použité ikonky odpovídají vždy nejvyššímu (tj. platinovému) stupni, znázorněnému pomocí barvy pozadí. Jako poslední je pak uvedena ikonka, která se zobrazuje, pokud je dané ocenění prozatím neznámé.

| Ikonka | Název | Popis |
|---|---|---|
|  | Dobrá muška | Toto ocenění je uděleno za prolétnutí 10, 100, 400, 1 000 obručemi. |
|  | Krátkozraký | Toto ocenění je uděleno za minutí 5, 10, 20, 50 obručí. |
|  | Straka | Toto ocenění je uděleno za sebrání 10, 50, 200, 500 bonusů. |
|  | Nemotorný | Toto ocenění je uděleno za náraz do překážky 10, 20, 50, 100 krát. |
|  | Dveře dokořán | Toto ocenění je uděleno za odemčení všech dostupných oblastí. |
|  | Zcestovalý | Toto ocenění je uděleno za navštívení všech dostupných oblastí. |
|  | Vytrvalý | Toto ocenění je uděleno za dokončení 5, 10, 50, 100 závodů. |
|  | Strašpytel | Toto ocenění je uděleno za vzdání 5, 10, 15, 20 závodů. |
|  | Nedosažitelný | Toto ocenění je uděleno za umístění se na prvním místě 3, 5, 10, 15 krát v řadě. |
|  | Smolař | Toto ocenění je uděleno za umístění se na posledním místě 3, 5, 10, 15 krát v řadě. |
|  | Důsledná příprava | Toto ocenění je uděleno za otestování té samé trati 10 krát během tréninkové fáze. |

**Tabulka 6.4** Všechna dostupná ocenění ve hře (první část).

| Ikonka | Název | Popis |
|---|---|---|
|  | Lukrativní byznys | Toto ocenění je uděleno za výdělek 1 000, 5 000, 12 000, 30 000 mincí. |
|  | Skrblík | Toto ocenění je uděleno za našetření 1 000, 2 000, 5 000, 10 000 mincí. |
|  | Nejlepší vybavení | Toto ocenění je uděleno za maximální vylepšení koštěte. |
|  | Mocný mág | Toto ocenění je uděleno za seslání 10, 50, 100, 200 kouzel. |
|  | Renesanční člověk | Toto ocenění je uděleno za zakoupení všech dostupných kouzel. |
|  | Doleť až na vrchol! | Toto ocenění je uděleno za dosažení první příčky v globálním žebříčku. |
|  | Nadčlověk | Toto ocenění je uděleno za dosažení nejvyšší možné hodnoty v libovolné statistice. |
|  | Neznámé ocenění | Toto ocenění zatím není známé. |

**Tabulka 6.5** Všechna dostupná ocenění ve hře (druhá část).

# 7 Závěr

V této závěrečné kapitole se ohlédneme za procesem vytváření naší hry. Nejprve vyhodnotíme, jak dobře se nám podařilo naplnit požadavky a cíle, které jsme si stanovili v sekci 1.3. Následně nastíníme, kam bychom chtěli ve vývoji hry dále směřovat.

## 7.1 Shrnutí a zhodnocení

Výstupem této práce je demo verze hry, která svým obsahem splňuje všechny požadavky definované v sekci 1.3, jak vyplývá z kapitoly 4 popisující výslednou implementaci:

- Hra byla vytvořena pomocí herního enginu Unity (**P1**) a běží na PC se systémem Windows (**P2**), což jsme ověřili také experimentálně v sekci 5.2. Pokud to dávalo smysl, volili jsme během vývoje tvorbu vlastního obsahu a vlastních řešení různých systémů (**P7**).
- Ve hře hráč závodí na koštěti (**P8**), postupně si buduje kariéru (**P9**) a vylepšuje své schopnosti (**P11**) s cílem stát se nejlepším závodníkem na světě (**P12**).
- Hra se přizpůsobuje schopnostem hráče (**P13**) tím, že se jednotlivé tratě procedurálně generují s parametry odpovídajícími úrovni hráče (**P18**–**P19**) a soupeři dělají různé množství chyb (**P21**–**P22**).
- Během závodů se hráč může volně pohybovat (**P14**) v rámci trati vyhrazené obručemi a ochrannou bariérou (**P16**), u toho sbírat bonusy (**P20**) a používat minimapu na obrazovce k usnadnění navigace (**P17**).
- Za dobré umístění v závodě hráč získává mince (**P10**), za které si může zakoupit vylepšení koštěte (**P15**) či kouzla, která pak může používat v závodech (**P23**).
- Do závodu si hráč může zvolit maximálně 4 různá kouzla (**P25**), která pak může sesílat za letu (**P24**) a rozhlížením se pro ně vybírat vhodný cíl (**P24a**). Pravým tlačítkem myši je možné resetovat pohled do výchozího směru (**P24b**).
- Sesílání kouzel spotřebovává manu a kouzla se poté určitou dobu dobíjí, než je možné je seslat znovu (**P26**). Některá kouzla se sesílají na soupeře (**P27**) a pokud takové kouzlo míří na hráče, na obrazovce se ukáže indikátor (**P28**).
- Veškerý pokrok v kariérním režimu hry se perzistentně ukládá (**P3**). Navíc má však hráč možnost režimu rychlého závodu, který je jednorázový a bezestavový (**P29**).
- Ve hře je možné zvolit si jazyk (**P4**). Uživatelská rozhraní jsou responzivní a přehledná (**P6**), což jsme ověřili také experimentálně (viz sekce 5.2.3).

Dále je součástí hry audio, které dle účastníků experimentu popsaného v sekci 5.2.3 odpovídá žánru a posiluje zážitek ze hry (**P5**).

- Ve hře se nachází scéna (dostupná pouze pomocí cheatů) demonstrující procedurální generování levelů (**P30**).

Kromě dílčích požadavků jsme během vývoje hry naplnili také všechny stanovené cíle. Nyní každý z nich podrobněji rozebereme.

**C1 Vytvořit návrh hry v podobě GDD (Game Design Document).**

Ještě než jsme s vývojem začali, sepsali jsme si GDD, abychom měli všechny zamýšlené části hry přehledně na jednom místě. Poté jsme jej průběžně doplňovali a aktualizovali, aby pokaždé zachycoval aktuální stav vývoje. Jeho poslední verze se pak nachází v elektronické příloze této práce ve složce `gdd/` (a stručný popis je přímo v příloze A.1).

**C2a Vytvořit demo verzi hry, která splňuje vytyčené požadavky P1–P30.**

Naplnění tohoto cíle jsme popsali již výše. Výsledná hra svým obsahem a způsobem implementace splňuje všechny vytyčené požadavky.

**C2b Vytvořit demo verzi hry, která je snadno rozšiřitelná (z hlediska nové funkcionality i nového obsahu).**

Během vývoje jsme se snažili vše navrhovat tak, aby to bylo co nejlépe rozšiřitelné. Kapitola 4 mimo jiné popisuje výsledné hierarchie tříd, použití prefabů a `ScriptableObject`ů. V kapitole 3 navíc uvádíme argumenty pro některá rozhodnutí.

Dále jsme se pokusili umožnit změnu co nejvíce parametrů v editoru (např. chybovost soupeřů, generování levelu), díky čemuž je navíc hra snadno modifikovatelná.

**C3 Vytvořit pro hru tutoriál.**

Součástí hry je také tutoriál, který hráče postupně seznamuje s důležitými herními mechanikami. Je přitom rozdělen na části (popsané v sekci 2.10), aby bylo možné hráčům zobrazit vždy jen to, co zrovna v danou chvíli potřebují vědět. Implementace pak byla popsána v sekci 4.9.

Během tutoriálu se využívá kombinace zobrazování textu na obrazovce a zvýrazňování relevantní části obrazovky. Některé kroky jsou navíc interaktivní, tzn. že hráči nemohou pokročit dál, dokud neprovedou požadovanou akci. Z výsledků experimentu (viz sekce 5.2.3) se pak zdá, že jsou tutoriály navržené dobře a hráčům skutečně pomáhají. Účastníci hodnotili velmi pozitivně, že je hra dobře učí, jak se má ovládat, a poskytuje potřebné informace srozumitelnou formou. Na základě zpětné vazby jsme pouze upřesnili instrukce v úvodním tutoriálu (viz sekce 5.3).

**C4 Provést experimenty, sbírat během nich herní analytiky a zapracovat zpětnou vazbu.**

V kapitole 5 jsme se věnovali popisu všech provedených experimentů. Během druhé části jsme nechali účastníky hrát hru zcela nezávisle po dobu alespoň 30 minut. Přitom jsme sbírali informace o důležitých událostech ve hře, které jsme následně analyzovali (viz sekce 5.2.4). Z otevřených otázek dotazníku jsme pak posbírali zpětnou vazbu (viz sekce 5.3). Nejdůležitější poznámky jsme již zapracovali a řadu dalších námětů plánujeme přidat do budoucna. Navíc jsme zapracovali také zpětnou vazbu z počátečního interního testování (viz sekce 5.1).

**C5 Zveřejnit demo verzi na `itch.io`.**

Z výsledné demo verze hry jsme vytvořili build, který jsme následně zveřejnili na `https://michelle2.itch.io/brooom`. Kromě něj jsme na stránku přidali také stručný popis a snímky ze hry. V budoucnu vytvoříme a přiložíme ještě krátký trailer.

## 7.2 Budoucí plány

V rámci této práce se nám podařilo vytvořit poměrně obsáhlou demo verzi, která zpracovává zásadní herní mechaniky a dává představu o tom, jak by mohla výsledná hra vypadat. Z výsledků experimentu se pak ukázalo, že by mohla mít hra potenciál zaujmout netriviální množinu lidí a mělo by tedy smysl v jejím vývoji nadále pokračovat.

V rámci první série experimentů jsme posbírali cennou zpětnou vazbu, kterou bychom chtěli dále do hry zapracovat. Navíc jsme v kapitole 2 nastínili celou řadu obsahu nad rámec demo verze. Všechny zmíněné části bychom rádi v budoucnu do hry přidali. Nyní však uvedeme pouze několik významnějších milníků, kterých bychom chtěli v budoucím vývoji hry dosáhnout.

- *Funkční odlišení oblastí* – Chtěli bychom do trati přidat dynamické překážky, které by byly specifické pro konkrétní oblasti. Díky tomu by se oblasti mezi sebou nelišily pouze vzhledem. Mohlo by se jednat jak o fyzické překážky (např. chapadla vynořující se z vody), tak o environmentální vlivy (např. tornádo přitahující hráče, písečná bouře omezující viditelnost).

- *Režim hry více hráčů* – Na základě výsledků experimentu se ukázalo, že by velká řada účastníků uvítala možnost hry více hráčů. Rádi bychom tedy tento režim přidali, ale nejprve bude třeba důkladně zvážit jeho návrh a způsob provedení.

- *Lepší generování levelů* – Chtěli bychom se pokusit implementovat některé sofistikovanější postupy procedurálního generování levelů, které by umožnily větší závislosti mezi tratí a prostředím. Díky tomu by mohly být výsledné levely zajímavější a více přizpůsobené žánru hry. Současně bychom chtěli přidat další optimalizace levelu (např. LOD prvků prostředí).

- *WebGL build* – Prozatím jsme cílili pouze na systém Windows jako cílovou platformu, ale do budoucna bychom chtěli podporu rozšířit také na WebGL.

Bude to ovšem vyžadovat jisté úpravy. Jedním problémem je kurzor myši, pro který jsme použili vlastní obrázek, ale při stisknutí `ESC` pro pozastavení hry se změní zpět na ten výchozí. Druhým problémem je pak zvuk, který se zasekává během načítání scén, jelikož využíváme FMOD, který je ve WebGL pouze jednovláknový. Pro oba zmíněné problémy již máme promyšlená možná řešení, ale museli bychom je ještě detailněji prozkoumat.

- *Zveřejnění na Steam* – Časem bychom chtěli hru zveřejnit na některé významnější platformě pro distribuci her, např. jako hru s předběžným přístupem na Steam. Ještě předtím bychom ji však významně rozšířili a provedli další sérii experimentů pro zhodnocení těchto změn.

# Literatura


1. Harha Studios. *Broom Race – Aplikace na Google Play* [online]. [cit. 2025-05-16]. Dostupné z: https://play.google.com/store/apps/details?id=com.JuhaniPaaso.BroomRace&hl=en&gl=US.
2. Dotpixel. *Broom Race – Aplikace na Google Play* [online]. [cit. 2023-06-06]. Dostupné z: https://play.google.com/store/apps/details?id=co.dotpixel.broomrace.
3. SMU Guildhall. *Hex Rally Racers ve službě Steam* [online]. [cit. 2025-05-16]. Dostupné z: https://store.steampowered.com/app/1903640/Hex_Rally_Racers/.
4. Přispěvatelé Wikipedie. *Mario Kart — Wikipedie: Otevřená encyklopedie* [online]. [cit. 2025-05-30]. Dostupné z: https://en.wikipedia.org/wiki/Mario_Kart.
5. Independent Arts Software. *Bibi Blocksberg ™ – Big Broom Race 3 ve službě Steam* [online]. [cit. 2025-05-16]. Dostupné z: https://store.steampowered.com/app/947260/Bibi_Blocksberg___Big_Broom_Race_3/.
6. UNIVRS, Inc. *Little Witch Academia: VR Broom Racing* [online]. [cit. 2025-05-16]. Dostupné z: https://lwa-vr.com/en/.
7. UNIVRS, Inc. *Little Witch Academia: VR Broom Racing ve službě Steam* [online]. [cit. 2025-05-16]. Dostupné z: https://store.steampowered.com/app/1294450/Little_Witch_Academia_VR_Broom_Racing/.
8. Avalanche Software; Warner Bros. Games. *Hogwarts Legacy – Home* [online]. [cit. 2025-05-16]. Dostupné z: https://www.hogwartslegacy.com/en-gb.
9. Unbroken Studios; Warner Bros. Games. *Home | Harry Potter: Quidditch Champions* [online]. [cit. 2025-05-16]. Dostupné z: https://www.quidditchchampions.com/en-gb.
10. Unity Technologies. *Real-Time 3D Development Platform and Editor| Unity* [online]. [cit. 2025-05-19]. Dostupné z: https://unity.com/products/unity-engine.
11. Pears, Max. *Design Pillars – The Core of Your Game* [online]. 2017-09-02. [cit. 2025-05-27]. Dostupné z: https://www.maxpears.com/2017/09/02/design-pillars-the-core-of-your-game/.
12. Csikszentmihalyi, M. *Beyond Boredom and Anxiety*. San Francisco: Jossey-Bass, 1975.
13. Schell, J. *The Art of Game Design: A Book of Lenses*. CRC Press, 2014. Druhé vydání.
14. Nadeo; Ubisoft. *Trackmania: Welcome to the club* [online]. [cit. 2025-05-29]. Dostupné z: https://www.trackmania.com/?lang=en.
15. Unknown Worlds Entertainment. *Subnautica ve službě Steam* [online]. [cit. 2025-05-29]. Dostupné z: https://store.steampowered.com/app/264710/Subnautica/.

16. Blizzard Entertainment. *World of Warcraft* [online]. [cit. 2025-05-29]. Dostupné z: https://worldofwarcraft.blizzard.com/en-us/.

17. Bohemia Interactive. *Ylands* [online]. [cit. 2025-05-29]. Dostupné z: https://ylands.com/.

18. Sporka, Adam J. *Music Design for Game Development Studios*. Acaremi, 2022. isbn 978-80-11-02314-0.

19. Přispěvatelé Redditu. *Reddit – I hate the broom controls on PC* [online]. [cit. 2025-05-30]. Dostupné z: https://www.reddit.com/r/hogwartslegacyJKR/comments/12aoo1p/i_hate_the_broom_controls_on_pc/.

20. Electronic Arts Inc. *The Sims Video Games – Official EA Site* [online]. [cit. 2025-05-30]. Dostupné z: https://www.ea.com/games/the-sims.

21. Unity Technologies. *Unity - Manual: Audio in WebGL* [online]. [cit. 2025-05-30]. Dostupné z: https://docs.unity3d.com/2021.3/Documentation/Manual/webgl-audio.html.

22. Unity Technologies. *Unity documentation* [online]. [cit. 2025-06-03]. Dostupné z: https://docs.unity.com/en-us.

23. Unity Technologies. *Unity – Manual: Prefabs* [online]. [cit. 2025-06-09]. Dostupné z: https://docs.unity3d.com/2021.3/Documentation/Manual/Prefabs.html.

24. Nystrom, Robert. *Prototype – Design Patterns Revisited (Game Programming Patterns)* [online]. [cit. 2025-06-21]. Dostupné z: https://gameprogrammingpatterns.com/prototype.html.

25. Unity Technologies. *Unity – Manual: ScriptableObject* [online]. [cit. 2025-06-09]. Dostupné z: https://docs.unityhttps://docs.unity3d.com/2021.3/Documentation/Manual/class-ScriptableObject.html.

26. Přispěvatelé Wikipedie. *Publish–subscribe pattern — Wikipedie: Otevřená encyklopedie* [online]. [cit. 2025-06-09]. Dostupné z: https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern.

27. Unity Technologies. *Make assets addressable – Unity Learn* [online]. [cit. 2025-06-09]. Dostupné z: https://learn.unity.com/tutorial/make-assets-addressable.

28. Unity Technologies. *Unity – Scripting API: Resources* [online]. [cit. 2025-06-09]. Dostupné z: https://docs.unity3d.com/2021.3/Documentation/ScriptReference/Resources.html.

29. Unity Technologies. *Unity – Manual: Streaming Assets* [online]. [cit. 2025-06-09]. Dostupné z: https://docs.unity3d.com/2021.3/Documentation/Manual/StreamingAssets.html.

30. Unity Technologies. *Unity – Scripting API: PlayerPrefs* [online]. [cit. 2025-06-09]. Dostupné z: https://docs.unity3d.com/2021.3/Documentation/ScriptReference/PlayerPrefs.html.

31. Unity Technologies. *Unity – Scripting API: JsonUtility* [online]. [cit. 2025-06-09]. Dostupné z: https://docs.unity3d.com/6000.1/Documentation/ScriptReference/JsonUtility.html.

32. Přispěvatelé Wikipedie. *Perlin noise — Wikipedie: Otevřená encyklopedie* [online]. [cit. 2025-06-04]. Dostupné z: https://en.wikipedia.org/wiki/Perlin_noise.

33. Přispěvatelé Wikipedie. *Gaussian filter — Wikipedie: Otevřená encyklopedie* [online]. [cit. 2025-06-21]. Dostupné z: https://en.wikipedia.org/wiki/Gaussian_filter#Digital_implementation.

34. Přispěvatelé Wikipedie. *Voronoi diagram — Wikipedie: Otevřená encyklopedie* [online]. [cit. 2025-06-04]. Dostupné z: https://en.wikipedia.org/wiki/Voronoi_diagram.

35. Přispěvatelé PCG Wiki. *Whittaker Diagram – Procedural Content Generation Wiki* [online]. [cit. 2025-06-04]. Dostupné z: http://pcg.wikidot.com/pcg-algorithm:whittaker-diagram.

36. Cochran, William G. *Sampling Techniques*. John Wiley & Sons, 1977. Třetí vydání.

37. Přispěvatelé Wikipedie. *Random walk — Wikipedie: Otevřená encyklopedie* [online]. [cit. 2025-06-04]. Dostupné z: https://docs.unity3d.com/2021.3/Documentation/Manual/UnderstandingFrustum.html.

38. Unity Technologies. *Unity – Manual: Understanding the View Frustum* [online]. [cit. 2025-06-09]. Dostupné z: https://en.wikipedia.org/wiki/Random_walk.

39. Unity Technologies. *Unity – Scripting API: Camera.layerCullDistances* [online]. [cit. 2025-06-09]. Dostupné z: https://docs.unity3d.com/2021.3/Documentation/ScriptReference/Camera-layerCullDistances.html.

40. Unity Technologies. *Unity – Manual: Level of Detail (LOD) for meshes* [online]. [cit. 2025-06-09]. Dostupné z: https://docs.unity3d.com/2021.3/Documentation/Manual/LevelOfDetail.html.

41. Unity Technologies. *Unity – Manual: Billboard Renderer component* [online]. [cit. 2025-06-09]. Dostupné z: https://docs.unity3d.com/2021.3/Documentation/Manual/class-BillboardRenderer.html.

42. Unity Technologies. *Unity – Manual: Using occlusion culling with dynamic GameObjects* [online]. [cit. 2025-06-09]. Dostupné z: https://docs.unity3d.com/2021.3/Documentation/Manual/occlusion-culling-dynamic-gameobjects.html.

43. Melder, Nic. A Rubber-Banding System for Gameplay and Race Management. In: Rabin, Steve (ed.). *Game AI Pro*. CRC Press, 2013. Dostupné také z: https://www.gameaipro.com/GameAIPro/GameAIPro_Chapter42_A_Rubber-Banding_System_for_Gameplay_and_Race_Management.pdf.

44. Jiménez, Eduardo. *The Pure Advantage: Advanced Racing Game AI* [online]. 2009-02-03. [cit. 2025-06-05]. Dostupné z: https://www.gamedeveloper.com/design/the-pure-advantage-advanced-racing-game-ai.

45. Jiménez, E.; Mitchell, K.; Serón, F. J. Capture and Analysis of Racing-Gameplay Metrics. *IEEE Software*. 2011, roč. 28, č. 6, s. 46–52.

46. Côté, Carle. Reactivity and Deliberation in Decision-Making Systems. In: Rabin, Steve (ed.). *Game AI Pro*. CRC Press, 2013. Dostupné také z: https://www.gameaipro.com/GameAIPro/GameAIPro_Chapter11_Reactivity_and_Deliberation_in_Decision-Making_Systems.pdf.

47. Dawe, Michael; Gargolinski, Steve; Dicken, Luke; Humphreys, Troy; Mark, Dave. Behavior Selection Algorithms: An Overview. In: Rabin, Steve (ed.). *Game AI Pro*. CRC Press, 2013. Dostupné také z: https://www.gameaipro.com/GameAIPro/GameAIPro_Chapter04_Behavior_Selection_Algorithms.pdf.

48. Game Maker's Toolkit. *The Genius AI Behind The Sims* [online]. 2023-06-30. [cit. 2025-06-05]. Dostupné z: https://www.youtube.com/watch?v=9gf2MT-IOsg.

49. Reynolds, Craig W. Steering Behaviors For Autonomous Characters. In: *Game Developers Conference*. San Francisco, CA, 1999.

50. Fray, Andrew. Context Steering: Behavior-Driven Steering at the Macro Scale. In: Rabin, Steve (ed.). *Game AI Pro 2*. CRC Press, 2015. Dostupné také z: https://www.gameaipro.com/GameAIPro2/GameAIPro2_Chapter18_Context_Steering_Behavior-Driven_Steering_at_the_Macro_Scale.pdf.

51. Radaev, Eugene. *Colorlink: Unity editor tool for quickly changing the game's color palette* [online]. [cit. 2025-06-10]. Dostupné z: https://github.com/leth4/Colorlink.

52. Audiokinetic. *Wwise* [online]. [cit. 2025-06-12]. Dostupné z: https://www.audiokinetic.com/en/wwise/overview/.

53. Firelight Technologies Pty Ltd. *FMOD* [online]. [cit. 2025-05-30]. Dostupné z: https://www.fmod.com/.

54. Nystrom, Robert. *Singleton – Design Patterns Revisited (Game Programming Patterns)* [online]. [cit. 2025-06-10]. Dostupné z: https://gameprogrammingpatterns.com/singleton.html.

55. *Unity Issue Tracker – Scene Async Load Is Not Async* [online]. [cit. 2025-06-10]. Dostupné z: https://issuetracker.unity3d.com/issues/async-load-is-not-async.

56. Unity Technologies. *About Localization | Localization | 1.5.4* [online]. [cit. 2025-06-13]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.localization@1.5/manual/index.html.

57. Borgers, Ben. *opensheet: An API to get a Google Sheet as JSON, no authentication required.* [online]. [cit. 2025-06-13]. Dostupné z: https://github.com/benborgers/opensheet.

58. Staskevicius, Norbert. *simple-tooltip: A very simple, free Unity asset that allows you to add a tooltip component to any game object* [online]. [cit. 2025-06-13]. Dostupné z: https://github.com/snorbertas/simple-tooltip.

59. Giardini, Daniele. *DOTween (HOTween v2)* [online]. [cit. 2025-06-11]. Dostupné z: https://dotween.demigiant.com/.

60. Unity Technologies. *Unity – Manual: Order of execution for event functions* [online]. [cit. 2025-06-11]. Dostupné z: https://docs.unity3d.com/Manual/execution-order.html.

61. Přispěvatelé Unity fóra. *Animation event not triggering – Unity Engine – Unity Discussions* [online]. [cit. 2025-06-11]. Dostupné z: https://discussions.unity.com/t/animation-event-not-triggering/655247.

62. Unity Technologies. *TextMesh Pro Documentation* [online]. [cit. 2025-06-19]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.ugui@2.0/manual/TextMeshPro/index.html.

63. Unity Technologies. *About Cinemachine* [online]. [cit. 2025-06-19]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.cinemachine@2.8/manual/index.html.

64. Unity Technologies. *Newtonsoft Json Unity Package* [online]. [cit. 2025-06-19]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.nuget.newtonsoft-json@3.1/manual/index.html.

65. Firelight Technologies Pty Ltd. *FMOD for Unity (2.02)* [online]. [cit. 2025-06-19]. Dostupné z: https://assetstore.unity.com/packages/tools/audio/fmod-for-unity-2-02-161631.

66. Font, Frederic; Roma, Gerard; Serra, Xavier. Freesound technical demo. In: *Proceedings of the 21st ACM international conference on Multimedia*. ACM, 2013.

67. *Google Fonts* [online]. [cit. 2025-06-19]. Dostupné z: https://fonts.google.com/.

68. Unity Technologies. *Input System* [online]. [cit. 2025-06-23]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.5/manual/index.html.

69. hexdump. *Executing first scene in build settings when pressing play button in editor? – Unity Engine – Unity Discussions* [online]. [cit. 2025-06-27]. Dostupné z: https://discussions.unity.com/t/executing-first-scene-in-build-settings-when-pressing-play-button-in-editor/489673/10.

70. Costa, Pablo. *SceneAutoLoader with additive loading* [online]. [cit. 2025-06-27]. Dostupné z: https://gist.github.com/pdcp1/6e7f7315fc000de55676.

71. Drachen, Anders; Mirza-Babaei, Pejman; Nacke, Lennart E. *Game User Research*. Oxford University Press, 2018.

72. Azarova, Mayya. *The Hawthorne Effect or Observer Bias in User Research* [online]. [cit. 2025-07-05]. Dostupné z: https://www.nngroup.com/articles/hawthorne-effect-observer-bias-user-research/.

73. Hodent, Celia. *The Gamer's Brain: How Neuroscience and UX can impact Design (GDC 2015)* [online]. [cit. 2025-07-06]. Dostupné z: https://www.youtube.com/watch?v=XIpDLa585ao&t=710s.

74. Phan, Mikki; Keebler, Joseph; Chaparro, Barbara. The Development and Validation of the Game User Experience Satisfaction Scale (GUESS). *Human Factors: The Journal of the Human Factors and Ergonomics Society*. 2016, roč. 58. Dostupné z DOI: 10.1177/0018720816669646.

75. Přispěvatelé Wikipedie. *Likertova škála — Wikipedie: Otevřená encyklopedie* [online]. [cit. 2025-07-13]. Dostupné z: https://cs.wikipedia.org/wiki/Likertova_%C5%A1k%C3%A1la.

76. Unity Technologies. *Unity Analytics* [online]. [cit. 2025-07-09]. Dostupné z: https://docs.unity.com/ugs/en-us/manual/analytics/manual/overview.

77. GameAnalytics. *Player Insights at Your Fingertip – GameAnalytics* [online]. [cit. 2025-07-09]. Dostupné z: https://www.gameanalytics.com/analytics.

78. Přispěvatelé Wikipedie. *Harry Potter for Kinect — Wikipedie: Otevřená encyklopedie* [online]. [cit. 2025-07-10]. Dostupné z: https://en.wikipedia.org/wiki/Harry_Potter_for_Kinect.

# A Přílohy

Veškeré elektronické přílohy této práce jsou součástí archivu `prilohy.zip`. Vnitřní struktura tohoto archivu pak vypadá následovně:

.
- **build/** – spustitelný soubor hry spolu s doplňkovými soubory
- **experimenty/** – adresář se soubory k experimentům
  - **build/** – verze hry použité pro experimenty spolu s aplikací pro testování srozumitelnosti ikonek
  - **data/** – soubory obsahující herní analytiky posbírané během experimentu
  - **grafy/** – grafy vytvořené z dat posbíraných v experimentu
  - **vyhodnoceni/** – tabulky s vyhodnocením dat z experimentů
  - **zpracovani/** – zdrojový kód použitý pro zpracování dat z experimentu a pro vytvoření grafů
- **gdd/** – Game Design Document popsaný více v sekci A.1
- **src/** – adresář obsahující implementační části práce
  - **Audio/** – FMOD projekt s audio událostmi, používá jej Unity projekt
  - **Brooom/** – Unity projekt samotné hry Brooom (vytvořený v editoru verze 2021.3.23f1)
- **tex/** – adresář obsahující zdrojové soubory textu práce pro LaTeX
  - **src/** – samotné .tex soubory
  - **img/** – obrázky použité v práci
- **abstract-cs.pdf** – soubor obsahující český abstrakt práce
- **abstract-en.pdf** – soubor obsahující anglický abstrakt práce
- **thesis.pdf** – soubor obsahující text práce ve formátu PDF/A
- **README.txt** – soubor popisující strukturu elektronických příloh

## A.1 Game Design Document (GDD)

Ještě před zahájením implementace jsme si sepsali GDD, který jsme pak neustále udržovali aktuální. Díky tomu zachycuje jak finální vizi, tak aktuální stav vývoje (pomocí různobarevných značek). Lze ho najít v elektronické příloze v podadresáři `gdd/`.

Kapitola *Implementace* popisuje nápady, jak by bylo možné implementovat různé prvky hry. Kapitola *Další nápady* pak uvádí seznam dalších prvků, které by mohly být do hry přidány, ale nebylo o tom ještě rozhodnuto a nebyly tedy vůbec součástí prvotního návrhu.

## A.2 Seznam použitých licencovaných zdrojů

Většina zdrojů, které jsme ve hře použili, je buď naší vlastní výroby, nebo zveřejněná pod licencí Creative Commons 0, která umožňuje jejich volné použití. Některé však vyžadují uvedení původu, a tak je zmíníme v této příloze:

- https://freesound.org/s/252681/ – Magic sound 1.wav, *JomelleJager* (Attribution 3.0),
- https://freesound.org/s/219534/ – wind tree silver birch 02 140215.wav, *klankbeeld* (Attribution 4.0),
- https://freesound.org/s/73370/ – Bees.wav, *Benboncan* (Attribution 4.0),
- https://freesound.org/s/531713/ – bees.wav, *paulotantastico* (Attribution 3.0),
- https://freesound.org/s/696181/ – 20230627.all.the.bees.wav, *dobroide* (Attribution 4.0).