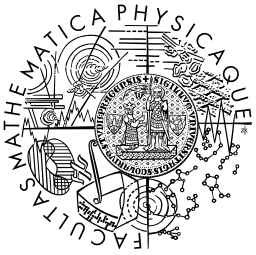


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Karolína Kuchyňová

Master of the Carpet: 3D akční hra z pohledu hráče

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce:	Mgr. Pavel Ježek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Obecná informatika

Praha 2018

Prohlašuji, že jsem tuto bakalářskou práci vypracovala 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 Brně dne 17.7.2018 Podpis autora

Ráda bych poděkovala vedoucímu své bakalářské práce Mgr. Pavlu Ježkovi, Ph.D. za čas, který mé práci věnoval, za to, že byl vždy ochoten mi se vším poradit, a především za to, že mi dodával optimismus ve chvílích, kdy jsem propadala skepsi a pochybovala, že se mi podaří práci úspěšně dokončit.

Největší díky pak patří mé rodině za neocenitelnou podporu a pomoc se vším od testování herních úrovní přes návrh modelů objektů až po doplňování čárek za věty vedlejší vložené.

Název práce: Master of the Carpet: 3D akční hra z pohledu hráče

Autor: Karolína Kuchyňová

Katedra: Katedra distribuovaných a spolehlivých systémů

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

Abstrakt: Cílem práce je reimplementovat starou DOSovou hru Magic Carpet s využitím moderních prostředků pro tvorbu her poskytovaných vývojovým prostředím Unity. Ve hře hráč představuje kouzelníka na létajícím koberci, jehož úkolem je shromáždit určené množství many. Přitom se pohybuje 3D herním světem, používá kouzla, staví hrady, kde svou manu ukládá, a bojuje s různými druhy příšer nebo nepřátelskými kouzelníky. Detailní rozbor původní hry je k dispozici v úvodu práce. V rámci analýzy jsou diskutovány možnosti tvorby konečného, ale neohraničeného herního světa a způsoby implementace umělé inteligence a pohybového systému pro příšery a nepřátelské kouzelníky. Herní svět je generován procedurálně, proto je zvláštní kapitola věnována i této problematice. Součástí práce je také rozšíření původního editoru Unity o okno Level Designeru, kde je možné vytvářet nové herní úrovně.

Klíčová slova: 3D hra, Magic Carpet, procedurální generování terénu, Unity

Title: Master of the Carpet: A 3D first-person action game

Author: Karolína Kuchyňová

Department: Department of Distributed and Dependable Systems

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

Abstract: The goal of this thesis is to reimplement the old DOS game Magic Carpet using modern game development means provided by the Unity engine. In the game, the player controls a wizard on a flying carpet, whose task is to collect specified amount of mana. He travels through a 3D game world, casts spells, builds castles, where his mana is stored, and fights various types of monsters or enemy wizards. A detailed analysis of the original game can be found in the introductory part of the thesis. Within problem analysis, we discuss the possible approaches to creating a finite but borderless game world and ways of implementing artificial intelligence and its movemenent system for monsters and enemy wizards. The game world terrain is generated procedurally therefore a special chapter is devoted to this topic. The work also contains an extension of the original Unity Editor called Level Designer, where new game levels can be created.

Keywords: 3D game, Magic Carpet, procedural terrain generation, Unity

Obsah

1. Úvod

Tématem práce je počítačová hra Magic Carpet z roku 1994. Ve své době byla průkopnická kvalitou grafiky i propracovaností herního systému. Hráči i kritici byli nadšeni z bohatosti herního světa s širokou škálou kouzel a nepřátel. Hra je i nyní přístupná buď s využitím DOSBoxu, nebo na některých herních platformách, ovšem stále v původní podobě. Ačkoli celková koncepce hry je stále stejně atraktivní, zastaralý vzhled hry by nejspíše většinu dnešních hráčů odradil.

Cílem práce je tuto starší „klasiku“ reimplementovat s využitím dnešních vyspělejších možností vývoje počítačových her v podobě prostředí Unity. Není přitom snahou dokonale napodobit detaily původní hry, ale spíše vytvořit herní základ celku, zachovávající všechny významné koncepty, na který bude možné snadno navázat a jednotlivé původní nebo nové aspekty dotvořit.

1.1 Původní hra Magic Carpet

Nejprve se seznámíme s původní hrou, pokusíme se popsat její hlavní rysy a to, jak z pohledu hráče funguje, abychom se mohli rozhodnout, co přesně bude cílem naší implementace.

1.1.1 Vznik a vývoj

Hru Magic Carpet vytvořila v roce 1994 společnost Bullfrog Productions. Hra sklidila velký ohlas u recenzentů i mezi hráči mimo jiné díky své originalitě, na tehdejší dobu pokročilé grafice a propracovanému příběhu. Není proto divu, že ještě téhož roku se dočkala rozšíření v podobě Magic Carpet Plus, které k původním padesáti herním úrovním přidávalo dalších dvacet pět. Následujícího roku vyšel také druhý díl pod názvem Magic Carpet 2: The Netherworlds. Dnes jsou oba díly pro Windows nebo MacOS přístupné na platformě GOG.com.

1.1.2 Základní koncepce

Hra má podobu leteckého simulátoru. Hráč přestavuje kouzelníka na létajícím koberci, který cestuje různými magickými světy a jeho úkolem je obnovit v nich rovnováhu. Toho docílí shromážděním dostatečného množství many ve svém hradu. Mana ve světě může mít různou podobu – tvoří samostatné kuličky, prostupuje obyvatele a také ji v sobě mají příšery, které svět ohrožují. Hráč prozkoumává světy, pomocí kouzel sbírá manu a bojuje s příšerami. Později ve hře také narazí na další nepřátelské kouzelníky, kteří manu chtějí získat pro sebe. V jednotlivých herních úrovních se postupně objevují nové typy příšer a hráč má také možnost získat nová kouzla, pomocí kterých jim může čelit.

Hráč může v průběhu hry přepínat mezi dvěma obrazovkami. Na základní obrazovce, kterou můžeme vidět na obrázku 1.1, sleduje své bezprostřední okolí. Alternativou je pak obrazovka přehledová, kterou představuje obrázek 1.2. V její levé části má hráč k dispozici mapu celého herního světa, vpravo pak vidí svůj inventář kouzel a v menší části také dění okolo sebe.

Obrázek 1.1: Základní obrazovka.

Obrázek 1.2: Přehledová obrazovka.

1.1.3 Terén a pohyb v něm

Herní svět je 3D. Jeho zobrazení vychází z trojúhelníkové sítě a využívá Gouraudovo stínování založené na interpolaci barev ve vrcholech trojúhelníku. Tímto způsobem je řešen terén a stavby. Postavy a příšery v terénu jsou animované 2D sprity.

Hráč létá nad různými typy terénu – zelenými pláněmi, mořskou hladinou, pouští a tak dále. Svět má čtvercový půdorys a nijak neomezuje hráčův pohyb. Pokud hráč překročí hranici čtverce, objeví se na jeho protilehlé straně. V terénu se mohou nacházet přírodní objekty například menší stromy nebo velké kameny. Dále v něm kouzelníci budují své hrady.

Hráč sleduje okolí pohledem hlavního hrdiny. Pohybem myši je ovládán pohled kamery a současně i otáčení do stran leteckého simulátoru. Posunutí myši do strany odpovídá natočení křídel letadla. To způsobí, že se kouzelník otáčí na místě, dokud polohu pomyslných křídel myší opět nevyrovná. Šipkami na klávesnici se pak hráč nad terénem pohybuje. Vždy letí nad zemí, a pokud je v cestě překážka, automaticky vyletí nad ni, nikdy tak nemůže do ničeho narazit.

Své bezprostřední okolí může hráč sledovat na malé kruhové mapce v levém horním rohu obrazovky. Pohled hráče do okolí zobrazuje obrázek 1.3. Celou mapu světa si může zobrazit po stisku klávesy *Enter*. Ukázku mapy můžeme vidět na obrázku 1.4. Hráč na mapě kromě okolního terénu vidí pohybující se obyvatele světa, příšery a nepřátelské kouzelníky. Mapa také prozrazuje polohu všech vybudovaných hradů a skrytých kouzel. Jednotlivé postavy a objekty jsou v ní vyznačeny různě barevnými čtverečky.

Obrázek 1.3: Herní svět.

Obrázek 1.4: Velká mapa světa.

1.1.4 Kouzelníkův hrad

Zásadní význam pro každého kouzelníka má jeho hrad. Zde jsou totiž uschovány jeho zásoby many. Pro obnovení rovnováhy daného světa a úspěšné dokončení herní úrovně se počítá pouze mana uložená v hráčově hradu. Kouzelník hrad vytvoří pomocí kouzla. Tímto kouzlem lze také hrad postupně vylepšovat až na úroveň sedm. Každé vylepšení ovšem vyžaduje více many než předchozí. Podobu hradu můžeme vidět na obrázku 1.5.

Obrázek 1.5: Kouzelníkův hrad.

Hráčem získanou manu do hradu dopravují horkovzdušné balóny. Od vytvoření hradu je k dispozici jeden a při přestavbě hradu na pátou úroveň přibude další. Balóny manu sbírají jen ve chvíli, kdy se ještě do hradu vejde. Kapacita pro ukládání many hradu je omezená, zvyšuje se podle úrovně hradu, což nutí hráče hrad průběžně vylepšovat.

Hrad ani balóny nejsou nezničitelné, ale mají omezenou životnost. Zničený balón se v hradu obnoví, ale vytrácí všechnu přenášenou manu do okolí. Zbořený hrad si musí kouzelník postavit znovu sám. S vyšší úrovní hradu se zvyšuje jeho odolnost vůči útokům. Navíc poskytuje hráči ochranu, v jeho bezprostředním okolí je nezranitelný a zranění z útoků za něj přebírá hrad.

Důležité je, že pokud je kouzelník zabit, ale jeho hrad stojí, kouzelník nezemře, ale opět ožije ve svém hradě. To platí jak pro hráčova hrdinu, tak pro nepřátelské kouzelníky, se kterými bojuje.

1.1.5 Mana a život

Svůj život a aktuální zásobu many může hráč sledovat na liště v horní části obrazovky. Její vzhled můžeme vidět na obrázku 1.6. Lišta zobrazuje také aktuální stav hráčova hradu a balónu/ů, pokud je má, včetně úrovně hradu a počtu balónů. Poškozené hrady a balóny se neopravují, ale zraněný kouzelník se sám od sebe pomalu léčí.

Pro zdolání každé úrovně hry je nutné nasbírat určité množství many, která se v jejím světě nachází. Potřebné množství je na horní liště vyznačeno bílými body na ukazatelích many hráče i hradu. Mana se ve světě vyskytuje v podobě žlutých kuliček. Aby ji kouzelník ovládl, musí ji zasáhnout svým kouzlem, poté se mana přebarví na jeho barvu a může být balónem odnesena do hradu. Ukázku hráčem získané many nabízí obrázek 1.7.

Obrázek 1.6: Lišta s manou a životem hráče, jeho hradu a balónu.

Jinou možnost zisku many nabízejí obyvatelé. Hráč může kouzlem získávat jejich domy a tím i podporu a manu. Nad domy se pak objeví jeho vlajka, jak vidíme na obrázku 1.8. Takto získaná mana ovšem nemůže být dopravena do hradu, a tudíž se nezapočítává k potřebnému množství pro dokončení herní úrovně.

Obrázek 1.7: Hráčem získaná mana.

Obrázek 1.8: Hráčem získaný dům.

1.1.6 Kouzla

V průběhu hry může hráč objevit až dvacet čtyři různých kouzel. Ta se nachází ve světě ukrytá v amforách. K jejich získání stačí, když hráč amforu sebere. Kromě znalosti kouzla potřebuje kouzelník k jeho provedení také dostatek many. Navíc platí, že čím je kouzlo pokročilejší, tím více many je ke kouzlení potřeba. Použitím kouzla se kouzelníkova aktuální zásoba many dočasně sníží a teprve postupně se vrací na původní úroveň.

Po stisku klávesy *Enter* se hráč může podívat na inventář svých získaných kouzel. Ukázku inventáře vidíme na obrázku 1.9. Zde může hráč zvolit, jaké kouzlo bude kouzelník mít v pravé a levé ruce a pak ho používat pomocí pravého a levého tlačítka myši.

Mezi základní kouzla patří především *Obarvení many* (*Posses*), umožňující kouzelníkovi získávat kuličky many, a *Hrad* (*Castle*), díky němuž buduje a vylepšuje svůj hrad. Nezbytností pro boj s nepřáteli jsou útočná kouzla jako *Ohnivá střela* (*Fireball*) nebo *Bouře blesků* (*Lightning Storm*).

Ve vyšších herních úrovních má hráč k dispozici také silná kouzla měnící herní svět jako *Sopka* (*Volcano*), které v okolním terénu vztyčí sopku chrlící ohnivou lávu, nebo *Zemětřesení* (*Earthquake*), které roztrhá část pevniny a vytvoří hlubokou trhlinu.

Poslední kategorií jsou kouzla, která zlepšují aktuální stav hráč například tím, že ho léčí, umožňují mu pohybovat se zvýšenou rychlostí nebo ho učiní neviditelným před nepřáteli.

Obrázek 1.9: Inventář hráčových kouzel a přehled všech kouzel ve hře.

1.1.7 Příšery

Magické světy, kterými hráč prochází, ohrožují různé typy nestvůr. Na své cestě může potkat třináct druhů příšer. Ty se mohou pohybovat po souši, létat v povětří, nebo se skrývat pod vodní hladinou. Ve vzduchu kouzelníka napadají orli, roje včel nebo draci. Ve vodě na něj čekají krakeni. Po souši se plazí nebezpeční červi, pobíhají krabi nebo číhají trollové. Jednotlivé druhy se liší nejen tím, kde se vyskytují, ale také rychlostí, jakou se ve svém prostředí pohybují, a způsobem, kterým na kouzelníka mohou zaútočit. Ukázku podoby příšer můžeme vidět na obrázcích 1.10a a 1.10b.

Příšery se náhodně pohybují svým domovským prostředím, buď jednotlivě, nebo v rojích či hejnech. V případě, že se v jejich okolí objeví kouzelník, zaútočí na něj, a dokud od nich dostatečně daleko neuteče, pronásledují ho. Mohou na něj útočit zobáky či žihadly nebo jej na dálku zasáhnout svými kouzly, kameny a šípy.

Některé druhy mají i zajímavější dovednosti. Mořští krakeni umějí udělat kouzlo, kterým kouzelníka k sobě připoutají a po určitou dobu mu znemožňují uletět do bezpečí. Džinové naopak v okamžiku svého ohrožení mohou použít teleport a zmizet z kouzelníkova dosahu. Krabi zase sbírají a požírají kuličky many. Díky nim se vyvíjejí, zvětšují, mohou používat silnější útočná kouzla a na nejvyšším stupni vývoje také klást vejce, z nichž se líhnou další generace těchto tvorů.

Jednotlivé druhy příšer jsou různě odolné vůči hráčovu útoku. Pokud se kouzelníkovi podaří příšeru zabít, zůstanou po ní kuličky many. Jejich počet závisí na druhu nestvůry.

(a) Suchozemský červ.

(b) Mořský kraken.

Obrázek 1.10: Ukázka příšer.

1.1.8 Nepřátelští kouzelníci

Kromě hlavního hrdiny se v některých světech nacházejí i další kouzelníci. Ti s ním soupeří o získání many. Mají v zásadě stejné možnosti jako hráč. Také používají kouzla, sbírají manu a shromažďují ji ve svých hradech. Mohou bojovat s příšerami, mezi sebou nebo napadnout hráče. Za celou hru jich hráč může potkat šest. Každý z nich má svoji charakteristickou barvu, kterou obarvuje manu a je vyznačen v mapách. Jednoho z nich můžeme vidět na obrázku 1.11.

Kouzelníci se liší mírou inteligence, a tím i úrovní své strategie. Navíc mají k dispozici některá z možných kouzel hráče. Může se tak stát, že v nějakém kole mohou používat pokročilejší kouzla než hráč sám. Stejně jako hráč, pokud jsou zabiti, ale jejich hrad stojí, obnoví se v něm. Jedinou možností, jak se jich zbavit natrvalo, je tedy nejprve zničit jejich hrad a až poté samotné kouzelníky.

Obrázek 1.11: Nepřátelský kouzelník.

1.1.9 Obyvatelé vesnic

Kromě příšer a nepřátelských kouzelníků v magických světech žijí i obyčejní, mírumilovní obyvatelé. Dělí se na prosté lidi, stavitele a obchodníky. Většinou se všichni pohybují v okolí své vesnice, stavitelé tam budují nové domy pro rostoucí populaci a obchodníci cestují mezi osadami a nabízejí své zboží. Jakmile město dosáhne určité velikosti, vytvoří si vlastní vojenskou posádku, která je chrání. Ukázku obyvatel a jejich osady najdeme na obrázku 1.12.

Obrázek 1.12: Obyvatelé vesnice.

Hráč může obyvatele získávat na svou stranu tak, že nad jejich domky umisťuje kouzlem svou vlajku. Pak se mu vojáci města v bojích snaží pomáhat. Naopak pokud kouzelník na lid útočí, obrací pak svou vojenskou sílu proti němu.

1.2 Požadavky pro novou implementaci

Poté, co jsme prošli základní aspekty hry, můžeme se u každého z nich rozhodnout, v jaké podobě ho chceme v nové verzi implementovat. Základní herní koncepci sbírání many, budování hradů a bojů s nepřáteli pomocí kouzel popsanou v části 1.1.2 samozřejmě zachováváme. Stejně tak i třídimenzionální herní svět včetně jeho neomezené podoby (část 1.1.3).

V nové verzi plánujeme zjednodušit ovládání pohybu hlavního hrdiny. Ovládání kamery zde bude odpovídat otáčení na místě místo přesného chování leteckého simulátoru popsaného v podkapitole 1.1.3. Důvodem změny není náročnost napodobení původního ovládání, ale snaha o větší komfort hráče méně zdatného v řízení simulátoru.

Z výchozí hry zůstává i stavba hradů (část 1.1.4), včetně jejich vylepšování na vyšší úrovně, s nimiž hrad dovoluje uložit více many. Stále také platí, že manu do hradu přenáší horkovzdušné balóny. Nepřátelé nebo příšery mohou hrady i balóny napadat. Kouzelník se po zabití v hradu obnoví, ale oproti originálu mu hrad za jeho života žádnou další ochranu neposkytuje.

V nezměněné podobě chceme zachovat nabízené přehledy nejbližšího okolí a stavu hráče v podobě malé mapky a lišty s ukazateli v horní části obrazovky. Hráči zůstává možnost přejít na přehledovou obrazovku se stejným obsahem. Detailní popis podoby přehledové lišty je uveden výše v podkapitole 1.1.5 a přehledovou obrazovku zmiňuje část 1.1.2.

Původní hra nabízí širokou škálu kouzel i typů příšer (podkapitoly 1.1.6 a 1.1.7). Jelikož cílem nové implementace je pouze vytvořit herní základ, zvolíme pro každou z těchto oblastí pouze pár příkladů představujících různé základní typy. Na druhou stranu chceme umožnit rozšíření základu do původní podoby, takže při implementaci musíme myslet na možnost jednoduchého doplnění dalších příšer nebo kouzel.

Pro celé fungovaní hry je nezbytné, aby měl kouzelník k dispozici kouzlo pro obarvení many i pro stavbu hradu. Za reprezentanta útočných kouzel bylo zvoleno kouzlo *Ohnivá střela* (*Fireball*). Velkou zajímavostí a originálním aspektem původní hry jsou kouzla měnící terén herního světa. Tuto skupinu bude v naší implementaci představovat kouzlo *Sopka* (*Volcano*). Tuto sadu mohou doplnit jednodušší kouzla měnící pouze vnitřní stav hráče jako například léčení nebo zrychlení.

U příšer je charakteristické omezení jejich pohybu jen na konkrétní živel. V nové implementaci proto chceme mít zástupce každého prostředí. Pro souš jsme zvolili červy a kraby, pro vzduch létající červy a vosy a mořské příšery budou reprezentovat krakeni. Mezi takto zvolenými zástupci stojí za větší pozornost krakeni a krabi. Krakeni totiž kromě běžného útoku mohou použít také poutající kouzlo. Krabi se zase vyznačují pojídáním many a s ním spojeným vývojem.

V nové implementaci chceme mít i nepřátelského kouzelníka (viz podkapitola 1.1.8). Ten by se měl chovat podobně jako hráč. Tedy sbírat manu, budovat hrady a bojovat s příšerami i hráčovým hrdinou. Jeho umělá inteligence by měla

být dostatečně pokročilá, aby se mohl rozhodovat o svých akcích směřujících ke komplexnějším cílům jako například vylepšení úrovně hradu nebo poražení nepřítele v boji.

Naopak jsme se rozhodli nevytvářet obyvatele vesnic (jejich popisu se věnuje část 1.1.9). Netvoří natolik významnou složku hry a z pohledu hráče ji většinou ovlivňují minimálně. Je zde samozřejmě prostor tuto mezeru vyplnit při dalším rozšiřování hry.

S hrou samotnou úzce souvisí také tvorba herních úrovní. Součástí práce je proto také vytvoření rozšíření základního editoru poskytovaného Unity, které by mělo usnadnit práci při návrhu herních úrovní. Jde především o úpravy terénu, možnost umístění herních objektů, tedy many nebo kouzel, a v neposlední řadě také přidávání příšer a nepřátelského kouzelníka.

1.3 Shrnutí cílů práce

Požadavky na novou implementaci původní hry Magic Carpet detailně rozvedené v části 1.2 můžeme shrnout do následujících bodů:

1. Herní svět je 3D a hráč se v něm může neomezeně pohybovat.
2. Cílem hráče je získat určité množství many, kterou v podobě kuliček shromažďuje ve svém hradu. Do hradu manu dopravují horkovzdušné balóny.
3. Hráč má dispozici sadu kouzel, která se postupně rozšiřuje. Implementována budou minimálně tato kouzla:
   (a) kouzlo pro obarvení a získání many,
   (b) kouzlo pro vybudování hradu,
   (c) *Ohnivá střela* (*Fireball*),
   (d) *Sopka* (*Vulcano*).
4. V herním světě se vyskytují následující druhy příšer:
   (a) suchozemští červi,
   (b) krabi,
   (c) krakeni,
   (d) vosy,
   (e) létající červi.
5. S hráčem soupeří nepřátelský kouzelník, který se chová podobně jako on sám. Také shromažďuje manu ve svém hradu, používá kouzla a bojuje s příšerami i hráčem.
6. Přehled o herním světě hráči nabízí přehledová obrazovka s mapou herního světa a jeho inventářem kouzel. V herní obrazce má k dispozici malou mapu bezprostředního okolí a lištu s ukazateli svého stavu.
7. Součástí práce je editor pro vytváření herních úrovní. Ten umožňuje:

(a) upravit terén herního světa,

(b) umísťovat kuličky many a kouzla,

(c) přidávat příšery a nepřátelského kouzelníka.

2. Vývojové prostředí Unity

Původní hra Magic Carpet vznikala na počátku devadesátých let minulého století jako nápad Glena Corpse, programátora firmy Bullfrog, která vytvářela hry pro tehdy populární šestnáctibitové domácí počítače Amiga, Atari ST a další.

Základem hry byl fraktálový generátor krajiny využívající Gouraudovo stínování. Pro vytváření některých herních postav byla použita pokročilejší technologie, například program 3D Max pro renderování a animace. V té době však neexistoval žádný univerzální systém, který by byl použitelný k vytvoření celé hry.

V současné době takových systémů existuje celá řada a některé z nich jsou dokonce pro nekomerční použití dostupné zdarma. Mezi nejčastěji používané patří například Unreal Engine, Unity nebo CryEngine. Vzhledem k předchozím zkušenostem s programováním v jazyce C# a požadavku na volnou dostupnost herního enginu jsme zvolili platformu Unity. V této kapitole si představíme základní principy jejího fungování.

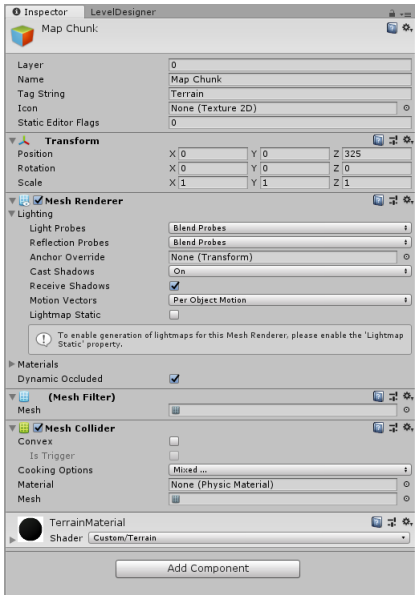
2.1 Úvod

Unity je multiplatformní herní engine pro vývoj 2D i 3D her nejrůznějších žánrů. První verzi uvedla společnost Unity Technologies v roce 2005. Dnes lze v Unity tvořit pro nejrůznější platformy přes klasické počítačové Windows, Linux a MacOS, mobilní jako iOS a Android až po virtuální realitu. Engine poskytuje podporu pro veškeré aspekty herního vývoje – simulaci fyziky herního světa, zpracování 2D i 3D grafiky, animace, umožňuje importovat 3D modely objektů, textury, práci se zvukovými efekty i tvorbu uživatelského rozhraní. Aktuálně je možné engine používat v rámci jedné ze čtyř licencí, přičemž varianta Personal je k dispozici zdarma.

2.2 Scéna

Model herního světa v Unity představuje tzv. scéna. Jednotlivé scény mohou odpovídat herním úrovním nebo různým prostředím, mezi kterými se hráč pohybuje. Základními prvky scény jsou herní objekty (`GameObjects`). Ty mohou být přímo viditelné jako modely postav a dílky terénu, nebo bez fyzické reprezentace jako kamera, zdroj světla nebo speciální efekty. Objekty jsou v rámci scény uspořádány hierarchicky.

Vlastnosti herních objektů určují komponenty, které jsou k nim připojeny. Základní komponentou, kterou má každý objekt vždy je `Transform`, ten určuje pozici objektu v 3D herním světě, včetně jeho natočení a případného škálování do základních směrů. Aby byl objekt vidět, musí mezi jeho komponentami být `MeshFilter` a `MeshRenderer`, které zobrazují nastavený mesh (trojúhelníkový model povrchu) objektu obarvený dle jeho materiálu. Aby spolu objekty mohly fyzikálně interagovat, musí mít k dispozici komponentu `Collider`. Jednotlivé komponenty se vzájemně ovlivňují a lze je zapínat a vypínat. Ukázku přehledu komponent konkrétního herního objektu můžeme vidět na obrázku 2.1.



Obrázek 2.1: Přehled komponent objektu.

2.3 Skriptování

Programování pro Unity je založeno na skriptování. Zdrojový kód, označovaný jako skript, je interpretován virtuálním strojem. Unity používá implementaci běhového prostředí Mono. Umožňuje používat skripty napsané v jazycích C# a JavaScript, v dřívějších verzích také Boo, což je jazyk vycházející z Pythonu. Jednou z výhod skriptování je, že při úpravě zdrojových souborů není třeba znovu kompilovat celý projekt, ale pouze změněné soubory, což výrazně zrychluje práci programátora.

Vytvořené skripty je pak možné připojovat k herním objektům jako jejich komponenty. V běhovém čase Unity vytváří instance tříd definovaných ve skriptech a na nich následně volá jejich jednotlivé metody. Třídy, jejichž instance lze použít jako komponenty, musí být potomky třídy `MonoBehaviour`. Základními metodami těchto tříd jsou `Start` a `Update`. Metoda `Start` se zavolá pouze jednou před začátkem hry a je tak vhodným místem pro inicializaci. Celá hra probíhá po jednotlivých snímcích. V každém snímku se na všech používaných skriptech volá metoda `Update`. Po skončení `Update` fáze probíhá ještě `FixedUpdate` a `LateUpdate`. V nich se počítá s tím, že všechny metody `Update` už proběhly. Lze tak například v `Update` vyřešit veškeré fyzikální simulace pohybu a teprve v dalších fázích polohu kamery, která bude snímat objekty po ukončení jejich interakcí.

Překreslování jednotlivých snímků se provádí až po ukončení všech výpočtů. Další výpočetní cyklus může být zahájen ihned po vykreslení scény, případně je možné vyčkat na dokončení zobrazení scény grafickou kartou. Mezi jednotlivými snímky tedy nemusí být pravidelné rozestupy, záleží na aktuálním zatížení procesoru a výpočetní náročnosti všech volaných metod `Update`. Pro zjištění reálného času mezi dvěma snímky potřebného pro fyzikální výpočty slouží proměnná `Time.deltaTime`.

V rámci kódu jednotlivých skriptů lze kromě stavu herních objektů zjišťovat také stav periferních zařízení počítače a okamžitý stav běhového prostředí

jako například velikost obrazovky. Stav klávesnice, informace poskytované senzory myši případně dalších vstupních zařízení jako joystick nebo touchpad jsou vhodným způsobem filtrovány tak, aby všechna tato zařízení bylo možno programově obsluhovat unifikovaným způsobem. Je také možné sledovat pohyb a polohu kurzoru na obrazovce, včetně možnosti převodu 2D souřadnic kurzoru myši v okně na souřadnice v herním světě.

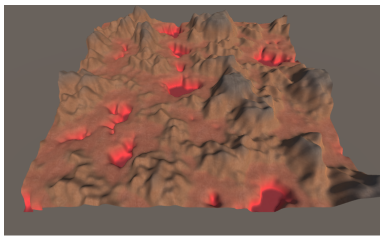
Skripty mohou pracovat s dalšími komponentami herního objektu, ke kterému jsou připojeny, nebo komponentami dalších objektů na scéně. Prostřednictvím skriptů lze také připojovat k objektu nové komponenty nebo vytvářet kopie existujících herních objektů včetně jejich komponent metodou `Instantiate`. Opakem je metoda `Destroy`, která umožňuje rušit komponenty, případně celé herní objekty.

2.4 Grafika

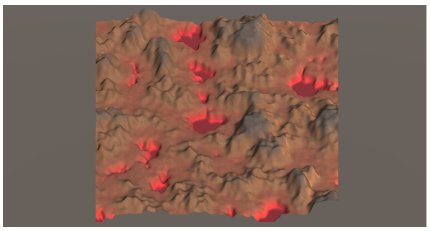
V každém snímku hry dochází k renderingu neboli vykreslení všech viditelných detailů scény na obrazovku. Výsledná barva konkrétního místa na scéně závisí na materiálu povrchu, osvětlení a úhlu pohledu. Na scéně se proto kromě pozorovaných herních objektů musí nacházet také zdroj světla a kamera.

Aby byly předměty na scéně viditelné, musí zde být umístěn jeden nebo více zdrojů světla (objektů obsahujících komponentu `Light`). Mezi základní typy světel patří všesměrová bodová světla, světelné kužely a směrová světla tvořená svazkem rovnoběžných paprsků. U všech světel je možné zvolit jejich intenzitu, barvu, polohu a nasměrování.

Scéna se zobrazuje na displej prostřednictvím kamery (respektive objektu s komponentou `Camera`). Kamera ukazuje to, co by viděl pozorovatel, kdyby se nacházel na jejím místě. Základním parametrem kamery je její zorný úhel. Na rozdíl od skutečné kamery umožňuje potlačit zobrazení objektů, které jsou příliš blízko nebo příliš daleko od ní. Unity nabízí dva typy kamer – ortografickou a perspektivní. Při perspektivním zobrazení se vzdálené předměty jeví menší, zatímco u ortografické se jejich velikost v závislosti na vzdálenosti nemění. Ukázku pohledů z obou kamer můžeme vidět na obrázcích 2.2 a 2.3. Umístění i natočení kamery se může dynamicky měnit. Lze použít i více kamer a pohledy z nich zobrazovat současně nebo mezi nimi přecházet.



Obrázek 2.2: Perspektivní pohled na terén.

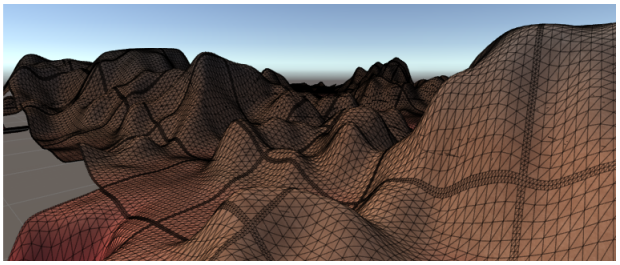


Obrázek 2.3: Ortogonální pohled na terén.

Jak kamera zobrazí povrch daného objektu, závisí především na jeho materiálu. Skript, který vypočte konečnou barvu pixelu, se nazývá shader. Barvu určuje na základě vlastností materiálu a osvětlení. Krom vestavěných shaderů lze použít

i shader vlastní. Materiál tělesa určuje, jaký shader má být použit pro výpočet barvy jeho povrchu. Další vlastnosti materiálu závisí na tom, jaké parametry pro svůj výpočet zvolený shader vyžaduje.

Pro potřeby zobrazení je těleso nahrazeno povrchovou sítí tvořenou trojúhelníky označovanou jako mesh. Ta je uložena v komponentě `MeshFilter`. Příklad meshe vidíme na obrázku 2.4. Druhou komponentou nutnou pro zobrazení povrchu tělesa je `MeshRenderer`. V něm je určen způsob zpracování osvětlení a materiál tělesa.



Obrázek 2.4: Mesh terénu.

Pro objekty, které nemají stálý povrch například plameny nebo oblaka, není vhodné použít meshe. Vytváří se proto jako systém částic v rámci komponenty `ParticleSystem`. Tato komponenta typicky zobrazuje větší množství malých částic, které rychle vznikají a zanikají. V průběhu jejich existence se může měnit jejich tvar, velikost i barva. Jejich efektním využitím mohou být exploze nebo kouzla.

2.5 Fyzika

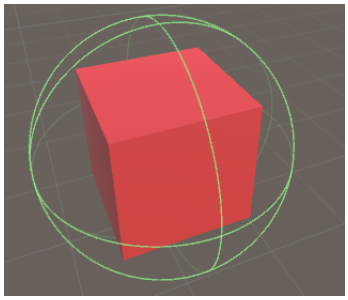
Herní engine Unity zahrnuje i podporu simulace pohybu těles v homogenním gravitačním poli a simulaci vzájemného silového kontaktního působení herních objektů.

Základní komponentou, která je využívána při modelování fyzikálně mechanických dějů, je `Rigidbody`, představující model tuhého tělesa. Mezi vlastnosti této komponenty patří hmotnost, moment setrvačnosti, odpor prostředí, simulace působení gravitace a schopnost detekovat kolize s ostatními herními objekty zahrnujícími tuto komponentu.

Pokud se má herní objekt obsahující `Rigidbody` pohybovat, je třeba na něj působit silou (prostřednictvím metod `AddForce` pro posuvný pohyb, případně `AddTorque` pro otáčení) a není vhodné pohyb simulovat jednotlivými změnami hodnot pozice v komponentě `Transform`.

Tělesa na sebe mohou působit prostřednictvím komponenty `Collider`. Tato komponenta představuje „neprostupný obal“ okolo herního objektu, který přenáší působící síly z okolí. Tvar `Collideru` lze přizpůsobit tělesu, které obklopuje. Ukázku `SphereCollideru`, tedy `Collideru` ve tvaru koule, můžeme vidět na obrázku 2.5. `Collidery` neslouží jen pro simulaci nárazů pevných těles, ale také jako triggery, umožňující detekovat dotyk dvou `Colliderů`. V rámci skriptů pak lze reagovat na události spojené s kolizí nebo na vstup do oblasti triggeru.

Dobře propracovaná je i podpora modelování soustavy těles spojených klouby. U těchto spojení lze nastavit velký počet parametrů od omezení rozsahu úhlu



Obrázek 2.5: Collider ve tvaru koule.

otáčení až po velikost síly, která je třeba k roztržení spojení. Pokud působíme na jedno z těles takové soustavy silou nebo momentem, silový účinek se přenese i na ostatní tělesa soustavy a pohyb celku působí přirozeným dojmem.

Ne vždy potřebujeme fyzikálně naprosto věrný model. Například pro simulaci pohybu po scéně z pohledu hráče, kdy je třeba provést okamžité změny polohy nebo rychlosti bez ohledu na působící setrvačnost. Takovéto chování umožňuje komponenta `Character Controller`. Díky ní se objekt může pohybovat v terénu, aniž by se propadl, a reagovat na překážky.

2.6 Uživatelské rozhraní

Unity nabízí širokou podporu pro tvorbu uživatelského rozhraní. Jednotlivé prvky rozhraní jsou reprezentovány herními objekty na scéně. Všechny musí být v hierarchii scény potomky objektu `Canvas`, který představuje plátno, na něž se objekty rozhraní kreslí.

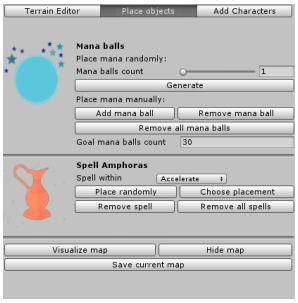
Celé plátno lze dále dělit pomocí panelů (`Panel`). V rámci plátna nebo panelu lze komponenty rozhraní různě ukotvovat, otáčet a škálovat. Lze také využít automatické rozložení horizontálně, vertikálně nebo do mřížky.

Základními neintraktivními komponentami uživatelského rozhraní jsou `Text`, `Image` a `Raw Image`. `Text` umožňuje zobrazení popisku se zvolenou barvou, fontem a podobně. `Image` i `Raw Image` představují obrázky. Rozdíl spočívá v tom, že pro `Image` je podkladem buď materiál, nebo Sprite (2D grafický objekt libovolného tvaru), zatímco `Raw Image` zobrazuje přímo texturu.

Ovládací prvky pak představují nejrůznější tlačítka (`Button`), vstupní pole (`Input field`), přepínače (`Toggle`), posuvníky (`Slider`) a tak dále.

Podobně jako například Windows API generuje Unity pro všechny ovládací prvky základní události v rámci jmenného prostoru `EventSystem`. K základním patří přesunutí kurzoru na objekt a z objektu, kliknutí nebo uvolnění myši nad objektem, změny stavu ovládacích prvků a tak dále. Pro události je možné registrovat různé obsluhy. Sadu standardních událostí lze rozšířit o vlastní.

V rámci Unity lze utvářet nejen uživatelské rozhraní finální hry, ale také měnit vzhled Unity editoru, ve kterém se hra vyvíjí. K tomu slouží tzv. *Immediate mode GUI* (IMGUI), které představuje alternativu k normálnímu GUI ve hře, založenému na herních objektech. Místo tvorby mnoha objektů se celý vzhled popíše pouze v kódu ve speciální metodě `OnGUI`, která se volá v každém snímku. V kódu lze vytvářet prvky rozhraní obdobné těm ze standardního GUI a lze také reagovat na události s nimi spojené. *Immediate mode GUI* lze použít buď pro



Obrázek 2.6: Okno editoru vytvořené pomocí IMGUI.

vytváření vlastních oken editoru, nebo pro definici zobrazení informací o herních objektech. Okno editoru definované prostředky *Immediate mode GUI* můžeme vidět na obrázku 2.6.

3. Analýza

V předchozích kapitolách jsme si stanovali požadavky na implementaci a seznámili jsme se s vývojovým prostředím Unity. Nyní se pokusíme definovat základní problémy, které je potřeba s využitím prostředků vývojového prostředí vyřešit. Nastíníme různé varianty řešení a vyhodnotíme, který přístup považujeme za nejvhodnější.

3.1 Herní svět

V rámci požadavku (1) chceme vytvořit konečný herní svět (herní mapa ho má celý zobrazovat), ve kterém se hráč i ostatní postavy mohou neomezeně pohybovat, tedy nerozeznávají v něm žádnou nepřekročitelnou hranici.

3.1.1 Tvar světa

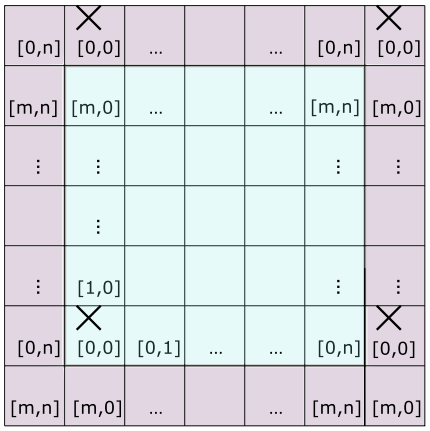
Jednou z realizací, která požadované chování umožňuje, je vytvoření herního světa na povrchu koule. Modelování světa na kouli však způsobí potíže například s výpočty pohybů v centrálním gravitačním poli, které budou podstatně složitější než využití vestavěného modelu gravitace poskytnutého enginem Unity. Další problémy se objeví při vytváření 2D mapy světa na kouli, kdy v okolí pólů dochází k výraznému zkreslení. Aby zobrazení povrchu terénu na obrazovce vypadalo přirozeně, musela by být koule a tím celý herní svět dostatečně rozlehlý, takže by nakonec stejně působil dojmem světa plochého. Jediná výhoda světa na povrchu koule (neomezený pohyb všemi směry) by byla dosažena za cenu vytvoření celého fyzikálního modelu, který by pro vlastní hru byl zcela nepodstatný.

Proto jsme se rozhodli pro snadněji implementovatelnou variantu, kterou je svět umístěný na rovině. Zde můžeme plně využít model homogenního gravitačního pole poskytovaný enginem Unity, je snadnější generace terénu i zobrazení 2D mapy a umísťování objektů. Obyčejná rovina však nesplní požadavek na neomezený pohyb v konečném světě. Teoreticky by bylo možno stále generovat nové segmenty terénu v okolí hráče. V tomto nekonečném, stále dále generovaném světě by nebyl neomezený pohyb problém, ale nebylo by možné zobrazit celou jeho mapu a pravděpodobně by to ani nebylo přínosem pro hratelnost hry. Další možností je omezený herní svět v rovině. Ten má však pevné hranice a musíme zde vyřešit problém, jak poskytnout hráči iluzi neomezeného pohybu. Tím se zabýváme v následujících podkapitolách.

3.1.2 Zdvojení okrajů světa

Prvním možným postupem, jak realizovat neomezený svět v konečné ploše, je po každém překročení hranic hráče přesunout na protilehlý konec herního světa. Přesunu v rámci jediného snímku si hráč pochopitelně nevšimne. Je však potřeba ošetřit situaci, kdy je hráč hranici blízko a očekává, že před sebou uvidí celé své okolí, tedy i protilehlý okraj mapy, kam se může za chvíli dostat. Možným řešením je kolem skutečného herního světa, ve kterém se hráč pohybuje, vytvořit ještě okrajovou zónu, která bude vždy kopírovat protilehlou část mapy.

Tuto možnost ilustruje obrázek 3.1. Světlá část představuje dílky herního světa, fialová pak kopie okrajových dílků. Ty jsou rozmístěny tak, aby na sebe protilehlé strany navazovaly. Čísla v jednotlivých čtvercích odpovídají souřadnicím dílků v mapě. Hráč se pohybuje pouze ve světle modré ploše, ale v okrajových oblastech vidí i fialovou část, která ukazuje oblast, do níž se s dalším dílkem může dostat. Je nutné volit hraniční zónu dostatečně širokou tak, aby její konec vypadal z pohledu hráče jako přirozený obzor.



Obrázek 3.1: Mapa se zdvojenou okrajovou částí.

Tato varianta „přechodu“ hráče přes okraj mapy nám vyhovuje. Navíc je v ní poloha všech dílků mapy neměnná a umožňuje tak používat klasické prostorové souřadnice Unity, které má každý objekt uloženy ve své komponentě `Transform`.

Problém ale nastává v situaci, kdy se v okolí hranice nenachází jen hráč, ale i jiné pohybující se objekty například nepřátelé nebo kouzla. Jejich přesun přes hranici lze stejně jako u hráče snadno vyřešit pouhou změnou polohy, ale otázkou je, jak tato akce bude vypadat sledována hráčem.

Pro toho by totiž bylo zjevně matoucí, kdyby v kopii hraničního dílku žádný pohyb nepozoroval, zatímco po jeho překročení by se objevili nepřátelé. To vyžaduje složitější řešení přemisťování ostatních pohyblivých objektů přes hranice světa.

Jednou z možností by bylo kopírovat nejen herní dílek na hranici, ale i všechny objekty, které se na něm aktuálně nacházejí. Pak by se ovšem jeden objekt mohl na scéně vyskytovat až ve čtyřech kopiích. Tuto situaci ukazuje křížek v obrázku 3.1 nacházející se v původním dílku mapy o souřadnicích [0,0] a také v jeho třech kopiích v protějších rozích. Tento přístup vyžaduje pečlivé ošetření situací, kdy mají kopie původního objektu vznikat a zanikat.

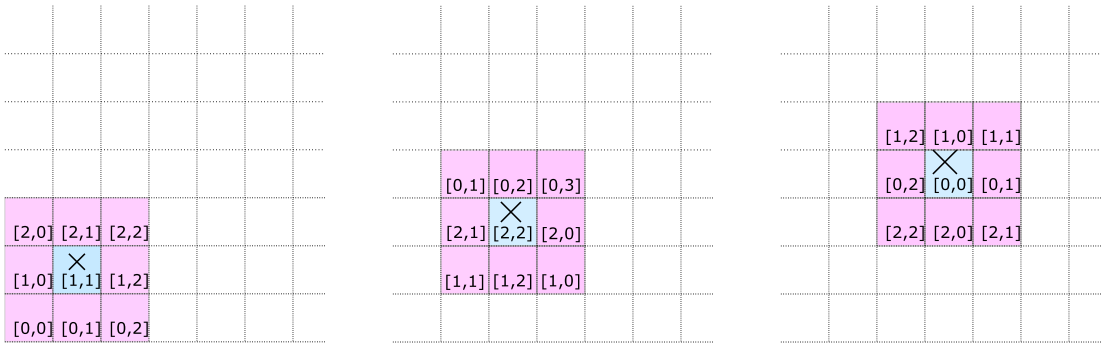
Druhou možností by bylo přesunout pohyblivé objekty, které jsou na dohled hráče za hranicí, do duplicitní (v obrázku fialové) oblasti. Pak by podoba přechodů přes hranici těchto objektů navíc závisela na vzdálenosti od hráče.

Přestože popsané přesouvání hráče řeší zadaný problém, přináší novou komplikaci v podobě nutnosti rozlišování různých druhů přesunů ostatních pohyblivých objektů mezi dílky. Proto hledáme ještě jiný možný přístup.

3.1.3 Přemisťování světa podle polohy hráče

Alternativou přesouvání hráče po překročení hranice světa je přesouvat celý herní svět podle aktuální polohy hráče. Zde neřešíme, co se stane, když se hráč ocitne poblíž hranic zobrazovaného světa, neboť hráč je v každém okamžiku v jeho středu.

Tento přístup pro čtvercovou mapu s devíti dílky ukazuje obrázek 3.2. Vybarvené čtverce představují zobrazené dílky mapy, na světle modrém políčku stojí hráč. Poté, co se hráč přesune do dalšího dílku, zbývající dílky se přeskupí tak, aby jimi hráč zůstával obklopen. Čísla udávají souřadnice dílků v herní mapě.



(a) (b) (c)

Obrázek 3.2: Svět pohybující se s hráčem.

Zde samozřejmě vzniká rozpor mezi tím, že jeden díl mapy se může nacházet v herním světě na mnoha různých pozicích (čtverečcích čárkované mřížky na obrázku). To vede k nutnosti používat složitější souřadný systém, respektive odlišovat polohu objektů v herním světě Unity, která je neomezená, a polohu v rámci konečné herní mapy rozdělené na dílky. Mezi oběma variantami musíme být schopni přecházet.

Pokud se hráč pohybuje dlouho jedním směrem, hodnota některé jeho souřadnice v souřadném systému Unity bude stále narůstat. To by způsobovalo, že vzdálenost sousedních míst bude vypočtena se stále menší přesností. Řešením je poté, co hráč v některém směru překročí zvolený násobek velikosti světa, přesunout celý svět zpět do počátku. K tomu stačí umístit na příslušné místo hráče a herní svět jej znovu obklopí.

Opět musíme zodpovědět otázku, jak se v takovémto světě budou pohybovat ostatní objekty. Oproti variantě s přesuny hráče zde ovšem existuje jednoduché řešení, které žádné problémy nepřináší. Tím je, aby si každý pohybující se objekt v sobě nesl informaci o své aktuální poloze v mapě světa a věděl, nad kterým jejím dílkem se má nacházet. V každém snímku pak musí kontrolovat, kde je příslušný dílek umístěn a v situaci, kdy se dílek přesunul nebo úplně zmizel, jeho činnost napodobit.

3.1.4 Generování terénu

Poslední rozhodnutí o podobě herního světa se týká způsobu jeho generování. Zde máme dvě základní možnosti. Buď ručně předpřipravit podobu jednotlivých dílků mapy, nebo tento úkol nechat na počítači a terén generovat procedurálně.

Především pro rozsáhlý herní svět se jeví jako lepší druhá varianta, neboť zde by bylo časově náročné vytvořit neopakující se vzhled terénu ručně. V naší implementaci jsme proto zvolili procedurální generování, doplněné o možnost dalších úprav výšek vygenerovaného prostředí v rámci editoru návrhu mapy.

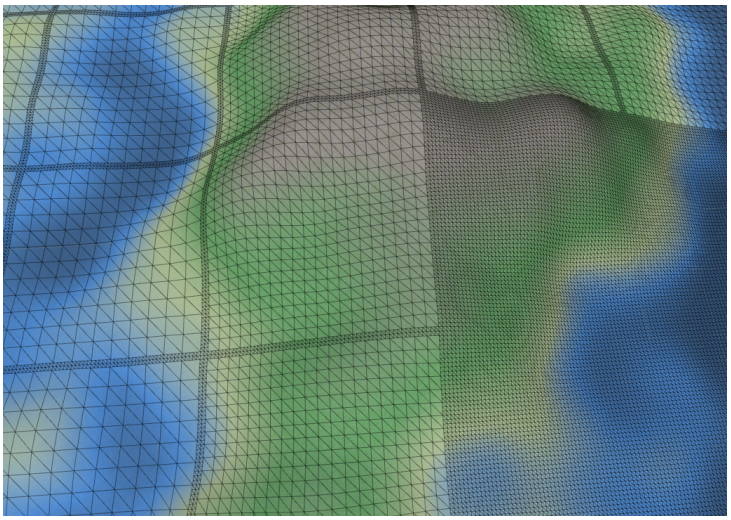
3.2 Dění mimo okolí hráče

V rozsáhlém herním světě je výpočetně náročné zejména jeho zobrazování a také řízení chování všech herních objektů. Přitom hráč vnímá pouze dění ve svém bezprostředním okolí a zbytek herního světa nevidí. Toho můžeme využít a mimo jeho zorný úhel prostředí i akce zjednodušit. Tato technika se označuje jako úroveň detailu (*level of detail* – LOD). Typicky se využívá pro zjednodušení zobrazování, ale její principy lze aplikovat i na řízení chování herních objektů. Úvod do problematiky použití LOD při programování umělé inteligence nabízí Brom a další v článku [1].

V dalších podkapitolách se budeme věnovat zjednodušení zobrazování i jednotlivých dějů, které mohou probíhat mimo okolí hráče. Jsou jimi pohyb herních objektů, stavba hradů, sběr many a souboje, kterých se hráč neúčastní. Pro každý z nich zvážíme možnosti jeho zjednodušení a vyhodnotíme jejich přínos pro naši implementaci.

3.2.1 Grafika

Náročnost zobrazování herního světa závisí na počtu objektů v něm a také na úrovni jejich detailu. Vykreslování objektů ve velké vzdálenosti je možné zjednodušit tím, že zmenšíme počet vrcholů jejich meshe. Můžeme postupovat dvěma způsoby – buď algoritmicky redukujeme vrcholy v původním modelu, nebo máme pro objekt připravených více modelů s různými úrovněmi detailu.



Obrázek 3.3: Přechod mezi různými úrovněmi detailu terénu.

První z nich lze využít u terénu, kdy dílek, na kterém hráč stojí, a jeho nejbližší okolí mají detailnější mesh, než dílky vzdálenější. Podrobnost terénu může klesat postupně. Síť pokrývající terén je pravidelná a konstruována tak, aby bylo možno

její hustotu snadno měnit. Ukázku různě detailních meshů poskytuje obrázek 3.3. Díly terénu, na které hráč nedohlédne, navíc můžeme změnit na neaktivní a Unity pak nemusí řešit jejich vykreslování vůbec.

Měnit meshe postav a herních objektů s rostoucí vzdáleností je komplikovanější, protože ty v sobě tolik pravidelnosti nemají a je složité rozhodnout, který vrchol je možné vynechat. Zde by bylo vhodnější mít pro objekt připravených více modelů s různou úrovní detailu.

Samotný grafický engine Unity vykreslování také automaticky urychluje například tím, že stíny objektů se zobrazují teprve, když je kamera dostatečně blízko.

Kvůli rychlosti grafických výpočtů chceme použít různé úrovně detailů pro terén. Vytvářet různé modely objektů pro různé vzdálenosti od hráče nepovažujeme za nutné vzhledem k tomu, že ostatní objekty mají v porovnání s terénem vrcholů daleko méně.

3.2.2 Pohyb

V herním světě se kromě hráče pohybují nejrůznější příšery, nepřátelský kouzelník, eventuálně také horkovzdušné balóny, sbírající manu do hradů svých majitelů. Pohyb objektů je tak zdaleka nejčastější akcí mimo okolí hráče a řeší se nepřetržitě v průběhu celé hry. Proto je zásadní rozhodnout, jakým způsobem jej budeme v okolí hráče i mimo něj řešit. Podle požadavku (6) zadání se na přehledové mapě zobrazuje aktuální poloha všech herních objektů. Musí tedy být v každém okamžiku jasné, kde se nacházejí.

Pro pohyb herních objektů je v Unity k dispozici dobře propracovaný fyzikální model (blíže popsaný v části 2.5). S jeho využitím lze dosáhnout přirozeně vypadajícího pohybu i po nerovném povrchu terénu. Pro neaktivní herní objekty se však fyzikální výpočty neprovádějí. Pokud by se tímto způsobem měly pohybovat všechny herní objekty, je nutné, aby měly komponenty `Rigidbody` a `Collider`. V případě terénu postačí komponenta `Collider`, která zabraní propadnutí objektu terénem vlivem gravitace. Určitou nevýhodou je, že v tomto případě všechny objekty i dílky terénu musí být stále aktivní.

Alternativou je vlastní simulace fyziky pohybu. Výhodou je plná kontrola pohybu i u nezobrazovaných objektů. Velkou nevýhodou je nutnost vlastních výpočtů kolizí, s nimiž se nám téměř jistě nepovede dosáhnout kvality výpočtů Unity. Pro pohyb vzduchem mimo zorné pole hráče, který se zobrazuje pouze na přehledové mapě, jsou výpočty polohy podstatně jednodušší. U příšer pohybujících se po zemi s omezením na typ terénu je většina výpočetního výkonu potřebná na určení správného směru pohybu na rozhraní dvou prostředí (viz část 3.4.6). Tomu se nevyhneme ani v případě, že se příšera nachází mimo dohled hráče.

Kombinací obou přístupů je varianta, kdy pro pohyb sledovaný hráčem použijeme fyziku Unity a pro pohyb mimo hráčovo okolí vlastní výpočty, ve kterých jen přepočítáváme teoretickou polohu objektu v mapě. Rozdílnost rychlostí obou druhů pohybů hráč na mapě pravděpodobně vůbec nepostřehne. Problematický by byl přechod ze stavu, kdy se těleso nezobrazuje a pouze se mění hodnoty jeho souřadnic, zpět do zobrazovaného herního světa. U složitějších objektů a zejména u objektů složených z pohyblivých dílů je obtížné určit výšku nad terénem a zabránit tomu, aby se v okamžiku aktivace nepřekrývaly části dvou objektů.

Kvůli převažujícím výhodám fyzikálního modelu Unity jsme se rozhodli využít

ho pro veškerý pohyb. Tato volba je sice spojena s nutností zobrazovat v každém okamžiku celý herní svět a související vyšší výpočetní náročností, ale při rozumné velikosti mapy na běžném počítači, který splňuje hardwarové požadavky Unity, by neměla hru citelně zpomalovat. Při testování herní mapy o velikosti 10 na 10 dílků s dvaceti příšerami byla dosažená frekvence 60 snímků za sekundu.

3.2.3 Sbírání many a stavba hradu

Mimo okolí hráče může nepřátelský kouzelník sbírat manu, nebo stavět svůj hrad. Četnost i výpočetní náročnost obou těchto akcí jsou minimální. Přirozeným zjednodušením dějů je vynechání zobrazování použitého kouzla a okamžitá realizace jeho následků. Manu také sbírají balóny a krabi, to ovšem žádný speciální efekt neprovází nikdy, stačí, že se nachází dostatečně blízko.

3.2.4 Souboje

Také souboje mohou probíhat v nepřítomnosti hráče, pokud spolu bojují příšery a nepřátelský kouzelník. Mezi souboje můžeme zařadit také napadení hradu nebo balónu nesoucího manu kouzelníkem nebo příšerou. V úvahu můžeme vzít také potenciální střetnutí více počítačem ovládaných kouzelníků v případě dalšího rozšiřování hry.

Nejsnazší variantou by bylo u každého souboje rovnou rozhodnout výsledek a nastolit finální stav. Tedy zabít jednoho ze soupeřů nebo zničit hrad či balón. Výsledek souboje v tomto případě závisí na schopnostech a aktuálním stavu jeho účastníků, případně obsahuje také prvek náhody. Nevýhodou je, že takto simulovaný souboj celý proběhne v jediném okamžiku a nedovoluje tak hráči, aby se do něj v průběhu zapojil.

Reálnější průběh střetu nabízí varianta, kdy vynecháme pouze vytváření doprovodných efektů kouzel. Místo vykreslování kouzla jen spočítáme pravděpodobnost zásahu, nebo rovnou předpokládáme, že k němu došlo. Opětovné použití kouzla je omezeno uplynutím minimální doby. Díky tomu je délka viditelného i neviditelného souboje zhruba stejná. Navíc lze snadno přejít k normálnímu zobrazování souboje, pokud se k němu hráč přiblíží. Takováto bitva nemusí nutně končit zabitím jednoho z účastníků, ale může ji ukončit i útěk jednoho ze soupeřů.

Poslední možností je nerozlišovat mezi viditelným a neviditelným soubojem a i ve střetnutí, které hráč nevidí, vytvářet efekty kouzel a vyhodnocovat jejich kolize se soupeři. Při tomto přístupu nemusíme v kódu rozlišovat více variant soubojů. Jedná se ovšem o výpočetně nejnáročnější postup, kdy je potřeba zobrazovat celý částicový systém kouzla a řešit jeho kolize s okolím.

Pro naši implementaci jsme zvolili kompromisní variantu, tedy dělání kouzel bez doprovodných efektů. Ušetří se tak výpočty zobrazování částicových systémů kouzla, které by stejně nebyly vidět. Zároveň souboje nejsou tak časté (jedním z účastníků musí být kouzelník, kterých v herním větě není mnoho), aby bylo nutné použít nejjednodušší možnou implementaci.

3.3 Umělá inteligence

Problém vytvoření umělé inteligence je součástí bodů (4) a (5) zadání. Týká se implementace nepřátelských kouzelníků a příšer. Požadavky na tyto dva základní typy nepřátel se značně liší.

Chování příšer je podstatně jednodušší. Mají v zásadě za úkol pouze pohybovat se po určitém území a případně útočit na kouzelníka v nejbližším okolí. Určitým zpestřením tohoto chování je pouze evoluce krabů, kteří by krom normálního pohybu měli navíc ještě sbírat okolo ležící kuličky many.

Komplexnější by mělo být chování nepřátelských kouzelníků. Ti by měli působit dojmem, že „logicky“ postupují s cílem zvládnutí náročnějších úkolů. Například pro dosažení co nejvyšší úrovně hradu musí nejprve hrad postavit a také mít pokaždé dostatek many na jeho postupné vylepšování. Při sběru many se mohou rozhodovat, jestli sbírat volně ležící manu, nebo zabíjet příšery, po kterých mana také zůstává. V souboji s příšerami i hlavním hrdinou by měli takticky volit kouzla a uvážit, zda je lepší v boji setrvat, nebo se pokusit utéct.

Nyní představíme dva možné přístupy pro tvorbu umělé inteligence a vyhodnotíme, jak odpovídají našim požadavkům. V závěru pak vyřešíme, jak konkrétně budeme zvolený přístup implementovat.

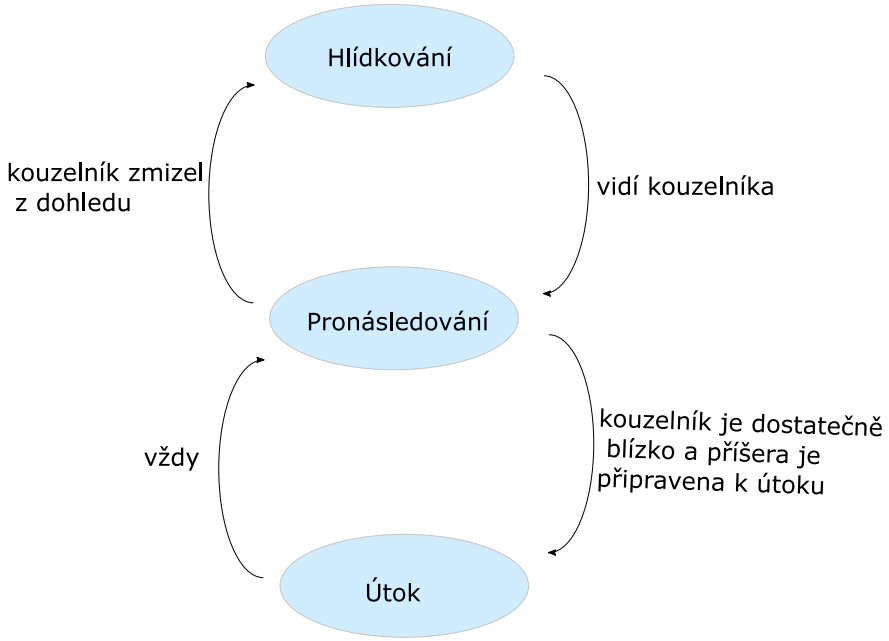
3.3.1 Stavové automaty

Základní možností implementace umělé inteligence jsou konečné stavové automaty. Automaty umožňují definovat různé stavy, v nichž se objekt může nacházet, mechanismus pro přechod z jednoho stavu do dalšího a určit pravidla, kterými se přechody mezi stavy řídí. Problematika tvorby umělé inteligence vycházející ze stavových automatů je popsána v kapitole 2 *Finite States Machines and You* knihy [2] a v kapitole 2 *State-Driven Agent Design* v knize [3], ze kterých jsme vycházeli.

Při vytváření umělé inteligence založené na automatech se veškeré její chování pokusíme rozdělit do jednotlivých stavů. Příkladem může být stav hlídkování na určitém území, stav, ve kterém nepřítel sbírá manu, stav, kdy bojuje a podobně. Dále musíme určit podmínky přechodů mezi zvolenými stavy. Například pokud příšera hlídkuje na svém území a do jejího zorného pole se dostane kouzelník, může přejít do stavu útoku nebo pronásledování. Naopak pokud hráče pronásledovala, ale ten jí zmizel z dohledu, vrátí se zpět k hlídkování. Často se ještě definuje jednorázové chování při příchodu do nebo opuštění daného stavu.

Pro naše požadavky na chování příšer se konečné automaty jeví ideální. Jejich mechanismus přesně odpovídá tomu, jak jsme jejich chování popsali. Snadno tak můžeme vytvořit stavový automat, který ho popisuje. Jeho ukázku nabízí schéma na obrázku 3.4. Vystačíme si s pouhými třemi stavy a jednoduchými podmínkami přechodů mezi nimi. Jednorázové akce při změně stavu zde nijak nevyužíváme. Základní podobu není problém rozšířit i pro složitější chování krabů nebo krakenů.

Výhoda použití konečných automatů spočívá především v jejich jednoduchosti, jak implementační, tak výpočetní. Navíc vzniklá struktura je přehledná a snadno udržovatelná i s možností přidání nových stavů. Jde ovšem o postup vhodný spíše pro simulaci jednoduššího chování, kde stačí vymezit pouze několik základních stavů a podmínek pro přechody mezi nimi.



Obrázek 3.4: Stavový automat příšery.

Naopak jejich použití při implementaci nepřátelských kouzelníků by bylo spíše komplikované. V jejich chování bychom museli rozlišit mnohem více stavů a definovat složité podmínky přechodů mezi nimi. Například po získání kuličky many je třeba se rozhodnout, jestli v jejím získávání pokračovat, nebo se už věnovat jiné činnosti směřující k vyššímu cíli. Pro tento typ chování je tak lepší využít jiné postupy.

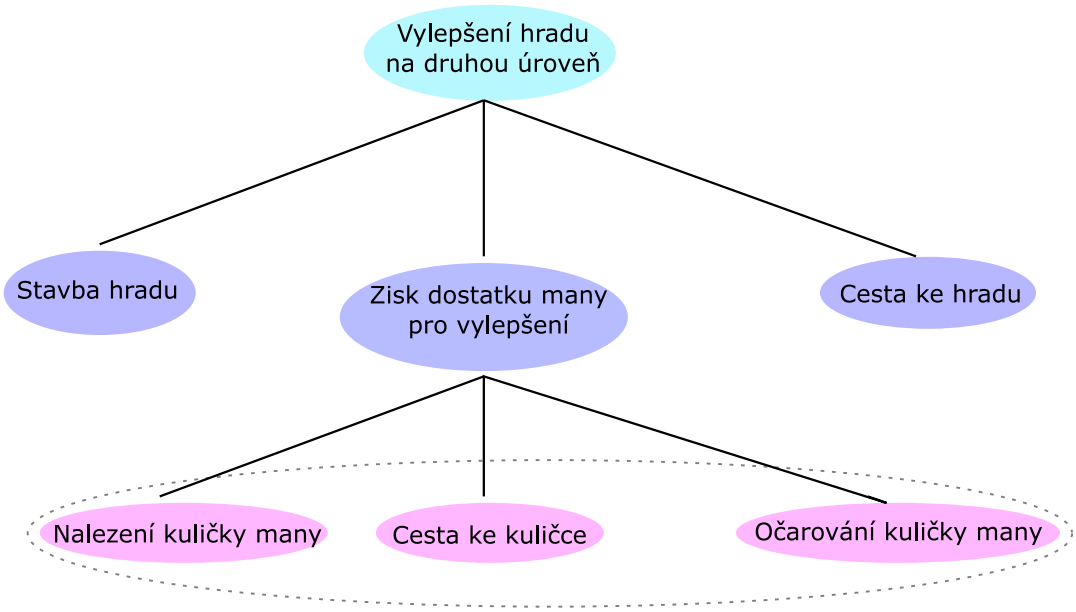
3.3.2 Chování řízené cíli

Druhým možným přístupem při tvorbě umělé inteligence je chování řízené cíli (*Goal-Driven Behavior*). Při jeho popisu vycházíme z kapitoly 9 *Goal-Driven Agent Behavior* knihy [3]. V této variantě je chování umělé inteligence rozloženo do hierarchie cílů. Ty můžeme rozdělit na jednoduché a složené. Příkladem jednoduchého cíle může být dosažení určitého místa v mapě nebo sebrání konkrétní kuličky many v dohledu. Jak název napovídá, složené cíle jsou takové, které se skládají z více dílčích cílů, ať už jednoduchých, nebo zase složených. Například ke splnění cíle získání určitého množství many bude potřeba několikrát zopakovat menší cíle v podobě hledání many v okolí a jejího sbírání.

Postup entity s umělou inteligencí, označované jako agent, je pak následovný. Snaží se dosáhnout svého nejvyššího cíle. Ten se typicky bude skládat z více podúkolů. Vybere si jeden z nich a ten se bude snažit vyplnit. Pokud i tento jde dále dělit, postupuje takto dále, až narazí na nějaký jednoduchý úkol, o jehož splnění se bude snažit v daném okamžiku.

Chování řízené cíli lze snadno implementovat v podobě zásobníkové datové struktury. Na zásobník bude agent přidávat nové cíle a odebírat ty, které se mu již povedlo splnit. Na vrcholu zásobníku je vždy aktuální úkol agenta. Pokud je na vrcholu složený cíl, k jehož dokončení je nejprve potřeba splnit nějaký podcíl, tento podcíl se přidá na vrchol zásobníku a stane se novým aktuálním úkolem agenta.

Systém požadavků na chování nepřátelského kouzelníka můžeme snadno převést do podoby popisované hierarchie cílů. Jako názornou ukázku tohoto postupu rozebereme splnění komplexního úkolu vybudování hradu na úrovni dva. Navrženou pyramidu úkolů ukazuje schéma na obrázku 3.5. Prvním krokem k hradu na úrovni dva je samotná výstavba hradu. Dále je třeba shromáždit manu potřebnou pro jeho vylepšení. Jde o další složený cíl, jehož podúkoly jsou nalezení nejbližší many, cesta k ní a její získání kouzlem. Celý postup musíme opakovat tak dlouho, dokud kouzelník nezíská dostatek many. Pak zbývá návrat k postavenému hradu a konečně jeho vylepšení.



Obrázek 3.5: Složený úkol pro vylepšení hradu.

Pokud není určen jediný cíl, o který agent usiluje celou hru, je potřeba, aby se vždy po určité době „zamyslel“ a určil, na jaký hlavní cíl se zaměří. Frekvence přehodnocování cílů by měla být dostatečně velká, aby agent zvládal dynamicky reagovat na měnící se podmínky, ale zároveň ne příliš vysoká, aby se stihl zvolenému cíli dostatečnou dobu věnovat. V našem případě by strategiemi, mezi kterými kouzelník volí, mohlo být například vylepšování hradu, boje s příšerami a kouzelníkem ovládaným hráčem, nebo naopak útěk před nimi.

Pro implementaci obyčejných příšer je použití složených cílů zbytečné. Veškeré jejich chování odpovídá spíše jednoduchým cílům stejné úrovně, těžko bychom tak rozumně sestavovali jejich hierarchii.

3.3.3 Závěr a vlastní implementace

Pro implementaci umělé inteligence nepřátel jsme se snažili volit přístup, který co nejvíce odpovídá tomu, jakým způsobem o jejich chování uvažujeme. V jednoduchého chování příšer můžeme rozlišit několik různých stavů a jako ideální implementace se nám proto jevil stavový automat. Naopak komplexní úkoly, které má vykonávat nepřátelských kouzelník, jsme se rozhodli reprezentovat jako hierarchii podúkolů v rámci chování řízeného cíli.

Základní možností implementace stavových automatů je stav reprezentovat jako abstraktní třídu, na níž budou definovány metody pro přechody z něj a vykonání akce v něm. Konkrétní stavy pak představují potomci takovéto třídy. Pokud zvolíme tento přístup, můžeme pro vytváření stavového automatu využít podporu Unity v podobě komponenty `Animator`.

My jsme se ovšem rozhodli pro jiné řešení, které více odpovídá tomu, že bychom chtěli umožnit snadné rozšiřování již popsaného chování příšer. Tím je definice stavů i přechodů jako virtuálních metod. Ukazatelem na aktuální stav příšery je pak delegát. Tato varianta umožňuje snadno změnit stavové chování příšery předefinováním dané metody, aniž bychom museli jakkoli měnit celkovou podobu automatu. Stejně tak lze do automatu jednoduše začlenit nový stav tím, že změníme definici některé již zavedené přechodové metody.

Chování řízené cíli je nejčastěji řešeno pomocí tzv. stromů chování (*Behavior Trees*). Zde má hierarchie cílů stromovou strukturu. Samotné splnitelné úkoly definují listy a vnitřní uzly stromu pouze určují způsob zpracování svých potomků. Pro tvorbu *Behavior Trees* v základní verzi Unity není vestavěna žádná podpora.

V naší implementaci používáme vlastní mírně odlišný zjednodušený postup tvorby cílů. Přizpůsobujeme se při něm konkrétním požadavkům, které máme na chování nepřátelského kouzelníka. Cíle představují potomci abstraktního předka, který deklaruje metodu jejich zpracování. V ní se buď plní samotný úkol, nebo se kontroluje, zda jsou splněny všechny potřebné podcíle.

3.4 Pohybový systém

S použitím umělé inteligence těsně souvisí také řízení samostatného pohybu entit v prostoru. To se týká příšer, nepřátelského kouzelníka i balónů shromažďujících manu. Potřebujeme zajistit, aby se herní objekty nejen dokázaly dostat na cílovou pozici, ale také aby cestou neprocházely překážkami nebo sebou navzájem.

Specifickým požadavkem naší hry je omezení pohybu některých druhů příšer pouze na určitý typ terénu. Konkrétně krakeni by měli zůstávat ve vodě a červi a krabi zase na souši. Ostatní příšery, stejně jako kouzelníci a balóny létají vzduchem a druh terénu pod nimi ani překážky v něm je nijak neomezují.

Při volbě vhodného řešení pohybového systému vycházíme z následujících předpokladů. V herním světě se nachází minimum statických překážek, které jsou většinou daleko od sebe. Představují je hrady kouzelníků a amfory s kouzly. Dynamickými překážkami jsou kuličky many a okolní pohybující se postavy. Z obvyklého množství many potřebného pro ukončení hry můžeme jejich počet shora omezit na tři sta. Posledním předpokladem je tvar pevniny, kde očekáváme spojitou plochu s jednoduchým okrajem bez složitějších výběžků nebo malých ostrůvků.

3.4.1 Řízení

Strategie, jak se má herní objekt pohybovat, se označují jako *Steering Behaviours*. Pro naši hru bychom chtěli implementovat následující chování. Příšery by měly umět chodit po zvolené cestě, pronásledovat kouzelníka nebo se pohybovat náhodně. Nepřátelský kouzelník by navíc měl být schopen utéct před útočníkem.

Nejjednodušší je chování balónu, u kterého pouze stačí, aby dorazil na zvolené místo s manou nebo zpět do hradu.

V naší implementaci řízení vycházíme z kapitoly 3 *How to Create Autonomously Moving Game Agents* knihy [3]. Používáme metodu *Seek* pro cestu maximální možnou rychlostí k cíli, ať už je cíl statickým bodem cesty, nebo jde o pohybujícího se nepřítele. Opakem je *Flee*, kdy agent postupuje co nejrychleji směrem od objektu. Strategie *Arrive* slouží k doražení do cíle s postupným zpomalováním a využije ji pro své cesty balón. Metoda *Wander* umožňuje náhodnou chůzi po okolí působící přirozeně oproti ustavičným náhodným změnám směru. Zvolená strategie určuje, jakým směrem a intenzitou, má na agenta působit síla, tu pak aplikujeme na komponentu `Rigidbody`.

3.4.2 Vyhýbání se statickým překážkám

Vyhýbání se překážkám můžeme rozdělit na dvě fáze. Nejprve je třeba detekovat, že se v okolí překážka nachází, a dále je třeba na tuto informaci zareagovat.

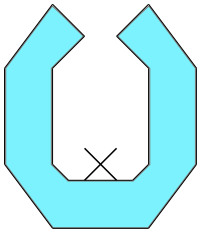
Jednou z možností detekce překážek je použít v každém kroku *Raycasting* nabízený Unity, tedy vyslat od agenta paprsek a vyhodnotit, zda je před ním překážka. Alternativou je mít připravený seznam překážek a pro každou otestovat, jak daleko od agenta se nachází. S velkým množstvím překážek by takovýto postup byl výpočetně náročný, ale vzhledem k tomu, že statických překážek očekáváme jednotky až malé desítky, není to problém. Navíc při výpočtu vzdálenosti můžeme snadno zohlednit vlastní souřadnice mapy, zatímco *Raycasting* by se složitě přizpůsoboval tomu, že protilehlé dílky mapy na sebe ve skutečnosti mají navazovat.

Dále je třeba na překážku zareagovat. Nejjednodušší variantou je posunout agenta zpět, aby tak k nárazu nedošlo. To ovšem způsobí, že se zasekne na místě, dokud nezmění cíl své cesty. Pro takovéto chování bychom nepotřebovali ani předchozí detekci překážky, ta by se mohla vyřešit prostřednictvím dostatečně velkého `Collideru` obklopujícího agenta.

Další variantou je, aby při přiblížení se k překážce začala na agenta působit síla směrem od ní. To by mu mělo umožnit vyhnout se jí a zároveň pokračovat směrem k původnímu cíli. Síla směrem od překážky by měla být nepřímo úměrná vzdálenosti agenta od ní. Výhodou tohoto přístupu je snadná implementace a kompatibilita s řízením pohybu taktéž pomocí sil. Výsledný pohyb je pak složením sil určených pro vlastní pohyb k cíli a sil zamezujících kolizi s překážkou. Obtížné je správně tyto síly vyvážit, aby agent do překážky nenarazil, ale ani si od ní neudržoval zbytečně velký odstup. Dalším nedostatkem této strategie je, že v komplexnější překážce by agent mohl uváznout. Takovouto situaci ukazuje obrázek 3.6, kde je agent vyznačen křížkem. V naší situaci naštěstí žádná taková překážka není, amfory jsou malé, hrady obdélníkové a jednotlivé překážky, náhodně umístěné na mapě, budou od sebe pravděpodobně dost daleko.

3.4.3 Vyhýbání se dynamickým překážkám

Bylo by možné se k dynamickým překážkám chovat stejně jako ke statickým, jak popisuje předchozí část. Zde ale můžeme s výhodou využít fyziku enginu Unity. Stačí, aby kolidující objekty měly `Rigidbody` a `Collider` a jejich srážka



Obrázek 3.6: Složitá překážka.

bude řízena Unity. Srážka přitom není o nic horším řešením než složitěji naplánované vyhnutí se překážce. Pro hráče vypadá zcela přijatelně, pokud příšera cestou zavadí o kuličku many a trochu s ní pohne, nebo pokud do sebe větší množství příšer naráží ve snaze dohnat kouzelníka.

3.4.4 Algoritmus A*

Alternativní často používanou metodou navigace je algoritmus A*. Zde herní mapu reprezentuje orientovaný graf. Agent pak cestuje mezi jeho vrcholy. Optimální cesta v grafu se určuje prohledáváním do šířky s využitím prioritní fronty. Hrany v grafu mohou být ohodnocené. To může představovat náročnost pohybu po určitém terénu. Detaily implementace algoritmu pro hledání cesty lze najít v kapitole 8 *Practical Path Planning* knihy [3].

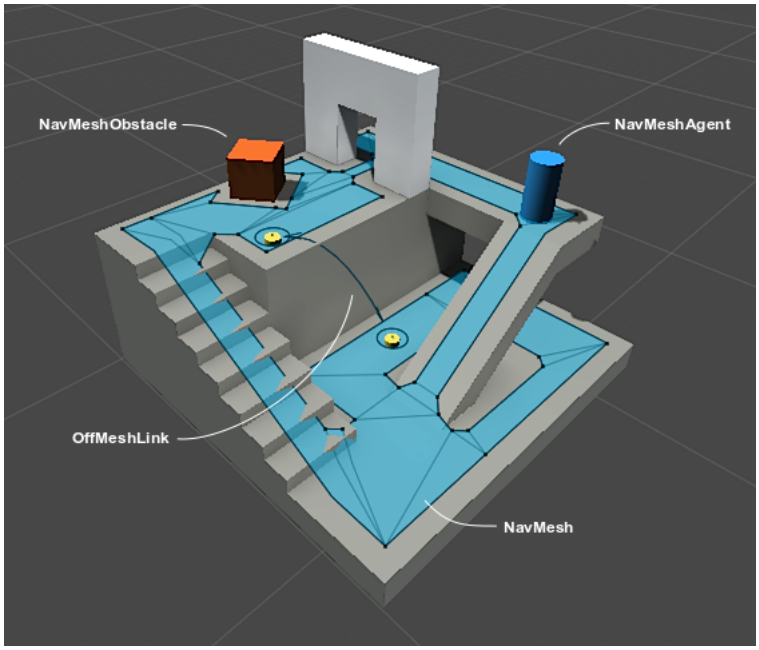
Výhodou použití A* pro naši hru je, že současně řeší problém překážek i omezení na určitý terén. Pro jeho implementaci stačí zrušit možnost průchodu po některých hranách. S ohodnocením hran bychom také mohli zavést různé rychlosti na různých površích. Díky prohledávání celého grafu by agent byl schopen najít optimální cestu i s uvážením překážek. Také by se dokázal orientovat i ve složitém tvaru terénu. Například kraken by tak dokázal najít cestu z téměř uzavřené zátoky a podobně. Další výhodou je, že v grafu lze snadno vytvořit správné propojení konců mapy prostým přidáním hran mezi protilehlými vrcholy.

Nevýhodou by byla paměťová náročnost udržování grafu celého světa s dostatečně detailním rozlišením. Složité by bylo i jeho počáteční vytvoření, při kterém je třeba odlišit souš a moře, a modifikace při přidávání překážek. Kvůli malému množství překážek by také většinou bylo hledání optimální trasy zbytečné, protože by šlo o přímou spojnici výchozího a cílového bodu.

3.4.5 NavMesh

Podpora pro navigaci agentů v prostoru je vestavěna i v prostředí Unity. Její součástí jsou komponenty `NavMesh`, `NavMeshAgent` a `NavMeshObstacle`. `NavMesh` obsahuje datovou strukturu popisující povrch, po kterém se agent může pohybovat. `NavMeshObstacle` definuje překážku v terénu. Samotný agent má pak komponentu `NavMeshAgent`, díky které v rámci `NavMeshe` hledá cestu k cíli. Ukázku `NavMeshe` s překážkou můžeme vidět na obrázku 3.7.

Při hledání cesty v `NavMeshi` se využívá algoritmus A*. Výhody jeho využití jsou tedy stejné, a to že řeší hledání cesty současně s vyhýbáním překážkám. Zde navíc také zaručuje, že agenti se vzájemně vyhýbají.



Obrázek 3.7: Použití `NavMeshe`. Převzato z [4].

Problémem je opět jeho využití v našem komplikovaném nespojitém herním světě. Zde by každý dílek musel mít svůj vlastní `NavMesh` dostatečně propojený s ostatními, aby fungoval přechod agentů přes hranici světa. Navíc podkladový mesh povrchu pro navigaci se typicky připravuje v editoru. Jeho vytváření v běhovém čase včetně správného vyřešení hranic terénů by bylo implementačně náročné. Oproti tomu výpočetní náročnost přípravy by nám nevadila, neboť by pouze prodloužila čas vytváření herní úrovně při jejím spuštění.

3.4.6 Omezení podle typu terénu

Kromě vyhýbání překážkám požadujeme u některých druhů příšer ještě omezení jejich pohybu pouze na konkrétní prostředí. Rozpoznání typu terénu, kde se objekt nachází, je možné podle výšky terénu v daném místě. Je však třeba zajistit, aby pohybující se objekt včas rozeznal hranici a zareagoval změnou směru a rychlosti.

Triviálním řešení tohoto problému je například přesunutí objektu při překročení hranice zpět do posledního místa, kde se objekt vyskytoval ve „správném“ prostředí. Nevýhodou tohoto řešení je, že objekt se zastaví vždy v prvním místě kontaktu s hranicí prostředí a pokud se nezmění cíl pohybu, objekt zůstává na místě, i když by mohl cíle dosáhnout jen malou změnou směru.

Herní engine Unity pro tyto situace nabízí použití komponenty `NavMesh`, která s použitím metody A* umožní objektům vyhnout se hranicím libovolného tvaru. Nevýhody použití `NavMeshe` jsou popsány v jemu věnované části výše.

V kapitole 3 *How to Create Autonomously Moving Game Agents* knihy [3] je pro tyto případy popsána metoda *Wall Avoidance*, která je použitelná pro případ vyhnutí se dlouhé rovné zdi. Přímá aplikace této metody pro naši hru je obtížná v tom, že procedurálně generovaný terén prakticky nemá žádné hranice prostředí, které by se daly aproximovat úsečkou rozumné délky.

Jako zajímavá se jeví modifikace metody *Wall Avoidance*, kde se kolizní úsečka

generuje až při přiblížení objektu k hranici prostředí. Při pohybu objektu se vzorkuje terén ve třech místech před objektem a v místě objektu. Pokud je některý ze vzorků za hranicí prostředí, proloží se třemi body ležícími před objektem rovina a vygeneruje se korekční síla, která se snaží vrátit objekt do správného prostředí. Tato síla působí ve směru gradientu roviny a její velikost je úměrná rychlosti pohybu objektu a vzdálenosti od hranice. V případě, kdy korekční síla nestačí na zabránění vstupu do nesprávného prostředí kvůli velké rychlosti a nevhodnému směru pohybu, se objekt pokusí vyhledat správné prostředí v nejbližším okolí a pokud není úspěšný, je přesunut zpět do posledního místa ve správném prostředí.

3.4.7 Závěr

Pro naši implementaci systému chování jsme se rozhodli použít kombinaci působících sil řízení všech pohybů. Tedy jak pro pohyb v zamýšleném směru, tak pro vyhýbání překážkám, eventuálně i omezení pohybu na určité území. Výhodou tohoto přístupu je, že všechna tři kritéria pohybu řešíme současně a stejným způsobem. Nárazy mezi dynamickými objekty ponecháváme na fyzikální simulaci enginu Unity.

4. Procedurální generování terénu

Jak jsme si stanovili v podkapitole 3.1.4, budeme terén herního světa naší hry vytvářet procedurálně. Procedurálně generovaný obsah je tvořen algoritmem na základě několika vstupních parametrů, a nemusí ho tak ručně připravovat herní designér. To je obzvláště výhodné u velkých herních světů, kde by vytváření rozsáhlého unikátního prostředí bylo velmi časově náročné. Procedurální generování lze využít pro mnoho aspektů hry přes textury a terén, až po hudbu nebo chování umělé inteligence. V této kapitole popíšeme jeho konkrétní využití pro tvorbu terénu v naší implementaci a projdeme jednotlivé fáze generování od počátečních náhodných čísel až po zobrazitelný díl herního světa.

4.1 Náhodná čísla

Základem pro procedurální generování je náhodnost. Obvykle není třeba generovat zcela náhodné jevy, ale pro praktické použití stačí více nebo méně kvalitní generátory náhodných čísel, z jejichž hodnot je potom možno odvodit chování algoritmu. Tato tzv. pseudonáhodná čísla vytvářejí posloupnost, která se zdá být náhodná, ale ve skutečnosti jsou generována nějakým deterministickým algoritmem.

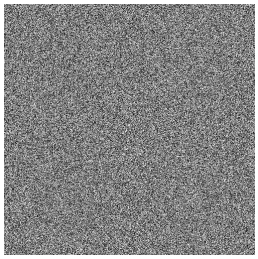
Generátory náhodných čísel implementované ve standardních knihovnách (jazyk C/C++, .NET) obvykle pracují s určitou vstupní hodnotou označovanou jako semínko (*seed*). Ta může být zvolena uživatelem nebo odvozena automaticky například ze systémového času počítače. Výhodou použití pseudonáhodných čísel je, že pro stejně zvolený vstup algoritmu lze dosáhnout stejného výsledku. „Náhodný“ výsledek tedy lze zopakovat.

4.2 Šumová funkce

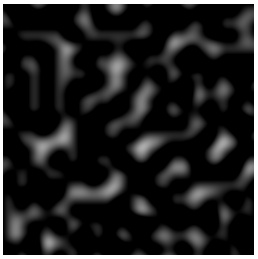
V našem případě tvorby terénu potřebujeme náhodně vygenerovat jeho výšku v jednotlivých bodech čtvercové sítě. Sousední místa ve skutečném světě se svojí nadmořskou výškou typicky příliš neliší. Aby vytvořený terén vypadal přirozeně, nemůžeme tedy použít přímo hodnoty poskytnuté generátorem náhodných čísel, které na první pohled vypadají jako zcela nekorelované. Je třeba použít nějakou spojitou šumovou funkci. Nejčastěji se používá metoda Kena Perlina z roku 1983 označovaná jako Perlinův šum, její vylepšenou verzi popisuje autor v článku [5]. Jejím základem je interpolace gradientů v mřížce. Srovnání nekorelovaného a spojitého šumu nabízí obrázky 4.1 a 4.2.

4.3 Další charakteristiky šumu

Samotná šumová funkce většinou generuje hodnoty mezi nulou a jedničkou. Pro dosažení přirozenějšího vzhledu můžeme šumovou funkci použít vícekrát s různou amplitudou a periodou a jednotlivé hodnoty sečíst do finálního výsledku. Jednotlivá volání funkce s různými parametry označujeme jako oktávy. Tento postup přináší dva nové parametry – persistenci a lakunaritu. Persistence udává,

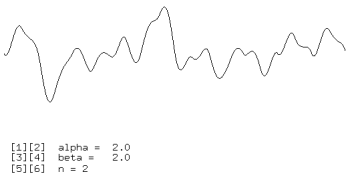


Obrázek 4.1: Nekorelovaný šum.

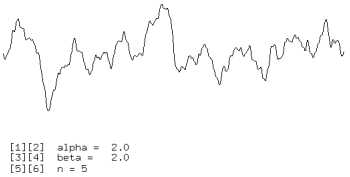


Obrázek 4.2: Spojitý šum.

kolikrát se zmenší amplituda v následující oktávě. Lakunarita naopak, kolikrát se zvýší v nové oktávě frekvence. Společně tak umožňují vytvářet různě velké detaily v horizontu terénu. Výsledky použití více oktáv na horizontu můžeme vidět na obrázcích 4.3 a 4.4, na obou je persistence i lakunarita rovna dvěma, každá další oktáva je tedy dvakrát hustší a má dvakrát menší amplitudu.



Obrázek 4.3: 1D šumová funkce s dvěma oktávami. Převzato z [6].



Obrázek 4.4: 1D šumová funkce s pěti oktávami. Převzato z [6].

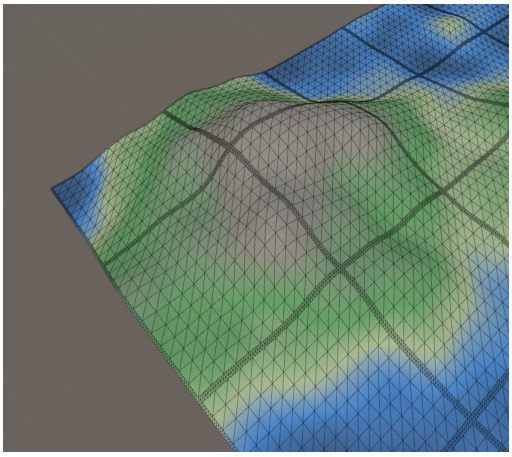
4.4 Skutečné výšky

Dalším krokem je pak úprava hodnot získaných ze šumové funkce. Správnou návaznost protilehlých částí mapy řešíme tím, že na všech okrajích základní herní plochy vytvoříme terén pokrytý vodní hladinou v nulové výšce. Je tedy potřeba zmenšit okrajové hodnoty získané ze šumové funkce pomocí takové spojité funkce, aby přechod k nulovým hodnotám na okrajích vypadal přirozeně. Další úpravou výchozích výšek je sjednocení výšek odpovídajících rozmezí, kde se nachází voda, na jedinou hodnotu, aby vodní hladina byla vodorovná. Nakonec zbývá jen připravené hodnoty vynásobit vhodným koeficientem a získat tak skutečné výšky terénu pro herní svět.

4.5 Mesh

Po určení výšek známe všechny tři souřadnice bodů mřížky terénu v herním světě. Dalším krokem je vytvoření trojúhelníkové sítě označované jako mesh, kterou 3D engine Unity dokáže zobrazit. Vrcholy jednotlivých trojúhelníků leží v uzlech čtvercové mřížky. Vzhled meshe vytvořeného v Unity ilustruje obrázek 4.5.

Mesh obsahuje informace o poloze jednotlivých vrcholů trojúhelníků. K nim přísluší souřadnice pro texturu označovaná jako *uv* (podle os na textuře, s tím, že



Obrázek 4.5: Mesh terénu.

x, y, z jsou již použity pro herní svět). Dále obsahuje informace o trojúhelnících. Typicky je trojúhelník popsán třemi indexy do pole vrcholů. V neposlední řadě je pak potřeba k trojúhelníkům doplnit jejich normály, které se využívají při stínování.

4.6 Úroveň rozlišení detailů

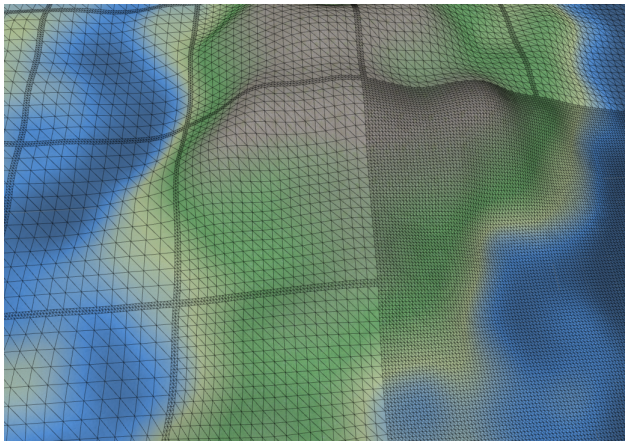
Pro snížení výpočetní náročnosti při zobrazování vzdálenějších míst v terénu popsané v části 3.2.1 je mapa rozdělena na díly, z nichž každý má vlastní mesh. Podle vzdálenosti od hráče můžeme jednotlivé dílky vykreslovat více či méně podrobně, případně nejvzdálenější dílky nezobrazovat vůbec. Různých úrovní zobrazení detailů může být více. Pro každou z nich je třeba mít vlastní mesh. Je nutné zajistit, aby na sebe sítě jednotlivých dílků navazovaly, a to i v případě, že každá má jinou úroveň detailů.

Sítě pro zobrazování s nižší úrovní detailů vytvoříme ze základní (nejpodrobnější) sítě tak, že v ní ponecháme pouze některé vrcholy. Vznikne tak řidší, opět čtvercová síť, ve které bude menší počet elementárních trojúhelníků. Přechody mezi různými úrovněmi detailu ukazuje obrázek 4.6, kde v pravé části je výchozí nejdetailnější mesh, uprostřed z původního meshe zůstává každý druhý vrchol a úplně vlevo jen každý třetí.

4.7 Textura a stínování

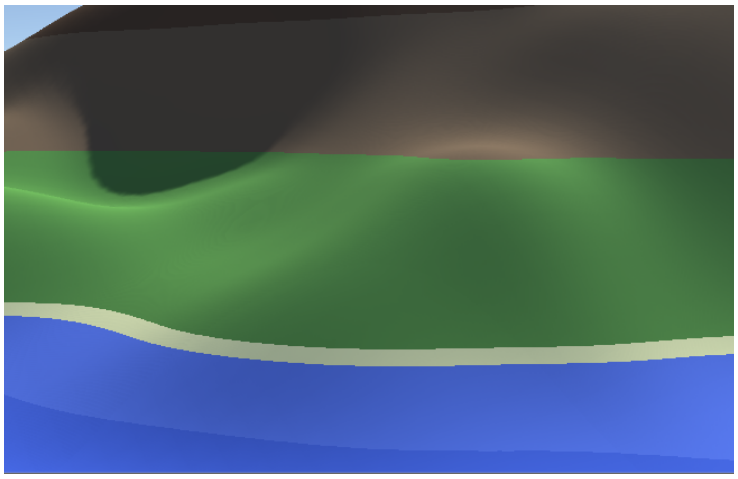
Posledním krokem přípravy terénu je přiřazení barev bodům meshe, což v prostředí Unity uděláme dodání vhodného materiálu. Na základě materiálu a aktuálního osvětlení se aplikací stínovacího modelu vypočítá barva v každém bodě povrchu. Díky stínování vzniká dojem hladkého přechodu mezi trojúhelníkovými ploškami meshe.

Terén herního světa by měl obsahovat různé povrchy jako vodu, travnatou plochu, nebo povrch horniny. Jednotlivé typy terénu tvoří výšková patra, která spojitě přechází jedno v druhé. Každému patru přiřazujeme jeho počáteční výšku, barvu a také texturu.



Obrázek 4.6: Přechod mezi různými úrovněmi detailu terénu.

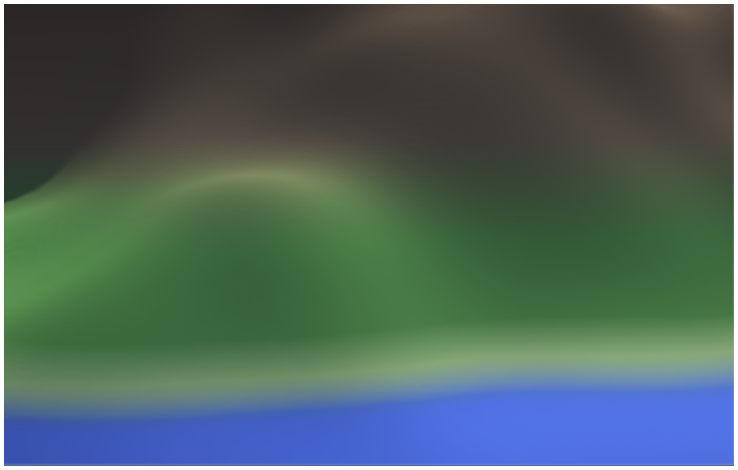
Na následujících obrázcích vidíme různě pokročilé varianty tvorby výškových pater terénu. V obrázku 4.7 jsou jim přiřazeny pouze barvy a ponechány ostré hranice přechodů mezi nimi. Lepší dojem vzbuzuje obrázek 4.8, kde na hranici pater dochází k mísení jejich barev, díky čemuž přechod působí přirozeněji. Obrázek 4.9 také používá plynulý přechod pater. Kromě barev navíc přiřazuje jednotlivým povrchům i textury. Díky nim se výsledná barva bodů povrchu liší i v rámci samotného patra.



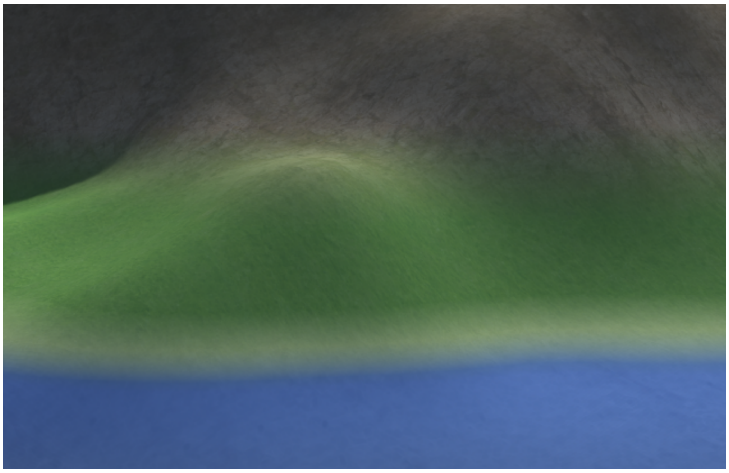
Obrázek 4.7: Barevná výšková patra s ostrou hranicí.

Textury se používají k vytvoření reálného vzhledu tělesa. Umožňují napodobit drobné detaily povrchu objektů, které by bylo obtížné nebo nevhodné modelovat ve třech rozměrech. Textura je bitmapa, která je „nanesena“ na povrch objektu. K jejímu správnému umístění slouží *uv* souřadnice ve vrcholech meshe. Ukázku použitých textur z balíčku Unity *Standard Assets* nabízí obrázky 4.10 a 4.11.

Při pokrývání terénu je často třeba texturu opakovat, protože není vhodné snažit se vytvořit bitmapu pokrývající celý herní svět. Přitom je nutné zajistit, aby opakováním nevznikaly viditelné artefakty. Pro správné škálování do různých směrů textury používáme triplanární mapování.



Obrázek 4.8: Barevná výšková patra s pozvolným přechodem.



Obrázek 4.9: Výšková patra s pozvolným přechodem doplněná o texturu.

Obrázek 4.10: Textura pro travnatý povrch.

Obrázek 4.11: Textura pro vodní hladinu.

5. Vývojová dokumentace

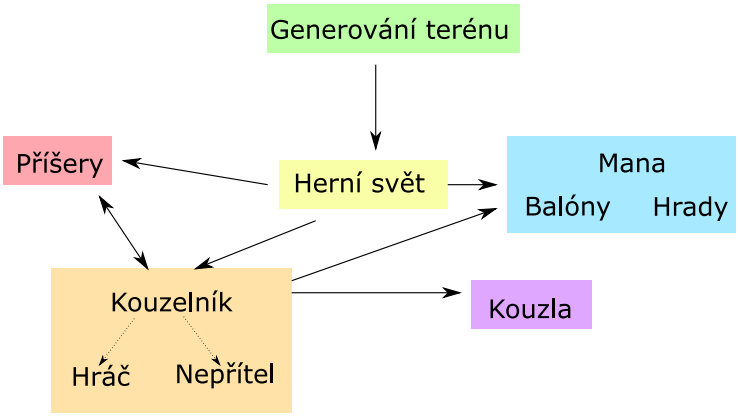
Projekt celé hry Master of the Carpet pro vývojové prostředí Unity verze 2018.1 se nachází na přiloženém DVD ve složce `MasterOfTheCarpet`. Všechny komponenty hry jsou uloženy ve složce `Assets` v následujících adresářích:

- V `Resources` se nachází předpřipravené herní objekty tematický rozdělené do dalších složek. Použité modely objektů byly vytvořeny programem *Blender* a vektorové obrázky byly převzaty z `https://www.freepik.com/`.
- Ve složce `Scripts` jsou uloženy všechny použité skripty. I jejich adresářová struktura je dále členěna.
- Složka `Scenes` obsahuje připravené scény hry.

V celém projektu se využívají tři scény:

1. *EditorScene* vytváří prostředí, v němž je možné pracovat s oknem *Level Designeru* a navrhovat nové herní mapy. Ve hře samotné se tato scéna neobjevuje. Skripty používané v rámci scény editoru popisuje část 5.9.
2. Scéna *Menu* ukazuje úvodní obrazovku a výběr herní úrovně. Skripty, které scénu obsluhují, se nachází v adresáři `Menu`. Jejich úkolem je pouze umožnit navigaci v prostředí a spuštění herní mapy ve scéně *GameScene*.
3. Ve scéně *GameScene* se vytváří celý herní svět na základě zvolené mapy.

V herní scéně se odehrává téměř veškerá logika hry. Ve zbytku dokumentace se budeme věnovat především mechanismům fungujícím v herním světě. Objekty a skripty ve scéně hry můžeme rozdělit do následujících vzájemně propojených skupin, jak ukazuje schéma 5.1.



Obrázek 5.1: Objekty scény hry.

Následující části se detailněji věnují jednotlivým oblastem.

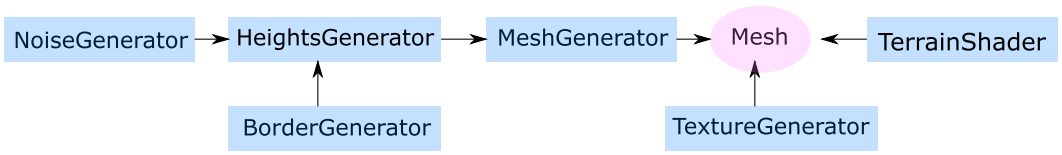
- Samostatná skupina skriptů řeší procedurální generování terénu. Zde se na začátku hry vytváří díly herní mapy. Skripty se používají také ve scéně editoru při tvorbě náhledu navržené mapy. Implementaci generování terénu přibližuje část 5.1.

- Herní svět zahrnuje správu jednotlivých dílů terénu. Veškeré fyzické objekty jako příšery, kouzelníci nebo hrady se pak vyskytují v rámci něj. Třídy, které herní svět spravují, popisuje část 5.2.
- Příšery se pohybují terénem a napadají kouzelníky. Detaily jejich implementace najdeme v části 5.5.
- Kouzelníci kouzly sbírají manu a do jejich hradu ji pak přenáší horkovzdušné balóny. Třídy, které mechanismus sběru many řeší, popisuje část 5.3.
- Kouzelníci se pohybují terénem, sbírají manu a staví hrady. Mohou také bojovat s příšerami a používat kouzla. Rozlišujeme mezi kouzelníkem ovládaným hráčem a jeho soupeřem řízeným počítačem. Detailní popis komponent entity kouzelníka najdeme v části 5.6.
- Ve hře se uplatňuje systém kouzel, která mohou kouzelníci používat. Jeho popisu je věnována část 5.4.

Kromě herního světa je důležitou součástí scény *GameScene* také uživatelské rozhraní, které s objekty světa úzce spolupracuje. Popisu jeho fungování se věnuje část 5.8.

5.1 Generování terénu

Herní terén je generován procedurálně. Postup jeho tvorby v naší implementaci ilustruje schéma 5.2.



Obrázek 5.2: Postup generování terénu.

Nejprve se ve třídě `NoiseGenerator` ze šumové funkce vygenerují relativní výšky terénu. Ty se kombinují s hodnotami ze třídy `BorderGenerator`, které zajistí, že v okrajových bodech mapy je výška nulová. Dále jsou relativní hodnoty transformovány a převedeny na absolutní v rámci třídy `HeightsGenerator`. Následuje tvorba podkladů pro tvorbu meshe ve třídě `MeshGenerator`. Barvu terénu určuje textura materiálu připravená třídou `TextureGenerator` a vlastní shader `Terrain`.

Všechny třídy, které se na přípravě terénu podílí, můžeme najít v adresáři `Terrain Generation`.

NoiseGenerator

Tvorba jednoho dílu terénu začíná ve třídě `Noise Generator`. Ta prostřednictvím metody `GenerateNoiseMap` vytvoří dvourozměrné pole čísel s hodnotami mezi 0 a 1, které odpovídají relativním výškám bodů v terénu. Parametry metody

jsou výška a šířka generovaného dílu a také jeho celková pozice v rámci mapy. Metoda pak vzorkuje Perlinův šum implementovaný Unity v daných bodech, přičemž pro každou oktávu se ještě přidává náhodné posunutí.

Vlastní parametry použitého šumu jsou metodě předány jako instance třídy `NoiseSettings`. Zde je uveden počet oktáv, jejich persistence (to, jak se v nové oktávě zmenšuje amplituda) a lakunarita (to, jak v nové oktávě narůstá frekvence). Dále také zvolený ofset celku, seed pro náhodné posunutí jednotlivých oktáv a celkové přiblížení.

BorderGenerator

Požadujeme, aby na sebe okraje herního světa navazovaly a umožňovaly tak hráči plynulý přesun mezi nimi. Řešením je od původních hodnot v mapě šumu spojitě odečítat hodnoty tak, aby se v okrajových polích blížily nule.

Metoda `GenerateBorderMap` třídy `BorderGenerator` vytvoří dvourozměrné pole hodnot pro daný díl tak, že postupně od středu směrem ke krajům mapy generované hodnoty rostou. Přitom postupuje tak, že každý bod x, s polohou vztaženou k celkový rozměrům mapy v maximové metrice, dosadí do funkce

$$\frac{x^a}{(x^a + (b - bx)^a)}.$$

HeightsGenerator

Třída `HeightsGenerator` prostřednictvím své metody `GenerateHeightMap` umožňuje vytvořit už finální výšky terénu v herním světě. Nejprve kombinuje hodnoty z `NoiseMap` s těmi z `BorderMap`, aby všechny okraje měly nulovou výšku a navazovaly na sebe. Následuje transformace pomocí animační křivky. Zde se nemění rozmezí hodnot výšek, ale dá se dále upravit tvar terénu. Nakonec se relativní výšky vynásobí zvoleným koeficient a vzniknou hotové výšky pro herní svět. Animační křivka a koeficientem pro násobení jsou oba obsaženy v instanci třídy `HeightMapSettings`.

MeshData a MeshGenerator

Veškerá data potřebná k výrobě meshe obsahuje třída `MeshData`. Jednotlivé instance odpovídají meshi konkrétních rozměrů s danou úrovní detailu zobrazení. Postupně se jako `Vector3` přidávají souřadnice jednotlivých vrcholů a *uv* souřadnice pro použitou texturu. Následně se přidávají trojúhelníky v podobě trojice indexů jejich vrcholů. Po zadání všech trojúhelníků je ještě třeba dopočítat jejich normály v metodě `CalculateNormals`. Hotový Mesh vytvořený na základě dat uložených ve třídě vrací funkce `CreateMesh`.

Novou instanci `MeshData` pro danou výškovou mapu a zvolenou úroveň detailu zobrazení vytváří třída `MeshGenerator` v metodě `GenerateMeshData`. Ta nejprve rozdělí jednotlivé vrcholy na ty, kterou jsou mimo mesh, na vnějším a vnitřním okraji jeho hranice, vnitřní a ty, které se vynechají v rámci malé úrovně detailu. Následně je předává instanci `MeshData`. Poté doplní ještě trojúhelníky a spočítá jejich normály. Tím jsou data zcela připravena pro vytvoření meshe.

TextureGenerator a TerrainShader

Terén herního světa je rozdělen na výšková patra, která odpovídají různým povrchům (například vodě, písku nebo skále). Jednotlivé typy terénů jsou popsány instancemi třídy `TerrainLevel`. Ta určuje výšku, ve které vrstva začíná (jde o relativní výšku vztaženou k nejnižší a nejvyšší výšce terénu), jak rychle přechází v další vrstvu, její barvu a intenzitu. Dále pak obsahuje texturu pro daný typ terénu a její měřítko.

Pro zobrazení terénu 3D enginem Unity je nutno připravit potřebné údaje. Jde zejména o barvu jednotlivých bodů, kterou vypočte shader. Aby se definované typy terénu promítly do jeho vzhledu, je třeba použít shader, který s nimi umí pracovat. Dále je třeba nastavit vlastnosti jednotlivých typů terénu v datových strukturách materiálu. Druhý z těchto úkolů má na starost třída `TextureGenerator` v rámci metody `ApplyToMaterial`. Zde se pro každou charakteristiku vrstvy vytvoří pole, jehož prvky pak odpovídají hodnotě dané charakteristiky v jednotlivých typech terénu.

`Terrain shader` totiž všechny informace o terénních typech očekává v jednotlivých polích. Zásadní je jeho metoda `surf`, která pro bod povrchu terénu vrátí jeho barvu. Shader prochází jednotlivé typy terénu a na základě výšky bodu určí míru ovlivnění bodu příslušným typem terénu. V rámci každého typu terénu se kombinuje jeho základní barva a barva vzorku z textury. Pro volbu polohy vzorku se používá tzv. triplanární mapování s cílem dosažení přirozenějšího vzhledu škálování textury do různých směrů.

Dalším úkolem třídy `TextureGenerator` je na základě definovaných terénů vytvořit texturu mapy světa. K tomu slouží metoda `CreateTextureFromNoiseMap`, která z výšek terénu určí příslušné barvy pro mapu.

5.2 Herní svět

Správu herního světa a práci se souřadnicemi v něm mají na starosti třídy definované skripty ve složce `Game World`. Informace o jednotlivých dílcích mapy obsahuje třída `MapChunk`. Všechny dílky spravuje třída `MapController` a seznamy jednotlivých objektů v herním světě vede `GameOverview`. Souřadnice konkrétního bodu na mapě popisuje instance třídy `Position`. Pro práci s nimi slouží třída `Coords`.

MapChunk

Instance třídy `MapChunk` představují jednotlivé dílky herní mapy. Jejich souřadnice v mapě je pevně určena, ale samotné umístění v herním světě se mění v závislosti na pohybech hráče. Fyzickou reprezentací dílku je `GameObject`, ke kterému třída přidává `MeshFilter`, `MeshRenderer` a `MeshCollider`. V rámci inicializace se pro dílek vyváří všechny meshe odpovídající definovaným úrovním detailu. V každém snímku se pak určuje vzdálenost dílku od hráče a volí se mesh se správným rozlišením. Může se také změnit jeho umístění tak, aby byl stále co nejblíže hráči. Veřejná metoda `GetTerrainHeight` umožňuje určit výšku terénu dílku ve zvoleném bodě. Dále je možné do terénu umisťovat další herní objekty

jako modely hradů, kuličky many, amfory nebo příšery prostřednictvím příslušných metody.

MapController a MapHeights

Popis herní mapy je uložen v instanci třídy `MapHeights`. Ta udává velikost herního světa i jednotlivých dílků mapy a obsahuje předpřipravené pole výšek terénu a definice jeho pater. Popisuje také rozmístění kuliček many, amfor s kouzly i příšer. V neposlední řadě pak určuje, jaké množství many je třeba získat pro dokončení herní úrovně, a definuje počáteční polohu hráče, eventuálně i nepřátelského kouzelníka.

Přípravu celého herního světa zajišťuje skript `MapController`, který se v herním světě vyskytuje v rámci objektu *Map*. Postupuje přitom dle informací v přiřazené instanci `MapHeights`. Nejprve připraví materiál a vytvoří jednotlivé dílky herní mapy. V rámci třídy jsou definované používané úrovně rozlišení detailů a vzdálenosti od hráče, při kterých se mezi nimi přechází. Dále v terénu rozmístí manu, kouzla, příšery a kouzelníky. Všechny takto vytvořené objekty se registrují v seznamech třídy `GameOverview`.

Dalšími úkoly třídy `MapController` jsou tvorba textury přehledové mapy vygenerovaného světa v metodě `GetWholeMapTexture` a změna výšek terénu v průběhu hry (například při budování hradu, nebo vzniku sopky), kterou řeší metoda `ChangeTerrainHeights`.

Position a Coords

Určování polohy objektů ve hře je komplikované, jelikož se celý herní svět posouvá podle aktuální polohy hráče. Je tak rozdíl mezi teoretickou pozicí v rámci herní mapy a skutečným umístěním v herním světě. Polohu objektu v herní mapě popisuje instance třídy `Position`. Ta určuje, ve kterém herním dílku se objekt nachází, a jeho relativní polohu v něm.

Pro práci s těmito odlišným souřadnými systémy slouží metody třídy Coords. V rámci metod `PositionToWorld` a `WorldToChunk` třída umožňuje převádět teoretickou pozici v mapě na tu v herním světě a naopak. Dále také dokáže najít nejkratší spojnici (metoda `FindShortestPathDirection`) a určit vzdálenost mezi body mapy. Metoda `UpdatePosition` řeší přechody objektů mezi různými dílky mapy.

5.3 Sběr many

Herní mechanismus sběru a ukládání many řeší trojice tříd `ManaBall`, `Balloon` a `Castle`, které se nachází ve složce `Game Objects`. Skript `Manaball` je součástí jednotlivých kuliček many. `Balloon` řídí horkovzdušný balón, který získanou manu přenáší, a `Castle` představuje kouzelníkův hrad, do nějž se mana ukládá. Všechny tři třídy implementují rozhraní `IMapPoint`, díky čemuž se zobrazují v přehledové mapě světa (viz 5.8).

ManaBall

Chování kuličky many řídí skript `ManaBall`. Po vytvoření kuličky je třeba nastavit její polohu pomocí metody `Setup`. Ve `FixedUpdate` každého snímku se pozice kuličky aktualizuje pro případ, že se kulička terénem pohybuje. Po zásahu kouzelníkovým kouzlem se volá metoda `ChangeOwner`, která změní barvu kuličky a přidá kuličku do seznamu kouzelníkem posbírané many v jeho komponentě `ManaManager` (viz 5.6). Pokud je kulička sebrána, dojde ke zrušení jejího herního objektu metodou `DestroySelf`.

Balloon

Úkolem horkovzdušného balónu řízeného skriptem `Balloon` je přenášet kouzelníkem získanou manu do jeho hradu. Pro kuličky many jej vysílá hrad metodou `CollectManaBall`. Poté, co se balón ke kuličce dostane, se pro její sběr použije metoda `StoreMana`. V okamžiku návratu do zámku mu balón shromážděnou manu předává metodou `ReleaseAllMana`. Přenosová kapacita balónu je omezená a určuje ji proměnná `maxManaStorage`.

Pohyb balónu řídí jeho komponenta `MovementController` (viz 5.7). S využitím metody `Arrive` třídy `SteeringBehaviors` se balón v každém snímku může pohybovat buď směrem k nové kuličce many, nebo se vracet do svého hradu.

Balón má omezenou životnost, která se zkracuje s každým jeho napadením. Výchozí hodnotu určuje proměnná `maxLife`. Reakcí na útok je volání `TakeDamage`, které snižuje životnost balónu. Pokud ji balón vyčerpá zcela, použije se metoda `DestroySelf`, balón uvolní získanou manu a obnoví se u svého hradu. Při zničení hradu zaniká v `DestroySelf` nadobro i balón.

Potomkem základní třídy `Balloon` je `PlayerBalloon`. Ten její chování rozšíří o spolupráci s herním objektem `StatusBar`, který ukazuje aktuální životnost a množství many shromážděné v balónu (viz 5.8). S využitím virtuálních metod `PlayerBalloon` nejprve vykoná akce předka a poté o změně stavu informuje `StatusBar`.

Castle

Úkolem herního objektu hradu a jeho skriptu `Castle` je spravovat kouzelníkem získanou manu. Hrad si udržuje seznam všech hráčem získaných, dosud nesesbíraných kuliček many a metodou `TryToSendBalloons` pro ně posílá balón, pokud to kapacita hradu i balónu dovolí. K ukládání přinesené many slouží metoda `StoreManaFromBalloon`.

Kapacita i maximální životnost hradu závisí na jeho aktuální úrovni. Aplikací kouzla *CastleSpell* se úroveň hradu zvyšuje (metoda `Upgrade`), naopak pokud se vyčerpá životnost hradu, jeho úroveň se sníží (metoda `Downgrade`). Při každém útoku na hrad se volá jeho metoda `TakeDamage`. Pokud je hrad zcela zničen, je odstraněn zavoláním metody `DestroySelf`. Spolu s hradem zanikne i jeho balón a uložená mana se v podobě kuliček uvolní do okolí.

Při změně úrovně hradu se mění i jeho vzhled, tedy herní objekt, který ho reprezentuje. Výchozí hodnoty maximální životnosti, úložné kapacity a také zvolené herní objekty pro jednotlivé úrovně zámku jsou ve třídě uloženy ve statických polích.

Základní třídu `Castle` rozšiřuje třída `PlayerCastle`, která se používá pro hráčův hrad a umožňuje změny svého stavu zobrazovat na přehledové liště `StatusBar` (viz 5.8).

5.4 Systém kouzel

Do systému kouzel můžeme zařadit abstraktní třídu `Spell` a všechny její potomky. Dále sem patří amfory, v nichž jsou kouzla uložena, a grafické reprezentace vyčarovaných kouzel, které představují potomci třídy `SpellBall`.

SpellAmphora

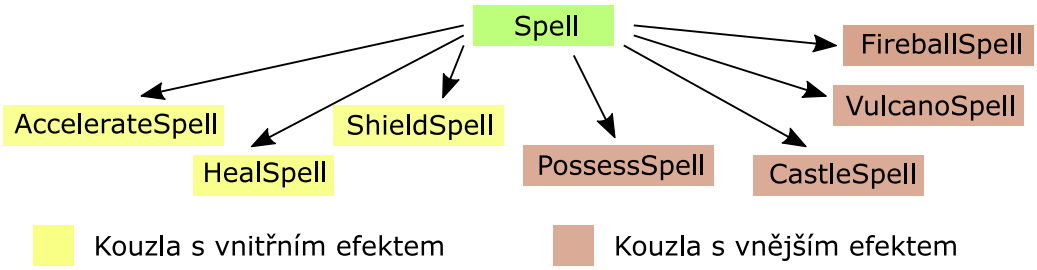
Chování a vlastnosti amfor s ukrytým kouzly definuje skript `SpellAmphora`. Ten implementuje rozhraní `IMapPoint` a umožňuje vyznačení amfor na mapě herního světa (viz 5.8). Hráč získává kouzlo ukryté v amfoře metodou `GetSpell`. Které kouzlo takto získá, určuje proměnná `spellIdx`.

Spell

Třída `Spell` představuje společný abstraktní základ všech kouzel. Základními vlastnostmi jsou `cooldown`, který udává minimální dobu, po jejímž uplynutí je možné kouzlo znovu použít, a `manaNeeded`, která určuje minimální množství many požadované pro vyvolání kouzla.

Virtuální metoda `IsReady` ověřuje pro danou instanci, zda je kouzlo připraveno k použití. Tedy zda má kouzelník dostatek many a není příliš brzy po předchozím vyvolání kouzla. Pro složitější podmínky používání kouzel může být tato metoda v potomcích předefinována. Stejně tak by každý potomek měl rozšířit virtuální metodu `Cast`, jejímž jediným parametrem je čarující kouzelník (instance třídy `Wizard`). V základní verzi metoda pouze spustí odpočet do dalšího nejbližšího možného použití kouzla.

Skripty obecného kouzla `Spell` i všech jeho potomků jsou uloženy ve složce `Spells`. Schéma systému tříd kouzel nabízí obrázek 5.3.



Obrázek 5.3: Systém tříd kouzel.

Implementovaná kouzla můžeme rozdělit do dvou skupin. První představuje kouzla, která nemají žádný vizuální efekt a pouze mění vnitřní stav kouzelníka. Jde o následující kouzla:

- `AccelerateSpell`, které po zvolenou dobu (vlastnost `duration`) urychluje pohyb kouzelníka vpřed,

- `HealSpell`, které navrátí zraněnému kouzelníkovi část života (kolik přesně určuje vlastnost `healAmount`),
- `ShieldSpell`, které po zvolenou dobu (vlastnost `duration`) chrání kouzelníka před plnou silou nepřátelských útoků.

Druhým typem jsou kouzla, která mají fyzickou podobu v herním světě a jejichž efekt vyvolá až dopad kouzla na určitý objekt. Takto se chovají kouzla `CastleSpell`, `FireballSpell`, `PossessSpell` a `VulcanoSpell`. Součástí jejich vlastností je objekt kouzla (`SpellBall`), které se má v herním světě objevit. V metodě `Cast` se pak pouze vytváří nová instance příslušného objektu před kouzelníkem. Doplňující vlastnost `Range` určuje, z jaké vzdálenosti by je měl používat nepřátelský kouzelník.

SpellBall

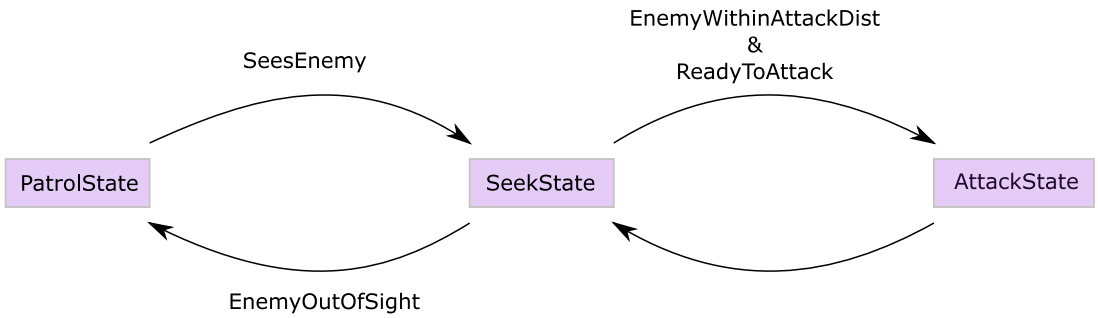
Vizuální efekty při vyvolání kouzla jsou realizovány vytvořením herního objektu s komponentou, jež je potomkem abstraktní třídy `SpellBall`. Vlastnostmi základní třídy jsou počáteční rychlost vytvořeného kouzla (`startSpeed`) a doba jeho existence (`lifeTime`). Třída také potomkům nabízí metodu `Focus` pro změnu směru pohybu kouzla k určenému cíli. Nárazy kouzel do objektů se detekují prostřednictvím `Collider`ů. Konkrétní implementace jsou následující.

- `CastleSpellBall` v místě dopadu do terénu vybuduje hrad, pokud kouzelník ještě žádný nemá. V opačném případě reaguje pouze na kolizi s hradem jeho vylepšením.
- `DuelSpellBall` zasaženého kouzelníka připoutá k tvůrci kouzla. Parametry `duration` a `maxDistance` udávají dobu trvání připoutání a maximální vzdálenost mezi útočníkem a obětí.
- `FireballSpellBall` při jakékoli kolizi s objektem vybuchne (vytvoří herní objekt `Explosion`). Pokud je objekt zranitelný, neboli implementuje rozhraní `IVulnerable`, pak mu ubere život. Míru poškození určuje vlastnost `damageAmount`.
- `PossessSpellBall` při nárazu přebarví volnou kuličku many a získá ji pro kouzelníka.
- `VulcanoSpellBall` v místě dopadu na terén vybuduje sopku neboli vytvoří herní objekt `Vulcano`.

5.5 Systém příšer

Obecným základem všech příšer je abstraktní třída `Monster`. Chování příšer určuje stavový automat implementovaný pomocí delegátů. Pro každý stav zde existuje metoda, která ho provede, a také metoda pro rozhodování o přechodu do dalšího stavu. Díky tomu, že obě tyto metody jsou virtuální, lze je v potomcích snadno měnit nebo rozšiřovat, aniž by bylo nutné měnit fungování ostatních metod. V každém snímku se provede akce definovaná v aktuálním stavu. Základní

podobu stavového automatu ukazuje schéma 5.4. To obsahuje následující stavy a přechody mezi nimi.



Obrázek 5.4: Základní podoba stavového automatu příšer.

- V `PatrolState` příšera hlídkuje po vytyčené trase (nastavené pomocí metody `SetPath`) nebo se pohybuje náhodně, pokud žádná trasa není definována.
- V metodě přechodu `PatrolStateTransitions` se kontroluje, zda se v blízkosti příšery nenachází její nepřítel. Vzdálenost, na kterou příšera objekt zaregistruje, určuje hodnota proměnné `withinSightDist`. Samotnou kontrolu okolí provádí metoda `SeesEnemy`. Ta také podle nastavených parametrů `attacksCastle` a `attacksBalloons` rozhoduje, zde příšera může napadnout i hrady a balóny. Pokud příšera nepřátelský cíl najde, přejde do stavu pronásledování.
- Při pronásledování (stav `SeekState`) se příšera snaží dostatečně přiblížit k nepříteli, aby na něj mohla zaútočit.
- Ze stavu pronásledování příšera přejde do útoku, pokud je na to připravena a její cíl je dostatečně blízko. Maximální vzdálenost pro útok udává proměnná `attackDist`. Pokud se naopak nepříteli podaří utéci, neboli dostane se od příšery dále než určuje parametr `releaseDist`, příšera se vrací k hlídkování.
- V rámci `AttackState` příšera útočí na cíl. V základní verzi se příšera musí dotknout svého cíle, který musí implementovat rozhraní `IVulnerable`, a volá jeho metodu `TakeDamage`. Sílu útoku určuje vlastnost `attackDamage`.
- Po útoku se v `AttackStateTransitions` příšera automaticky vrátí do stavu pronásledování. Dobu, za kterou bude opět připravena k útoku, udává proměnná `attackCooldown`.

Příšery implementují rozhraní `IVulnerable` a mohou tak být cílem útoku. Zranění utrpí voláním metody `TakeDamage`. Život příšery spravuje instance třídy `HealthManager` (viz 5.6). Pokud příšera zemře, použije se metoda `Die` a příšera se změní na kuličky many. Množství many, která po příšeře zůstane, udává vlastnost `manaBallsCount`.

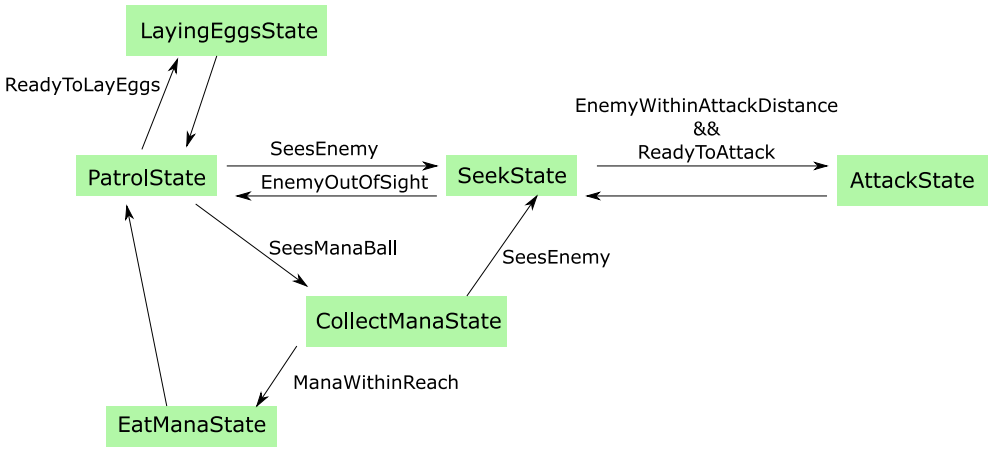
Pro pohyb příšer se používá komponenta `MovementController` (viz 5.7). V rámci instance třídy se volí parametry pohybu jako například prostředí, v

kterém se příšera pohybuje, nebo jeho rychlost. Třída `Monster` také naplňuje rozhraní `IMapPoint`, díky čemuž se poloha příšery zobrazuje na přehledové mapě světa (viz 5.8).

Implementované příšery

Potomky třídy `Monster` jsou následující konkrétní příšery. Příslušné soubory se nachází ve složce `Monsters`.

- `Wasp` základní chování příšer nijak nerozšiřuje.
- Ve třídách `Worm` a `FlyingWorm` je změněna podoba boje tak, že příšera útočí na dálku. Pokud je v jejím okolí hráč, zobrazuje se navíc podoba kuličky útočného kouzla. V opačném případě se stejně jako v základní verzi sníží život nepřítele přímo a nevyhodnocuje se zásah kouzlem.
- `Kraken` také mění podobu útoku. Kromě ohnivého útoku může příšera použít ještě speciální poutající kouzlo (`DuelSpellBall`). Minimální interval jeho opětovného použití určuje vlastnost `duelCooldown`. Obdobně jako v předchozí případě i zde existují dvě varianty použití kouzla podle toho, jak daleko od akce se nachází hráč.
- `Crab` rozšiřuje základní podobu stavového automatu o nové stavy, jak ukazuje schéma na obrázku 5.5. Při kontrole přechodů ze stavu hlídkování (`PatrolStateTransitions`) hledá krab ve svém okolí kuličky many metodou `SeesManaball`. Pokud nějakou kuličku najde, přechází do nového stavu `CollectManaState` a začne se k maně přibližovat. Přímo u kuličky změní stav na `EatManaballState` a kuličku pohltí. Poté se vrací zpět do stavu hlídkování.



Obrázek 5.5: Stavový automat kraba.

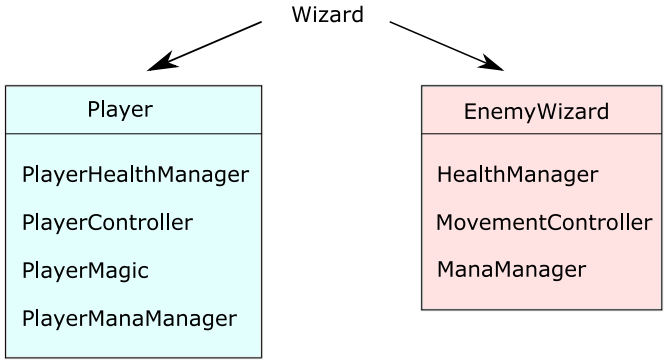
Při získání dostatečného množství many, krab pokročí na vyšší evoluční úroveň. Množství many potřebné k dovršení jednotlivých fází vývoje udává statické pole `evolutionManaRequirements`. Na nejvyšší úrovni přibývá v hlídkovacím stavu možnost snášet vejce přechodem do (`LayingEggsState`). V něm se vytváří nový objekt `CrabbEgg`, který se po definované době

(`hatchTime`) mění v nového kraba. Frekvenci kladení vajec omezuje parametr `layingEggsCooldown`.

S vývojem kraba souvisí i síla jeho útoku, která se postupně zvyšuje. Konkrétní hodnoty pro jednotlivé úrovně definuje pole `attackDamageLevels`. Krab útočí na dálku stejně jako předchozí příšery. Posbírané kuličky many navyšují základní množství many, které po zabití kraba zůstává.

5.6 Kouzelník

Ve hře se nachází dva typy kouzelníků. Jde o hlavního hrdinu ovládaného hráčem a jeho soupeře, nepřátelského kouzelníka. Kouzelníka hráče řídí skript `Player` a jeho soupeře skript `EnemyWizard`. Společným předkem obou je abstraktní třída `Wizard`. Fungování obou entit doplňují další komponenty, jejichž přehled nabízí obrázek 5.6.



Obrázek 5.6: Komponenty objektů kouzelníků.

- Informace o zdravotním stav kouzelníka udržuje a aktualizuje komponenta `HealthManager`. Třída `PlayerHealthManager` pouze rozšiřuje její fungování o zobrazování hráčova stavu na přehledové liště obrazovky.
- Správu získané many řeší skript `ManaManager`. I zde je jeho činnost rozšířena stejně jako v předchozím případě ve variantě `PlayerManaManager`.
- Řízení pohybu se u obou typů kouzelníků podstatně liší. `PlayerController` reaguje na vstupy hráče, zatímco `MovementController` řeší pohyb automatický. Pohybový systém celé hry je popsán v části 5.7.
- `PlayerMagic` je skript, který umožňuje hráči používat kouzla. Čarování nepřítele je přímo součástí třídy `EnemyWizard`.

Všechny skripty používané v rámci herního objektu kouzelníka se nachází ve složce `Wizard`.

Wizard a Player

Společný základ hráčova i nepřátelského kouzelníka představuje abstraktní třída `Wizard`. Třída naplňuje rozhraní `IMapPoint` (viz 5.8) a `IVulnerable` (popsáno dále), díky čemuž je poloha kouzelníka zobrazena na mapě a je možné ho zranit (metoda `TakeDamage`) nebo zabít (metoda `Die`).

Třída dále definuje abstraktní nebo virtuální metody pro reakci na kouzla zrychlení a duelu a informuje o jejich působení komponentu spravující pohyb (`MovementController` nebo v případě hráče `PlayerController`).

Další věcí, která je řešena společně pro oba typy kouzelníků, je odpočet času, po jehož uplynutí má kouzelník znovu k dispozici použité kouzlo.

Třída `Player` už jen doplňuje implementace poděděných metod v případě hráčem ovládaného kouzelníka.

ManaManager

O správu kouzelníkovy many se stará skript `ManaManager`. Ten vede seznam získaných kuliček many a zprostředkovává manipulaci s tímto seznamem v metodách `CollectManaBall` a `LoseManaBall`. Veřejná vlastnost `CurrentMana` informuje o aktuálním množství many získané kouzelníkem.

Rozšířením základní třídy `PlayerManaManager`. Ten plní všechny původní úkoly a navíc spolupracuje se stavovou lištou `StatusBar`, kterou informuje o změnách kouzelníkovy zásoby many (viz 5.8). Doplňuje tak chování původních virtuálních metod `CollectMana` a `LoseMana`.

PlayerMagic

Kouzlení hráčova kouzelníka zajišťuje třída `PlayerMagic`. Jejím hlavním úkolem je reagovat v metodě `Update` na kliknutí myší vytvořením zvoleného kouzla. Volbu toho, jaké kouzlo odpovídá jednotlivým tlačítkům myši, umožňuje metoda `ChooseSpell`.

Třída také kontroluje, zda se kouzelník nenachází dostatečně blízko nějaké amfory, aby získal kouzlo v ní ukryté. Nové kouzlo hráči zpřístupňuje metoda `AcquireNewSpell`.

HealthManager

Správu života entity řeší skript `HealthManager`. Používá se jak pro kouzelníky, tak pro příšery. Vlastníkem může být jakákoli entita naplňující rozhraní `IVulnerable`, které požaduje metody `TakeDamage` a `Die` a zpřístupnění informací o poloze a rychlosti objektu.

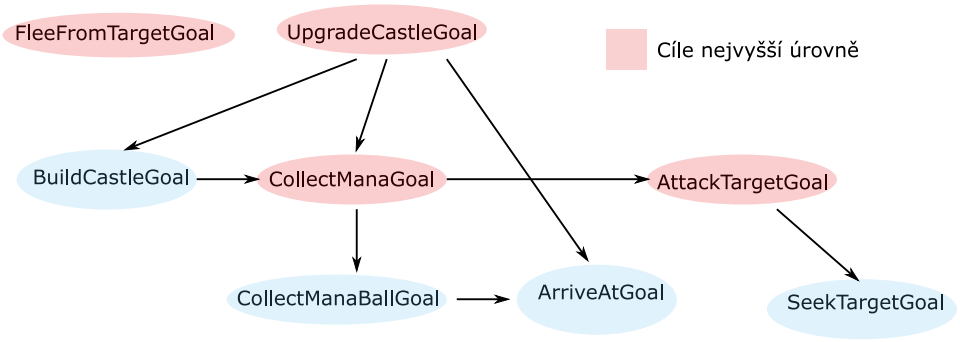
Základními vlastnostmi třídy jsou maximální zdraví (`fullHealth`) a rychlost, kterou se entita automaticky léčí (`healingRate`). Pro změnu množství života nabízí třída metody `Heal` a `TakeDamage`. Informace o aktuálním relativním zdraví nabízí metoda `GetCurrentCondition`.

`HealthManager` také umožňuje v metodě `TurnOnShield`, aby vlastník měl po určitou dobu zapnutý štít. Pokud je štít aktivován, síla veškerých nepřátelských zásahů se o čtvrtinu snižuje.

Potomkem třídy `HealthManager` je třída `PlayerHealthManager`, která k virtuálním metodám předka `TakeDamage` a `Heal` doplňuje aktualizace ukazatele zdraví hráče v objektu stavové lišty (viz 5.8). Zajišťuje také krátké zabarvení obrazovky do červena při zranění hráče.

EnemyWizard

Rozsáhlou logiku doplňující podděděné metody má v sobě třída `EnemyWizard`, která řídí chování nepřátelského kouzelníka. Jeho umělou inteligenci řešíme pomocí hierarchie cílů. Jednotlivé cíle implementujeme jako potomky abstraktní třídy `Goal` s jedinou metodou `Process`, která definuje postup vedoucí k dosažení cíle. Aktuálně rozpracované úkoly má nepřítel uloženy na zásobníku a v každém snímku zpracovává úkol na jeho vrcholu. Konkrétní volbu cílů ukazuje schéma 5.7. Jejich úkoly jsou následující.



Obrázek 5.7: Hierarchie cílů nepřítele.

- `FleeFromTargetGoal` je jednoduchý cíl, v rámci něhož se kouzelník v každém kroku pohybuje maximální rychlostí od daného objektu. Cíle je dosaženo v okamžiku, kdy je kouzelník v dostatečné vzdálenosti od zdroje ohrožení.
- `SeekTargetGoal` je opakem předchozího cíle. Zde je kouzelníkovým úkolem dostat se co nejblíže ke zvolenému objektu.
- Posledním cílem souvisejícím s pohybem je `ArriveAtGoal`. Ten slouží k přesunu a zastavení na určitém místě.
- V `AttackTargetGoal` je kouzelníkovým cílem zničit daný nepřátelský objekt. Může jít o příšeru, hráčova kouzelníka, jeho hrad nebo balón. V rámci úkolu je třeba nejprve vyplnit podúkol `SeekTargetGoal`, aby byl kouzelník při útoku dostatečně blízko nepřítele.
- `CollectManaBallGoal` dává kouzelníkovi za úkol sebrat konkrétní kuličku many. Ke splnění cíle je třeba nejprve se ke kuličce dostatečně přiblížit naplněním příslušného cíle `ArriveAtGoal`. Druhou podmínkou je pak připravenost kouzla *PossessSpell*.
- Cíl `CollectManaGoal` požaduje, aby kouzelník získal dostatečné množství many. K tomu může použít buď přímo cíl `CollectManaBallGoal`, pokud v okolí manu vidí, nebo cíl `AttackTargetGoal`, pokud se ji rozhodne získat zabitím příšery.
- Cíl `BuildCastleGoal` konkretizuje snahu o vybudování kouzelníkova hradu. K tomu je třeba mít k dispozici dostatečné množství many, kterou lze získat naplněním `CollectManaGoal`, a připravené kouzlo *CastleSpell*.

- Nejsložitějším úkolem je `UpgradeCasteGoal`, který vyžaduje vylepšení kouzelníkova hradu na zvolenou úroveň. V procesu jeho plnění je třeba vybudovat hrad prostřednictvím cíle `BuildCastleGoal`, nasbírat dostatek many (cíl `CollectManaGoal`) na jeho vylepšení a nakonec se prostřednictvím cíle `ArriveAtGoal` vrátit k hradu, u kterého je potřeba použít kouzlo *Castle-Ball*.

Kouzelník nemá po celou hru jediný hlavní cíl, ale rozhoduje se v metodě `ThinkGoals` mezi čtyřmi cíli nejvyšší úrovně, kterými jsou útok na nepřítele, útěk, vylepšování hradu a sbírání many. Rozhodování probíhá s periodou zvolenou v parametru `thinkTime`, nebo pokud nepřítel aktuálně nemá určený žádný cíl. V rámci úvah o další činnosti se pro každý z úkolů počítá jeho vhodnost v dané chvíli a jako nový hlavní cíl se zvolí ten, který byl vyhodnocen jak nejvýhodnější. Při rozhodování hraje hlavní roli aktuální stav kouzelníka a jeho vzdálenost od jednotlivých herních objektů. Prochází se proto všechny objekty registrované v seznamech třídy `GameOverview` a vyhodnocuje se jejich vzdálenost od hráče. Maximální vzdálenost, ve které kouzelník na objekt ještě reaguje, udává parametr `viewDist`.

5.7 Pohybový systém

V rámci pohybového systému řešíme změny polohy hráče i navigaci příšer, nepřátelského kouzelníka a balónů přenášejících manu. Ovládání hráče definuje skript `PlayerController`. Entity, jejichž pohyb je řízen automaticky, používají jako komponentu `MovementController`. Jejich správné směrování v herním světě má na starosti statická třída `SteeringBehaviors`.

PlayerController a CameraController

Pohyb hráče řídí třída `PlayerController`. V každém snímku mění polohu hráče a jeho rychlost v závislosti na vstupech z klávesnice. Podle pohybu myši určuje jeho natočení. Efekty kouzel, která ovlivňují pohyb hráče, řeší metody `StartDuel` a `TurnOnAccelerate`.

Kameru ve scéně ovládá skript `CameraController`. Jeho úkolem je měnit její polohu i otočení tak, aby odpovídaly aktuální pozici hráče.

MovementController

Pohyb všech ostatních pohyblivých herních objektů kromě hráče zajišťuje komponenta `MovementController`. V jejím rámci je možné zvolit maximální rychlost objektu (`maxSpeed`) a také prostředí, ve kterém se pohybuje (`type`). Pro objekty pohybující se vzduchem je relevantní také položka `heightAboveGround`, která určuje výšku nad zemí, v níž se objekt pohybuje. Pro „vylepšení“ pohybu létajících červů jsou ještě doplněny dva parametry `heightOscillation` a `heightOscFrekv`, které popisují výškové oscilace (amplitudu a frekvenci) způsobující jejich vlnění.

Vlastní pohyb na základě působící síly řeší metoda `MoveWithForce`. U objektů pohybujících se po zemi nebo po vodě se do výsledného pohybu zahrnuje i snaha

o vyhnutí se překážkám a omezení na zvolený typ terénu. V každém snímku se navíc kontroluje správná poloha objektu v mapě i herním světě.

Kromě pohybu samotného řeší třída i správné otočení ve směru pohybu a míření na nepřátelský cíl s korekcí podle jeho rychlosti(metoda `ShootDirection`). Pohyb objektu také ovlivňuje kouzlo *Duel*. Jeho efekt se aktivuje a deaktivuje v metodách `StartDuel` a `EndDuel`.

SteeringBehaviors

Směr pohybu objektu se určuje ve statické třídě `SteeringBehaviors`. Na základě informací v `MovementControlleru` a zvolené strategii pohybu je vypočtena síla, kterou je na objekt potřeba působit. Výpočty ve třídě vychází z kapitoly 3 *How to Create Autonomously Moving Game Agents* knihy [3]. Implementovány jsou následující strategie pro pohyb.

- `Seek` působí maximální silou směrem ke zvolenému cíli.
- `Flee` se používá pro pohyb maximální rychlostí od zvoleného objektu.
- `Arrive` podobně jako `Seek` namíří objekt ke zvolené pozici. Navíc přidává postupné zpomalování tak, aby se objekt na pozice zcela zastavil.
- `Wander` slouží pro náhodný pohyb objektu.
- `AvoidObstacles` určí sílu, která upravuje pohyb objektu tak, aby se vyhýbal statickým překážkám v okolí.
- `TerrainBoundary` vypočítá sílu, která objektu znemožňuje pohyb mimo zvolené prostředí.

5.8 Uživatelské rozhraní a přehledy

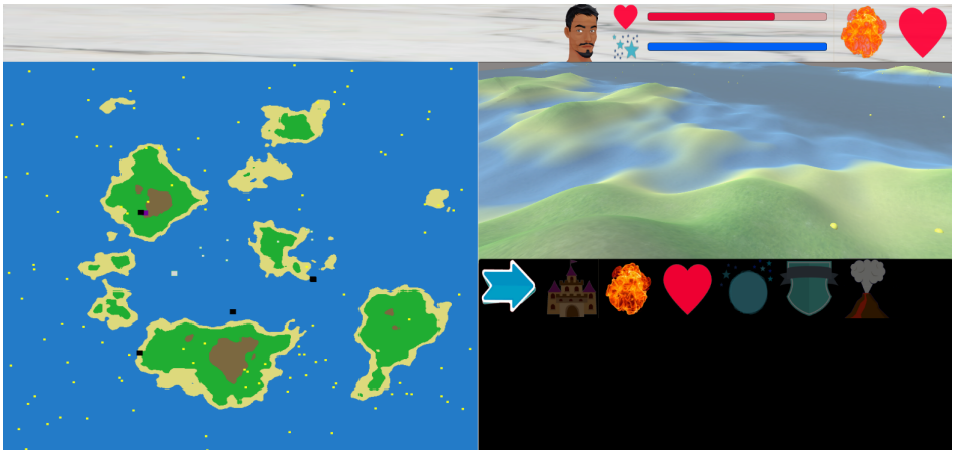
Hra obsahuje bohaté uživatelské rozhraní, kde hráč může sledovat svůj aktuální stav, přehledovou mapu světa nebo inventář svých kouzel. Rozlišujeme mezi herní a přehledovou obrazovkou. Vzhled obou vidíme na obrázcích 5.8 a 5.9.

V horní části obrazovky se vždy nachází přehledová lišta (objekt *StatusPanel*), ukazující stav hráče, jím zvolená kouzla a eventuálně také stav jeho hradu a balónu. Práci s lištou zajišťuje skript `StatusBar`. V herní obrazovce lištu doplňuje malá kruhová mapa zobrazující nejbližší okolí hráče. Jde o herní objekt *Minimap* řízený stejnojmenným skriptem.

Přehledová obrazovka je rozdělena na tři části. Její levou polovinu zabírá přehledová mapa světa v herním objektu *Map*. Její tvorbu a aktualizaci řeší skript `MapOverview`. V pravé horní části zůstává pohled hráče do okolí. Pravá dolní část obsahuje přehled hráčových kouzel v rámci objektu *SpellsOverview*. Ten spravuje skript `SpellOverview`.

Přepínání mezi herní a přehledovou obrazovkou řídí třída `OverviewScreen` v metodách `ShowOverview` a `HideOverview`.

Obrázek 5.8: Základní herní obrazovka.



Obrázek 5.9: Přehledová obrazovka.

StatusBar

Přehledovou lištu se stavem hráče, jeho hradu a balónu zobrazovanou v panelu *StatusPanel* spravuje skript `StatusBar`. Změny hráčova zdraví nebo množství jím získané many mu oznamují objekty `PlayerHealthManager` a `PlayerManaManager` popsané v části 5.6. Třída také porovnává aktuální zásobu many v hráčově hradu s hodnotou požadovanou pro výhru. Při jejím dosažení pak zobrazí obrazovku výhry (*VictoryImage*). Naopak při nezvratné smrti hráčova kouzelníka ukáže obrazovku prohry (*GameOverImage*).

Poté co hráč vykouzlí svůj hrad a balón, vznikají instance tříd `PlayerCastle` a `PlayerBalloon`(viz část 5.3).Ty nejprve informují `StatusBar` o svém vzniku v metodách `OnCastleBuilt` a `OnBalloonCreated`, díky nimž se na liště zobrazí příslušné ukazatele. `StatusBar` pak dále reaguje na události spojené se změnami životnosti nebo uložené many hradu a balónu.

Lišta také zobrazuje nastavená kouzla, která má hráč v pravé a levé ruce. Nové kouzlo se nastavuje v metodě `ChooseSpell`. V každém snímku se upravuje barva zobrazovaného kouzla tak, aby odpovídala aktuální dostupnosti kouzla. Kouzlo, na které nemá hráč dostatek many, je černé. V průběhu trvání minimálního intervalu pro opakované použití kouzla barva postupně přechází z černé zpět

k původní.

MapOverview a Minimap

Třída `MapOverview` se v herním světě vyskytuje v rámci objektu *Map*. Jejím úkolem je tvorba přehledové mapy včetně vyznačení polohy herních objektů jako jsou kuličky many, příšery nebo kouzelníci a jejich hrady a balóny. Samotnou mapu terénu získává od třídy `MapController` (viz 5.2) voláním její metody `GetWholeMapTexture`.

Všechny objekty, jejichž umístění se má v mapě zobrazit, musí implementovat rozhraní `IMapPoint`, které určuje, na jakém místě a jakou barvou a velikostí se mají zobrazovat. Nové objekty mapy se registrují metodou `AddMapPoint` a při zrušení mizí pomocí `RemoveMapPoint`.

Přehledová mapa se zobrazuje na přehledové obrazovce v rámci objektu *OverviewScreen*. Pokud je obrazovka aktivní, v každém snímku se do mapy terénu zakresluje poloha všech registrovaných objektů.

Třída `Minimap` ovládá prvek uživatelského rozhraní, který na hlavní obrazovce zobrazuje malou mapu hráčova nejbližšího okolí. Tu v každém snímku skládá z výřezů mapy celého světa, kam dokresluje další objekty v metodě `DrawMapPoints`. Navíc zobrazovaný objekt mapy otáčí tak, aby jeho orientace odpovídala natočení hráče.

SpellOverview

Skript `SpellOverview` je součástí objektu `SpellsManager` v hlavní scéně hry i editoru. Umožňuje zde spravovat seznam všech implementovaných kouzel. Informace o jednotlivých kouzlech představují instance třídy `SpellOverviewInfo`, která k samotnému skriptu kouzla přidává jeho název a ikonu.

Na začátku hry je pro každé kouzlo uvedené v seznamu vytvořeno tlačítko v panelu *SpellsOverview* na přehledové obrazovce. V obsluze kliknutí na tlačítko `SpellClick` se vybrané kouzlo nastaví do ruky kouzelníkova hráče. Na začátku hry žádné z tlačítek není aktivní. K jeho odemčení slouží metoda `UnlockSpell`.

MonsterOverview

Stejně jako v případě kouzel i pro příšery existuje skript `MonsterOverview`, který udržuje jejich přehled. Skript se nachází jako komponenta objektu *MonsterManager* ve scéně editoru i hlavní hry. Třída obsahuje seznam všech implementovaných příšer popsaných instancemi třídy `MonsterOverviewInfo`, která kromě samotného objektu příšery obsahuje ještě její název a ikonu. Doplňující informace se využívají při tvorbě záložky přidávání příšer v *Level Designeru*.

5.9 Skripty rozšíření editoru

Skripty editoru jsou umístěny v adresáři `Editor`. Vlastní nové okno editoru nazvané *Level Designer* vytváří stejnojmenný skript, který je potomkem vestavěné třídy `EditorWindow` a používá jmenný prostor `UnityEditor`. Uživatelské

rozhraní editoru funguje na principu *Immediate mode GUI*. Vykreslování obsahu okna probíhá v metodě `OnGUI`. V panelu okna se nabízí tři záložky pro tvorbu terénu, rozmísťování předmětů a přidávání postav. Jejich obsah je definován v metodách `DrawTerrainGenerationEditor`, `DrawObjectPlacementEditor` a `DrawEnemyEditor`. V panelech se používají popisky, tlačítka, obrázky a také pole, v nichž designer nastavuje parametry generované herní mapy.

Při přípravě herní mapy návrhář pracuje pouze v režimu editoru. Pro interaktivní práci s objekty na scéně při jejich tvorbě nebo rušení je třeba používat speciální skripty editorů pro jednotlivé objekty a jejich metody `OnSceneGUI`. Ty se volají ve chvíli, kdy je ve scéně vybrán objekt, pro který je editorový skript určen. Takto funguje třída `TerrainEditor`, díky níž při zvolení jakéhokoli objektu ve scéně můžeme detekovat kliknutí na místo v mapě. Jako reakce se pak do terénu umísťuje nový předmět nebo se modifikují výšky ve zvoleném bodě.

Mapu vytvořenou v editoru spravuje třída `EditorMapController`. Ta v metodě `GenerateEditorMap` generuje jednotlivé díly mapy na základě designerem zvolených parametrů uložených v instanci třídy `MapSettings`. Příprava terénu probíhá principem popsaným v části 5.1. Dále třída umožňuje měnit vzhled mapy změnou výšky terénu ve zvoleném bodě (metoda `ChangeTerrainHeights`) a doplňovat do mapy další herní objekty, příšery a polohu kouzelníků. Designerovu hotovou mapu ukládá metoda `SaveEditorMap` jako instanci třídy `MapHeights`. Ta obsahuje pole výšek všech dílů mapy a její rozměry, popis definovaných výškových pater terénu, dále informace o rozmístěné amfor a kuliček many a konečně také polohu hráče a eventuálně i nepřátelského kouzelníka. Při ukládání mapy se vytváří vyskakovací okno `SaveMapPopUp`.

Příšeru umístěnou v mapě reprezentuje skript `EditorMonster`. Ten umožňuje poznamenávat k příšeře body cesty (objekty se skriptem `PathPoint`), po které by měla procházet, a na požádání je zobrazovat a skrývat. Nový průchozí bod trasy přidává metoda `AddPathPoint` volaná z metody `OnSceneGUI` editorového skriptu `MonsterEditor` po kliknutí na příslušné místo v mapě.

6. Uživatelská dokumentace

Tato kapitola představuje uživatelskou dokumentaci ke hře Master of the Carpet. Provádí hráče jejím spuštěním, popisuje hru a její ovládání. Soubory hry jsou uloženy na přiloženém DVD.

6.1 Spuštění

Hru není třeba nijak instalovat a je možné ji rovnou spustit ze souboru `MasterOfTheCarpet` v adresáři `MasterOfTheCarpet-Game`. Pro běh hry je nutné, aby tento adresář obsahoval také všechny ostatní stávající soubory. Verze na DVD je určena pro 64-bitový systém Windows verze 7 SP1 a novější. Vyžaduje procesor s podporou instrukční sady SSE2, což jsou všechny novější procesory Intel i AMD, a grafickou kartu s hardwarovou podporou DirectX10.

6.2 Menu

Po spuštění hry se objeví úvodní obrazovka, kterou můžeme vidět na obrázku 6.1. Stisk libovolné klávesy přenese hráče na obrazovka s výběrem herní mapy, jejíž vzhled ilustruje obrázek 6.2. Ze seznamu si uživatel může zvolit, jakou herní mapu chce vyzkoušet. Předpřipravených je šest základních herních úrovní. Pro návrat ze hry zpět do menu slouží klávesa *Escape*. Po opuštění rozehrané hry se již do předchozího stavu nelze vrátit. Celou hru je možné ukončit stiskem tlačítka *Exit* v obrazovce výběru mapy.

Obrázek 6.1: Úvodní obrazovka.

Obrázek 6.2: Výběr mapy.

6.3 Popis hry

Ve hře hráč představuje kouzelníka na létajícím koberci. Jeho úkolem je nashromáždit ve svém hradě dostatečné množství many a tím obnovit rovnováhu magického světa.

Mana se ve světě nachází v podobě kuliček. Ty hráč kouzlem získává a horkovzdušný balón je pak odnáší do jeho hradu. Další manu v sobě ukrývají příšery, které svět ohrožují. Pro její získání kouzelník nejprve musí svými kouzly monstrum zabít.

K výhře nestačí, aby kouzelník potřebnou manu získal, ale je také třeba, aby byla uložena v jeho hradu. Kouzelník jej buduje pomocí kouzla. Opětovným použitím kouzla může hrad postupně vylepšovat až na nejvyšší, čtvrtou, úroveň.

Společné s hradem vzniká také balón, který do něj přináší hráčem získanou manu. Balón může najednou přenášet jen omezené množství many. Stejně tak kapacita hradu je omezena a je třeba ji navyšovat jeho vylepšováním.

V magickém světě jsou v amforách ukryta kouzla, která kouzelník může získávat. Poloha amfor je vyznačena v mapě. Pro zisk kouzla stačí, aby kouzelník amforu sebral. Každé kouzlo vyžaduje, aby kouzelník měl k dispozici dostatek many. Potřebná mana se u různých kouzel liší, stejně jako minimální interval mezi jejich opětovným použitím.

Na kouzelníka číhá nebezpečí v podobě různých druhů příšer. Ty se mohou pohybovat po souši, létat vzduchem, nebo plavat na vodní hladině. Některé druhy neútočí jen na hráče, ale mohou napadnout také jeho hrad nebo balón.

Zraněný kouzelník se sám od sebe pomalu léčí, naopak poničený hrad nebo balón už opravit nelze. Po zničení hradu nebo balónu se v něm uschovaná mana rozsype do okolí. Zatímco rozbitý balón se sám v hradu obnoví, nový hrad si musí kouzelník znovu vyčarovat. Pokud kouzelník přijde o veškerý život, ale jeho hrad stojí, oživí se v něm. V opačném případě umírá definitivně.

V některých herních světech může hráč narazit také na nepřátelského kouzelníka. Ten s hráčem soupeří a snaží se získat manu pro sebe. Také používá kouzla a staví hrad, ve kterém svou manu shromažďuje, bojuje s okolními příšerami a může napadnout i hráče.

6.4 Herní obrazovka a ovládání

Hráč se pohybuje terénem pomocí šipek a pohybem myši otáčí kameru. V jednom okamžiku má k dispozici až dvě kouzla, každé v jedné ruce kouzelníka. Příslušné kouzlo se provede po kliknutí pravým, respektive levým tlačítkem myši. Pokud má kouzlo vnější podobu, vytváří se vždy uprostřed obrazovky před hráčem. Shrnutí ovládání poskytuje tabulka 6.1.

šipka nahoru/dolů	zrychlení pohybu směrem vpřed/vzad
šipka doleva/doprava	úkrok doleva/doprava
pohyb myší	otočení kamery
levé/pravé tlačítko myši	vyčarování kouzla v levé/pravé ruce kouzelníka
mezerník	zobrazení a opuštění přehledové obrazovky
Escape	návrat do menu

Tabulka 6.1: Shrnutí ovládání hry.

Základní herní obrazovku představuje obrázek 6.3.

V levém horním rohu obrazovky se nachází malá mapa hráčova nejbližšího okolí. V jejím středu je poloha hráče a dále jsou v ní různobarevně vyznačeny další objekty. Jde o výřez mapy celého světa, popsané v sekci 6.5. Směrování mapky se mění současně s otáčením hráče.

O stavu hráče informuje přehledová lišta v horní části obrazovky. Zde je možné sledovat jeho aktuální život a posbíranou manu. Když je ukazatel many naplněn,

Obrázek 6.3: Herní obrazovka.

hráč jí získal dostatek pro obnovení rovnováhy kouzelného světa. Lišta také zobrazuje stav hráčova hradu a balónu, pokud je hrad postaven.

V pravé části přehledové lišty jsou vyznačena kouzla, která má hráčův hrdina v pravé a levé ruce. Pokud je ikona kouzla potemnělá, kouzlo ještě není připraveno k použití. To může být způsobeno nedostatkem many, nebo nutným časovým intervalem mezi opětovným použitím téhož kouzla. V druhém případě se ikona postupně vrací do původní nezastíněné podoby.

6.5 Přehledová obrazovka

Zobrazení přehledové obrazovky se zapíná a vypíná mezerníkem. Vzhled obrazovky ukazuje obrázek 6.4.

Obrázek 6.4: Přehledová obrazovka.

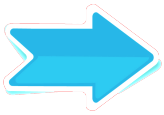
V jeho levé části se nachází mapa celého herního světa. V ní jsou různými barvami vyznačeny volné kuličky many (žlutá barva), amfory s kouzly (oranžová

barva) a příšery (černá barva). Dále zobrazuje polohu hráče, eventuálně i nepřátelského kouzelníka a jejich hrady a balóny. Hráče reprezentuje fialová barva a nepřítele červená. Pokud kouzelník získá kuličku many, i ona je v mapě vyznačena jeho barvou.

V pravé horní části může hráč stále sledovat své okolí. V pravé dolní části je pak přehled kouzel, která kouzelník může získat. V základní verzi je těchto kouzel sedm. Jejich detailnější popis nabízí část 6.6. Potemnělá jsou kouzla, která hráč zatím nenašel. Ta, která již má k dispozici, jsou zvýrazněna. Kouzla hráč volí kliknutím na ně. Podle toho, jestli použije levé, nebo pravé tlačítko myši, se kouzlo přiřadí buď do levé, nebo do pravé ruky kouzelníka.

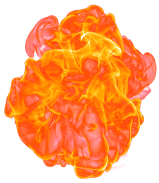
6.6 Seznam kouzel

Jednotlivá kouzla představujeme ve stejném pořadí, v jakém jsou seřazena na přehledové obrazovce.



Zrychlení (*Accelerate*) na omezenou dobu dovoluje kouzelníkovi pohybovat se vyšší rychlostí vpřed.

Hrad (*Castle*) v místě dopadu vytvoří kouzelníkův hrad, pokud ještě žádný nemá. Pokud kouzelník svůj hrad již vybudoval, může ho pomocí kouzla vylepšovat na vyšší úrovně. Nároky na manu pro provedení kouzla se s úrovní hradu postupně zvyšují.



Ohnivá střela (*Fireball*) je útočné kouzlo, které po zásahu zraňuje příšeru nebo nepřátelského kouzelníka.

Léčení (*Heal*) navrátí zraněnému kouzelníkovi část jeho ztraceného života.

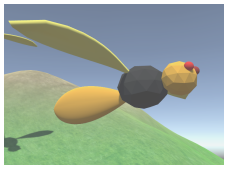
Obarvení many (*Possess*) slouží k zisku kuliček many. Stačí, aby se jí kouzlo dotklo, a kulička se přebarví na kouzelníkovu barvu a stává se jeho.

Štít (*Shield*) na omezenou dobu tlumí veškeré útoky na kouzelníka.

Sopka (*Volcano*) v místě dopadu kouzla vytváří sopku. Ta chrlí žhavou lávu a ohrožuje jí všechny živé bytosti kolem.

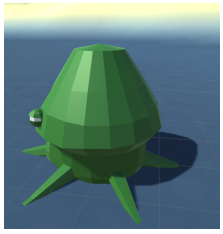
6.7 Seznam příšer

V základní verzi se hráč může setkat s následujícími druhy příšer.

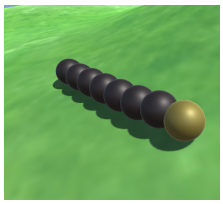


Vosy útočí nablízko, létají vzduchem a napadají nejen kouzelníka, ale také jeho hrad nebo balón.

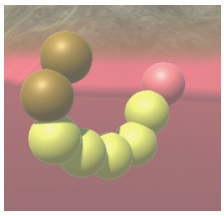
Krabi pobíhají po pevnině a pojídají kuličky many ve svém okolí. S rostoucím ziskem many se vyvíjejí a sílí jejich ohnivý útok. Na nejvyšším evolučním stupni pak kladou vajíčka, z nichž se líhnou nové generace.



Krakeni se pohybují po mořské hladině. Kromě běžného útočného kouzla mají k dispozici také poutací kouzlo. Po zásahu je kouzelník po určitou dobu nucen zůstat v okolí krakena a stává se tak pro něj snadnou obětí.



Červi se pomalu plazí po zemi. Kouzelníka letícího kolem mohou napadnout svým ohnivým útokem. Stejně tak jeho hrad, pokud jim stojí v cestě. Jsou odolní vůči útokům a odměnou za jejich zničení je velké množství many.



Létající červi jsou vzdušnou obdobou pozemních. Navíc napadají i kolem letící balóny s manou.

7. Dokumentace herního designera

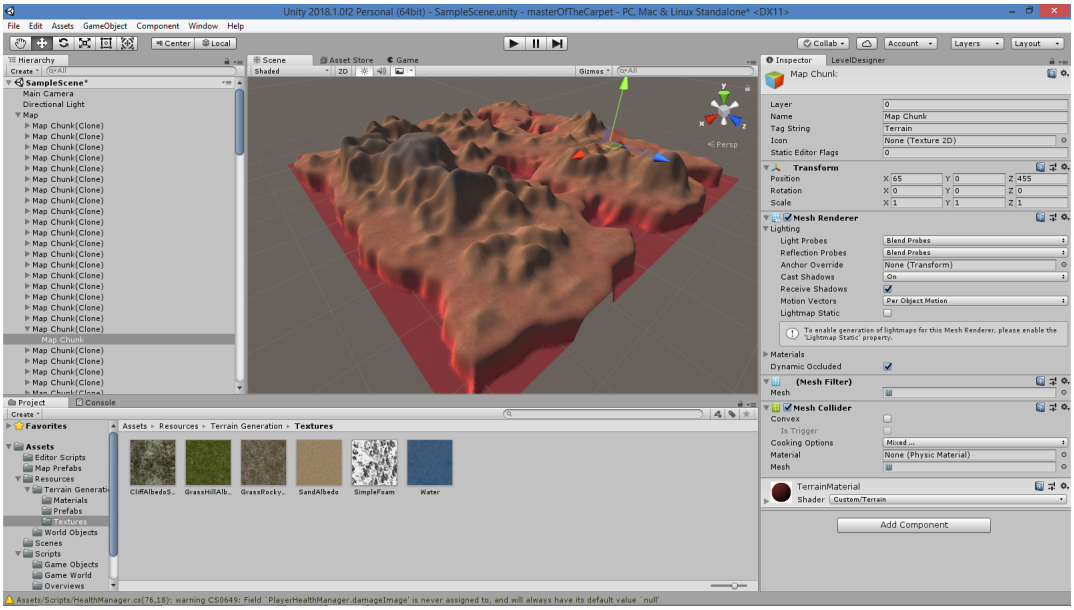
Návrh jednotlivých herních úrovní hry Master of the Carpet se odehrává ve standardním prostředí Unity editoru rozšířeném o speciální okno editoru nazvané *Level Designer*.

Pro návrh je nutné mít nainstalováno prostředí Unity. Doporučena je verze 2018.1 nebo novější, zpětně kompatibilní. Připravená scéna editoru *EditorScene* se nachází v adresáři `masterOfTheCarpet/Assets/Scenes` na přiloženém DVD. Po kliknutí na soubor se scéna otevře přímo v prostředí Unity. Speciální okno Level Designeru lze najít v záložce *Windows* →*Level Designer*. Jeho polohu stejně jako polohu všech ostatních oken si může uživatel zvolit.

7.1 Editor Unity

Vývojové prostředí Unity lze zdarma stáhnout z oficiálních stránek [7], kde je také k dispozici návod k jeho instalaci [8].

Pro návrh herní úrovně je potřeba pouze základní znalost prostředí Unity. Proto zde představíme jednotlivé součásti Unity editoru pouze zběžně. Detailní informace je možné najít na stránkách manuálu editoru [9]. Obrázek 7.1 ukazuje základní vzhled a jednotlivá okna editoru.



Obrázek 7.1: Náhled editoru Unity.

Okno projektu (*Project Window*) nabízí pohled na adresářovou strukturu se všemi soubory, které se v projektu hry využívají. Kromě skriptů jsou zde například modely objektů, textury nebo materiály.

Hierarchie scény (*Hierarchy Window*) zobrazuje všechny herní objekty přítomné na scéně. Ty mají stromovou strukturu a kliknutím na rodiče je možno zobrazovat a skrývat potomky.

V okně inspektoru (*Inspector Window*) se zobrazují detailní informace o zvoleném herním objektu scény nebo souboru vybraném v okně projektu. Ukazuje se zde přehled komponent objektu s jejich vlastnostmi, které je možné v inspektoru upravovat.

Déle se zastavíme u scény (*Scene View*), neboť s ním by měl designer aktivně pracovat. Toto okno představuje interaktivní náhled na herní svět. K práci s náhledem slouží nástroje v levém horním rohu obrazovky, které ukazuje obrázek 7.2.

Obrázek 7.2: Nástroje pro manipulaci se scénou.

Pomocí nástroje ruky se stisknutým levým tlačítkem myši je možno se pohybovat v náhledu. Pro výběr objektů scény slouží tlačítko posuvníku. Zvolený objekt je barevně ohraničen a vyznačen v hierarchii scény. Ostatní nástroje lišty se při tvorbě herní úrovně nevyužijí. K otočení pohledu scény je třeba držet pravé tlačítko myši, nebo použít ikonu v pravém horním rohu scény, pomocí které lze zvolit natočení kamery a přepínat mezi perspektivní a ortogonální projekcí.

7.2 Návrh herní úrovně

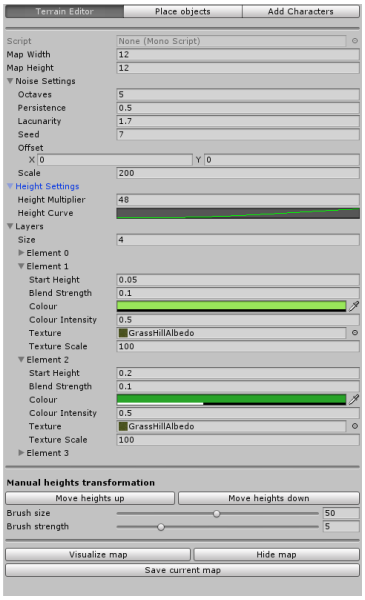
Celý návrh úrovně probíhá s využitím vlastního okna *Level Designeru*. Nejprve je třeba nastavit parametry zobrazovaného terénu v sekci *Terrain Editor*. Tlačítkem *Visualize map* pak lze příslušnou herní mapu zobrazit. V panelu *Place objects* lze doplnit rozmístění many a kouzel. V poslední záložce *Add characters* se pak přidávají příšery a nepřátelský kouzelník.

Když je návrhář s podobou herního světa spokojen, může ho uložit kliknutím na tlačítko *Save current map*. Při ukládání doplní jméno mapy, které se pak bude zobrazovat ve hře při výběru herní úrovně. Po uložení vytvořený terén zčerná a je možné ho skrýt tlačítkem *Hide map*. Všechny vytvořené herní mapy jsou uloženy ve složce `Resources/Map Prefabs` v okně projektu. Zde je možné ověřit správné uložení vytvořené mapy, eventuálně nadále nepotřebné mapy smazat.

Následně je hru třeba nechat sestavit. K tomu vede následující postup. V horní liště zvolíme *File→Build settings...*. V nově otevřeném okně můžeme zachovat výchozí nastavení a vybrat tlačítko *Build*. Poté je třeba zvolit složku, do které chceme soubory hry uložit. Hotový spustitelný soubor ve složce pak ponese její jméno. Při spuštění vytvořené hry se v sekci výběru herní úrovně objeví i nově připravená mapa.

7.2.1 Návrh terénu

Pro tvorbu terénu je v okně *Level Designeru* určena záložka *Terrain Editor*. Její podobu můžeme vidět na obrázku 7.3. Zde návrhář může zvolit velikost herní mapy, nastavení parametrů šumu, z nějž je terén procedurálně generován, a definovat jednotlivé vrstvy terénu. Mapu vytvořenou dle zadaných parametrů je možné zobrazit zmáčknutím tlačítka *Visualize map* a skrýt tlačítkem *Hide map*.



Obrázek 7.3: Terrain Editor

Rozměry herního světa

V polích *MapWidth* a *MapHeight* návrhář může zvolit rozměry herní mapy. Jednotkami jsou počty dílků. Velikost samotných dílků návrhář nastavovat nemůže, neboť ta těsně souvisí s tím, jaké pro ně lze vytvářet úrovně detailů, a při nekompatibilní velikosti dílku by nebylo možné vytvářet jejich meshe s požadovaným zjednodušením. Rozměry mapy se měly pohybovat v rozmezí od 5 do 20 dílů.

Parametry šumu

V záložce *Noise Settings* lze zvolit parametry šumu, ze kterého terén vzniká. Počet oktáv ovlivňuje, jak detailně je vzniklá krajina členěná. První oktáva tvoří základní horizont terénu, další do něj přidává drobnější detaily a tak dále. S tím souvisí i parametry persistence a lakunarita. Persistence udává, jak moc výrazné budou detaily v následující oktávě. Lakunarita určuje, jak se zvětšuje frekvence těchto detailů. Základní nastavení používá pět oktáv s hodnotou persistence 0.5 a lakunaritou rovnou 2. Srovnání terénů s různým počtem oktáv vidíme na obrázku 7.4.

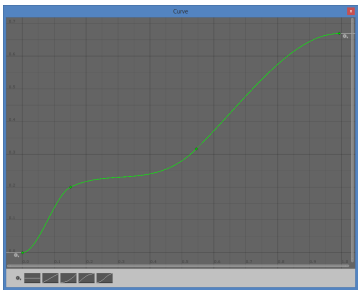
(a) (b)

Obrázek 7.4: Terén s různými počty oktáv.

Hodnotami v *Offsetu* se dá celý herní svět posouvat v základních směrech. *Seed* představuje náhodné semínko pro použitou šumovou funkci. Změnou jeho hodnoty lze získat zcela novou mapu. Scale udává velikost přiblížení šumové funkce.

Nastavení výšek

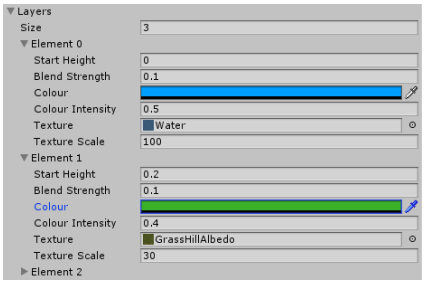
Nastavení výšek lze najít v záložce *Height Settings*. Zde se mění pouze dva parametry. Nejprve je možné transformovat původní relativní výšky pomocí křivky výšek (*Height Curve*). Její podobu vidíme na obrázku 7.5. Původní výška se použije jako x-ová souřadnice a transformuje se na příslušnou y-ovou hodnotu. Lze docílit například toho, že všechny výšky odpovídající vodě se zobrazí na stejnou hodnotu, a tím vznikne rovná vodní hladina, nebo můžeme vytvořit krajinu bez kopců, když vyšší hodnoty transformujeme na nižší. Spojitost funkce zachová spojitý vzhled terénu i po transformaci. *Height Multiplier* představuje koeficient, kterým se přenásobují vygenerované relativní výšky mezi 0 a 1. Ve výchozím nastavení je jeho hodnota zvolena jako 40.



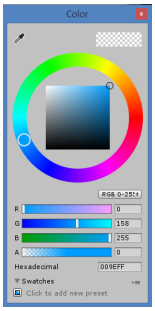
Obrázek 7.5: Křivka pro transformaci výšek.

Vrstvy terénu

V záložce *Layers* je možné vytvořit libovolný počet typů terénu. Jednotlivé typy terénu pak tvoří výšková patra krajiny. Druhy terénu mají celou řadu parametrů, jejichž vzhled v editoru vidíme na obrázku 7.6. *Start Height* určuje relativní výšku (mezi 0 a 1), na které terén začíná, *Blend Strength* udává, jak široký výškový pás kolem počáteční výšky terén zabírá. Barvu terénu volíme v parametru *Colour*. Lze ji vybrat pomocí kapátka nebo po kliknutí pomocí barevné palety, jejíž podobu vidíme na obrázku 7.7.



Obrázek 7.6: Parametry vrstev terénu.



Obrázek 7.7: Barevná paleta.

Každý terén také musí mít zvolenou texturu a její měřítko. Texturu lze najít mezi soubory v okně projektu a přenést do příslušného pole. Základní textury pro terén jsou připraveny ve složce `Resources/Terrain Generation/Textures`. Sem lze také přidat novou vlastní texturu. Výsledná podoba terénu je kombinací zvolené barvy a textury. Poměr zvolené barvy ve výsledku udává *Colour intensity*.

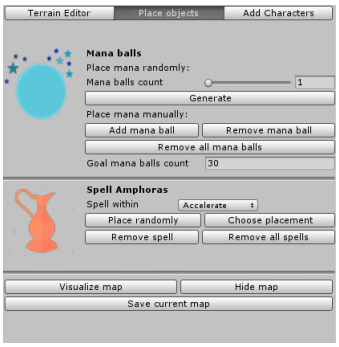
Manuální změny výšek

Na již vytvořené mapě lze ještě lokálně snížit nebo zvýšit terén zapnutím tlačítek *Move heights up* nebo *Move heights down*. Dále stačí v hierarchii zvolit objekt *Map* a klikat do míst, jejichž výšku chceme upravit. Poloměr kruhu, v němž se výšky mění, udává *Brush size* a to, jak velká změna výšky je, pak *Brush strength*. Při modifikaci výšek je nutné mít nastavenu perspektivní kameru.

7.2.2 Umístění herních objektů

V rámci návrhu úrovně designer rozmisťuje do herního světa kuličky many a amfory s kouzly. Potřebné nástroje se nachází v záložce *Place Objects*, jejíž podobu vidíme na obrázku 7.8.

Při ručním přidávání objektů je potřeba mít nastavenu perspektivní kameru a v hierarchii mít zvolen objekt *Map*.



Obrázek 7.8: Editor pro umísťování objektů

Rozmístění many

Rozmístit kuličky do mapy náhodně lze hromadně. Stačí určit jejich počet (*Mana balls count*) a zvolit tlačítko *Generate*. Alternativou je volit polohu každé kuličky zvlášť. K tomu stačí zmáčknout tlačítko *Add mana ball* a klikat na cílová místa na mapě. Takto lze doplnit další manu k předchozí náhodně vygenerované. Ke zrušení zvolené kuličky na scéně slouží tlačítko *Remove mana ball*. Všechny umístěné kuličky many lze najednou odstranit pomocí tlačítka *Remove all mana balls*. V poli *Goal mana balls count* návrhář volí, kolik many musí hráč posbírat, aby vyhrál.

Rozmístění kouzel

Dále by měly být v herní mapě umístěny amfory s kouzly, která hráč po objevení amfory dostává k dispozici. Umístit amforu lze ručně po volbě tlačítka

Choose placement nebo její polohu ponechat náhodě s *Place randomly*. Kouzlo skryté v umísťované amfoře určuje pole *Spell within*, kde je možné vybrat z přehledu všech implementovaných kouzel. Odstranit rozmístěné amfory lze opět buď jednotlivě jejich vybráním ve scéně a zmáčknutím tlačítka *Remove spell*, nebo všechny najednou s *Remove all spells*.

7.2.3 Přidávání postav

Poslední součástí herního světa jsou příšery a eventuálně také nepřátelský kouzelník. Pro jejich doplnění a možnost výběru výchozí pozice hráče je určena záložka *Add characters*, kterou můžeme vidět na obrázku 7.9.

Obrázek 7.9: Panel pro přidávání postav

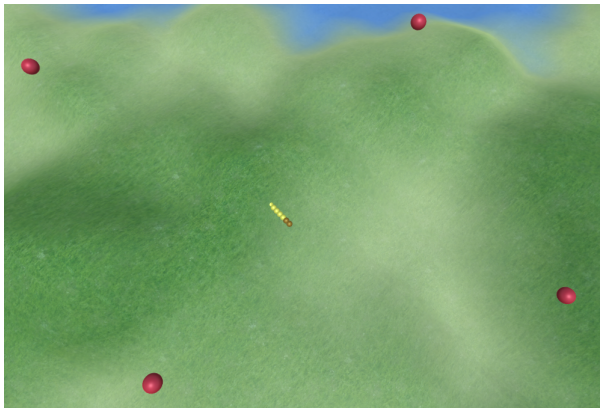
V levé části se zobrazuje seznam všech implementovaných příšer. Z nich si designer vybere a klikáním na mapě umisťuje zvolenou příšeru. Po vybrání ve scéně je možné s jednotlivými příšerami dále pracovat a určit trasu, po které se mají pohybovat.

Dráha příšery je v editoru vyznačena červenými body, jak ukazuje obrázek 7.10. Trasu zvolené příšery lze zobrazovat a skrývat použitím tlačítek *Show path* a *Hide path*. Nové body prodlužující dosavadní trasu se přidávají tlačítkem *Add path point*. Naopak k odstranění vybraného bodu dráhy slouží tlačítko *Remove path point*.

Při určování trasy je třeba volit body tak, aby mezi sousedními mohla příšera procházet pokud možno přímo. Pokud příšera žádnou trasu stanovenu nemá, bude se ve hře pohybovat náhodně.

Nepřátelského kouzelníka je možné do herní mapy umístit stejně jako příšery. Na rozdíl od nich se však v herním světě může nacházet nejvýše jednou, takže zvolením nové pozice starou opouští. Odstranit vytvořeného kouzelníka lze použitím tlačítka *Remove enemy wizard*. Nepřátelský kouzelník se pohybuje dle svého uvážení, takže pro něj se žádná trasa nepřipravuje.

Posledním krokem je volba počáteční pozice hráče. Pokud není nastavena, hráč začíná ve středu levého dolního dílku mapy. Jinou výchozí pozici je možno



Obrázek 7.10: Body vyznačující dráhu pohybu příšery.

vybrat po stisku tlačítka *Set player position*, pro obnovení výchozího postavení slouží tlačítko *Reset*.

7.3 Vlastní rozšíření

Základní nabídku kouzel a příšer může návrhář dále rozšiřovat. Na vytvoření nového kouzla nebo příšery může pracovat přímo v projektu hry v prostředí editoru Unity. Pokud jsou součástí rozšíření i nové skripty, může pro jejich tvorbu využít libovolné standardní vývojové prostředí pro jazyk C# jako je MS Visual Studio nebo MonoDevelop.

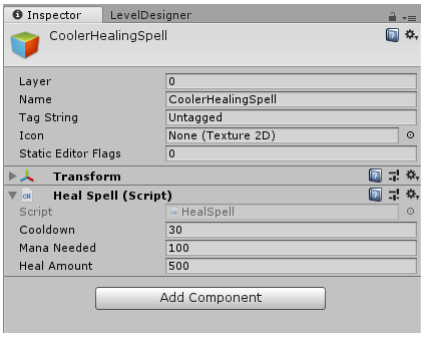
7.3.1 Přidání nového kouzla

Nové kouzlo lze do hry přidat následujícím způsobem.

1. Nejprve vytvoříme nový prázdný herní objekt v záložce horní lišty *GameObject* →*Create Empty*. Přehled jeho komponent si zobrazíme v okně inspektoru. Do položky *Name* napíšeme název nového kouzla. Dle zavedené konvence by měl název končit slovem *Spell*, například *HealSpell*, *LightningSpell* a podobně.

2. Nyní je třeba přidat jako komponentu skript definující chování kouzla. Zde můžeme využít nějaký již existující skript a ten dále upravovat. Lze například vytvořit lepší léčivé kouzlo, které také bude vyžadovat více many. Předpřipravené skripty kouzel je možné nalézt ve složce `Scripts/Spells` v okně projektu. Alternativou je vytvořit vlastní skript s novým chováním. Detaily k tomuto postupu jsou uvedeny v části 7.3.2.

3. Po přidání skriptu můžeme v inspektoru nastavovat vlastnosti kouzla. Vždy volíme minimálně, jaké množství many je pro použití kouzla potřeba (*Mana Needed*), a kolik vteřin musí uplynout mezi dvěma použitími kouzla (*Cooldown*). Je třeba, aby stanovená doba byla nenulová. Další nastavované vlastnosti se již odvíjí od povahy kouzla. Ukázku hotového přehledu nového léčivého kouzla poskytuje obrázek 7.11.



Obrázek 7.11: Přehled komponent nového kouzla.

4. Nově připravené kouzlo pak můžeme přenést z panelu hierarchie do složky `Resources/Spells` v okně projektu a z hierarchie herní scény smazat.

5. Nakonec je třeba kouzlo přidat do přehledu kouzel herní scény a scény editoru. Příslušné scény pojmenované *EditorScene* a *GameScene* najdeme ve složce `Scenes`. V okně hierarchie scény zvolíme objekt *SpellsManager*. V jeho komponentě *Spell Overview* vybereme pole *Spells*. Zde je seznam všech kouzel, která jsou v editoru nebo hře k dispozici.

6. Do seznamu přidáme nový prvek zvětšením jeho velikosti (parametr *Size*) o jedna a stiskem klávesy *Enter*. Do nového prvku se automaticky překopírovává obsah předchozího. Ten je třeba upravit. Do položky *Spell* z okna projektu přeneseme naše nově vytvořené kouzlo. Doplníme jeho název (*Name*), pod kterým se bude zobrazovat v editoru.

7. Dále je možné přidat obrázek (texturu), který bude kouzlo znázorňovat v přehledu kouzel ve hře. Aby textura zapadala do koncepce přehledu, měla by mít průhledné pozadí. Textury použité pro ostatní kouzla jsou uloženy ve složce `Resources/Overview/Spells/SpellImages`. Sem je také vhodné texturu nového kouzla nahrát a pak přenést do položky *Image*.

8. Stejné změny musíme udělat v objektu *SpellManager* ve scéně editoru i hry. Pro správné fungování je nutné, aby kouzla v přehledu byla vždy ve stejném pořadí. Nakonec je třeba novou podobu obou scén uložit.

7.3.2 Skript nového kouzla

Skripty kouzel se nacházejí ve složce `Scripts/Spells`. Právě sem je tedy vhodné přidat nový vlastní skript. Vytvořit nový skript můžeme přímo v editoru Unity kliknutím pravým tlačítkem myši uvnitř složky v okně projektu a volbou *Create→C# Script*. Soubor by se měl po otevření automaticky spustit v nastaveném vývojovém prostředí jazyka C#.

Nové kouzlo by mělo být potomkem třídy `Spell` nebo některého jejího potomka. Detaily tříd kouzel popisuje vývojová dokumentace v části 5.4. Nová třída by měla rozšiřovat poděděnou metodu `void Cast(Wizard wizard)`, která se volá při použití kouzla a jako argument předává čarujícího kouzelníka. Může také předefinovávat původní virtuální metodu `bool IsReady(Wizard wizard)`, pokud pro použití kouzla existuje další podmínka krom dostatku many a zvoleného

časového odstupu od posledního použití kouzla. Zbytek záleží zcela na programátorovi.

7.3.3 Přidání nové příšery

Při tvorbě nové příšery můžeme buď upravovat nějakou již existující, nebo začít s jejím návrhem zcela od začátku. Podle toho buď jednotlivé komponenty pouze upravujeme, nebo ve druhém případě postupně přidáváme. Proces přípravy nové příšery můžeme rozdělit na následující kroky.

1. Pokud chceme pracovat s již existující příšerou, přeneseme ji z adresáře `Resources/Monsters` v okně projektu do hierarchie scény. V případě, že vyvářímе monstrum zcela nové, můžeme do adresáře nejprve importovat vlastní model, který chceme použít, (k jeho přípravě lze použít například Blender). Poslední možností je místo modelu použít nějaké primitivní těleso předpřipravené v Unity (záložka *GameObject→3D Object*). Dvojitým kliknutím na náš objekt v hierarchii se na něj zaměří kamera scény a v panelu inspektoru se zobrazí jeho vlastnosti.

2. Nejprve zvolíme jméno naší nové příšery (položka *Name*). Automaticky vytvořené komponenty *Transform*, *Mesh Filter* a *Mesh Renderer* nemusíme nijak upravovat.

3. Jako první nastavíme komponentu *Capsule Collider*. Měníme výšku (*Height*) a poloměr (*Radius*) kapsle tak, aby co nejlépe obklopovala model příšery.

4. Dalším nastavovanou komponentou je *Rigidbody*. Její výchozí hodnoty nemusíme nijak měnit. Je třeba pouze přepnout tlačítko používání gravitace (*Use Gravity*) podle toho, zda se má monstrum pohybovat po zemi, nebo létat nad ní.

5. Dále přidáváme nebo přenastavíme komponentu skriptu *Health Manager*. Zde se nastavuje život příšery v položce *Full Health*. Dále změnou *Healing Rate* na nenulovou hodnotu umožňujeme, aby se zraněná příšera postupně sama léčila. Její zdraví se každou vteřinu zvýší o zvolenou hodnotu.

6. Dalším potřebným skriptem zajišťujícím pohyb příšery je *Movement Controller*. Zde nejprve v položce *Type* volíme, zda půjde o příšeru suchozemskou (*Ground*), vodní (*Water*) nebo létající vzduchem (*Air*). Pokud jsme zvolili poslední variantu, je pro nás relevantní také položka *Height Above Ground* (zde volíme, jak vysoko nad zemí se má příšera pohybovat) a *Height Oscillation* (nenulová hodnota znamená, že příšera bude ve vzduchu létat po sinusové dráze se zvolenou amplitudou). Nakonec ještě nastavujeme maximální rychlost (*Max Speed*).

7. Posledním krokem je přidání skriptu popisujícího chování příšery. Ten vybereme v okně projektu ze složky `Scripts/Monsters`, kam můžeme také doplňovat vlastní skripty příšer (tvorbu těchto skriptů popisuje část 7.3.3). V inspektoru pak nastavíme jednotlivé vlastnosti definované skriptem. Vždy jde minimálně o počet kuliček many, které se objeví po zabití příšery, (*Mana*

Balls Count) a rychlost, jakou příšera útočí (*Attack Cooldown*). Dále vybíráme vzdálenosti, ve které příšera rozezná kouzelníka (*Within Sight Dist*), odkud na něj může útočit (*Attack Dist*), a jak daleko od ní musí kouzelník být, aby pronásledování vzdala (*Release Dist*). Pro monstra útočící zblízka je specifikována také síla útoku (*Attack Damage*).

8. Připravenou příšeru můžeme následně přenést z panelu hierarchie do složky `Resources/Monsters` v okně projektu a z herní scény smazat.

9. Nakonec je třeba monstrum přidat do přehledu příšer herní scény a scény editoru. Příslušné scény najdeme ve složce `Scenes`, jde o *EditorScene* a *GameScene*. V obou dvou v okně hierarchie scény zvolíme objekt *MonsterManager* a v jeho komponentě *Monster Overview* vybereme pole *Monsters*. Zde je seznam všech příšer, která jsou v editoru nebo hře k dispozici.

10. Do seznamu přidáme nový prvek zvětšením jeho velikosti (parametr *Size*) o jedna. Po rozbalení nového prvku na konci seznamu do položky *Monster Prefab* z okna projektu přenese naši nově vytvořenou příšeru. Doplníme její název (*Name*), pod kterým se bude zobrazovat v editoru.

11. Dále je možné přidat obrázek (texturu), která se zobrazuje v editoru při přidávání příšer do vytvářené herní mapy. Textury použité pro ostatní příšery jsou uloženy ve složce `Resources/Monsters/Images`. Sem je také vhodné texturu pro novou příšeru nahrát a pak přenést do položky *Image*.

12. Stejné změny musíme udělat v objekt *MonsterManager* ve scéně editoru i hry. Pro správné fungování je nutné, aby příšery v obou přehledech byly v tomtéž pořadí. Nakonec je třeba novou podobu obou scén uložit.

7.3.4 Skript nové příšery

Skripty chování příšer jsou umístěny ve složce `Scripts/Monsters`. Sem můžeme doplnit i vlastní nově definované chování příšery. Vytvořit nový skript můžeme přímo v editoru Unity kliknutím pravým tlačítkem myši uvnitř složky v okně projektu a volbou *Create→C# Script*. Soubor by se měl po otevření automaticky spustit v nastaveném vývojovém prostředí jazyka C#.

Nová třída příšery by měla být potomkem třídy `Monster` nebo některého z jejích potomků, na jehož chování se bude co nejlépe navazovat. Struktura chování příšery má podobu stavového automatu. V nejzákladnější podobě se střídají pouhé tři stavy – hlídkování, pronásledování kouzelníka a útok na něj. Detailní popis stavů a pravidla přechodů mezi nimi lze najít ve vývojové dokumentaci v podkapitole 5.5.

Programátor může předefinovat původní chování příšery v daném stavu, nebo vytvářet stavy vlastní. Stavy i přechody mezi nimi jsou definovány ve virtuálních metodách typu `void`. Dle zavedené koncepce názvy metod definujících stavy končí slovem `State` a metody určující přechody mezi nimi slovem `StateTransitions`.

Jako ukázku uvádíme část skriptu chování ve třídě `Crab`, která rozšiřuje chování obyčejné příšery o stav sbírání kuliček many. Pro přechod do nového stavu pak přepisuje základní přechody ze stavu hlídkování (`PatrolState`).

```
public class Crab : Monster {
 protected virtual void CollectManaState() {
       //  insert code for actions to perform in the state
        CollectManaStateTransitions();
    }
 protected virtual void CollectManaStateTransitions() {
       // insert code describing transitions to states
       // already defined in the Monster class
    }

 // change old state transitions to be able enter the new state
 protected override void PatrolStateTransitions() {
         // transition to the new state
        if (SeesManaball(out goalManaball)) {
            currentState = new MonsterStateExecute(CollectManaState);
        }
       // transitions defined in the Monster class
        base.PatrolStateTransitions();
    }
}
```

Závěr

V následující části projdeme jednotlivé body zadání vytyčené v části 1.3 a pokusíme se zhodnotit jejich naplnění.

1. Herní svět, který je konečný, ale bez fyzické hranice, jsme se rozhodli vytvořit v rovině s použitím více dílků mapy, které se neustále přeskupují do okolí hráče. Zvolené řešení se ukázalo jako zcela funkční při pohledu hráče do okolí i z hlediska přesunů ostatních objektů. Procedurálně generovaný terén pak působí vcelku přirozeně.

2. Hra bez problémů využívá základní mechanismus budování hradů a ukládání posbírané many prostřednictvím balónů.

3. Hráč má k dispozici všechna požadovaná kouzla s odpovídajícími efekty. Na kouzlu sopky jsme úspěšně ověřili možnosti změny terénu v průběhu hry, i když v současné podobě není jeho použití ve hře z hlediska hráče příliš smysluplné. Předem definovanou sadu kouzel jsme doplnili o další kouzla pro léčení, urychlení pohybu a obranu.

4. Vytvořili jsme všech pět zadaných typů příšer. Jejich chování je v souladu s minimálními požadavky spíše jednodušší a implementovali jsme jej pomocí stavového automatu. Pro pohyb směrem k cíli se současným vyhýbáním se překážkám a omezením na zvolený terén jsme použili vážené skládání sil.

5. Nejsložitější byla tvorba nepřátelského kouzelníka. V naší verzi sice dělá všechny základní činnosti, ale jeho chování působí hodně roboticky. Není také schopen dostatečně flexibilně reagovat na měnící se situaci v okolí. Na druhou stranu problematika tvorby dostatečně pokročilé umělé inteligence, která by věrně napodobovala chování skutečného hráče, je velice složitá a nad rámec této práce.

6. Hráč má k dispozici přehledovou obrazovku s výběrem kouzel a mapou světa, jejíž vzhled odpovídá naší původní představě. Stejně tak zcela vyhovuje stavová lišta a malá mapa v herní obrazovce.

7. Posledním požadavkem bylo vytvoření editoru, v němž by bylo možné navrhovat jednotlivé úrovně hry. Ten je implementován jako rozšíření standardní podoby editoru Unity. Z toho vyplývá nutnost, aby designer měl k dispozici prostředí Unity a byl schopen základní navigace v něm. Převažující výhodou je ovšem příjemná práce v tomto prostředí. Naše rozšíření pak umožňuje snadno absolvovat všechny kroky návrhu herní mapy, od tvorby terénu, přes rozmisťování kuliček many a amfor s kouzly až po přidávání nepřátel.

Možná rozšíření

Přestože současná implementace hry Master of the Carpet naplňuje námi stanovené požadavky, ponechává mnoho prostoru pro další vylepšování. V současné

podobě jde spíše o herní základ, při jehož tvorbě jsme se snažili udělat přidávání nových prvků co nejlehčím. Následuje výčet možností, jakými směry by se další úpravy mohly ubírat.

- Očekává se přidávání nových herních úrovní k pěti základním v současné verzi. Pro jejich návrh by měl herní designer využívat námi vytvořené rozšíření editoru Unity. Kromě toho by bylo vhodné obohatit, v současné podobě spíše chudou, nabídku kouzel a nepřátelských monster. Základní návod, jak přitom postupovat, nabízí část 7.3.1, respektive 7.3.3.

- Jednotlivé objekty hry mají mnoho parametrů, které je možno upravovat přímo v editoru Unity. Jde například o rychlost a sílu jednotlivých druhů příšer nebo nároky na manu pro provedení kouzel nebo stavbu hradu. Hodnoty těchto vlastností byly voleny pouze na základě odhadů, jak budou ve hře působit, a bylo by dobré je upravit dle zpětné vazby hráčů, kteří hru otestují.

- Zajímavým prvkem obohacujícím základní chování příšer by bylo nějakým způsobem implementovat chování celé skupiny příšer dohromady, například vytvořit roj útočících vos. Bližší seznámení s problematikou chování skupin nabízí zdroj [1] v kapitole 3 a [2] kapitola *Flocks and Crowds*.

- Více pozornosti by si také jistě zasloužilo chování nepřátelského kouzelníka. Minimálně by bylo vhodné rozšířit jeho sortiment kouzel, která by pak mohl strategicky používat. Mohli bychom také doplnit parametry jeho „osobnosti“. Například to, jak moc se snaží na ostatní útočit, nebo naopak se konfliktům vyhnout. Takto by bylo možné vytvořit více kouzelníků s různými charaktery a bylo by zajímavé sledovat vývoj hry, kde by takoví kouzelníci útočili nejen na hráče, ale bojovali i mezi sebou.

- Pokud by naším cílem bylo přiblížit se původní hře Magic Carpet co nejvíce, můžeme kromě kouzel a příšer doplňovat další drobnosti, na které nám zatím nezbyl prostor. Například to, že ve svém hradu je kouzelník v bezpečí a nepřátelské útoky jsou odráženy, nebo celý koncept prostých obyvatel světa, který je popsán v podkapitole 1.1.9.

- Zážitek ze hry by jistě prohloubily hezčí modely kouzelnických hradů a příšer, které jsou v současnosti spíše ilustrativní. Stejně tak bychom mohli doplnit zvukové nebo působivější vizuální efekty doprovázející používání kouzel. Například vytvoření spáleného povrchu terénu v místě dopadu ohnivé střely. Oživujícím a dosud neimplementovaným prvkem originálu je také výskyt *triggerů*, míst, jejichž návštěva má nějaký speciální efekt. Mohou odkrýt polohu kouzla na mapě, nebo způsobit, že se hráč ocitne v obklíčení nepřátel a podobně.

- V současné podobě hra postrádá téměř veškeré animace. Ať už jde o pohyb příšer, nepřítele nebo například postupnou výstavbu hradu. Další práci lze tak zaměřit právě tímto směrem a navíc s výhodou použít podporu pro tvorbu animací, kterou prostředí Unity nabízí.

- Významným a rozsáhlým rozšířením, které by celé hře dodalo další rozměr, je přidání režimu více hráčů. Ti by sdíleli herní svět a každý by ze svého počítače ovládal jednoho kouzelníka. Cílem této verze hry by mohlo být buď co nejdříve získat požadovaný objem many, nebo jednoduše porazit všechny soupeře.

Seznam použité literatury

[1] C. Brom, J. Lukavský, O. Šerý, T. Poch, and P. Šafrata. Affordances and level-of-detail ai for virtual humans. *Proceedings of Game Set and Match 2*, 2006. [cit. 2018-05-08].

[2] R. Barrera, A. S. Kyaw, and T. N. Swe. *Unity 2017 game AI programming: leverage the power of Artificial Intelligence to program smart entities for your games*. Packt Publishing Ltd., Birmingham, UK, 2018.

[3] M. Buckland. *Programming game AI by example*. Wordware Publishing, Inc., Plano, TX, 2005.

[4] Unity Technologies. Navigation system in unity. `https://docs.unity3d.com/Manual/nav-NavigationSystem.html`, 2017. [cit. 2018-04-22].

[5] Ken Perlin. Improved perlin noise. `https://mrl.nyu.edu/~perlin/paper445.pdf`, 2002. [cit. 2018-01-16].

[6] Pavel Tišnovský. Perlinova šumová funkce a její aplikace. `https://www.root.cz/clanky/perlinova-sumova-funkce-a-jeji-aplikace/`, 2007. [cit. 2017-12-08].

[7] Unity Technologies. Unity store. `https://store.unity.com/`, 2017. [cit. 2017-10-28].

[8] Unity Technologies. Installing unity without the hub. `https://docs.unity3d.com/Manual/InstallingUnity.html`, 2017. [cit. 2017-10-28].

[9] Unity Technologies. The main windows. `https://docs.unity3d.com/Manual/UsingTheEditor.html`, 2017. [cit. 2018-05-16].

[10] R. Watkins. *Procedural content generation for Unity game development: harness the power of procedural content generation to design unique games with Unity*. Packt Pub, Birmingham, UK, 2016.

[11] F. Sapio. *Unity UI Cookbook*. Packt Publishing Limited, Birmingham, UK, 2015.

[12] Martin Schmidt. Magic carpet download. `http://www.oldgames.sk/game/magic-carpet/`, 2009. [cit. 2018-02-11].

[13] Wikipedia contributors. Magic carpet (video game). `https://en.wikipedia.org/wiki/Magic_Carpet_(video_game)`, 2018. [cit. 2017-09-21].

[14] Návod - magic carpet 1. `http://www.abecedaher.cz/index.php?page=navod&page2=zobrazit&id_hry=0000002037`. [cit. 2017-09-25].

[15] Redakce Games.cz. Retro gamesplay – magic carpet. `https://games.tiscali.cz/video/retro-gamesplay-magic-carpet-300857`, 2017. [cit. 2017-10-25].

[16] Jonathan Gordon. Behind the scenes of magic carpet. https://www.gamestm.co.uk/features/behind-the-scenes-of-magic-carpet/, 2015. [cit. 2017-12-05].

[17] Hady ElHady. Top game engines in 2018. https://blog.instabug.com/2017/12/game-engines/, 2018. [cit. 2018-03-18].

[18] Wikipedia contributors. Unity (game engine), 2017. [cit. 2017-11-05].

[19] Unity technologies. Unity. https://unity3d.com/, 2017. [cit. 017-10-05].

[20] Unity technologies. Unity manual. https://docs.unity3d.com/Manual/index.html, 2018. [cit. 2017-10-05].

[21] Sebastian Lague. Game developement tutorials. https://www.youtube.com/channel/UCmtyQOKKmrMVaKuRXz02jbQ, 2017. [cit. 2017-12-08].

[22] Asbjørn. Game dev tutorials Brackeys. https://www.youtube.com/channel/UCYbK_tjZ2OrIZFBvU6CCMiA, 2017. [cit. 2018-02-06].

[23] Gabriel Aguiar Prod. Tutorial | indie games. https://www.youtube.com/channel/UCtb1s859RTxx-RIgFs5ZVQA, 2018. [cit. 2018-03-26].

[24] Jimmy Vegas. Start your game developement. https://www.youtube.com/channel/UCRMXHQ2rJ9_0CHS7mhL7erg, 2018. [cit. 2018-03-27].

A. Přílohy

Přiložené DVD obsahuje následující soubory a adresáře:

- `\MasterOfTheCarpet` – adresář obsahující projekt hry Master of the Carpet pro prostředí Unity.
- `\MasterOfTheCarpet-Game` – adresář obsahující spustitelnou hru Master of the Carpet v souboru `MasterOfTheCarpet.exe`.
- `\prace.pdf` – elektronická verze této práce ve formátu PDF/A-1a.
- `\MOTC.chm` – automaticky vygenerovaná referenční příručka projektu hry.