MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Simona Ondrčková

Destroy The Castle: 3D hra inspirovaná hrou Magic Carpet

Katedra distribuovaných a spolehlivých systémů

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

Studijní program: Informatika

Studijní obor: Programování a softwarové systémy

Praha 2017

Chtěla bych poděkovat všem, kteří mi pomohli v psaní této práce. Především děkuji Mgr. Pavlovi Ježkovi, Ph.D., který měl se mnou velkou trpělivost a během konzultací mi velmi pomohl.

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 …… dne…… podpis

Název práce: Destroy The Castle: 3D hra inspirovaná hrou Magic Carpet

Autor: Simona Ondrčková

Katedra: Katedra distribuovaných a spolehlivých systémů

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

Abstrakt: Cílem bakalářské práce je vytvořit počítačovou hru na bázi hry Magic Carpet. Hra má dva hlavní programátorsky zajímavé aspekty: umělou inteligenci a editor. Umělá inteligence je založená na přístupech, které umožňují více typů přemýšlení nepřátel. Editor umožňuje tvorbu vlastních úrovní zámku, kouzel a monster.

Jako další cíl si bakalářská práce klade zjistit, zda se takovýto typ hry dá vytvořit v Unity a jaké to přináší problémy nebo naopak výhody. Postupně se zaměřujeme na jednotlivé problémy nebo rozhodnutí, které byly při programování potřeba udělat a jak jsme je vyřešili v rámci Unity.

Klíčová slova: 3D hra, kouzla, Zničte zámek, editor, Unity

Title: Destroy the Castle: A 3D Magic Carpet-like Game

Author: Simona Ondrčková

Department: Department of Distributed and Dependable Systems

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

Abstract: The goal of the thesis is to create a computer game based on a game called Magic Carpet. The game has two main interesting aspects from the programming point of view: artificial intelligence and an editor. The artificial intelligence uses different approaches to create distinctively thinking enemies. The editor allows a designer to create new castle levels, spells and monsters.

Another goal of the thesis is to make sure, that this kind of game can be created using Unity game engine, and to determine it's advantages and disadvantages. Gradually, we focus on individual problems and decisions that had to be made and how we solved them using Unity game engine.

Keywords: 3D Game, spells, Destroy the Castle, editor, Unity

Obsah

1 Úvod

Cílem této bakalářské práce je naprogramování hry v C#. V rámci programování her je možné hru naprogramovat úplně od základu, to znamená vytvořit si vlastní reprezentaci scény, grafických objektů, řešit renderování grafiky, kolize objektů, atd.

Tyto mechanismy u většiny her fungují podobně. Nezávisle na tom, zda je hra ve stylu adventury nebo RPG, je nutné vyřešit, aby do sebe postavičky nenarážely (kolize objektů), aby se hezky zobrazily (reprezentace scény a grafických objektů) a uměly se pohybovat. Protože to většina her dělá víceméně stejně, není třeba naprogramovat tuto část hry pokaždé, ale naopak je obvyklé použít již předpřipravený software. Tomuto softwaru říkáme game engine.

Při výběru game enginu je pro nás důležité, aby byl populární, aby se v něm dalo programovat v C# a aby byl zdarma. Tato kritéria splňuje např. UrhoSharp, Xenko a Unity. Všechny jsou zdarma a dají se použít ve spojení se C#, některé jako např. Unity umožňují programování i v jiném jazyce (Unity v Javascriptu). Hlavním kritériem tedy bude popularita. Na stránkách blogu Microsoftu [1] můžeme najít seznam nejlepších her týdne. Z posledních 5 her týdne (26.7.16–7.9.16), byly 4 naprogramované v Unity, poslední byla naprogramovaná v XNA, což je game engine, který se používá převážně pro 2D hry. Tento blog je ale zaměřený na programování v .NETu, takže autor blogu vybírá pouze hry s game enginem, který je možné použít na této platformě. Další známkou popularity Unity je například to, že v prvním kvartálu roku 2016 bylo z nejlepších 1000 zdarma dostupných mobilních her 34% vytvořeno pomocí Unity [2].

Chtěli bychom zjistit, zda je jeho velká popularita skutečně opodstatněná, a zhodnotit, jak se v takovémto game enginu programuje. Také bychom chtěli zhodnotit, jak se používá ve spojení se C# a jak ulehčuje programování her. Abychom pokryli co nejvíce výhod a nevýhod, vybrali jsme si hru, která by se dala zařadit do více herních žánrů (kombinuje jejich různé prvky). Jednou z takovýchto her je hra Magic Carpet. Je to totiž kombinace střílečky, leteckého simulátoru a akčního RPG. Touto hrou bychom se chtěli inspirovat při tvorbě naší hry.

1.1 Magic Carpet

Hra Magic Carpet byla vytvořena v roce 1994 studiem Bullfrog Productions. V této 3D hře hráč ovládá svou vlastní postavičku kouzelníka, který bojuje proti ostatním kouzelníkům. Cílem je zabít všechny ostatní kouzelníky a zůstat jako poslední naživu. Tím se hra podobá akčním RPG nebo FPS (First-person shooter) střílečkám. Jen místo zbraní hráč používá kouzla. Tak jako v běžných střílečkách máme určitý počet nábojů do zbraně, v této hře máme manu, kterou můžeme použít pro tvorbu kouzel.

Pro běžné akční RPG je typické, že se hráč pohybuje po zemi nebo na koni, ve hře Magic Carpet se ale létá na létajícím koberci. To umožňuje větší volnost

a variabilitu pohybu. Z toho hlediska se hra řadí i ke zmíněným leteckým simulátorům. Naopak rozdílem od akčního RPG je to, že v typickém RPG má každý hráč určité množství many, která mu většinou v průběhu času automaticky stoupá. Je to v podstatě jednou z vlastností hráče. Ve hře Magic Carpet ji hráč automaticky nedostane, ale je ji třeba získat. Ukládá se hráči ve speciální budově, tzv. zámku. Ze zámku se pak čerpá a lze ji proměnit v kouzla.

Kromě zdroje kouzel mana ve hře reprezentuje životní sílu objektů. Pokud hráč zemře a má stále manu (životní sílu) v zámku, umožní mu se oživit a pokračovat dále ve hře. Pokud je však jeho zámek zničen a není tedy kde udržovat životní sílu hráče, hráč se nemá jak oživit a umírá.

[Obrázek 1 – Screenshot ze hry Magic Carpet 2 ukazující balon a manu (principiálně se shoduje s podobou balonu a many v Magic Carpet)]

Kouzelník není jediná bytost ve hře, která má vlastní životní sílu – kromě něj se ve hře vyskytují monstra, která se volně pohybují po světě. Pokud je kouzelník zabije, jejich životní síla z nich vypadne ve formě mana balls. Mana ball je fyzická reprezentace many (na obrázku 1 vyznačeno červenou šipkou), která se ale nedá přímo proměnit v kouzla. Nejdříve se musí proměnit v astrální manu. Kromě nových mana balls (vytvořených po zabití monster), se ve hře vyskytují volné mana balls, které jsou ve světě již od začátku.

Proces proměny mana balls v manu je automatický a děje se uvnitř zámku hráče. Je však potřeba si mana balls přivlastnit. Hráč má k tomu uzpůsobené speciální kouzlo, které nestojí žádnou manu. Pokud se přivlastněné míče dostanou do blízkosti hráčova zámku, zámek je nasaje do sebe a promění v použitelnou manu. Jak ale mana balls dostaneme do zámku, pokud jsou daleko? Na to jsou ve hře balony.

Ty se samy ovládají a odletí od zámku pro mana balls, vtáhnou je do sebe a pak se vrací k zámku, do kterého je přenesou. Aby balony věděly, kam mají jít, je důležité si mana balls přivlastnit. Poté, co si hráč mana ball přivlastní, balon sám vyletí a vyzvedne ho. Každý balon má určitou kapacitu, kolik mana balls dokáže vyzvednout. Do zámku se vrátí, když má naplněnou kapacitu nebo když už není žádný přivlastněný mana ball, který by mohl vyzvednout.

Zámky mají také určitou kapacitu many, kterou v sobě dokáží udržet. Pokud zámek získá více many, než je schopen unést, mana z něho vytryskne a zůstane v jeho okolí ve formě mana balls. Tuto kapacitu i kapacitu balonů má hráč možnost zvýšit vylepšením zámku.

Vylepšení má i další výhody, jako je například větší život zámku. Pokud se někdo pokusí vylepšený zámek zničit a zničí mu život, zámek se jen degraduje na nižší úroveň a nepřítel musí zničit i ten, a teprve pak je zámek doopravdy zničen. Pouze pokud nepřítel zničí zámek první úrovně, je zámek opravdu zničen.

Prostředí hry netvoří jen monstra čekající na kouzelníky. Naopak jsou zde postavičky, které si žijí vlastním životem. Některé jsou nebezpečné a útočí na toho, koho vidí, ostatní jen proletí kolem a kouzelníka ignorují. Další si zase pamatují akce hráče, a pokud na ně zaútočil, budou se mu snažit pomstít. Hra obsahovala i teleporty nebo tajné spínače, jež hráče někam přenesly nebo na něj spustily blesky.

1.2 Specifikace hry

V předchozí kapitole jsme popsali, jak vypadá hra Magic Carpet, v této kapitole si přiblížíme, jakou hru bychom chtěli naprogramovat a v čem se budeme inspirovat původní hrou Magic Carpet a v čem se budeme lišit.

Stejně tak jako originál bude naše hra ve 3D, kde hráč bude ovládat svou postavičku, která bude umět létat a jejím cílem bude zničit všechny protihráče (nepřátelské kouzelníky). Dále ze hry Magic Carpet převezmeme hlavní koncept manového systému. To znamená, že v naší hře budou také zámky, ve kterých se mana balls promění v manu, také budou k dispozici balony, které odletí od zámku pro mana balls a přinesou je zpět, aby se transformovaly na manu. Mana balls bude znovu nutné si přivlastnit. Dalším konceptem, který použijeme, je vylepšování zámků. Vylepšování umožní zvýšit kapacitu zámku, balonů, zvýšit životní sílu i obranné schopnosti zámku. Zámky se totiž budou bránit útokům nepřítele pomocí jednoduchého kouzla. Zámek k tomu bude používat vlastní manu, které má neomezeně a kterou není třeba nikde získávat, nedá se ale použít na nic jiného než na obranu zámku, která funguje automaticky. Pokud bude zámek pod útokem, tak vždy po nějaké době – té říkejme cooldown perioda – vystřelí na svého útočníka. Tuto cooldown periodu bude možné snížit vylepšením zámku.

Pokud jde o monstra, použijeme dvě různé varianty jejich chování, které se vyskytovaly i v původní hře. Jednou z nich jsou monstra, která zaútočí hned, jakmile hráče spatří. Druhou variantou jsou monstra, která hráče ignorují a věnují se své vlastní činnosti, brání se pouze v případě, že jim hráč zabije někoho z jejich

populace. V každém případě monstra nebudou útočit na ostatní monstra nebo na zámky.

Na rozdíl od originálu hry bude v naší hře k porážce nepřítele stačit zničení zámku. Aby hráč vyhrál, bude muset zničit všechny ostatní zámky. Umožnuje to alternativní přístup k hraní hry. Pokud bude hráč umět dobře létat, mohl by se teoreticky vyhnout jakémukoli střílení na nepřátelského kouzelníka. Další změnou je to, že mana se nevyskytuje volně ve světě, ale lze jí získat pouze zabitím nepřátel.

Nepřátelští kouzelníci budou mít rozumnou umělou inteligenci, aby se chovali přibližně jako reální protihráči: budou schopni zaútočit na hráče, bránit svůj zámek a posbírat si manu, pokud jí nebudou mít dost. Umělé inteligenci se bude podrobněji věnovat kapitola 1.2.3.

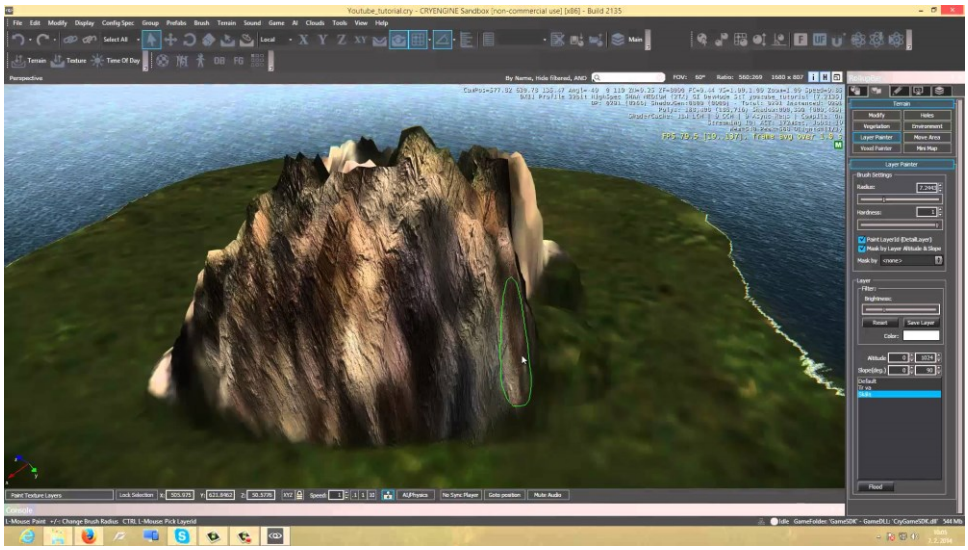
1.2.1 Motivace editoru

V současné době je typické, že se ke hrám přidává editor, který umožňuje jejich úpravu. Obvykle se v editorech dá upravit hráč, jeho protivníci (pokud hra vůbec nějaké protihráče má) a svět, ve kterém se hra hraje.

Úprava protivníků má několik variant. U jednodušších editorů se obvykle upravuje pouze počet protivníků, u složitějších editorů i jejich vzhled a chování. Výhodou editoru je, že je možné vytvořit si nové herní úrovně a hra je zábavná i při druhém a dalším hraní, protože si ji hráč přizpůsobí. Další výhodou je, že je možné oddělit vzhled objektů od jejich fungování. Stačí naprogramovat, že monstra budou útočit na kouzelníky, ale to, jestli je monstrem had nebo kostlivec, už není důležité. Podobu monster bývá možné v editorech upravit tím, že se vytvoří nový (3D) model monstra.

1.2.2 Specifikace editoru

V naší hře bychom se chtěli držet principu zmíněného v předchozí kapitole. Vytvoříme si tedy náš vlastní editor, který umožní vytvořit nová monstra a kouzla. Zároveň by bylo vhodné umožnit také větší flexibilitu hry – například bychom mohli umožnit designerům vytvořit kostlivce a dát je pouze do jedné části světa, ve které se hraje. Tím by se svět stal mnohem různorodější. K tomu tedy vytvoříme další část editoru, která umožní změnit následující hru.



[Obrázek 2 – Úprava terénu (https://www.youtube.com/watch?v=wDUm6l4C_Xo)]

V předchozí kapitole jsme zmínili, že editory slouží především k úpravě světa, ve kterém se hraje. Kromě úpravy oblastí pro monstra je možné upravovat i terén.

Úprava terénu spočívá v umožnění plynulé změny tvaru terénu, např. vytvořením hory nebo údolí. To se zpravidla dělá tak, že hráč má k dispozici štětec (zelená oblast v obrázku 2), kterým nastaví, zda chce zvyšovat nebo snižovat terén. Potom kliknutím na vybranou oblast zvýší nebo sníží terén. Okolí terénu se upraví, aby přechod vypadal plynule. To znamená, že pokud zvýšíme terén v jednom bodě a hned vedle v druhém bodě, hory se spojí do jedné, jak je vidět na obrázku 2.

V rámci editoru by mělo být možné upravit úrovně zámku, pak upravit terén a oblasti monster, jak je zmíněno výše. V neposlední řadě musí editor umožňovat výběr nepřátelských kouzelníků.

Druhou částí editoru bude editace samotných objektů. Prvním vyrobitelným objektem je kouzlo. Bude u něj možné nastavit poškození, manu, typ, efekt i ikonu. Dalším objektem je monstrum, které by mělo být téměř plně nastavitelné – od vzhledu, poškození, až po základní animace. Poslední, co půjde upravit, je umělá inteligence nepřátel. Mělo by být možné částečně upravit jejich chování. Dále by hra měla být naprogramovaná tak, aby se umělá inteligence dala snadno rozšiřovat.

1.2.3 Specifikace umělé inteligence

Naše hra je single player, je tedy potřeba vymyslet, jak se budou chovat počítačoví nepřátelé. V rámci počítačových her je možné umožnit počítačově ovládanému hráči, aby měl veškeré dostupné informace (kdo má kolik životů, many, kde se kdo vyskytuje atd.). Pokud by počítačový hráč využil všechny tyto poznatky, bylo by téměř vždy možné (pokud na to má počítač dostatečnou výpočetní kapacitu) najít způsob, jak hráče porazit.

Chceme ale, aby hra byla zábavná a hráč měl možnost vyhrát. Měla by vytvořit iluzi nového světa a hráče motivovat k tomu, aby se co nejvíce do hry vložil. Postavy

v ní by se měly chovat co nejrealističtěji. Proto je potřeba vymyslet jiný způsob chování nepřítele, ideálně takový, který by se co nejvíce podobal chování lidského protihráče. A to počínaje věcmi, které nám mohou připadat úplně samozřejmé (například aby hráč nestřílel do vlastního zámku), až po rozhodnutí, která nejsou předem jasná ani pro lidské hráče (např. pokud mám málo zdraví, mám raději zaútočit a pokusit se porazit nepřítele nebo se vyléčit?).

1.2.4 Umělá inteligence ve hře Zničte zámek

V naší hře Zničte zámek by měl nepřítel umět několik věcí: zaútočit na zámek, vyléčit se, bránit vlastní zámek nebo např. získat manu, když už žádnou nemá. Dále by měl být schopný reagovat na to, co se zrovna děje, tj. například pokud na něj útočí monstrum, měl by být schopen se rozhodnout, jaká je nejvhodnější reakce (např. utéct nebo se bránit). Nestačí však, aby tato jednotlivá chování uměl, je třeba se správně rozhodnout, kdy bude něco dělat nebo naopak kdy něco dělat přestane. Představme si situaci, kdy hráč A zaútočí na hráče B. Mezitím, ale hráč C zaútočí na zámek hráče A. Hráč A by tedy měl hráče B nechat být a měl by se vrátit, aby bránil svůj zámek.

Je důležité, aby umělá inteligence brala v úvahu vzdálenosti. Pokud je třeba zabít monstrum, aby kouzelník získal manu, je vhodné vybrat jedno z monster, které je blízko, nikoli obletět polovinu světa. Dále by umělá inteligence měla brát v úvahu cenu kouzel a pokud možno jimi neplýtvat. Musí umět vybrat to nejlevnější kouzlo, které způsobí co největší poškození. Také by neměla vybírat kouzla, na které nemá dostatek many.

Jaké jsou možnosti implementování umělé inteligence a jaký způsob jsme si vybrali, bude podrobně rozebráno v kapitole 2.3.2.

1.3 Shrnutí cílů bakalářské práce

V této bakalářské práci si stanovujeme následující cíle:

1. Vytvořit hru Zničte zámek v Unity
   1.1 Vytvořit v ní počítačem ovládané protihráče (umělá inteligence)
   1.2 Vytvořit oblasti monster.
2. Naprogramovat editor hry a herních objektů
   2.1 Vytvořit editor následující hry
   2.2 Umožnit tvorbu nových kouzel a monster
   2.3 Umožnit úpravu terénu.
   2.4 Umožnit úpravu nepřátel a jejich chování
3. Zjistit výhody a nevýhody programování takovéto hry v Unity
   3.1 Zhodnotit její programování v rámci Unity.

2 Analýza a architektura

Jedním z cílů naší bakalářské práce je vytvořit počítačovou hru, inspirovanou hrou Magic Carpet. V předchozí kapitole bylo nastíněno, jak by hra měla fungovat. Přesto je mnoho rozhodnutí, která se v průběhu programování musí udělat a nejsou specifikovaná v zadání. Tato rozhodnutí je možné rozdělit podle prvků hry, ke kterým se vážou (např. hráč, kouzlo, nepřítel…). V každé podkapitole probereme rozhodnutí, která se týkají jednoho prvku. Zároveň si u jednotlivých prvků detailněji popíšeme, jak budou fungovat.

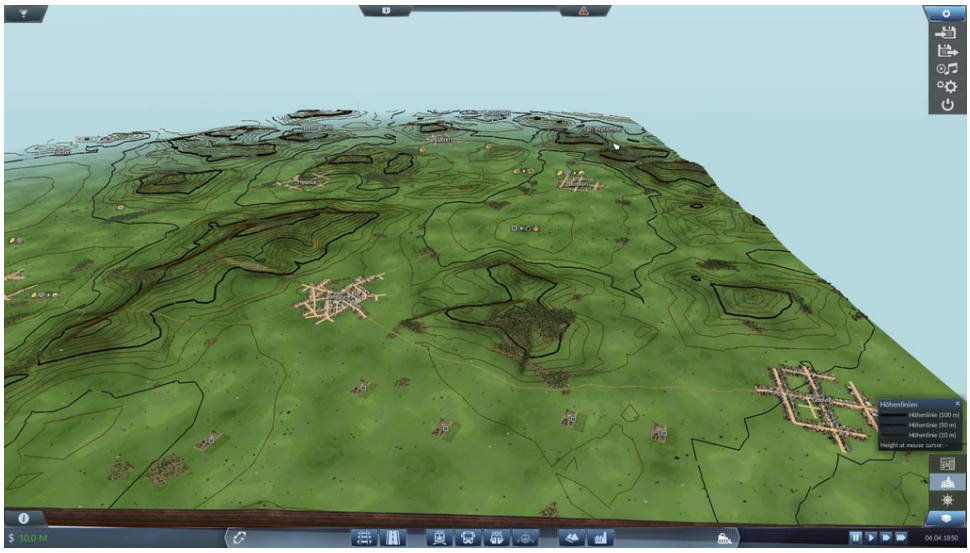
2.1 Herní svět

Jedním z prvních rozhodnutí při tvorbě hry byla podoba herní světa, s níž souvisí vymezení jeho hranic. Náš svět totiž není nekonečný, a je tedy potřeba určit, jakým způsobem ho ohraničíme. Rozhodovali jsme se mezi třemi různými možnostmi: ohraničené prostředí, omezená oblast tvářící se jako neomezená a planeta.

2.1.1 Ohraničené prostředí

Ohraničené prostředí reprezentuje nějakou oblast, většinou obdélník nebo čtverec, která má kolem sebe hranici. Tato hranice by mohla být ve hře reprezentována např. hustou mlhou nebo blesky. Ty by mohly hráči i ublížit, pokud by se přes ně pokusil projít. Zajímavé na této možnosti je to, že každý areál ve hře může mít jinou úroveň nebezpečí např., pokud budou zámky v rohu, jsou z více stran kryté bariérou, a tedy chráněny, zatímco na zámky uprostřed lze útočit odkudkoli. Tato možnost ale výrazně omezuje svobodu hráče. Nemůže se totiž pohybovat libovolným směrem. A pokud by hranice útočila, mohl by za to být dokonce trestán.

Příkladem hry, která používá tento způsob ohraničení, je Transport fever. Obrázek 3 ukazuje, kde je vidět hranice mezi světem a nicotou.

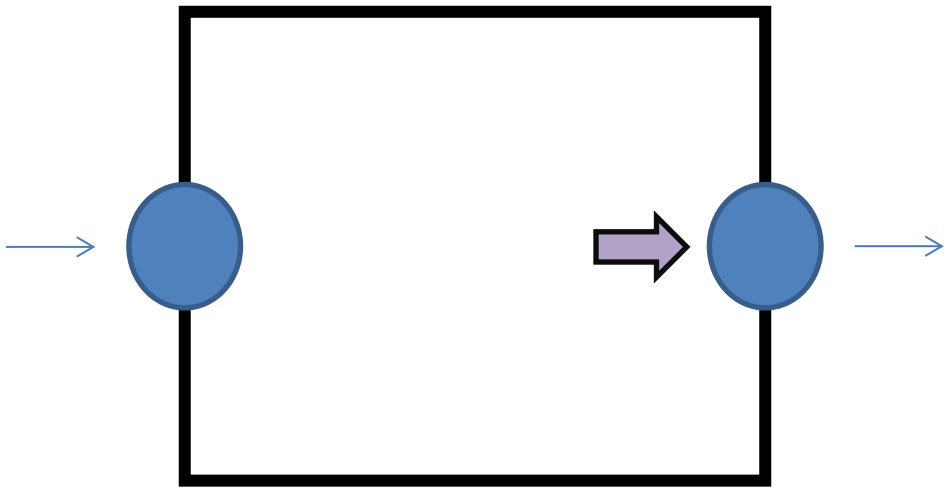


[Obrázek 3 - Mapa světa ve hře Transport fever (https://www.transportfever.net/filebase/index.php/Entry/2361)]

2.1.2 Omezená oblast tvářící se jako neomezená

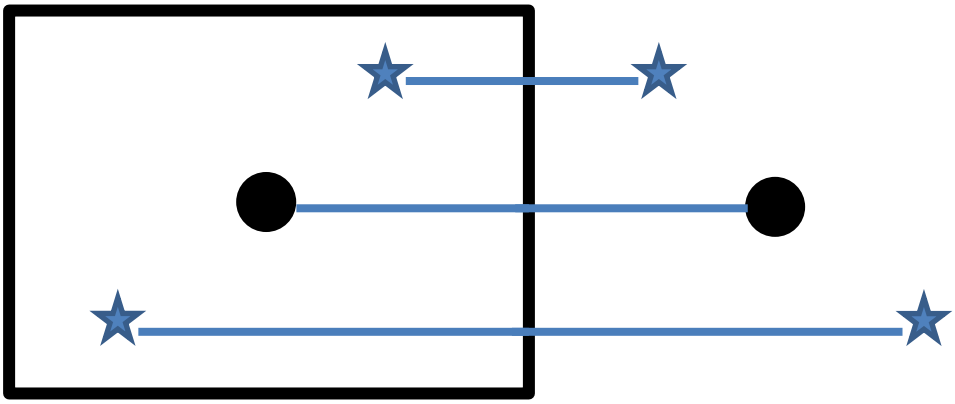
Tato metoda také používá obdélník jako podklad, ale místo vytvoření hranice kolem, je hráč, který se dostane na konec oblasti, teleportován zpět na její začátek. To samotné by bylo pro implementaci poměrně jednoduché – stačilo by pouze změnit pozici hráče, když se přiblíží k pomyslné hranici. Nevýhodou je ale to, že by samotná teleportace byla pro hráče velmi nepříjemná. Není totiž intuitivní, že by se najednou měl přemístit jinam. Hráč by byl jednak přemístěn bez varování, a navíc by nevěděl kam. Mohl by se např. chtít schovat daleko od přišer, ale teleportace by ho přesunula přímo do údolí kostlivců. Bylo by nutné dát hráči možnost zjistit, kam se bude teleportovat. Takovou možnost představuje zrcadlení.

Na obrázku symbolizuje modrá elipsa hráče. Poté, co by došel na hranici (označeno fialovou šipkou) oblasti (černý čtverec), teleportoval by se automaticky na začátek oblasti.



[Obrázek 4 – Teleportace hráče při přechodu bariéry]

Zrcadlení by fungovalo tak, že by za pomyslnou hranici oblasti zrcadlově zobrazilo původní oblast, jak je vidět na obrázku 5. Obrázek zobrazuje pouze zrcadlení za jednu stranu, ne za všechny čtyři. Černá koule symbolizuje nepřítele a hvězdy monstra. Hráč by tedy za koncem hranice viděl začátek oblasti nebo celou oblast, což by způsobilo, že by mu prostředí připadalo plynulé. Hráč by se na konci oblasti teleportoval na začátek nové oblasti, ale měl by pocit, že šel dál, jako by tam žádná hranice nebyla.



[Obrázek 5 – Zrcadlení objektů]

Dojmu zrcadlení dosáhneme pomocí toho, že budeme mít nějaké pole se všemi objekty a terénem, a pokud se hráč přiblíží k jedné z hranic, budeme zobrazovat objekty z tohoto pole a na opačné straně mapy je vymažeme. Například na obrázku 5 vidíme kolečko, které zmizí z původního místa, pokud se hráč přiblíží k pravé hranici, a zobrazí se proti němu.

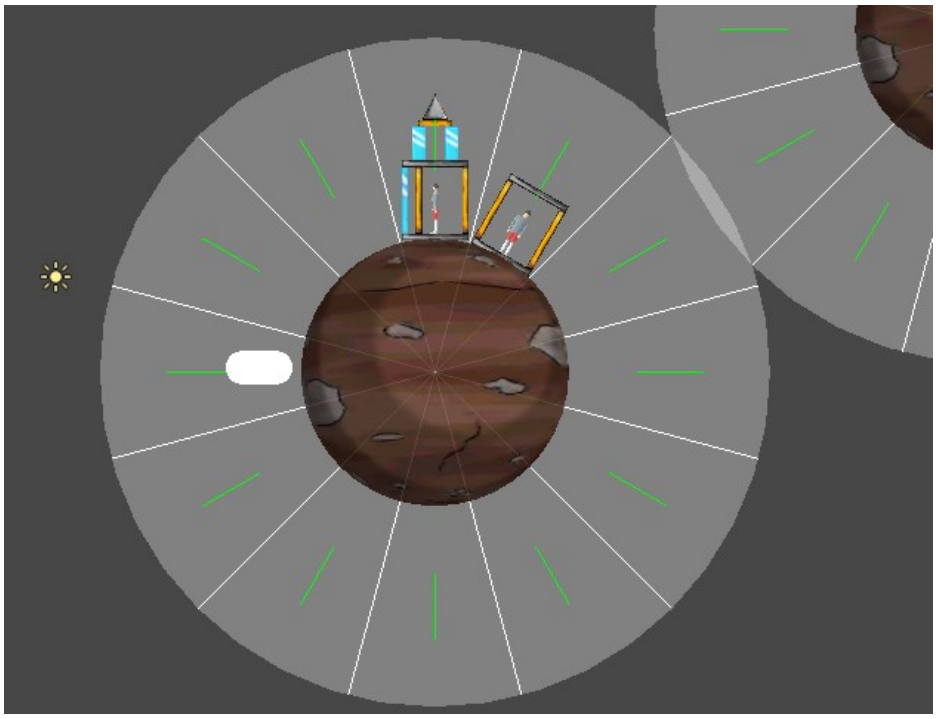
Pokud se hráč otočí a nemá v dohledu žádnou překážku, může se stát, že i když je na hranici (tzn., že by se měly zobrazit objekty za ní) a ohlédne se, uvidí všechny objekty na svých původních místech. V rámci generování objektů bude tedy třeba brát v úvahu i rotaci objektu, nejen jeho pozici. V podstatě je třeba vypočítat zorné pole objektu a zobrazovat objekty, které spadají do tohoto zorného pole.

Problém by mohl nastat, kdyby byla příliš malá mapa oproti zornému poli. V tom případě by se mohlo stát, že by hráč viděl objekt několikrát za sebou. To je možné snadno vyřešit zmenšením zorného pole nebo zvětšením mapy.

2.1.3 Planeta

Dalším způsobem, jak vytvořit ohraničenou oblast, která vypadá plynule, je místo zrcadlení vzít strany obdélníku a slepit je, čímž vznikne koule neboli planeta. Hlavní výhodou tohoto přístupu je, že není třeba řešit žádné zobrazování objektů. Všechny objekty se na mapě zobrazují na svých původních místech.

Nevýhodou této techniky je zejména gravitace. Najednou nestačí objekty přitahovat jen k zemi, ale k bodu v prostoru (jádru koule). S tím souvisí rotace objektu. Je potřeba, aby objekt stál vždy rovně vzhledem k zemi.



[Obrázek 6 – Gravitace (https://forum.unity3d.com/threads/faux-gravity-physics-sillyness.144309)]

Objekt (osobu, jak je vidět na obrázku), je nejdříve potřeba přitáhnout k zemi – to se dělá pomocí vektoru od objektu směrem ke středu jádra (na obrázku 6 vyznačené zelenou čarou pro objekty uprostřed každého sektoru). Tento vektor se přičte ke směru objektu a zajistí, že objekt spadne až na zem. Na obrázku vidíme, že osoba v druhém sektoru je částečně otočená, čehož nelze docílit pouhým působením vektoru, ale je potřeba upravit i rotaci. Bez toho by osoba zůstala natočená stejně jako osoba v prvním sektoru.

Poslední aspekt, který musíme vzít v úvahu, pokud zvolíme kouli, je to, že najednou by se všechny objekty měly pohybovat po kružnici, nikoli přímo z místa na místo. Je nutné kopírovat zemský povrch, protože jinak by se objekty pohybovaly skrz zem.

2.1.4 Shrnutí

V rámci hry je pro nás velmi důležité neomezovat pohyb hráče a umožnit mu, aby se mohl pohybovat jakýmkoli směrem. Proto jsme vyloučili první možnost.

Naším cílem je zjistit, jak se hra bude programovat v Unity s možností co nejoptimálnějšího využití jeho schopností. Unity se samo stará o zobrazování objektů, z čehož vyplývá, že ho nebudeme přepisovat (abychom dosáhli zobrazení objektů na správném místě), ale využijeme planety, kde se objekt vždy zobrazuje na jednom místě. Problémy s gravitací se pokusíme vyřešit vlastním naprogramováním gravitace, o čemž podrobně pojednáme v kapitole 4.5.1.

2.2 Hráč

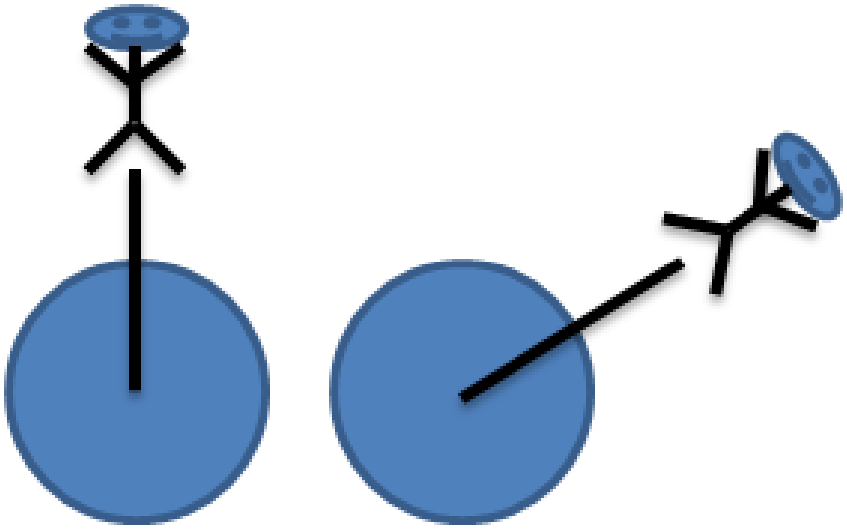
Jedním z nejdůležitějších prvků ve hře je hráč. V rámci jeho tvorby jsme se museli rozhodnout, jakým způsobem budeme indikovat život a manu, bojovat a létat.

2.2.1 Život a mana

Hráč bude ve hře ovládat vlastního kouzelníka, který má vlastní život a manu. Je samozřejmé, že hráč by měl být stále informován o tom, jaký je stav jeho života a many. Ve hrách se poměrně běžně používají indikátory na obrazovce nahoře vlevo (obvykle obdélníčky zabarvené podle množství života/many). Tento prvek tedy aplikujeme i v této hře. Pokud bude mít hráč příliš málo many, dostane varování v podobě blikajícího indikátoru many.

2.2.2 Létání

V této hře, podobně jako v ostatních hrách, může hráč ovládat pohyb kouzelníka pomocí šipek – v tomto případě kouzelník létá. Jelikož se hra odehrává na planetě, bude potřeba na hráčův pohyb aplikovat gravitaci a tím zajistit, aby nemohl odletět z planety a následně spadnout někam do neznáma. Gravitace funguje tak, že hráče přitahuje k jádru planety. Kromě toho nutí hráče rotovat, aby jeho osa y byla v souladu s vektorem do jádra země. Potom ho ve směru tohoto vektoru přitahuje.



[Obrázek 7 – Obrázek ukazuje, jak aplikace gravitace rotuje hráče]

2.2.3 Střílení kouzel

Kromě létání bude hráč také velmi často bojovat, což provádí pomocí střílení kouzel. Pro výběr kouzla bude na straně obrazovky k dispozici jejich menu. Zde bude ikona každého kouzla a hráč si bude moci kliknutím myši vybrat. Jak budeme indikovat, že hráč kouzlo vybral? Jednoduchou variantou je změna ikonky kliknutím myší na ikonku kouzla. Poté postačí, aby hráč klikl na cíl, a kouzlo se spustí. Pokud

si hráč bude chtít střílení rozmyslet, bude možné kliknout zpět do menu a vybrat si jiné kouzlo nebo klepnout pravým tlačítkem myši a zrušit jakýkoli výběr kouzla.

Pokud se hráč pokusí vyvolat kouzlo, i když nemá dostatek many, kouzlo by se nemělo vytvořit. Zároveň je potřeba hráče nějakým způsobem upozornit, aby věděl, proč se mu kouzlo nevytváří. My jsme zvolili již zmíněný blikající indikátor many.

2.2.4 Fyzický boj

Zmínili jsme, jak bude hráč bojovat pomocí kouzel. Co se ale s hráčem stane, když nebude mít žádnou manu? Nemůžeme ho nechat bez možnosti obrany, protože by hru prohrál, přestože by jeho zámek stále stál. Nemohl by se nijak bránit nepřátelským střelám. V případě, že by mana došla i nepříteli, ani jeden z hráčů by nemohl nic dělat a hra by přitom běžela do nekonečna.

Je tedy potřeba vymyslet nějaký způsob, jakým může hráč manu získat. V této hře se nevyskytuje volná mana – je možné ji získat pouze zabitím nepřátel. Jednou z variant je fyzický boj. Hráč bude mít možnost přiletět do těsné blízkosti k monstrům, a když bude přímo u nich, zaútočit na ně. To je riskantní, protože monstra se můžou bránit a hráč bude v dosahu jejich útoku. Pokud se mu podaří je zabít, získá jejich manu, kterou si bude moci obarvit. Díky tomu bude vždy existovat způsob, jak hru dohrát.

Je důležité, aby byl fyzický boj znevýhodněn oproti kouzelnému boji. Proto i poškození, které způsobí, bude minimální. Rozhodli jsme se, že pokud hráč udeří před sebe, měl by zranit všechna monstra v dosahu, nejen to jediné před ním.

2.3 Nepřátelé

Nepřátelé neboli počítačem ovládání hráči by se měli chovat co nejpodobněji reálným protihráčům. Toho docílíme tak, že si nejdříve rozdělíme činnost nepřítele na dvě oblasti: pohyb a umělou inteligenci. Pohyb se soustředí pouze na to, jak se nepřítel bude hýbat, umělá inteligence by mu měla říci kam (případně se bude pohybovat náhodně). Umělá inteligence řeší ostatní chování, tedy například zda je teď vhodné zaútočit na zámek nebo ne. Toto rozdělení si můžeme dovolit díky tomu, že kouzla fungují na dálku a není tedy nutné, aby byl nepřítel při střelbě na nějakém konkrétním místě.

2.3.1 Pohyb

Nepřítel musí umět létat po planetě. Existuje několik způsobů, jak toho docílit – např. pohyb v předurčených výškách nebo vznášení se nad objektem.

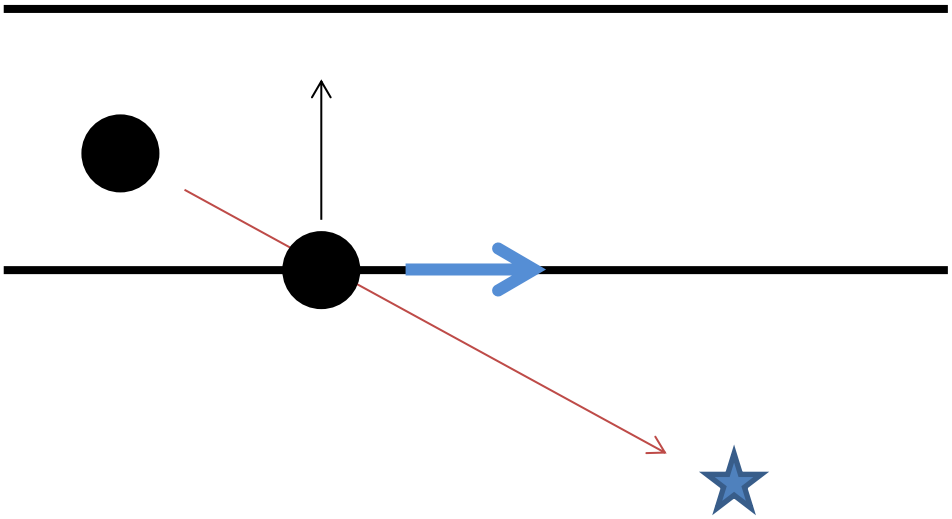
Pohyb v předurčených výškách

Pohyb v předurčených výškách je technika, ve které máme předem zadanou minimální a maximální výšku nad zemí. V každém okamžiku sledujeme, jakou pozici má nepřítel. Zjistíme vzdálenost ke středu planety (odečteme poloměr planety) a tím získáme informaci o tom, jak je nepřítel vysoko. Pokud je příliš nízko nebo

příliš vysoko, pouze mu k dalšímu pohybu přidáme směr nahoru nebo dolů. (Pokud by původně směřoval doprava a byl by pod minimální výškou, pohnul by se následně nahoru a doprava).

Nevýhodou této techniky je, že nekopíruje terén. Pokud bychom měli vysokou horu, mohlo by se stát, že do ní nepřítel vrazí, protože jeho minimální výška bude nižší než vrchol hory.

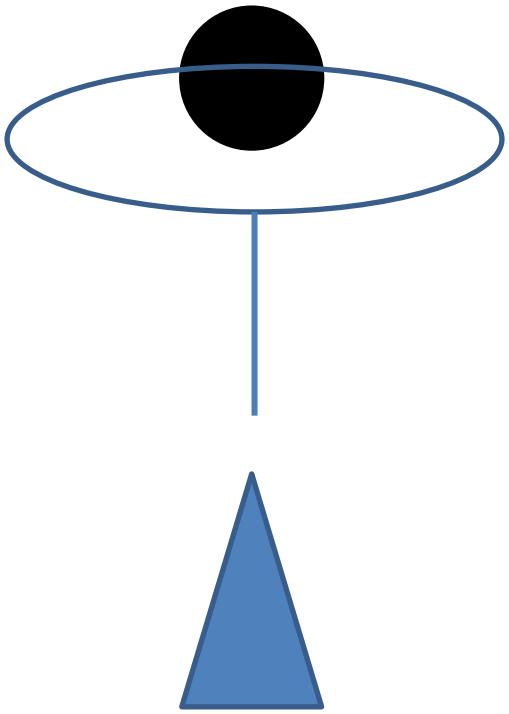
Dalším problémem je realizace fyzického boje. Stejně jako u hráče (viz kapitola 2.2.4) musí nepřítel umět zaútočit na monstra zblízka. Pokud nepříteli (černá koule na obrázku 8) zadáme, že se má přiblížit k monstru (hvězda na obrázku 8), bude problém, že je monstrum na zemi. Základní směr nepřítele bude dolů a dopředu (vyznačeno červenou šipkou na obrázku 8), ale jakmile se snese na úroveň minimální výšky, přičte si směr nahoru (černá šipka na obrázku 8), což vynuluje jeho směr dolů a ve výsledku se ve výšce své minimální výšky bude pohybovat rovně (modré šipka) a monstrum přeletí. Potom se otočí a zkusí to z druhé strany, ale výsledek bude stejný. Takto se k monstru nikdy nedostane. Lze to vyřešit dynamickou minimální výškou, která se při pronásledování monstra sníží na nulu.



[Obrázek 8 – Pohyb nepřítele k monstru při pohybu v předurčených výškách]

Vznášení se nad objektem na zemi

Kromě pohybu v předurčených výškách se nabízí další řešení – vznášení se nad objektem. Tato technika funguje tak, že nepřítel má pod sebou naváděcí objekt (trojúhelník na obrázku 9), který se pohybuje po zemi. Samotný nepřítel (černá koule na obrázku 9) se udržuje v určité výšce (nazveme ji ideální výška nad objektem) nad tímto objektem (modrý trojúhelník na obrázku 9) a pohybuje se náhodně v malém okruhu (šedý kruh na obrázku).



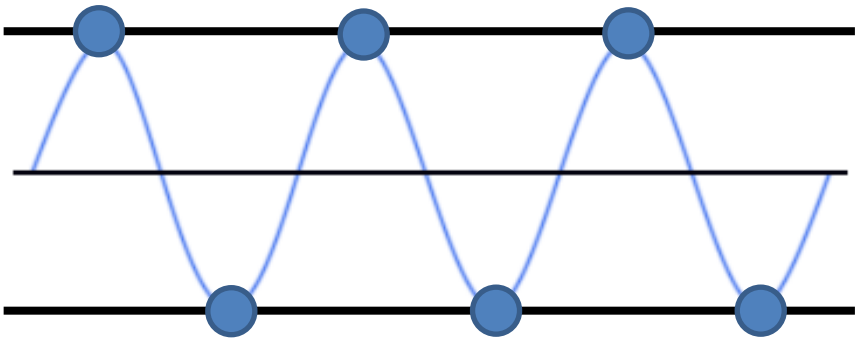
[Obrázek 9 – Vznášení se nad objektem]

Hlavní výhodou je, že nepřítel kopíruje terén planety. Pokud jeho naváděcí objekt narazí na horu, vystoupá na její vrcholek a nepřítel, který letí nad ním, také vyletí nahoru.

Nastává zde otázka, jak postupovat, když chceme někoho pronásledovat ve výšce. Naváděcí objekt by totiž vzlétl, což by způsobilo, že i kdyby naváděcí objekt dostihl cíl, nepřítel by se nad cílem pouze vznášel. Je to opačný problém než u fyzického boje v předchozí kapitole a je možné ho také vyřešit dynamickou výškou. Dalším řešením by byla aplikace gravitace na naváděcí objekt, čímž bychom zajistili, že bude stále na zemi, i když by měl pronásledovat někoho ve vzduchu. Ideální výšku potom můžeme zvýšit nebo snížit podle výšky pronásledovaného objektu. Tato variabilita ideální výšky vyřeší i problém předchozí kapitoly (jak sletět až na zem k monstru) a to tak, že pokud nepřítel pronásleduje monstrum, sníží se mu ideální výška na nulu, což mu umožní sletět až k zemi.

Shrnutí

Žádná ze zmíněných technik nemá výraznou výhodu ani nevýhodu. Výsledkem létání v předurčených výškách je dráha letu v podobě sinusoidy (obrázek 10), což pro let nepřítele není příliš vhodné, ale je to naopak vhodné pro let balonu.



[Obrázek 10 – Let v předurčených výškách]

Rozhodli jsme se tedy použít obě techniky. Pro pohyb nepřítele použijeme vznášení se nad objektem. K létání balonu použijeme pohyb v předurčených výškách. Více k této variantě bude popsáno v kapitole 2.7.1.

2.3.2 Umělá inteligence

K tomu, jak určit, co by měl nepřítel dělat, se lze postavit dvěma intuitivními přístupy. Prvním z nich jsou předem zadané úkoly, které se nepřítel pokusí udělat za jakýchkoli okolností. Druhým přístupem je reagovat na okolí – například obranou, pokud na hráče útočí monstrum, nebo ochranou zámku, pokud je v ohrožení. Úkoly tedy mohou být vyvolané buď prostředím, nebo „vnitřně“.

Dále rozlišujeme, co je důležité (je třeba tomu dát prioritu) a co je vedlejší. Úkoly jsme rozdělili na hlavní (většinou vnitřně vyvolané) a vedlejší (méně důležité, spíše reaktivního formátu.) Existují samozřejmě výjimky, například když je nepřítelův hrad pod útokem, obrana je jeho reakcí, i když je to jedna z jeho hlavních činností.

Mise

Úkoly, které nepřítel bude vykonávat, nazveme mise a rozdělíme je na hlavní a vedlejší. Hlavní mise mají za úkol řešit klíčové chování ve hře: útok na zámek, obranu zámku, útok na hráče, sbírání many a vyléčení se. Většina z nich (kromě obrany zámku) je vyvolaná vnitřně – to znamená, že podmínky, které musí být pro jejich spuštění splněny, jsou závislé pouze na stavu nepřítele. Například pro vyléčení se je nutné mít dostatečně málo zdraví, pro útok na hráče je třeba mít dostatek many, aby měl hráč čím útočit.

Vedlejší mise by měly být spíše reaktivního formátu. Měly by zjišťovat, „co se zrovna děje“ a jestli je vhodná doba, aby se spustily. Můžeme mít např. vedlejší mise obarvení many, která bude hlídat, jestli je v okolí nějaká mana, která by se dala obarvit. Pokud ano, tak nám mise oznámí, že je připravená. Další vedlejší mise, která řeší útok na monstra, bude připravená, pokud se v okolí budou vyskytovat nějaká monstra, na která bude možné zaútočit.

Jak bude nepřítel vybírat z připravených misí? Jak by se měl chovat, pokud má například dostatek many a málo života? Má se nejdřív vyléčit, nebo zaútočit na hráče? K tomu budou ve hře priority, které zajistí z připravených misí výběr té nejdůležitější.

U vedlejších misí postačí náhodný výběr, protože mají většinou stejnou důležitost. Hra však bude snadno rozšiřitelná i na vedlejší mise s prioritami.

2.4 Kouzla

Předtím, než se budeme věnovat jednotlivým typům kouzel, je potřeba rozhodnout, jak budou hráč a jeho protivníci kouzla střílet. Ve hře Magic Carpet je možné střílet kouzla úplně kamkoli, například do vzduchu a předpovídat pohyb nepřítele, který se následně s kouzlem střetne. Výhodou tohoto způsobu je, že má hráč úplnou volnost. Může např. kolem nepřítele vytvořit kruh ohnivých kouzel a efektivně ho tak uvěznit uvnitř. Další možností je vytvořit kolem zámku hradbu z kouzel.

Hlavním problémem s tímto typem střílení je to, že je třeba odhadnout, kam chce hráč střílet. Obrazovka hry je ve 2D, takže když hráč klikne myší na nějaké místo, není jasné, do jaké vzdálenosti chce vystřelit. Hráč může například kliknout mezi sebe a nepřítele, ale nemusí být zřejmé, jestli chce poslat kouzlo těsně před nepřítele, nebo těsně před sebe.

Řešením tohoto problému se budeme zabývat v kapitole: 4.3.1, kde nastíníme všechny varianty, které jsme vyzkoušeli.

2.4.1 Létající kouzla

Létající kouzla jsou kouzla, která se pohybují od kouzelníka k cíli. Tento typ kouzel může vždy ublížit jen jednomu cíli najednou, a to tomu, který trefí jako první. Po kontaktu s cílem kouzlo okamžitě vyprchá. Pokud se kouzlo do ničeho netrefí, bude létat tak dlouho, dokud nevyprchá jeho účinek, a potom zmizí.

Zajímavé rozhodnutí, které jsme museli udělat, se týkalo toho, zda bude kouzlo moci ublížit kouzelníkovi, který ho vyvolal, nebo zda by měl mít vlastní imunitu? Imunita je přirozenější, ale na druhou stranu kouzla bez imunity umožňují nepřátelskému kouzelníku dělat různé úhybné manévry, při kterých se může z nepozornosti zabít sám. Nakonec jsme zvolili tuto možnost.

2.4.2 Kouzla na místě

Kouzla na místě se objeví přímo tam, kam hráč klikne. Není možné se jim vyhnout, ale dá se z nich vystoupit. S tím souvisí, že existují dva typy těchto kouzel. Prvním typem jsou taková kouzla, která cíli ublíží jen jednou a poté, i když je s nimi cíl stále v kontaktu, mu už znovu život neuberou. Druhý typ kouzel ubírá život opakovaně. Tento typ nazýváme kouzelná stěna. Když do něj někdo vstoupí poprvé, ubere se mu život a následně mu začne běžet bezpečná doba, ve které může v kouzlu

setrvávat bez újmy. Jakmile ale bezpečná doba vyprší, pokud neukončí kontakt s kouzlem, opět se mu ubere život.

Kouzla mají předem určenou dobu, jak dlouho budou setrvávat na daném místě, a ta je nezávislá na tom, kolik monster/hráčů kouzlo ovlivní. Tento počet je omezen jen tím, s kolika objekty přijdou do kontaktu.

2.4.3 Jedovatá kouzla

Kromě výše zmíněných typů kouzel existují také jedovatá kouzla. Ta fungují tak, že pokud jednou zapůsobí na nějaký objekt, otráví ho a vždy po určitém čase mu znovu uberou život. To se děje po určitou předem stanovenou dobu. Život, který objektu ubere otrava, bude nastavitelný, takže bude možné vytvořit jedy, které jsou slabé, ale jejichž počáteční útok je obrovský, nebo naopak takové, které napoprvé neudělají skoro žádné poškození, ale postupně uberou značnou část života. Jedovatá kouzla mohou být buď létající – mohou vždy otrávit jen jednu osobu na jedno kouzlo, nebo na místě – otráví kohokoli, koho se v průběhu svého trvání dotknou.

2.5 Mana

Již jsme zmínili, jaká kouzla lze použít, ale k tomu, abychom vůbec nějaké kouzlo mohli vyvolat, je potřeba mana. Ta má ve hře tři podoby: mana balls neboli přírodní mana, normální mana a speciální mana zámku.

Normální mana je mana, kterou má každý hráč ve svém zámku. Ta se využívá k vyvolávání kouzel. Každý hráč má určité maximum many, kterou dokáže v zámku udržet. Pokud se do zámku dostane více many, než je zámek schopen uchovat, zbytek many vyletí ze zámku v podobě mana balls. Stejně tak pokud je zámek/balon zničen, jeho přebytečná mana ze zámku/balonu vyletí.

Mana balls budou malé koule, které cestují po světě. Každá koule v sobě bude mít určité množství normální many, kterou je možné proměnit v kouzla. Aby se hráč k normální maně dostal, musí si přivlastnit mana ball, což udělá speciálním zbarvujícím kouzlem typu kouzlo na místě. Mana balls mohou být přebarvené kdykoli, i když už patří nějakému hráči.

Vzniknou tím, že monstrum zemře. Vytrysknou z jeho mrtvoly a začnou se pohybovat po světě. Budou běžně podléhat gravitaci planety. To znamená, že se budou hromadit např. na spodku údolí. Pro jednoduchou orientaci ve hře jsme se rozhodli, že každý mana ball má v sobě stejné množství normální many. Pokud je potřeba vytrysknout méně nebo více many, než je násobek many v mana balls, výsledek se zaokrouhlí.

Poslední typ many je speciální mana zámku, ke které se hráč nikdy nedostane. Je to mana, kterou zámek používá pro svou obranu. Ta mu nikdy nedojde, ale není ovladatelná pro kouzelníka.

2.6 Zámek

Zámky jsou domovem pro hráče, skladují manu a umožňují hráčům se oživit. Aby bylo ve hře možné zámky od sebe odlišit, rozhodli jsme se, že každý zámek bude mít nějakou věž nebo jinou jeho část, která se zabarví do barvy hráče. Tato barva je také barvou, kterou hráč používá pro obarvení many.

2.6.1 Boj

V rámci zadání bylo řečeno, že se zámky musí umět bránit. To uděláme tak, že zámek bude mít své speciální létající kouzlo, které střílí po nepříteli. Jak často bude střílet a kolik života toto kouzlo ubere, závisí na úrovni zámku. Vždy se zvýšením úrovně se může poškození a doba mezi střílením zkracovat, ale nemusí. Vše by mělo být nastavitelné v editoru.

2.6.2 Vylepšení a degradace

Vylepšování zámku by mělo umožnit úpravu mnoha parametrů. Těmito parametry jsou kapacita balonu, maximální zdraví zámku, prodleva mezi střílením, poškození zámku a kolik many stojí následující vylepšení. Dále jsme se rozhodli, že samotné vylepšení by mělo zámku vylepšit zdraví. O kolik se mu vylepší zdraví, bude také nastavitelné v editoru v rámci tvorby zámku.

Hráči je potřeba nějakým způsobem ukázat, na jaké úrovni je zámek. Pro prvních pět úrovní jsme se rozhodli nechat zámek postupně rozrůstat přidáváním různých objektů. Například při přechodu z úrovně dvě na úroveň tři zámku přibude věž. Po páté a každé další úrovni se zámek při vylepšení pouze zvětší.

Kromě vylepšování zámku se ve hře vyskytuje také degradace zámku. Nastává tehdy, pokud zámek přijde o všechen svůj život. V tom případě se propadne o jednu úroveň a bude mít maximální život předchozí úrovně. Pokud je zámek na první úrovni a přijde o všechen život, je zničen a hráč prohrává.

2.7 Balon

Balony ve hře přenášejí mana balls do zámku, kde se promění v normální manu. Je to objekt, který bydlí v zámku, a pokud má hráč nějaký přivlastněný mana ball, vyletí za ním a pokusí se ho sebrat. V rámci zadání hry nebylo přesně určeno, jak by měl balon manu přitahovat nebo jak si vybírat, pro který přivlastněný mana ball se balon vydá. Hra bude používat nejjednodušší variantu, a to že balon přiletí k nejbližšímu přivlastněnému mana ball.

2.7.1 Sbírání many

V kapitole 2.3.1 byly popsány možné varianty pohybu nepřítele. Tyto varianty se nabízejí i pro pohyb balonu. První variantou je létání v předurčené výšce, druhou variantou je létání nad naváděcím objektem, který se pohybuje po zemi. Hlavním úskalím první varianty bylo, že nekopíruje terén, což u balonů nevadí, protože pro ně není problém mít dostatečně vysokou minimální výšku. Druhou výzvou byl způsob,

jak sletět na zem. U nepřítele je tento problém důležitý při útoku na monstra. U balonu pro sběr mana balls. Tento problém je možné obejít, pokud by se balon snesl k zemi na jediném místě – v zámku (řešilo by se to jako speciální stav balonu). Nad mana ballem by balon stál v minimální výšce a přitahoval ho gravitací.

Sbírání many by fungovalo tak, že by balon přiletěl k místu, kde je mana ball, všechny mana balls v okolí by přitáhl gravitací, a kdyby nad ně doletěl, mana balls by vystoupaly vzhůru a balon by je vstřebal. Mana balls, které hráč nemá přivlastněné, by byly také přitahovány gravitací míče, ale balon by je nevstřebával. Kdyby balon odletěl, spadly by zase na zem.

Druhou variantou by byl balon, který se snáší k zemi v místě many. Mana balls by se tedy kvůli balonu nijak speciálně nepohybovaly. Balon by pro ně sletěl až na zem a vyzvedl si je. S touto variantou by se mohlo použít i vznášení se nad objektem nebo vznášení se v předurčených výškách s variabilní minimální výškou.

Žádná z možností nemá oproti ostatním nijak výrazné výhody nebo nevýhody. Vybrali jsme variantu se vznášením se v předurčených výškách a balonem, který zůstává ve vzduchu nad mana balls, protože nám připadalo zajímavé mít pro nějaký objekt dva body gravitace, jádro planety, které drží mana ball na zemi, a balon, který ho nutí vyletět.

2.7.2 Útok a obrana

Další věcí, kterou je třeba rozhodnout, je, zda bude na balon možno útočit a pokud ano, jak se může balon bránit. Hlavním důvodem pro možnost útoku na balon by byla příležitost ukrást protihráči manu. Kdyby nepřítel sestřelil hráčův balon, mana by z něj vytryskla a nepřítel by si ji mohl přivlastnit. Pokud bychom zvolili tento způsob, je třeba rozhodnout, co mají balony dělat, pokud zemřou.

Ze zadání je jasné, že každý hráč musí mít balon pro získání many, jinak by nebylo možné ve hře pokračovat. Po zničení je tedy nutné balon nějakým způsobem obnovit. To by se dalo udělat tak, že se balon po svém zničení a vypuštění many, kterou v sobě držel, teleportuje zpět do zámku. Tím bychom získali možnost zpomalit sběr many protihráče.

Hráč A by hráči B zničil balon, který vyletěl pro mana balls hráče B. V zámku hráče B by se pak vytvořil nový balon, ale cestu by musel uletět znovu. To by se dalo ještě vylepšit tím, že by měl balon určitou oživovací periodu, a místo toho, aby se nový balon zrodil v zámku okamžitě, zrodil by se až po chvíli a zdržel by protihráče ještě o to déle.

Balon tedy bude možné zničit, ale bude mít balon možnost se útoku balon bránit? Rozhodli jsme se řídit podle skutečných balonů, které se nemohou bránit, a nechali jsme balony bezbranné. Na hráčově rozhodnutí bude, zda se bude věnovat obraně svých balonu nebo je ponechá napospas nepříteli s tím, že v určitém okamžiku mu mana dojde a nebude už moci útočit na hráčovy balony.

2.8 Monstrum

Kromě nepřátel jsou monstra jediné další bytosti ve hře, které mohou hráče ohrozit. Každé monstrum má vlastní život a obranu. Pokud na něj někdo zaútočí, spočítá se útok nepřítele, a v případě, že je větší než obrana monstra, ubere se mu život.

2.8.1 Boj

V rámci boje je nejprve třeba rozhodnout, kdo útočí na koho. Jak jsme již zmínili v předchozích kapitolách, monstra neútočí sama na sebe. Je to proto, že kdyby na sebe monstra útočila, navzájem by se zabila a hráč by vlastně nemusel nic dělat. Na zámky zase neútočí proto, že by byl zámek hráče neustále pod útokem a hráč by se musel neustále věnovat jeho obraně.

2.8.2 Pohyb

Ve hře je také potřeba určit, jak se monstra budou pohybovat. Používáme zde dva typy pohybu. Když nepřítel není nikde v dosahu, monstrum se bude pohybovat libovolně, když ale nepřítele uvidí, začne ho pronásledovat. V rámci hry nazýváme tyto dva typy pohybu mód procházka (wandering) a mód pronásledování (chasing).

Mód procházka

Mód procházka je mód, který je aktivní, pokud monstrum není v ohrožení a zároveň nejsou v bezprostřední blízkosti žádní nepřátelé. Monstrum by se tedy mělo nějakým způsobem procházet.

Nejjednodušší možností pohybu je náhodný pohyb. Ten by bylo možné implementovat například tak, že každých x sekund se monstrum otočí o náhodný počet stupňů a následně se vydá tím směrem. Problém s tímto přístupem spočívá v tom, že se monstrum může dostat kamkoli. Nedaly by se proto vytvořit předem určené oblasti monster, jak bylo požadováno v zadání (cíl 1.2). Kostlivci by mohli z údolí utéct a na hráče by tam nikdo nečekal, naopak jinde v bezpečné oblasti by se kostlivci hromadili ve velkých počtech.

Náhodný pohyb monstra je tedy nezbytné nějakým způsobem omezit. Mohli bychom například přimět monstrum sledovat, jak daleko je od určitého centrálního bodu, a pokud by bylo příliš daleko, dát mu pokyn, aby se vrátilo. Nevýhodou této varianty je ale to, že pokud má každé monstrum v oblasti přesně vymezenou hranici, stačí stát těsně za hranicí a žádné monstrum se hráče nedotkne. To se dá vyřešit např. tím, že každé monstrum bude mít trochu odlišnou hranici od hranice oblasti. Všechna monstra i nadále zůstanou přibližně v dané oblasti, ale některá hranici mírně překročí, jiná se k ní zase nikdy úplně nepřiblíží.

V módu procházka bychom také chtěli umožnit nastavení, aby se monstra pohybovala po předem určené trase. Ta by fungovala tak, že monstrum by mělo předem zadané destinace, ke kterým má dojít, a pokaždé, kdy by dosáhlo jedné, by

přešlo na další v pořadí. V okamžiku, kdy by došlo na poslední destinaci, přesunulo by se zpět na první.
Praktičtější variantou se nakonec místo sledování vzdálenosti ukázala předem zadaná destinace, ke které se monstrum snaží dojít, a to jak pro mód náhodné procházky, tak pro pevně určenou trasu. Více se touto problematikou zabývá kapitola 4.4.1. Sledování vzdálenosti využijeme v módu pronásledování.

Mód pronásledování

Mód pronásledování, jak název napovídá, je aktuální, když je v blízkosti monstra nepřítel, na kterého by monstrum mohlo zaútočit. Konkrétní způsob pronásledování nepřítele je velmi jednoduchý – stačí zjistit, jakým směrem od monstra nepřítel je, a tím směrem se vydat.

Zajímavější je, co bude monstrum dělat, pokud má kolem sebe dva nepřátele. Jednou možností je jít za tím, který je nejblíže. Tento způsob je ale velmi předvídatelný. Druhou možností je pokračovat v pronásledování již sledovaného nepřítele (pokud někoho již sleduje). Zde je zase nevýhodou to, že pokud hráče monstrum jednou začne sledovat, není možné mu už utéct.

Obě možnosti mají přibližně stejně silnou nevýhodu, a proto jsme se rozhodli je implementovat obě a umožnit hráči jejich nastavení. Monstrum bude mít v rámci editoru nastavitelný způsob, který použije, a hra by měla být rozšiřitelná na změnu módu pronásledování za běhu.

V popisu módu procházka jsme zmínili, že se monstrum bude pohybovat pouze v předem dané oblasti. Stejné pravidlo bude aplikováno i v rámci pronásledování. Pokud se tedy hráč dostane mimo vymezenou oblast, monstrum ho přestane pronásledovat a vrátí se zpět.

2.8.3 Zrak

V rámci pronásledování je zajímavé vzít v úvahu úhel pohledu monstra. Místo toho, aby se monstrum rozběhlo za nejbližším nepřítelem, kterého potká, přihlédne se k tomu, zda ho ze své pozice vůbec uvidí. Hráči se tak otevírá možnost se za monstrum vplížit a zabít ho zezadu. Jakým způsobem lze určit, jak široký je úhel zorného pole monstra? To by mělo být jednoduše nastavitelné při tvorbě nového monstra. Zadal by se úhel pohledu, který by v případě plných 360 stupňů zajistil, že monstrum vidí všude.

2.8.4 Narození monster

Již bylo zmíněno, že je důležité, aby se monstra pohybovala pouze ve vymezené oblasti, a proto je přirozené, aby se také rodila pouze v předem určené oblasti. Budou se v této oblasti rodit všechna monstra na stejném místě, nebo každé někde jinde? Zrodí se všechna najednou, nebo se budou rodit postupně? Předtím, než začneme programovat, je třeba si na tyto otázky odpovědět.

Výhodou zrození monster na jednom místě je to, že by se například dala vytvořit jeskyně, v níž se rodí kostlivci, kteří se pak pohybují po celém údolí. Problémem ale je, že pokud by zrozená monstra dostatečně rychle neodešla z rodícího areálu, docházelo by ke kolizi a monstra by padala na sebe. Další nevýhodou je, že by bylo zcela zřejmé, kde monstra vznikají, a dalo by se jim tak snadno vyhnout. Lepší řešení nabízí varianta zrodu monster libovolně v rámci dané oblasti.

Pokud bychom se rozhodli zrodit všechna monstra najednou, bude pro hráče velmi těžké se po mapě pohybovat, protože by na něj hned všechna monstra zaútočila. Monstra tedy budeme vytvářet postupně v předem určených intervalech. Tím ale hra opět získává na předvídatelnosti, protože nové monstrum se zrodí například každých pět vteřin. Proto do hry zaimplementujeme nějaký náhodný generátor, který rození monster rozprostře do daného intervalu.

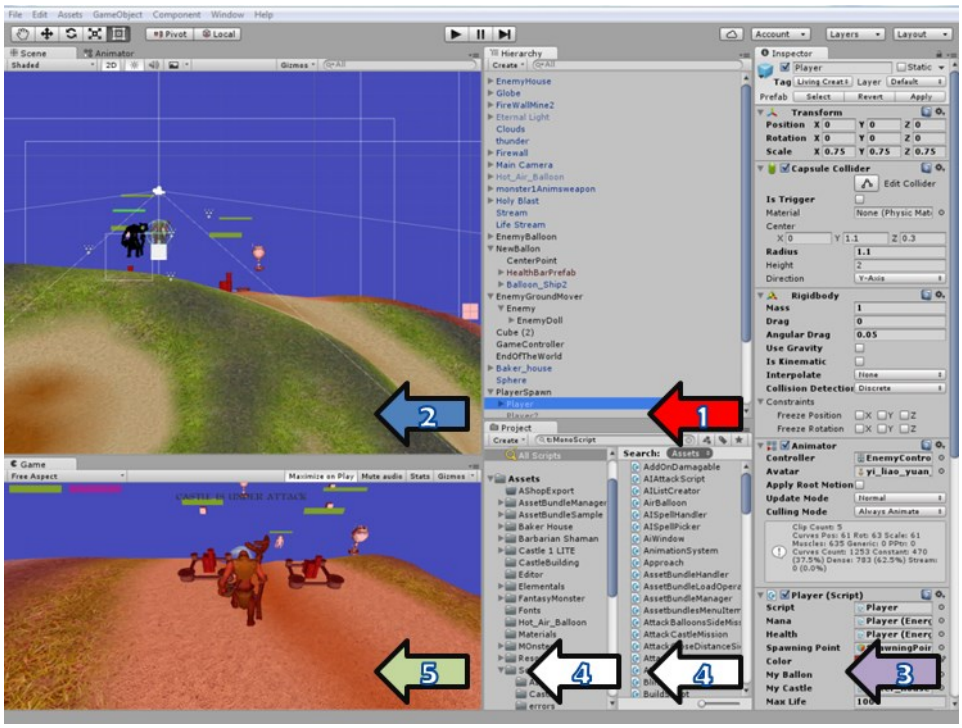
Monstra jsou hlavním zdrojem many, je tedy potřeba, aby ve hře stále nějaká byla a hráč mohl manu získat. V tuto chvíli pokud hráč zabije všechna monstra v dané oblasti, ztratí možnost získat manu. Proto jsme se rozhodli hru nastavit tak, že monstra se budou rodit nepřetržitě (až na vybrané oblasti). Tím bychom se ale vrátili zpět k problému, že bude ve hře příliš mnoho monster. Tomu je možné se vyhnout tak, že v každé oblasti stanovíme maximum živých monster, která se v ní mohou pohybovat. V okamžiku, kdy počet monster dosáhne stanoveného limitu, nová monstra se už nebudou rodit. A jakmile hráč nějaké monstrum zabije, v nejbližším rodícím období se zrodí nové monstrum, ale pouze do stanoveného počtu.

3 Popis Unity

Unity 3D je game engine vyvinutý společností Unity Technologies, který se používá pro tvorbu her na PC nebo mobilních zařízeních. Jeho první verze vyšla v roce 2005 a od té doby byla mnohokrát vylepšena. Nejnovější verzí je nyní verze Unity 5. V našem projektu využíváme Unity 5.2.

Dle diplomové práce o implementaci multiplayerové strategie v Unity je od roku 2012 Unity nejpopulárnější game engine pro tvorbu 3D aplikací pro mobilní zařízení [4].

Unity 3D mohou používat jak programátoři, tak designeři. Jeho prostředí se podobá programům na modelování, jako například Maya nebo 3D Studio Max. V Unity se nejdříve vytvoří prostor – to znamená, že se do prostředí vloží různé objekty, světla, kamera atd. Potom se jednotlivým objektům přiřadí chování pomocí skriptů. Když je aplikace hotová, programátor vybere cílovou platformu a Unity vytvoří spustitelný .exe soubor. Skripty pro Unity se mohou psát v C# nebo v Javascriptu.



[Obrázek 11 - Unity]

Na obrázku 11 můžeme vidět, jak Unity vypadá. Uprostřed horní obrazovky je seznam všech objektů ve scéně (označeno červenou šipkou s číslem 1- panel hierarchie). Vlevo nahoře je zobrazena scéna (na obrázku označena modrou šipkou s číslem 2), ve které se zmíněné objekty nachází. Kliknutím na jeden z objektů (na obrázku na objekt Player) zobrazíme všechny skripty a komponenty, které jsou k němu připojené. Ty vidíme v další části – Inspektor (fialová šipka s číslem 3).

Ve čtvrté části je seznam objektů a skriptů, které jsou k dispozici a mohou se do Unity vložit (označená dvěma bílými šipkami). Mohou to být např. objekty, které jsme stáhli z Unity obchodu (Asset Store) nebo které jsme sami vytvořili. Pátou částí je pohled hlavní kamery scény (zelená šipka).

Herní svět je v Unity 3D reprezentován scénou, která je plná objektů. Každý objekt má k sobě přiřazené `transform`, což je komponent, který určuje pozici, velikost a rotaci objektu. Kromě toho má ještě pro sebe specifické komponenty. Z těch zmíníme například collider, který definuje tvar objektu pro kolize [8]. Jednou z funkcí collideru je zajistit, aby do sebe objekty nenarážely, nebo na to alespoň upozornit. Například pokud do Unity vložíme krychli, bude mít v sobě automaticky box-collider.

3.1.1 Scéna

Scéna v Unity představuje všechny objekty, které jsou v danou chvíli aktivní nebo mohou být aktivované. Převážně se snažíme, aby v rámci scény byly přítomné všechny objekty, které spolu budou muset nějakým způsobem spolupracovat. Sem jsou všechny objekty umístěny. Scénu si můžeme představit jako různé úrovně hry nebo jednou scénou může být hlavní menu a druhou scénou samotná hra.

Přepnutí mezi scénami umožnuje přechod mezi jednotlivými částmi hry – např. ze hry do menu – nebo u her, které mají různé oblasti, například přechod do jeskyně.

Při přepnutí z jedné scény do druhé, jsou všechny objekty původní scény zničeny. To se dá upravit pro vybrané objekty.

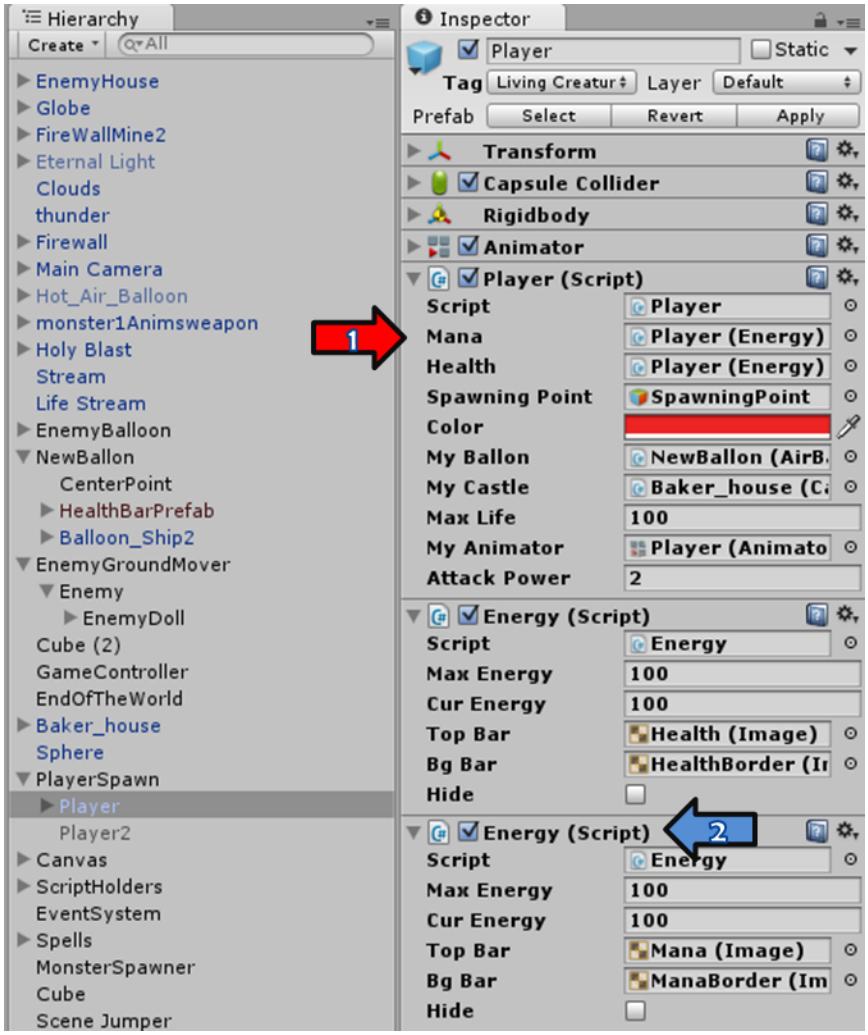
3.1.2 GameObject

`GameObject` je základním objektem v Unity. Dají se k němu připojit různé komponenty jako skripty nebo collidery. Abychom mohli tyto objekty kopírovat a nemuseli k nim znovu připojovat skripty, dá se `GameObject` uložit do takzvané prefab. (To je vlastně `GameObject` spolu se skripty, případně dalšími `GameObject`, které jsou jeho potomky). Je obvyklé si pro každý objekt ve hře vytvořit prefab.

3.1.3 Inspektor v Unity

Inspektor v Unity ukazuje všechny komponenty, které jsou připojené na objekt (viz obrázek 12). To umožňuje nejen snadno přidávat komponenty k objektům, ale také je mezi sebou propojovat.

Obrázek 12 je příkladem, kde jsou detailně zobrazeny komponenty přidané na objekt `Player`. Jeden ze skriptu připojených k objektu `Player` je skript `Player`. U něj jsou zobrazeny kolonky pro jednotlivé veřejné proměnné, jež skript obsahuje. Jednou z těchto proměnných je Mana (označeno červenou šipkou s číslem 1), která je typu `Energy` (není na obrázku poznat). Do této proměnné můžeme buď přiřadit hodnotu skriptem, nebo vzít skript `Energy` (označeno modrou šipkou s číslem 2) a přesunout ho do kolonky proměnné mana. Výhodou přesunutí je, že není třeba hledat `GameObject`, na kterém se skript nachází v kódu.



[Obrázek 12 - Inspektor]

3.1.4 Plugins

Unity má vlastní obchod AssetStore, ze kterého si můžeme stáhnout prefabs nebo pluginy (sada skriptů). V rámci této hry používáme plugin pro úpravu terénu Spherical Terrain.

3.1.5 Skripty

Skript v Unity je kód napsaný v Javascriptu nebo C#, který se připojí k objektu. Pokud se skript nemusí připojit ke konkrétnímu objektu, je potřeba vytvořit prázdný objekt, který slouží pro držení skriptu.

Skripty v Unity většinou dědí `MonoBehaviour`, protože potom ho lze připojit na objekt. Pokud skript není typu `Monobehaviour`, může být pouze přiřazen do proměnné. `Monobehaviour` dále umožňuje přetížit předpřipravené virtuální funkce – popíšeme si pět často používaných virtuálních funkcí MonoBehaviour: `Awake()`, `Start()`, `Update()`, `FixedUpdate()` a `LateUpdate()`.

Funkce `Awake()` a `Start()` se spustí při tvorbě objektu, ke kterému je skript připojen, dají se použít například k přiřazení hodnot do proměnných. Rozdíl mezi nimi je takový, že `Awake()` se vždy spustí před `Start()` a spustí se i v případě, že je objekt vypnutý (disabled), zatímco `Start()` se spustí pouze tehdy, pokud je objekt zapnutý (enabled).

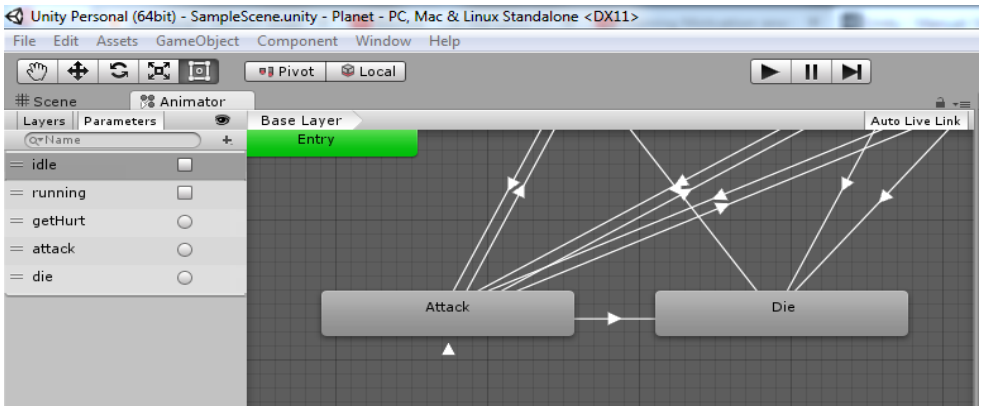
Všechny varianty Update funkcí se volají automaticky pomocí Unity v průběhu programu. Používají se převážně pro pohyb ve hře. Funkce `Update()` se volá každý frame. Funkce `FixedUpdate()` se používá, pokud má objekt v sobě `RigidBody()` (komponenta pro pohyb). Funkce `FixedUpdate()` může běžet vícekrát v rámci jednoho framu podle toho, kolikrát je to nastaveno v rámci fyzického enginu. Funkce `LateUpdate()` se volá po všech ostatních Updatech.

Pro potomky `Monobehaviour` nelze vytvořit parametrický konstruktor. To se dá poměrně snadno obejít sadou funkcí jako `Create()` nebo `Begin()`, v rámci kterých se předají dané parametry. Problém je, že mezi vytvořením objektu a přiřazením parametrů existuje prodleva. Během této prodlevy po něm může jiný objekt požadovat některý z později přiřazených parametrů. V rámci programování je třeba vzít toto chování v úvahu.

3.1.6 Animace

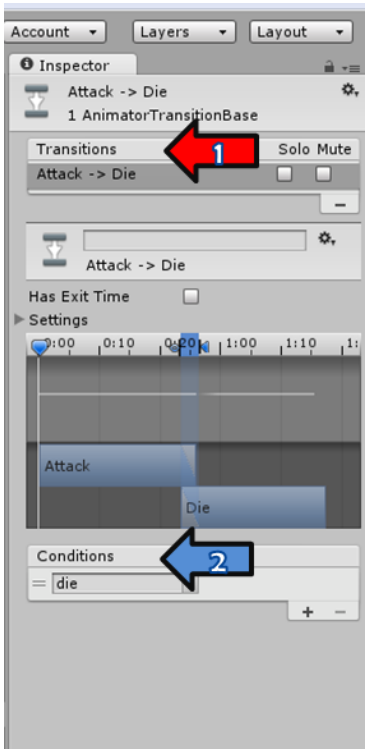
V Unity existují dva způsoby, jak lze pracovat s animacemi. Původní, starší je Legacy, druhou, novější variantou je Mecanim, který používá animátor pro práci s animacemi. K programování této hry používáme pouze Mecanim a jeho animátor. Existují další zajímavé vlastnosti Mecanimu, jako např. kopírování animací mezi podobnými objekty, ty ale v naší práci nevyužijeme, a proto se jim nebudeme podrobněji věnovat.

Pro práci s animacemi slouží v Unity (Mecanimu) Animator Controller. Funguje jako konečný automat, jehož stavy jsou animace nebo sady animací a přechody mezi stavy jsou přechody mezi animacemi. U každého přechodu je možné nastavit různé podmínky pomocí speciálních proměnných. Existují čtyři typy těchto proměnných: float, int, bool a trigger. Float a int jsou běžné číselné proměnné a přechod se u nich nastavuje například tak, že hodnota je větší nebo menší než nějaká pevná hodnota. Bool je také klasická booleovská proměnná (na obrázku 12 proměnná running a idle.) Přechod mezi animacemi se spustí, pokud proměnná je true/false a bude běžet tak dlouho, dokud bude booleovská proměnná nastavená na true/false. Posledním typem proměnné je trigger. Pokud se přepne na true, spustí se přechod, který je na něj navázán, a potom se trigger automaticky přepne zpět na false. To má za následek, že nová animace se spustí pouze jednou.



[Obrázek 13 – Animátor]

Na obrázku 13 můžeme vidět stavy `Attack` a `Die`. Vlevo na obrázku jsou proměnné.



[Obrázek 14 – Přechod animací]

Na obrázku 14 je zobrazení přechodu z animace `Attack` na animaci `Die`. Slouží k tomu trigger `die`, který můžeme vidět na obrázku 13. V tabulce Transitions (označené červenou šipkou s číslem 1) vidíme jeden přechod z `Attack` na `Die`. Podmínky tohoto přechodu jsou uvedeny v tabulce Conditions (označené modrou šipkou s číslem 2), kde je trigger `die`.

Důležitou vlastností přechodu animace je „Has Exit Time“. V případě, že je zapnutá, přechod animace počká, dokud původní animace neskončí, a teprve pak se spustí. Tento postup se používá například při animaci zranění a následného pohybu,

kdy chceme docílit, aby animace zranění proběhla celá a teprve potom postava pokračovala v běhu.

4 Implementace v rámci Unity

Třetím cílem naší práce je zhodnotit, jak se naše hra programovala v Unity a jaké výhody a nevýhody v jejím programování Unity přineslo. V této kapitole se zaměříme na to, jak jsme hru naprogramovali, ve kterých aspektech nám Unity pomohlo a kde naopak přinášelo různá úskalí nebo nevýhody.

Nejdříve pojednáme o celkové struktuře hry a poté se budeme věnovat jednotlivým objektům ve hře a tomu, jakým způsobem jsme je implementovali. Dále představíme implementaci pohybu ve hře a prostředí hry.

4.1 Struktura hry Zničte zámek

Druhou kapitolu jsme rozdělili na dílčí části podle objektů, které mají ve hře různé chovaní – např. od monstra očekáváme něco jiného než od hráče. Jsou to následující podkapitoly: Herní svět, Hráč, Nepřátelé, Kouzla, Mana, Zámek, Balon a Monstrum. Toto rozdělení se nabízí i v programování. Chceme totiž, aby každý objekt, který se svým chováním od ostatních dostatečně liší, měl vlastní třídu a funkce, jež jsou pro něj specifické.

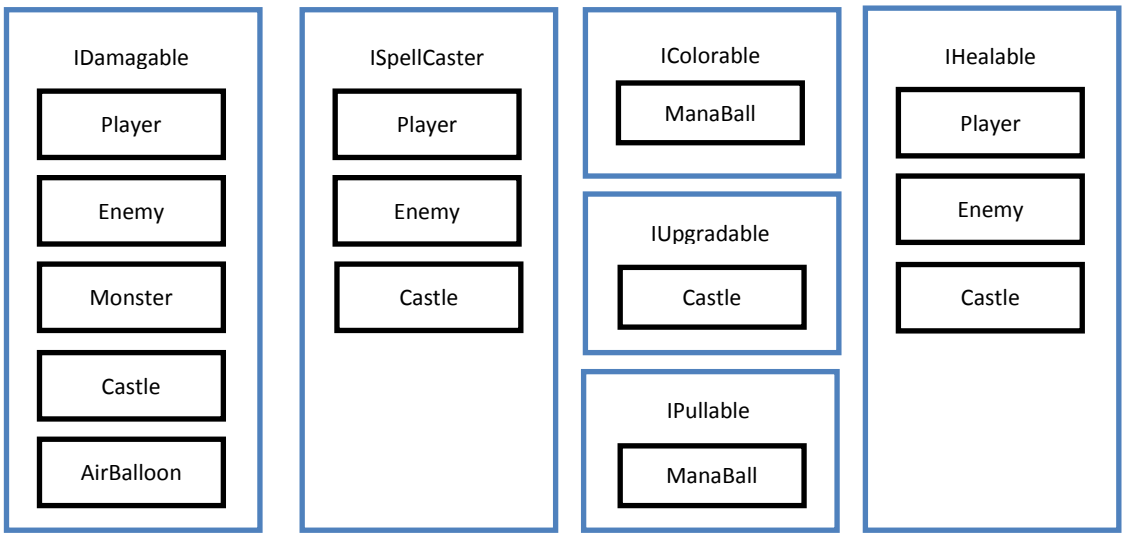
4.2 Interface

V rámci naší hry máme několik hlavních typů objektů, a to monstrum, hráč, nepřítel, kouzlo, balon a zámek. Pro komunikaci mezi těmito objekty používáme převážně interface. Níže je tabulka s hlavními interface ve hře.

`IDamagable`	Umožňuje ostatním objektům na daný objekt útočit a zranit ho.
`ISpellCaster`	Umožňuje objektu používat kouzla.
`IColorable`	Umožňuje objektu být obarven a zároveň patřit nějakému jinému objektu. Používá se pro mana balls.
`IHealable`	Umožňuje objektu být vyléčen.
`IUpgradable`	Umožňuje objektu být vylepšen. Používá se pro zámky.
`IPullable`	Umožňuje objektu být přitahován jinými objekty. Používá se pro mana balls.

[Tabulka 1 – Interface]

Na obrázku 15 jsou zobrazené hlavní interface (vyznačené modrým rámečkem) a třídy (vyznačené černým rámečkem), které je implementují.



[Obrázek 15 – Třídy a jejich interface]

Výhodou použití interface je to, že je můžeme snadno přiřadit k nějaké třídě. Například kdybychom chtěli, aby všechna monstra mohla použít kouzla, stačí, aby třída monster implementovala `ISpellCaster.`

Interface má ale v rámci Unity jednu nevýhodu – není vidět v inspektoru. Musí se tedy pokaždé přiřadit v rámci kódu. Máme například monstrum, které chce zaútočit na hráče. Je třeba mu přes kód předat `IDamagable` hráče, nestačí mu ho předat v inspektoru.

Doporučili bychom spíše nepoužívat interface a raději využít abstraktní třídy. Je ale nutné si dát pozor na to, že v C# lze dědit pouze od jedné třídy.

4.3 Implementace kouzel

V rámci implementace kouzel, bylo potřeba řešit převážně míření. Kolize detekuje samotné Unity. Stačí tedy v rámci kolize, zavolat funkci, která spustí efekt kouzla.

4.3.1 Míření

Jak jsme zmínili v kapitole 2.4, hlavním problémem při míření kouzel je předpovědět, kam hráč kliká. Vzhledem k tomu, že obrazovka hry je 2D a svět je ve 3D, je potřeba správně odhadnout hloubku třetí osy. K tomu v Unity slouží `Raycast`, což je paprsek, který se spustí na x a y souřadnici myši a postupuje ve směru osy z (která definuje hloubku). V okamžiku, kdy na nějaký objekt narazí (může být i zem), vrátí souřadnice tohoto bodu. Poté se od hráče vyšle kouzlo směrem k daným souřadnicím. Střetnutím s cílovým objektem se spustí efekt kouzla (pokud se cílový objekt shoduje s typem kouzla).

Aby hráč mohl střílet i mimo objekty, je potřeba vytvořit kontrolní plochu, o kterou se `Raycast` může zastavit a poslat zpět navádějící souřadnice. Jako kontrolní plocha by se dala použít planeta. Tato varianta nefunguje kvůli kombinaci následujících vlastností hry: hra se hraje na kouli a část hry se hraje ve vzduchu.

Kdyby se hra hrála pouze na planetě a neumožňovala by pohyb ve vzduchu, stačilo by použít jako kontrolní plochu pro `Raycast` planetu. Pokud by však hráč chtěl vystřelit do prostoru mezi cílové objekty, vyslal by se sice paprsek `Raycast`, ale ten by se neměl o co zastavit, tudíž by hra nezískala žádné souřadnice. Řešením této situace by bylo vytvořit stěnu, jež je rovnoběžná s obrazovkou hráče, která by zastavila paprsek a vrátila souřadnice jejich průsečíku. Použití stěny však zkresluje cílové souřadnice paprsku v závislosti na vzdálenosti cílového objektu od ní. Proto ani tato varianta nefunguje.

Dalším způsobem vysílání kouzel v Unity je využití schopnosti Unity přímo kliknout na objekt a detektovat jeho pozici. Po kliknutí na objekt, dá objekt vědět, kde se nachází, a pak se vyšle kouzlo z pozice hráče na pozici cílového objektu.

Monstra jsou objekty, jež se neustále pohybují, proto by je při použití této varianty téměř nikdy hráč netrefil. Monstrum se totiž přesune, zatímco kouzlo letí stále k původní pozici. Řešením je vytvořit samonaváděcí kouzla. Kouzlo by bylo informováno o svém cíli a pohybovalo by se stále v jeho směru. Aby měl cílový objekt šanci uniknout, byl by pohyb kouzla zpomalený.

Bylo by možné kliknout pouze na objekty, na které mohou kouzla působit, tedy na ty, jež implementují alespoň jeden z těchto interface: `IDamagable`, `IColorable`, `IHealable` a `IUpgradable` (to jsou všechny objekty kromě planety). Objekty reagují pouze na kouzla, s jejichž typem se shodují, i když se dá vystřelit i na objekty, které se s tímto typem neshodují – např. lze vystřelit kouzlo obarvení many na zámek, ale nemá to žádný efekt.

Pomocí této metody se nedá kouzlo vyslat mezi objekty pouze přímo na ně, přesto se tato varianta jeví jako nejlepší řešení, a proto ji použijeme.

Původní hra Magic Carpet neumožňuje hráči mířit, naopak kouzla letí automaticky k nejbližšímu agresivnímu cíli. V naší hře jsme chtěli umožnit hráči, aby si mohl sám vybrat cíl, na který bude střílet, což naše varianta umožňuje.

4.4 Implementace pohybu

V Unity existují dva hlavní způsoby, jak pohybovat objektem. Buď je možné použít `RigidBody`, nebo `Transform`. Hlavním rozdílem je, že `RigidBody` při výpočtu pohybu bere v úvahu fyziku. (Lze na něj působit silou z více směrů a pak se podle jejich kombinace pohne výsledným směrem). `Transform` umožňuje buď rovnou zadat výslednou pozici (přes `Transform.Position()`), nebo použít `Transform.Translate()`, což je funkce, která objekt pohne dopředu určitou rychlostí.

Ve hře používáme oba tyto systémy. `RigidBody` používáme kvůli gravitaci. Nejlepší ukázka toho, jak funguje, je u mana balls, které jsou na začátku vystřeleny určitým směrem a pak se pohybují zbytkovou energií, působí na ně gravitace a zároveň přitahování balonu, pokud jsou v jeho dosahu.

U ostatních objektů, jako jsou monstra nebo hráči, používáme většinou pohyb pomocí `Transform`. Ten využíváme pro čtyři způsoby pohybu: pronásledování, procházka, pohyb balonů a pohyb nepřátel.

4.4.1 Pohyb procházka

Pohyb procházky má dvě varianty. Náhodný pohyb a předem určenou trasu.

Původním plánem pro náhodný pohyb bylo náhodné otáčení monstra a pak chůze daným směrem. V praxi to ale vypadalo nepřirozeně. Další nevýhodou byla potřeba sledování, kdy monstrum dojde k hranici oblasti, a otočení, aby se nedostalo mimo oblast. Monstrum se pak chovalo nepřirozeně, protože když došlo k hranici oblasti, otočilo se směrem od ní, ale za chvíli se mohlo otočit zase zpátky k ní, takže se otáčelo stále dokola.

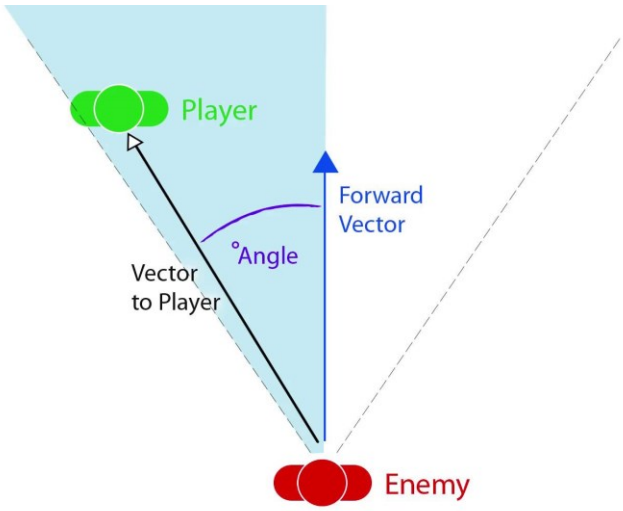
Místo toho jsme tedy použili jinou variantu, která vychází z plánu pro předem určenou trasu. Funguje tak, že si monstrum vybere destinace v rámci své oblasti a poté se pohybuje směrem k ní. Pokud své destinace dosáhlo, přejde na další v pořadí. Destinace jsou u předem určené trasy od začátku zadané. Pro náhodný pohyb se volí následující destinace náhodně, pokaždé když monstrum dorazí do cílové destinace.

4.4.2 Pohyb pronásledování

Samotná logika pronásledování je poměrně jednoduchá. Předpokládáme, že už víme, koho máme pronásledovat i jak velkou máme oblast. Potom stačí jednou za čas porovnat pozice mezi lovcem a cílem a otočit se k němu. Samotný pohyb je volání funkce, která jde stále dopředu. To je často používaná technika v Unity, kde místo aby se např. monstru řeklo, že se má pohnout určitým směrem, se pouze natočí a vždy pohne rovně.

Nyní jsme vyřešili pronásledování v případě, že monstrum ví, koho bude pronásledovat. To je však potřeba určit. Ve hře lze umožnit každému objektu, aby věděl o jakémkoli jiném objektu. Například monstra mohou získat seznam všech hráčů (což jsou objekty, které mohou pronásledovat). Z nich pak stačí vybrat toho, který je v zorném poli nejblíže. Vzhledem k tomu, jak výše uvedená technika funguje, není problém za běhu změnit cíl, a lze tedy plynule přejít z pronásledování jednoho hráče k druhému.

V rámci pronásledování může být vhodné vzít v úvahu, zda monstrum nepřítele vidí. Zorné pole si vyjádříme úhlem a pak stačí vypočítat úhel mezi oběma objekty a zjistit, zda je menší než zorné pole monstra (viz obrázek 16). Při tvorbě monstra může designer zadat zorné pole monstra. Tento úhel je dvojnásobek (fialového) úhlu na obrázku, protože reprezentuje celé zorné pole, nejen úhel od středu.



[Obrázek 16 – Zorné pole monstra (https://i.ytimg.com/vi/mBGUY7EUxXQ/maxresdefault.jpg)]

4.4.3 Vznášení se nad objektem

Jak jsme již zmínili v kapitole 2.3.1, vznášení se nad objektem je pohyb, který využívá nepřítel. Naším cílem je vytvořit objekt na zemi, který bude nepřítel následovat. V Unity lze vytvořit naváděcí objekt poměrně jednoduše. V tomto případě stačí vytvořit objekt, který je neviditelný, což je v Unity nastavitelné. Dále je třeba donutit nepřítele, aby tento objekt považoval za naváděcí objekt. K tomu je možné v Unity využít dědičnosti objektů. Nepřítel bude potomkem naváděcího objektu a bude se automaticky pohybovat jako naváděcí objekt.

Problémem mohou být kolize, kterým je nutné předejít. Kdyby totiž naváděcí objekt kolidoval s ostatními objekty, nepřítel by ve vzduchu kolidoval také (ale tam by nebylo s čím kolidovat). Abychom zajistili, že naváděcí objekt bude kolidovat se zemí, ale již s ničím jiným, vytvořili jsme skript, který ignoruje kolize s jinými objekty.

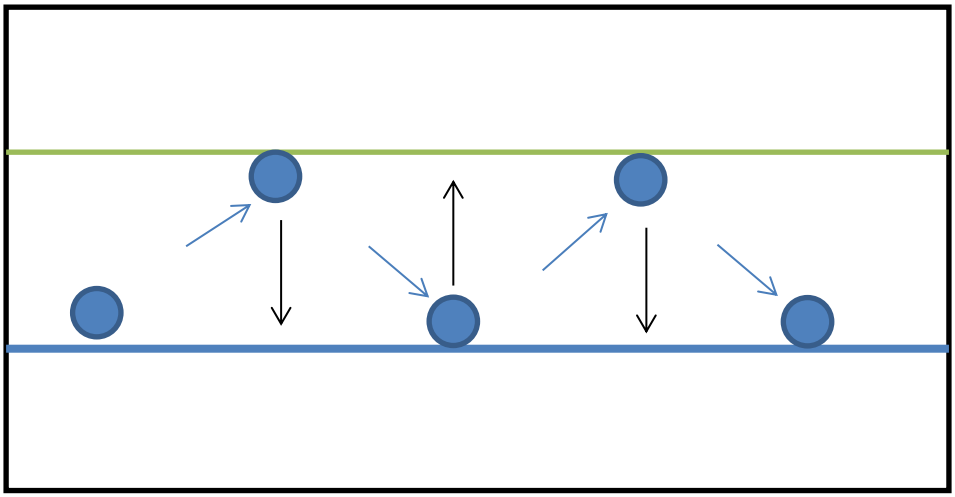
Dalším problémem je střílení kouzel přes naváděcí objekt. Jak už jsme zmínili v kapitole 4.3, střílení kouzel funguje tak, že klikneme na objekt, na který chceme poslat kouzlo. Pokud by však nepřítel stál blízko hráče, mohlo by se stát, že by chtěl hráč zaútočit na monstrum, ale naváděcí objekt by byl blíž a kouzlo by nevystřelilo. Naváděcí objekt není typu `IDamagable`, `IUpgradeable`, `IColorable` ani `IHealable`, a nemůže na něj tedy být sesláno žádné kouzlo. V Unity existuje speciální nástroj vrstvy, který je možné použít k řešení tohoto problému. Unity umožnuje vložit objekty do různých vrstev. Jedna z vrstev se jmenuje `Ignore Raycast`. Výběrem této vrstvy dosáhneme přesně požadovaného záměru – na objekt nebude možné kliknout a půjde skrze něj kliknout na jiný objekt.

4.4.4 Pohyb balonu

Pro let balonu je důležité vyřešit, jak bude balon létat v předurčených výškách, jak vyletí ze zámku a jak se zastaví nad mana balls. Nejdříve balon zjistí, kde jsou mana balls. Poté vypočítá pozici objektu, kdyby se nacházel v ideální výšce. Ta je cílovou pozicí balonu, čímž se zajistí vzlet ze zámku.

V rámci letu v předurčených výškách se balon pohybuje ve třech fázích: v ideální výšce, příliš vysoko a příliš nízko. Hraniční výšky se určí na začátku letu a v průběhu letu pak balon počítá svou aktuální výšku, kterou s těmito dvěma hodnotami porovná. Pokaždé, když se dostane do jednoho extrému (příliš vysoko/nízko), přidá se jeho `RigidBody` síla, která ho pošle nahoru/dolů. Výsledkem bude, že balon poletí v sinusoidě.

Popsanou situaci můžeme vidět na obrázku níže – zelená čára symbolizuje maximální výšku, modrá čára minimální výšku a pokaždé, když balon dosáhne jedné z těchto výšek, se k jeho původnímu směru přidá nový směr, jak naznačuje černá šipka.



[Obrázek 17 – Let balonu]

4.4.5 Pohyb hráče

Unity je pro ovládání pohybu hráče částečně předpřipravené. Automaticky detekuje, zda hráč používá k pohybu šipky nebo kombinaci WASD. Slouží k tomu funkce `Input.GetAxis()` pro vertikální a horizontální osu, která vrací hodnoty -1, 0, 1 podle toho, jaké tlačítko hráč stiskl, např. pro vertikální osu hodnota 1 znamená, že hráč stiskl šipku nahoru či tlačítko W.

Původně bylo naplánováno, pro pohyb použít, jak klávesnici, tak myš, ale protože myš se již používá na míření kouzel, tak jí nejde využít. Pohyb proto funguje tak, že šipky doleva a doprava ovládají rotaci, šipka nahoru určuje směr dopředu a nahoru a šipka dozadu určuje směr dozadu a dolů, analogicky pro kombinaci WASD.

Pokud je postavička hráče ve vzduchu a hráč krátce zmáčkne tlačítko dolů/D, postavička se pomalu snese k zemi.

4.5 Implementace prostředí

V kapitole 2.1 jsme si vybrali planetu jako prostředí, ve kterém se hra bude odehrávat. To nám ale přináší několik problémů, jako je např. vlastní gravitace, rotace objektů a udržování objektů ve výšce. V následujících podkapitolách se zaměříme na to, jak jsme tyto problémy vyřešili.

4.5.1 Gravitace

Unity umožňuje gravitaci, ale pouze takovou, která objekt přitahuje plošně k zemi. Musíme si ji tedy naprogramovat sami. K tomu nám pomůže komponent pro pohyb `RigidBody` (viz kapitola 4.4), na který je možné aplikovat sílu z určitého směru. Směr, kterým se nakonec `RigidBody` posune, se vypočítá součtem všech sil, které na něj působí.

Gravitaci vytvoříme tak, že při každém updatu přidáme sílu, jež stahuje `RigidBody` směrem k centru planety. To uděláme výpočtem vektoru od objektu do středu planety a v jeho směru aplikujeme sílu gravitace.

4.5.2 Rotace objektů

Kromě gravitace je potřeba vyřešit i rotaci, aby byly objekty vždy kolmo k centru země (viz kapitola 2.1). Při každém updatu vypočítáme, jaká rotace je pro danou pozici kolmo k zemi., a následně změníme rotaci objektu. Problém spočívá v tom, že rotaci mění nejen skript týkající se gravitace, ale také např. skript, který objektem pohybuje (např. monstra se musí otáčet podle toho, kde je nepřítel). Aby se neotáčela hlavou dolů nebo např. šikmo nahoru, pokud bude nepřítel mírně nad nimi, je vhodné vypnout rotaci (freeze rotation) `RigidBody.` Zmrazíme ji v některých osách a monstrum se bude moci otáčet např. pouze doleva a doprava.

U mana balls chceme docílit, aby svým pohybem kopírovaly terén. Jejich pohyb je naprogramován tak, že se stále pohybují pouze dopředu, to znamená, že pokud např. narazí do kopce, otočí se, a tím pak sjedou z kopce. Zde proto nesmí být vypnutá rotace, protože by mana balls nereagovaly na změnu terénu.

Vytvořili jsme tedy obě varianty pro obě využití.

4.5.3 Udržení objektů ve výšce

Udržení ve výšce je v rámci Unity implementováno poměrně jednoduchým způsobem – stačí spočítat, v jaké výšce se objekt nachází. To je možné provést buď tak, že se odečte `radius` od vzdálenosti od středu země (nevýhodou je, že se přitom nebere v úvahu tvar povrchu). V reálném životě bychom si to mohli představit jako nadmořskou výšku. Pokud má nějaká hora vyšší nadmořskou výšku, objekt do ní stejně narazí. Druhou variantou je mít objekt na zemi a počítat vzdálenost od něj (můžeme to nazvat nad-objektová výška). Pokud tedy bude objekt na hoře, která má „nadmořskou“ výšku 100 jednotek, a druhý objekt bude nad ním ještě o 10 jednotek

výš, má „nadmořskou výšku“ 110 jednotek, ale nad-objektovou výšku pouze 10 jednotek.

V rámci programování používáme obě dvě varianty. „Nadmořskou výšku“ u létání balonu a výšku nad objektem u pohybu nepřátel.

4.6 Implementace monster

Prvním aspektem, který bylo potřeba při tvorbě monster naprogramovat, byl způsob jejich pohybu, o kterém jsme pojednali v kapitolách 4.4.1 a 4.4.2.

V následujících dvou podkapitolách se budeme zabývat bojem a animací monster.

4.6.1 Boj

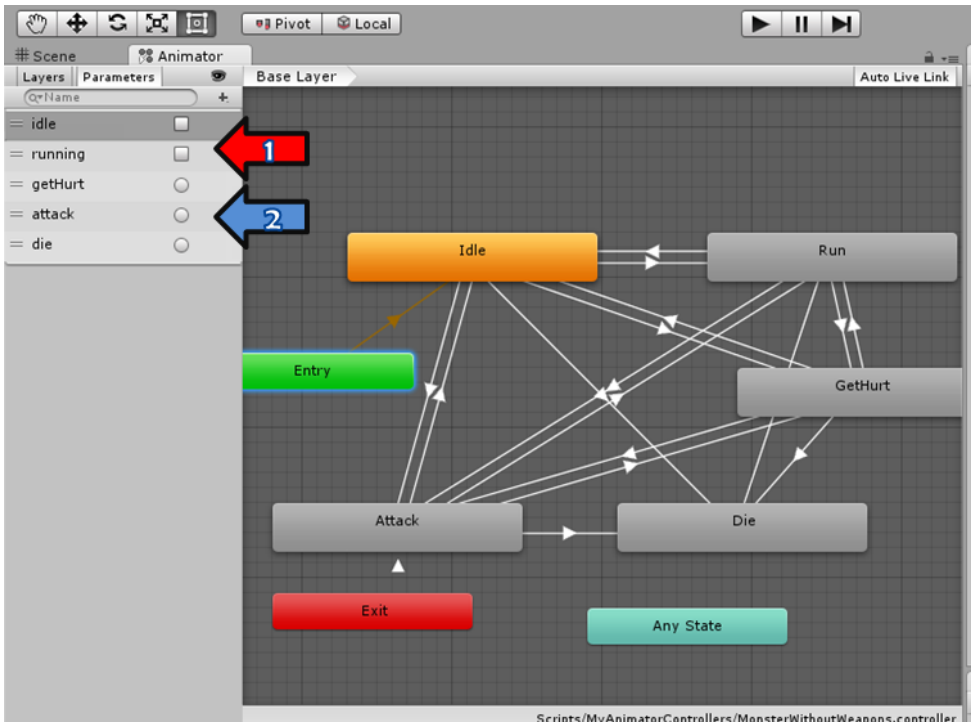
Monstra musí umět bojovat (viz kapitola 2.8.1.). Toho dosáhneme tak, že když se dostanou do blízkosti nepřítele/hráče, spustíme animaci útoku a informují cíl, že na něj zaútočili. Protože se ale předem neví, kdo přesně cíl je (zda je to hráč nebo nepřítel), použije se k tomu rozhraní `IDamagable`. To bude mít funkci `GetHurt()`, která se může zavolat, když je potřeba na objekt zaútočit. Zároveň zůstává na objektu, zda je schopen se ubránit (a tedy si ubere život) nebo ne (více k obraně proti útoku v kapitole 5.3.3).

Monstra také implementují `IDamagable`, aby na ně mohli hráči útočit.

4.6.2 Animace monster

Jak jsme již popsali v kapitole 3.1.6, v Unity slouží k animacím animátor, což je konečný automat animací. Pro tvorbu našich animací jsme vytvořili pět hlavních stavů: `Idle`, `Run`, `GetHurt`, `Die` a `Attack`.

Kromě těchto pěti stavů nám Unity přidá stavy `Entry`, `Exit` a `Any State` (jak je vidět na obrázku 18). Všechny objekty začínají ve stavu `Entry`. Pro spuštění nějaké animace je potřeba, aby existoval přechod ze stavu `Entry` do této animace. Přechod do animace `Exit` umožňuje ukončení animací. Alternativně se dají animace ukončit tak, že se přejde do stavu, ze kterého neexistuje další přechod. Tento způsob používáme v našem projektu. Stav `Any State` umožňuje vytvořit přechod z jakéhokoli stavu do konkrétního stavu. V našem projektu tento stav neuplatňujeme.



[Obrázek 18 – Animátor Monstra]

Pro přechody používáme pět hlavních proměnných `idle`, `running`, `getHurt`, `attack` a `die`. Proměnné `idle` a `running` jsou booleovské, všechny ostatní jsou typu trigger. To je důležité rozlišit při použití animací, které se mají spustit jen jednou, a animací, které se mají pouštět nepřetržitě, dokud je něco nepřeruší. Trigger používáme u animace, kterou chceme spustit pouze jednou. Například animaci `GetHurt` chceme spustit jednou, a to pokaždé, když se monstrum zraní. Na to použijeme trigger `getHurt`. Na druhou stranu animaci `Run` chceme spouštět nepřetržitě, dokud monstrum běží. V tomto případě stačí použít booleovskou proměnnou a nastavit přes ni přechod na stav `Run`. Unity bude automaticky pouštět animaci `Run`, dokud je proměnná `running` pravdivá a nenastal přechod do jiného stavu.

4.7 Implementace umělé inteligence

V předchozích kapitolách 2.3.2 jsme popsali, jak by se měla chovat umělá inteligence. Zmínili jsme, že bychom chtěli vytvořit mise, které bude nepřítel plnit. Tyto mise budou mít různou prioritu a nepřítel si vždy vybere tu s nejvyšší prioritou. Priorita misí je pevná, ale to, zda je daná mise k dispozici nebo ne, se mění v průběhu hry – např. mise vyléčení není potřebná, když má nepřítel plné zdraví. Mise tedy musí informovat o tom, jakou má prioritu, zda může začít a zda je splněná.

Výběr mise s nejvyšší prioritou se zajistí tak, že se ze seznamu misí vybere podmnožina misí, které jsou schopné se v tento okamžik spustit (vyhovují jejich startovací podmínky), a poté se seřadí dle nejvyšší priority a vybere se ta nejvyšší. Tato varianta by fungovala, ale problémem by bylo, že by bylo potřeba opakovaně

hry třídit seznam misí. To je zbytečné vzhledem k tomu, že mise priorit se nemění, pokud se nemění `Approach` (více podrobností k tomu v kapitole 4.7.2). Je tedy výhodnější seřadit mise hned na začátku a pak pouze procházet seznam od té s nejvyšší prioritou a zjišťovat, zda se může spustit.

Může se stát, že nepřítel bude vykonávat nějakou misi a přitom může začít jiná mise s vyšší prioritou. V tom případě by se měl nepřítel přepnout na misi s vyšší prioritou.

Vzhledem k tomu, že vykonání nějaké mise trvá určitou dobu (obvykle v řádu desítek sekund), není třeba sledovat možnost přepnutí se na novou misi každou milisekundu. Stačí mise s vyšší prioritou procházet vždy po několika sekundách.

4.7.1 Vedlejší mise

Kromě normálních misí musí nepřítel plnit i vedlejší mise (jako např. útok na monstrum, pokud je nablízku). Ty nás také musí informovat o tom, kdy mohu být spuštěny. Nemají ale prioritu, takže pokud bude k dispozici více vedlejších misí současně, vybereme si jednu náhodně. V případě, že by výběr vedlejších misí nastával příliš často, mohlo by se stát, že by nepřítel pokaždé vybral jinou misi a stále by mezi nimi přepínal a vlastně by v rámci žádné mise nic neudělal. Proto je třeba každou misi nechat nějakou dobu běžet, než se spustí nová mise.

4.7.2 Approach

Nepřátelé ve hře by měli být různorodí. Měli bychom mít agresivní nepřátele, kteří často útočí, a jiné naopak defenzivní. K tomu, abychom odlišili různé typy nepřátel, nám ve hře bude sloužit `Approach`. Třída `Approach` nám umožní změnit prioritu jednotlivých misí (například ofenzivní nepřítel bude mít prioritu útoku na hráčův zámek 5 a léčení své postavy 3, u defenzivního to bude naopak). Dále umožní nastavit jednotlivým misím, zda bude možné v jejich průběhu plnit vedlejší mise nebo ne. `Approach` bude moci u libovolné mise přepsat její chování nebo vlastnosti jejího spuštění a splnění. Umožní i vlastní způsob výběru vedlejších misí. Kromě misí bude možné také ve třídě `Approach` upravit, jak bude nepřítel vybírat kouzla. Implicitní způsob výběru kouzel je popsán v kapitole 4.7.3, ale hra je rozšiřitelná na mnoho jiných způsobů. Třída `Approach` bude rozšiřitelná na to, aby bylo možné měnit `Approach` nepřítele v průběhu hry.

4.7.3 Výběr kouzel

Nepřítel má k dispozici různá kouzla, která může ve hře použít. Chtěli bychom, aby se choval realisticky (vybíral pouze kouzla, na která má manu, a taková, která jsou adekvátní tomu, kolik života chceme ubrat). Použijeme tedy techniku, jež vybírá pouze z kouzel, na které má nepřítel manu.

Nepřítel se zpravidla snaží vybrat kouzlo, jehož poškození je co nejblíže k životu cílů. To znamená, že pokud cíl disponuje větším množstvím života, nepřítel

vybere kouzlo s největším poškozením. Pokud však cíl má života málo, např. jen hodnotu 5, budeme se snažit vybrat kouzlo, jehož poškození je také pouze 5.

Nepřítel vybere nové kouzlo místo původního, pouze pokud nové kouzlo je blíže k ideálnímu poškození a odebere méně many než původní kouzlo.

4.8 Balon

V kapitole 4.4.4 jsme popsali, jak funguje pohyb balonu. Nezmínili jsme ale, jak balon přitahuje manu. Chtěli bychom, aby na mana balls mohlo působit více balonů současně. K tomu se nabízí použití `RigidBody`. Zároveň jsme se rozhodli, že nechceme, aby se balony snesly k zemi, když přiletí nad mana balls, ale zůstaly ve vzduchu.

Je tedy potřeba na dálku informovat mana balls o tom, že balon je nad nimi, a přitáhnout je. Jednou možností je každému mana ball poslat signál (zavolat funkci), že se balon nachází nad ním. Jednoduché řešení by bylo spočítat ke každému mana balls vzdálenost od balonu a podle toho zjistit, zda je balon nad ním. Unity však nabízí výhodnější řešení, a to box-collider. Collidery nám v Unity umožňují detekovat kolize. Stačí tedy v okamžiku doteku collideru s mana balls aplikovat na jejich `RigidBody` sílu přitahování balonu. Ta musí soupeřit s ostatními balony a s gravitací. Čím blíže je mana balls balonu, tím silněji síla působí.

Collider ale běžně funguje tak, že když s ním nějaký objekt přijde do kontaktu, automaticky ustoupí, protože primární funkcí colliderů je, aby do sebe objekty nenarážely. To se dá obejít pomocí takzvaných triggerů. Ty umožňují zabránit automatické reakci při kolizi. Objekty neustoupí, ale místo toho se zavolá funkce `OnTriggerEnter()`, v rámci které může programátor napsat, co by se mělo dělat při střetnutí. My jsme do této funkce naprogramovali logiku přitahování mana ball. Kromě toho máme na balonu ještě klasický collider, aby jím objekty neprocházely.

Co se týče výběru mana balls, které bude balon přitahovat, vybere si balon ty, které jsou k němu nejblíže. V rámci letu za cílovým mana ball bude balon kontrolovat, zda není jiný mana ball blíže.

4.9 Implementace editoru

V předchozích kapitolách jsme již naznačili, které vlastnosti objektů by měly být v editoru upravitelné. V této kapitole si všechny vlastnosti shrneme a popíšeme si jejich implementaci. V menu hry budeme mít odkaz do editoru, který má čtyři hlavní části: editor monster, editor kouzel, editor umělé inteligence (to vše spadá pod editaci objektů) a nastavení hry.

Editor objektů nám umožní vytvořit ve hře nové objekty, a to konkrétně kouzla, monstra a nové nepřátele.

4.9.1 Tvorba monster

Tvorbu monster je potřeba rozdělit na dvě části: tvorbu samotného monstra a tvorbu spawneru.

Pro tvorbu monster jsme v Unity vytvořili speciální scénu, ve které umožníme hráči zadat vlastnosti monstra, konkrétně poškození, život, počet mana balls při úmrtí monstra, obranu a rychlost. Dále hráč vloží objekt monstra. V rámci tvorby jsme se rozhodli, že každé monstrum již v sobě musí obsahovat animace. Unity sice umožnuje přidat objektům animace dle jiného objektu, ale funguje to pouze pro objekty podobného vzhledu. Máme například kostlivce a člověka. Unity umožňuje přenést animace kostlivce na objekt člověka a naopak. Protože ale nevíme, jak vypadá monstrum, které si hráč nahrává do hry, nemůžeme kopírovat vlastní animace. Co kdybychom se např. snažili nakopírovat animaci kostlivce na draka?

Očekáváme, že monstrum designera bude mít na sobě animátor s pěti proměnnými, jež jsme zmínili v kapitole 4.6.2. Tyto proměnné jsou v kódu upravovány a na jejich základě se spouští animace monster.

Poté designer vybere, jakým způsobem bude monstrum pronásledovat nepřítele. Tedy pokud bude zapnutý `chaselock`, z jaké vzdálenosti začne nepřítele pronásledovat a z jaké útočit. Poslední vlastností je zorné pole nepřítele.

Po zadání těchto vlastností hra vytvoří nový objekt monstrum (dle objektu s animacemi) a přiřadí mu všechny skripty potřebné pro monstrum. Těm se upraví požadované vlastnosti. Tento objekt bude poté uložen jako prefab a vznikne tím prototyp monstra.

Je zde ale několik vlastností, které ještě monstru nebyly přiřazeny – např. to, zda se bude pohybovat po předem určené cestě nebo náhodně. Tyto vlastnosti se nedají přiřadit v tuto chvíli, protože nejsou závislé na monstru, ale na spawneru. Stejný typ monster se může jednou pohybovat po předem dané cestě a jindy náhodně.

Spawner

V rámci nastavení hry si designer vytvoří spawnery monster. Designer nejdřív vybere místo na planetě, které bude centrem spawneru. Poté se objeví okno, kde může upravit několik vlastností spawneru. Kromě běžných vlastností jako radius, pro které stačí vytvořit textové pole, je třeba umožnit hráči vybrat typ monstra. Na planetě se tedy objeví vzor každého typu.

Dále designer určí maximální počet monster ve spawneru, interval, v jakém se budou rodit, a zadá takzvaný `randomizer`. To je procento, s jakou pravděpodobností bude zrození monstra opožděno. Má na starost zajistit, aby se monstra rodila plynuleji, tedy ne přesně každé např. 3 sekundy, ale o něco dříve nebo později.

Rození monster implementujeme tak, že při vypršení intervalu rození vytvoříme seznam časů monster, které se mají zrodit. Pokud je `randomizer` vypnutý (nastavený na nulu), jsou všechny časy 0 a všechna monstra se zrodí okamžitě. Pokud ne, bude každému monstru vygenerováno zpoždění, poté se časy seřadí a pokaždé, když nějaký čas vyprší, se zrodí monstrum.

Dále si designer může vybrat, kolik monster se bude rodit současně. Pokud designer zvolí např. pět monster najednou, ale do maximálního počtu zbývají jenom tři, zrodí se pouze tři monstra. Designer si může zvolit, zda se nová monstra budou rodit, pouze když ještě nějaké monstrum v oblasti existuje. Potom může rozhodnout, zda monstra budou útočit na hráče i nevyprovokovaná. Poslední vlastností spawneru je určení cesty, po které se monstra budou pohybovat. Ta nemusí být zvolená vůbec – v takovém případě se monstra budou pohybovat náhodně.

Pokud chce designer umístit více různých monster na jedno místo, stačí vytvořit více spawnerů přes sebe.

4.9.2 Tvorba nepřátel

Ve hře lze také vytvořit nový typ protihráčů. Ty se tvoří na nové obrazovce a lze u nich nastavit život, mana, barvu a poškození bez kouzel. V rámci tvorby nepřátel je také možné vytvořit nový `Approach` (viz kapitola 4.7) a přiřadit ho k danému nepříteli. Při tvorbě Approach je možné změnit prioritu základních misí, to, zda umožňují vedlejší misi nebo ne, a některé jejich požadavky pro spuštění mise.

Barvy se přiřazují z předpřipraveného seznamu, aby byly navzájem dostatečně odlišné. Proto je počet hráčů omezen na počet barev v seznamu (8 barev).

4.9.3 Tvorba kouzel

Kouzla se podobně jako monstra tvoří na speciální obrazovce. Kouzlo se ve hře skládá z jeho vlastností (poškození, typ, apod.), ikony a efektu. Nejdříve necháme designera vybrat typ pohybu kouzla (létající/na místě) a typ útoku (jen při prvním doteku/opakovaně/jed). Tato kombinace vytvoří jeden z typů kouzel. Každý typ kouzla požaduje trochu jiné vlastnosti. Proto je potřeba vytvořit různá textová pole a ukazovat pouze ta, jež se shodují s typem kouzla, např. pro jedovatá kouzla: poškození kouzla, poškození jedu, jak často jed působí atd. To je v Unity možné udělat např. tak, že tlačítka budou předem připravená, ale schovaná, a pokud designer zvolí vhodnou kombinaci, přesunou se do popředí. Třída, která tvoří kouzla, má přístup ke všem tlačítkům a podle typu zjistí, která jsou aktivní a která ne.

Dále by měl designer zadat adresu ikony kouzla. Pokud se hře nepodaří ikonku načíst, měla by vytvořit nějakou implicitní. Potom designer zadá adresu objektu, který bude reprezentovat toto kouzlo v herním světě. Je samozřejmé, že by měl editor designera informovat, pokud se mu něco nepodaří načíst nebo pokud designer nezadal nějaké důležité informace.

4.9.4 Načtení nového objektu ze souborového systému

Jak v tvorbě kouzel, tak v tvorbě monster je potřeba načítat objekty designera za běhu hry. U kouzel to jsou animace kouzel a ikonky, zatímco u tvorby monstra je to přímo objekt monstra.

Unity nabízí několik možností, jak do něj za běhu vložit nové objekty a soubory. Prvním způsobem je vložení textury, která funguje tak, že stačí zadat

úplnou adresu textury a Unity ji načte. Nevýhodou je, že je to možné provádět pouze s obrázky (tedy ikonami) a ne například s particle systémy (kouzla) nebo Unity objekty (monstra). Druhým způsobem je použití složky Resources, do které se v průběhu tvorby hry vloží objekty, které se později mohou do hry stáhnout pomocí funkce `Resources.Load()`. V rámci načtení souboru si můžeme zvolit, jakého bude typu (což se samozřejmě musí shodovat s opravdovým typem objektu).

Do složky Resources je potřeba vložit objekt přes Unity inspektor v průběhu tvorby hry. V okamžiku, kdy je hra zkompilovaná se již do složky Resources nedá nic přidat.

Třetím způsobem vkládání nových objektů je použití Asset Bundles, což jsou balíčky objektů, které se do hry dají stáhnout za běhu hry. Nejprve je potřeba objekty z Unity speciálně exportovat a označit je jako Asset Bundle. Tím vznikne balíček s objekty, který se poté uloží na vzdálený server (pomocí předpřipravených skriptů). V rámci naší hry však nechceme použít vzdálený server, ale načíst soubor z disku. To se nám pomocí Asset Bundles nepodařilo, proto jsme zvolili variantu se složkou Resources.

Způsob, jak do hry vložit nový objekt (monstrum či kouzlo), je popsaný v kapitole 7.3.8.

4.9.5 Přenos objektu mezi scénami

V rámci tvorby editoru je potřeba přenášet objekty mezi scénami, např. při přechodu ze scény, kde se vytváří monstrum do nové herní scény, je nutné, aby vytvořené monstrum přežilo změnu scény. Implicitně je v Unity pravidlem, že se při změně scény všechny objekty předchozí scény zničí. K tomu existuje v Unity funkce `DontDestroyOnLoad()`. Tu je třeba zavolat v libovolném skriptu, jež je připojen na objekt, který má přežít transformaci scény.

Abychom se v následující scéně mohli odkázat na tento objekt a dále s ním pracovat, musíme ho nejdříve najít. K tomu existuje v Unity také funkce – `GameObject.Find()`. Je potřeba zkontrolovat, jestli vůbec takový objekt existuje, protože jsme mohli do této scény přejít i bez tvorby tohoto objektu.

5 Vývojová dokumentace hry

V této kapitole se budeme zabývat několika zajímavými třídami a jejich implementací. V kapitole 4 jsme pojednali o implementaci herních prvků s ohledem na Unity. V této kapitole se budeme věnovat konkrétním třídám, jejich účelu a propojení s ostatními třídami. Nebudeme zde zmiňovat všechny třídy, pouze ty, které implementují nějaký zajímavý nebo komplikovanější koncept, který není jasný z komentářů. Kromě této vývojové dokumentace bude v příloze vygenerovaná vývojová dokumentace.

Každá třída je naprogramovaná ve vlastním souboru.

Hra je rozdělena do 10 scén, jak je vyznačeno v následující tabulce.

Scéna	Popis
MainMenu	Scéna zobrazující hlavní menu. Tato scéna se objeví jako první, při zapnutí hry.
Editor	Scéna zobrazující hlavní menu editoru.
MonsterCreation	Scéna pro tvorbu nových monster.
spellcreationmenu	Scéna ve, které se tvoří kouzla.
SphericalTerrain/Scenes/SampleScene	Hlavní scéna hry. V této scéně se hra hraje. V této scéně se také do hry vkládají zámky a spawnery.
pauseScene	Scéna zobrazující pozastavenou hru.
GameOverScene	Scéna, jež se zobrazí, když hráč prohraje.
WinningScene	Scéna, jež se zobrazí, když hráč vyhraje.
InstructionScene	Scéna, jež popisuje, jak se hra ovládá.
AIList	Scéna, pro tvorbu umělé inteligence.
castlecreationscene	Scéna, pro tvorbu úrovní zámků.

[Tabulka 2 – Scény hry]

5.1 Gravitace

Pro gravitaci a rotaci objektů kolmo k zemi používáme dvojici skriptů `FauxGravityBody` a `FauxGravityAttractor.` Tyto skripty jsme naprogramovali dle tutoriálu o gravitaci v Unity (https://www.youtube.com/watch?v=gHeQ8Hr92P4).

`FauxGravityBody` se připojuje na objekty, na které má gravitace působit. `FauxGravityAttractor` se připoje na planetu. Ve `FauxGraavityBody` se určuje, zda se bude aplikovat gravitace i rotace nebo pouze jedna z nich. `FauxGravityBody` opakovaně ve své `Update` funkci volá skript `FauxBodyAttractor`.

`FauxGravityAttractor` nejdříve spočítá vektor kolmosti k planetě v bodě objektu (normalizovaný vektor od středu planety k objektu). Poté spočítá, kde je pro objekt poloha nahoře. Následně skript objekt rotuje směrem od své původní rotace

k nové rotaci, čímž docílí, že bude objekt stát kolmo k zemi. Ještě je potřeba přitáhnout objekt k zemskému povrchu. K tomu se použije `RigidBody` tak, že se na něj aplikuje síla ve směru rotace.

Skript FauxGravityBody zmrazí rotaci `RigidBody` (viz kapitola 4.5.2). To znamená, že v rámci simulace (např. jak objekt bude narážet do objektů) není možné rotovat `RigidBody`.

Pro mana balls je důležitý povrch terénu. Pohybují se totiž pouze zbytkovou silou a dle prostředí. Když tedy narazí na kopec, posunou se o něco výš, ale pak by se měly otočit a klesnout opět dolů. Se zmrazeným `RigidBody` to ale není možné. Proto máme pro mana balls speciální skript `FauxGravityBodyNonFreeze`, který zajišťuje, že se mohou volně pohybovat dle terénu. V ostatních aspektech funguje úplně stejně jako `FauxGravityBody`.

5.2 Umělá inteligence

V předchozích kapitolách 2.3.2 a 4.7. jsme popsali, jak by měla fungovat umělá inteligence. V této kapitole se zaměříme na to, jak jsme dané požadavky implementovali, jaké třídy k tomu používáme a co je v rámci umělé inteligence rozšiřitelné a upravitelné.

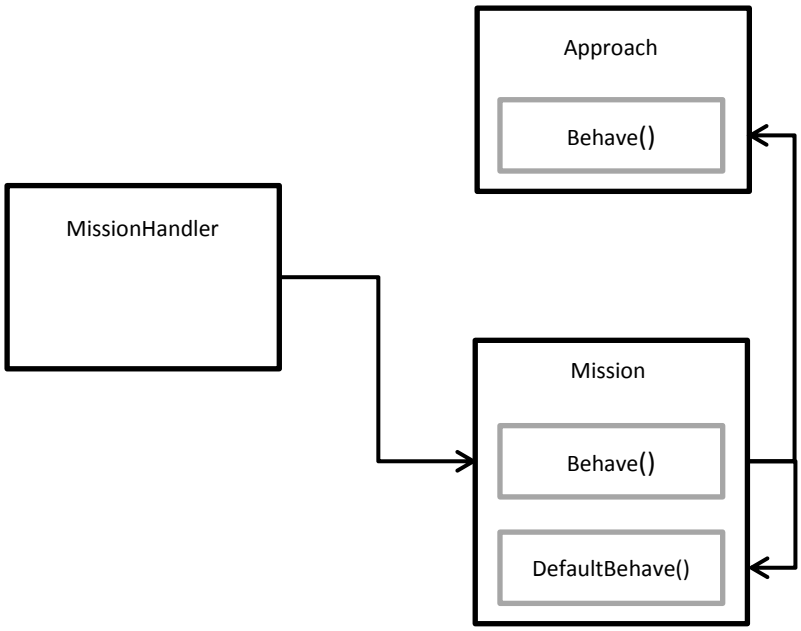
5.2.1 Třída Mission

V kapitole 4.7 jsme zmínili, že mise musí podat zprávu o tom, kdy je splněná, kdy se může spustit a jakou má prioritu. Priorita je jí určená na začátku. Zajímavé je, jak mise určí, kdy se může spustit. Každá funkce má vlastní seznam objektů, ke kterým potřebuje kontakt, a u nich zjišťuje potřebné informace o tom, zda se může spustit.

Ve hře je pět základních misí: vyléčit se (`Heal`), získání many (`GetMana`), útok na hráče (`Attack`), ochrana zámku (`ProtectCastle`) a útok na zámek (`AttackCastleMission`).

Lze dopsat další mise, pokud programátor vytvoří vlastní `Approach`, který bude mít tyto mise v seznamu vlastních misí. Všechny mise musí dědit třídu `Mission`, která je abstraktní. Tato třída implementuje interface `IMission`, který má funkce `CanStart()`, `IsFullfilled()`, `Behave()`.

Každá tato funkce je implementovaná tak, že pokud má `Approach` pro tuto misi nějaké speciální chování, spustí ho, a pokud ne, zavolá funkce `DefaultCanStart()`, `DefaultBehave() a DefaultISFullfilled()`. Pro funkci `Behave()`, je to znázorněno na obrázku 19. `MissionHandler` je třída, která volá funkce misí (v případě na obrázku `Behave()`). Mise potom zjistí zda Approach tuto funkci přetížil, pokud ano zavolá funkci třídy `Approach`, pokud ne, zavolá defaultní verzi. Pokud bude programátor vytvářet novou misi, neměl by přetížit funkci `CanStart()`, ale `DefaultCanStart()` (stejně tak i u ostatních funkcí).



[Obrázek 19 – Volání funkcí mise]

Vedlejší mise také dědí třídu `Mission`. Rozdíl je pouze v tom, že je `Approach` vrátí v seznamu vedlejších misí, a tedy budou vybírány jiným způsobem.

5.2.2 Výběr mise

Pro výběr mise používáme třídu `MissionHandler`. Ta má přiřazený `Approach`.

`MissionHandler` v pravidelných časových intervalech testuje, zda nenajde misi z předem připravených misí, která by měla větší prioritu než současná mise. Pokud ano, zastaví běh původní mise a spustí misi novou. Pokud ne, pokračuje v předchozí misi. Stejně tak bude hledat novou misi, pokud původní mise skončila nebo byla opuštěna. Pokud není k dispozici žádná nová mise, `MissionHandler` použije implicitní misi svého přístupu.

Stejně tak v pravidelných časových intervalech `MissionHandler` testuje, zda by mohl přejít na novou vedlejší misi. Nejdříve vždy otestuje, zda vůbec daná hlavní mise povoluje vedlejší mise. Pokud ano, tak vybere vedlejší misi a tu spustí.

Výběr vedlejší mise funguje náhodně, případně je možné naprogramovat jiný způsob volby vedlejší mise v třídě Approach. `MissionHandler` automaticky upřednostňuje výběr naprogramovaný v rámci třídy `Approach`, pokud takový výběr existuje. Vlastní výběr vedlejší mise je ve třídě `Approach` nutné označit pomocí booleovské proměnné `hasPickSideGoal`.

Aby `Missionhandler` nemusel procházet všechny mise, seřadí si je předem dle priority a prochází pouze ty, které mají větší prioritu než současná mise. Toto seřazení proběhne při spuštění hry nebo při každé změně přístupu.

Vedlejší mise by měla působit dojmem, že běží paralelně s hlavní misí. To by ale nemuselo fungovat, protože pokud by obě mise chtěly pohnout nepřítele jiným

směrem, nastal by konflikt. Proto používáme `counter` (implicitně nastavený na 5), který každé páté spuštění Update funkce spustí chování vedlejší mise, v ostatních případech spustí hlavní misi. Tím je zajištěno, že nepřítel vykonává převážně hlavní misi a jen občas vedlejší.

5.3 Monstrum

Monstrum je ve hře implementováno jako objekt, který má na sobě připojenou sadu skriptů. Prvním skriptem je třída `Monster` (viz kapitola 5.3.1) a skripty týkající se pohybu (viz kapitola 5.3.2), skript gravitace (viz kapitola 5.1.). Dále má objekt na sobě animátor pro animace (viz 4.6.2).

5.3.1 Třída Monster

Monstrum je ve hře implementováno třídou `Monster`, která implementuje interface `IDamagable`, aby se na monstra dalo útočit, poté `IHealable`, aby bylo možné je léčit. Kromě toho existuje interface `IMonster` (který třída monster implementuje), aby hra byla rozšiřitelná na další typy monster.

Monstrum má několik základních stavů: `idling`, `wandering`, `attacking`, `chasing`, `dead` a `presenting`

`Idling`, `wandering` a `chasing` souvisí pouze s pohybem (více v kapitole 5.3.2). `Attacking` je stav, který se nastaví, když monstrum začne útočit na hráče/nepřítele. K útoku dojde, pokud je hráč/nepřítel dostatečné blízko (v rámci `attackingDistance`). Další informace o boji jsou v kapitole 5.3.3. Stav `dead` je důležitý, protože před zničením objektu musíme nastavit animaci úmrtí, která určitou dobu trvá. Nestačí tedy zničit objekt okamžitě.

Stav `presenting` je potřebný kvůli tvorbě nových oblastí, ve kterých se monstra budou vyskytovat. Při tvorbě oblasti jsou hráči nabídnuta monstra, která může použít. Tato monstra jsou již plně funkční herní objekty se všemi skripty, včetně pohybových, a proto by se okamžitě začala někam pohybovat. Z tohoto důvodu je ve hře použitý speciální stav `presenting`, který zabraňuje jakémukoli pohybu monstra.

Tato třída se také stará o `chaselock`. Pokud je vypnutý, monstrum bude při pronásledování plynule přecházet k nejbližšímu cíli. Pokud je zapnutý, drží se stále jednoho cíle.

5.3.2 Pohyb

Pro pohyb monster máme dva hlavní skripty: pro pronásledování skript `Chase` a pro mód procházky `WanderMovementImproved`. Monstrum má několik stavů, jež umožňují rozlišit, které pohybové skripty mají zrovna běžet. Jak je zmíněno v předchozí kapitole, monstra disponují stavy `idling`, `wandering`, `attacking`, `chasing`, `dead` a `presenting`. Pro pohyb jsou důležité pouze první čtyři stavy. Skript `Chase` běží, pokud je monstrum ve stavu `chasing` nebo `attacking`. Skript

`WanderMovementImproved` funguje, když je monstrum ve stavu `idling` nebo `wandering`.

Stavy `presenting` a `dead` potřebují, aby se monstrum nehýbalo, proto u nich neběží žádný ze skriptů pohybu.

Chase

Pro mód pronásledování používáme třídu `Chase`. Skript `Chase` již předpokládá, že víme, koho budeme pronásledovat. To se určí v rámci třídy `Monster`. To znamená, že například i to, zda monstrum má nebo nemá `chaselock`, se řeší ve třídě `monstrum`, nikoli `Chase`.

Opakovaně v rámci funkce `Update()` v Unity monstrum zjišťuje, zda již nedosáhlo cíle. Pokud ne a stále cíl vidí, bude ho dále pronásledovat. Pronásledování dosáhne rotací směrem k cíli a pohybem dopředu.

Pokud se cíl schová (např. se dostane za monstrum s malým úhlem pohledu), přestane monstrum cíl pronásledovat a přepne se do stavu `idle`, což umožní jiným skriptům převzít kontrolu.

Skript také kontroluje, zda se monstrum nedostalo za hranici oblasti. Pokud ano, monstrum přestane pronásledovat cíl. Fungování skriptu `WanderMovementImproved` zajistí, že se monstrum vrátí zpět do své oblasti.

WanderMovementImproved

Tento skript řeší pohyb v módu procházky. Skript má určité destinace, ke kterým monstrum pošle. Poté, co monstrum dojde do cíle, zvolí novou destinaci. Pokud je monstrum v oblasti, která má předurčenou cestu, má tuto cestu v seznamu a pouze ho cyklicky prochází. Je potřeba, aby se předdefinované destinace nacházely v dané oblasti. Tento skript to již nekontroluje.

Pokud monstra nejsou v oblasti s předem určenou cestou, monstrum si pokaždé vybírá náhodnou destinaci v rámci oblasti. Monstra se pohybují nejkratší cestou k cíli, a to tak, že se opakovaně natočí do cíle a pak jdou rovně.

5.3.3 Boj

Monstra mají určitou schopnost se bránit. Každé monstrum má proměnnou `defence`, která určuje sílu jeho obrany. Když na něj někdo zaútočí, vygeneruje se pseudonáhodné číslo od 0 do 100. Pokud je větší než `defence`, bude monstrum zraněno a odebere si život, pokud je menší než `defence`, monstrum se dokáže ubránit. Díky tomu je možné, aby se monstrum ubránilo i některým kouzlům.

Proměnná `attackingDistance` určuje vzdálenost od cíle, v rámci které monstrum může útočit. Pokud je nepřítel/hráč blíže než `attackingDistance`, monstrum se přepne do stavu `attacking` a zaútočí na nepřítele.

5.3.4 Spawner

Oblasti, ve kterých se monstra vyskytují, jsou implementovány třídou `MonsterSpawner`. Jedním z jejích úkolů je dodat monstrům cíle, které mohou pronásledovat. Těmi jsou buď všichni hráči, nebo v případě, že monstra v oblasti útočí, jen pokud na ně bylo zaútočeno, tak seznam hráčů, kteří na ně zaútočili.

Implementace tvorby nových monster funguje tak, že `MonsterSpawner` v pravidelných intervalech kontroluje, zda se počet monster rovná maximálnímu počtu. Pokud ne, zapamatuje si, kolik nových monster je potřeba vytvořit. Pro každé nové monstrum vygeneruje čas (pokud je zapnutý `randomizer`, pokud ne, čas je 0). Potom si tyto časy seřadí od nejnižšího k nejvyššímu a opakovaně zjišťuje, zda je již nějaký z časů 0 (časy updatuje). Pokud ano, vytvoří nové monstrum.

5.4 Hráč

Hráč je reprezentován objektem hráče a skripty: skript `Player`, skripty energie, skript gravitace, skripty pohybu, skript útoku na blízko a skript umožňující kliknout na hráče.

Skript energie se jmenuje `Energy` a má za úkol starat se o zobrazení energií many nebo zdraví hráče. Má k sobě přiřazený indikátor zdraví nebo many, který je umístěn na levé straně obrazovky a ukazuje, kolik zdraví/many hráč má. Skript má v sobě několik základních metod jako přidání nebo ubrání energie, změna maxima a zvýšení energie na maximum (např. při regeneraci hráče) nebo pouze změna maxima.

Skript boje na blízko funguje tak, že pokud hráč zmáčkne mezerník (zaútočí) skript se podívá, jestli je někdo v blízkosti hráče, a pokud ano, zaútočí na něj. Animace útoku se spustí, i když není nikdo v dosahu.

Třída `Player` implementuje `ISpellcaster`, `IDamagable`,`IHealable` a `Monobehaviour`.

5.4.1 Pohyb

Unity má pro ovládání pohybu hráče předpřipravené funkce. Například pro pohyb existují v rámci Unity osy (`Input.getAxis`). Máme horizontální osu, která určuje pohyb doleva nebo doprava, a vertikální osu, která určuje pohyb dopředu nebo dozadu. Výhodou je, že není třeba řešit, jaké tlačítko hráč zmáčkne. V Unity jsou pro pohyb hráče automaticky přiřazeny jak šipky, tak kombinace WASD.

Skript starající se o pohyb hráče se nazývá `Movement`.

5.5 Kouzla

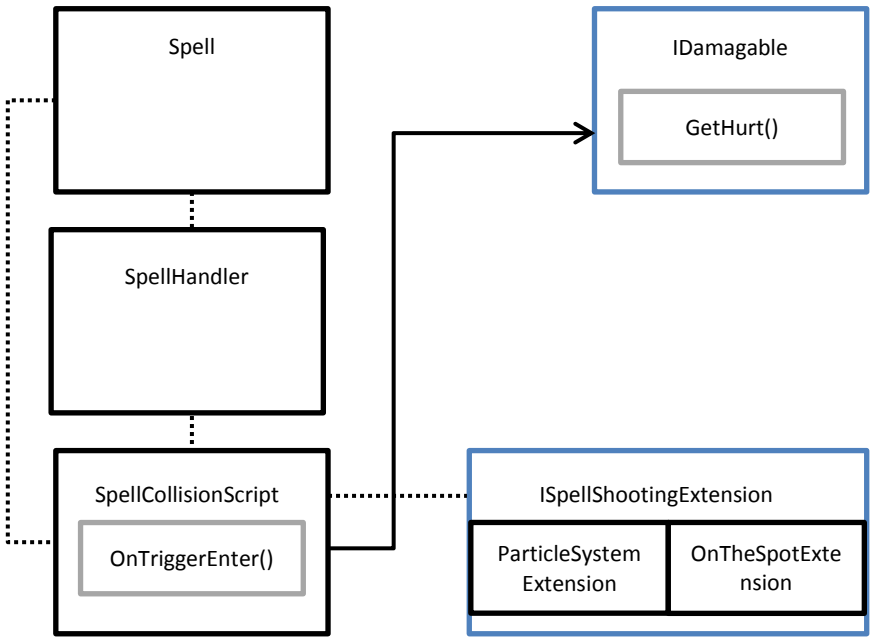
Kouzla jsou ve hře reprezentována třídou `Spell`. Ta nereprezentuje `MonoBehaviour`, což umožňuje vytvořit konstruktor, ale neumožňuje za běhu použít funkce `Update`/`FixedUpdate`. Dále to znamená, že ji nejde připojit na volné objekty. To při tvorbě nových kouzel přes editor není problém (viz 5.5.3), ale pokud chceme vytvořit nějaká kouzla v Inspektoru, není to proveditelné. Proto jsme si

vytvořili nový skript `SpellPlacer`, který dědí `MonoBahaviour` a který má proměnné pro všechny důležité vlastnosti kouzel. Tento skript vytvoří nové kouzlo dle vlastností, jež jsme zadali v Inspektoru Unity.

Vytvořili jsme další čtyři třídy, které dědí třídu `Spell: PoisonSpell` pro jedovatá kouzla, `WallSpell` pro kouzla stojící na místě, `HealingSpell` pro léčící kouzla, `UpgradingSpell` pro vylepšující kouzla a `RecolorSpell` pro obarvující kouzla.

Tyto třídy s kouzlem nijak nepohybují, drží pouze veškeré informace o kouzle jako poškození, typ, cena atd.

Postup vyvolání kouzla ukazuje obrázek 20. Hráč nebo nepřítel vybere kouzlo, které vystřelí (Spell na obrázku). Poté určí cíl. V tuto chvíli SpellHandler (na obrázku 20) pro hráče či AiSpellHandler pro nepřítele vyšle kouzlo, čímž vznikne `SpellCollisionScript` (vyznačeno na obrázku 20). Tento skript má za úkol kontrolovat objekty, do kterých narazí (k tomu slouží funkce `OnTriggerEnter()`), a zjišťovat, zda se shodují s typem kouzla. Pokud ano, začne na objekt kouzlo působit, např. vylepšující kouzlo nesmí reagovat, pokud trefí monstrum, ale pokud trefí zámek, musí mu oznámit, že se má vylepšit. To udělá tak, že dle typu kouzla zjišťuje, zda cíl implementuje požadovaný interface (na obrázku je znázorněno interface `IDamagable()`), a pokud ano, zavolá požadovanou funkci na daném interface (`GetHurt()` nebo `GetHurtPosion()` viz níže).



[Obrázek 20 – Střílení kouzel]

Pro obarvující kouzla, vylepšující kouzla a kouzla On Impact tímto práce končí. Vyvolá se jejich efekt na daném objektu a kouzlo poté zmizí. Dále se již nemusí nic udělat. Ve hře jsou však i kouzla, která působí opakovaně, a to jedovatá kouzla a kouzla na místě. Jedovatá kouzla fungují tak, že je v `IDamagable` interface

přidaná funkce `GetHurtPoison()`, která využívá toho, že v Unity lze spustit korutinu, která opakovaně po uplynutí časového intervalu ublíží cíli.

Kouzla na místě ale takto vyřešit nejde. Může do nich totiž vejít někdo jiný než původně, a naopak původní objekt z nich může vyjít, a v tom případě by kouzlo už nemělo původní objekt zraňovat. Proto jsme přidali skript `OnTheSpotExtension`. Když kouzlo na místě někoho zraní, přejde tento objekt do seznamu imunních objektů a po dobu mezi útoky kouzla mu nebude brán život. Pokud bude poté stále v dosahu kouzla, `SpellCollisionScript` se spustí a řekne znovu `OnTheSpotExtension`, že objekt je v dosahu. Ten zjistí, zda objekt patří do imunních objektů nebo ne. V tuto chvíli už do nich nebude patřit, takže ho kouzlo zraní. Tento skript se také stará o to, aby kouzlo na místě po svém životním cyklu zmizelo.

`OnTheSpotExtension` se spustí pro všechna kouzla typu na místě, tedy i léčivá a obarvující. U těch nebude skript nic dělat.

Pro létající kouzla je třeba, aby pronásledovala svůj cíl. To řeší třída `ParticleSystemExtension`, která je potomkem `ISpellExtension` (vyznačeno na obrázku 20 s modrým rámem, potomkem této třídy je i `OnTheSpotExtension`). `ParticleSystemExtension` kouzlo otáčí a posílá směrem ke svému cíli. V průběhu letu však `SpellCollisionScript` kontroluje, zda kouzlo do něčeho nenarazilo – pokud ano, informuje `ParticleSystemExtension`, kouzlo objekt zraní a zničí se.

Hráč si vybere kouzlo tím, že si zvolí jeho ikonu v menu. K tomu slouží skript `ButtonSpellAssigning`, jenž obsahuje informaci o tom, které kouzlo si hráč vybral. Přiřazení ikonky ke kouzlu řeší skript, který tvoří menu z ikonky (více podrobností viz níže). Poté hráč vybere cíl a klikne na něj, tím se spustí `SpellCollisionScript, jak je zmíněno výše`. Ten už ví, jaké kouzlo hráč seslal (tu informaci mu předá `ButtonSpellAssigning`).

5.6 Editor

Editor jsme si rozdělili na několik částí: editace monster, editace kouzel (patří do editace objektů), editace terénu a editace nepřátel. Každá z těchto částí má vlastní scénu a vlastní skripty, jež se jim věnují. Kvůli různým scénám je potřeba zajistit přenos objektů mezi scénami. K tomu slouží speciální třídy jako `SpellHolder a Level`, kde `SpellHolder` drží informace o kouzlech a `Level` o monstrech či dalších potřebných proměnných mezi scénami.

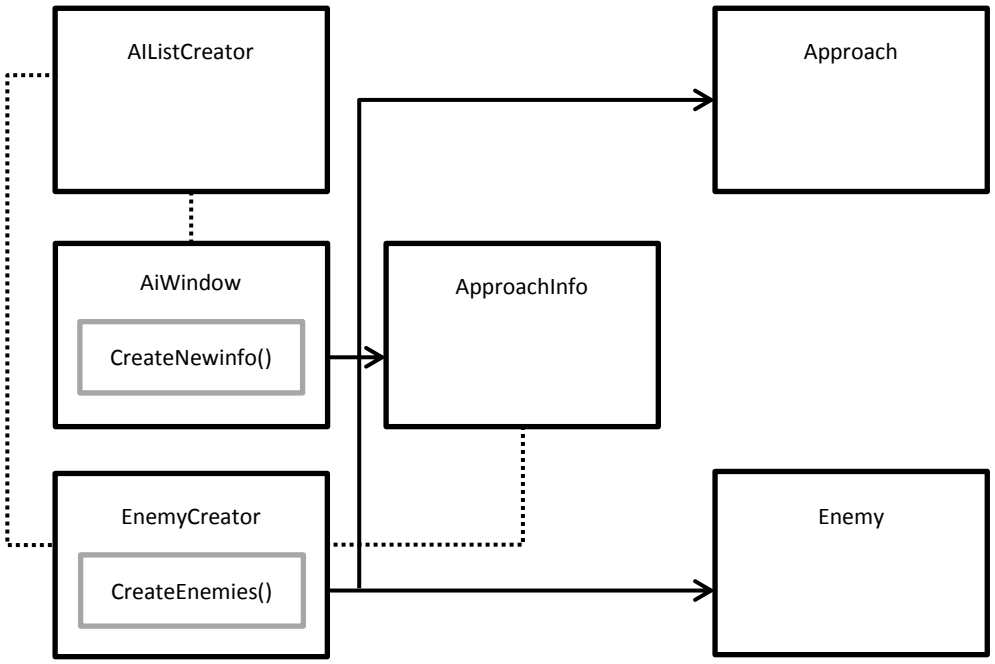
5.6.1 Editace objektů

Tvorba nových monster a kouzel funguje velmi podobně. Pro monstra existuje třída `MonsterCreation`, pro kouzla `SpellCreationHandler`. Fungují tak, že ve scéně tvorby nových kouzel/monster mají k sobě přiřazená všechna tlačítka a textová pole. Poté, co designer vyplní požadované informace, přesunou si tyto vlastnosti do proměnných a pak zavolají konstruktor u kouzel a vytvoří kouzlo.

Monstra dědí `Monobehaviour`, takže není možné volat parametrický konstruktor. Další problém s monstrem spočívá v tom, že monstrum není tvořeno jen jedním skriptem jako kouzlo, ale řadou skriptů, jako `Chase`, `Monster` apod. Některé vlastnosti jim není možné přiřadit již při tvorbě monstra, protože se dají přiřadit až při tvorbě oblastí. Třída `MonsterCreation` tedy nejdříve vytvoří objekt monstra (viz kapitola 4.9.4), poté k němu přiřadí skripty `Monster`, `WanderingMovementImproved`, `Chase`, skript gravitace a další skripty nutné pro fungování. Některé proměnné těchto skriptů však zůstávají prázdné, dokud hráč nevytvoří `spawner`, který do nich dodá dodatečné informace.

5.6.2 Editace nepřátel

Designer může vytvořit nové nepřátele, upravit jejich život, manu, útok bez kouzel, barvu i vytvořit nový `Approach`, který určuje chování nepřátel. Tvorba nových nepřátel probíhá ve vlastní scéně.



[Obrázek 21 – Třídy pro tvorbu nepřátel]

O vzhled této scény (tlačítka) se stará třída `AIListCreator` (vyznačeno na obrázku 21), který je potomkem třídy `ListCreator`. Druhým potomkem této třídy je třída `CastleListCreator`, která se stará o tlačítka v rámci scény pro tvorbu úrovní zámku (viz kapitola 5.6.3).

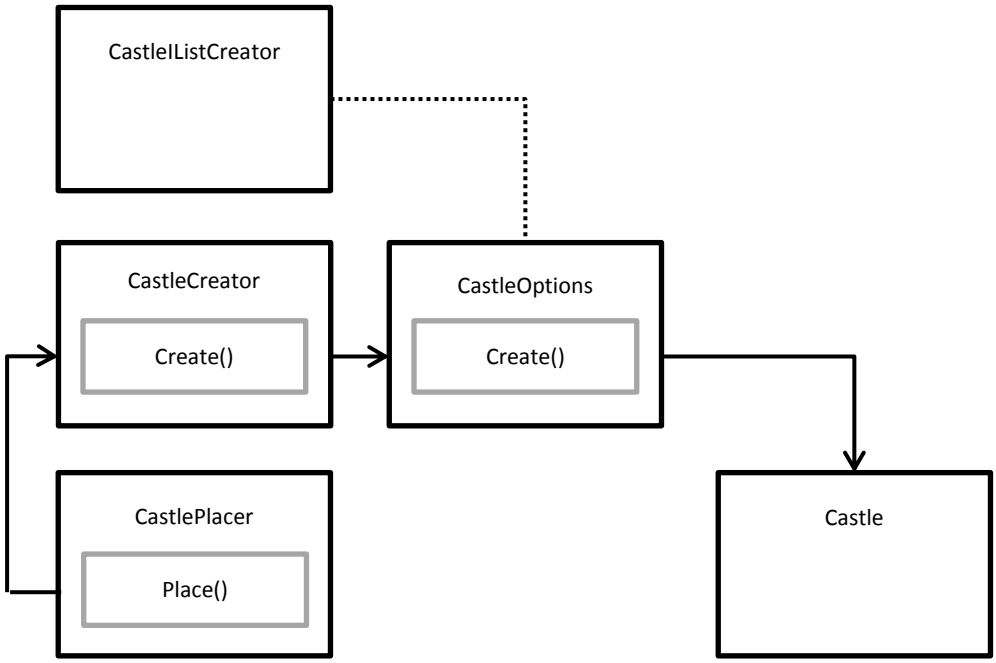
Pro tvorbu nového `Approach` existuje další třída `AIWindow`. Tato třída zobrazí okno pro vytvoření nového `Approach`, v rámci kterého lze určit prioritu misí, zda se povolují vedlejší mise a maximální množství života/many, pro spuštění některých misí např. maximální množství života, při kterém se ještě spustí mise léčení. Poté, co designer zadá všechny vlastnosti budoucí třídy `Approach`, klikne na tlačítko „Create new approach“, to zavolá funkci `CreateNewInfo()` (viz obrázek

21 – funkce jsou vyznačeny šedým rámem), která vytvoří instanci třídy `ApproachInfo`. Není totiž ještě možné vytvořit instanci třídy `Approach`, protože by jí chyběly odkazy na jiné třídy, např. na třídu `Enemy`, či `Castle`.

Poté, co designer zadá všechny požadované hodnoty, může pokračovat v editoru (přejde na tvorbu úrovní zámku). V tuto chvíli se požadované hodnoty týkající se nepřátel uloží do třídy `AIListCreator`, která přežije přechod scény. V hlavní scéně hry (ve které se hra hraje) je třída `EnemyCreator`, která převezme informace od tříd `AIListCreator` a `ApproachInfo` (pokud existují) a vytvoří nové nepřátele i jejich třídy `Approach` (viz obrázek 21).

5.6.3 Zámek

Další částí editoru je úprava úrovní zámku. Zámek je reprezentován třídou `Castle` a pro jeho vylepšení nebo degradaci slouží funkce `Upgrade()` a `Degrade()`. Při spuštění hry třída `CastleOptions` (vyznačeno na obrázku 22), která udržuje informace o tom, jaké úrovně zámek má, zavolá svou funkci `Create()` (funkce jsou na obrázku vyznačené šedou barvou) a vytvoří `Castle`. Tato funkce předá zámku informace o tom, kolik má úrovní a jak se má vylepšovat.



[Obrázek 22 – Tvorba zámků]

Pokud designer úrovně nemění, použije se implicitní `CastleOptions` (skript připojen k objektu ve scéně hry). Pokud však vytvoří vlastní úrovně zámku, použije se `CastleOptions` ze scény tvorby úrovní zámku. O vzhled této scény se stará třída `CastleListCreator`. Tato třída má v sobě všechna tlačítka a zároveň udržuje informace o tom, co designer zvolil. Když designer skončí s tvorbou nových úrovní,

hra přejde do hlavní scény hraní hry. `CastleListCreator` přežije přechod této scény a předá své informace třídě `CastleOptions`.

Počet úrovní je omezen na 10 s tím, že pro prvních 5 úrovní se mění vzhled zámku. Pro následující úrovně se zámek pouze zvětšuje. Vzhledem k tomu, že pro zničení zámku je třeba zničit zámek první úrovně (pokud je zámek na vyšší úrovni, jen degraduje), větší počet úrovní by byl zbytečný – hra by trvala příliš dlouho.

Kromě tvorby nových úrovní má designer také možnost zámky postavit (určit jejich pozici na planetě). Pokud designer měnil počet nepřátel nebo úrovně zámku, musí zámky postavit. K postavení zámku na planetu slouží třída `CastlePlacer`, která vytvoří nový GameObject na dané pozici a poté zavolá funkci `Create`() třídy `CastleCreator`, která (přes `CastleOptions`) daný zámek vytvoří (vše je vyznačeno na obrázku).

5.7 Modely ve hře

Součástí této hry není grafická úprava jednotlivých objektů. Proto v rámci programování této hry používáme již hotové grafické objekty, které jsou volně dostupné v Unity Asset Store.

Objekty	Popis	Autor
MOBA Game Witch Doctor (Barbarian Shaman)	Vzhled a animace pro objekty hráče a nepřátel.	MonsterSnail
Castle Supply LITE	Vzhled pro objekty zámku.	Aquarius Max
Elemental Free	Animace a vzhled pro kouzla	G.E. Team Dev
Fantasy Monster Skeleteon	Vzhled a animace pro objekt monstra (kostlivec)	Teamjoker
Monster	Vzhled a animace pro objekt implicitního monstra	3DMaesen
Toy Balloonship	Vzhled pro objekt balonu.	NV3D
Battle Skill Icon	Ikony kouzel	DR.Game

[Tabulka 3 – Modely ve hře]

6 Závěr

V rámci zadání hry jsme měli tři hlavní cíle: vytvořit hru Zničte zámek, vytvořit její editor a zjistit, jaké výhody a nevýhody má programování této hry v rámci Unity. Tyto cíle se nám podařilo splnit. Vytvořili jsme hru inspirovanou hrou Magic Carpet, ve které se vyskytují počítačem ovládaní protihráči a oblasti monster. Dále jsme ke hře naprogramovali editor, ve kterém lze vytvořit nová monstra, kouzla, upravit terén, upravit chování nepřátel a nastavit různé úrovně zámku. Pro tvorbu nových monster a kouzel je třeba otevřít Unity.

Nepodařilo se nám vytvořit nejvhodnější způsob pohybu hráče, protože hráč létá, ale k jeho ovládání můžeme použít pouze šipky (neboť myš se již používá ke střílení kouzel).

V průběhu programování hry se ukázalo, že pro určité postupy by bylo možné najít v rámci Unity příznivější řešení – např. volba interface jako základního prvku pro komunikaci mezi třídami nebyla příliš vhodná, protože není vidět v rámci inspektoru Unity. Vhodnějším řešením by bylo zvolit abstraktní třídy.

Hlavní výhodou Unity byla již připravená interakce mezi objekty, takže nebylo nutné řešit kolize. Další výhodou Unity byl předpřipravený systém pro animace – bylo tedy možné použít funkční animátor.

Použitím Unity se nám tvorba zjednodušila, a proto bychom dalším programátorům práci Unity doporučili.

6.1 Rozšíření hry

Hra by šla rozšířit několika způsoby:

- Monstra se stanou kouzelníky
  Zajímavým rozšířením hry by mohlo být umožnění monstrům střílet kouzla.
- Lepší umělá inteligence
  V rámci hry jsme naprogramovali základní chování umělé inteligence pomocí misí a jejich priorit. Výběr chování a vlastnosti těchto misí se dají upravit pro každého nepřítele individuálně nebo i hromadně ve třídě zvané `Approach`. Tímto způsobem je možné vytvořit nové a náročnější nepřátele.
- Interakce s objekty
  Dalším zajímavým rozšířením by mohly být objekty ve hře, např. stromy, kameny, případně různé budovy atd.
- Týmy
  V rámci misí si nepřátelé vybírají, na jakého jiného hráče budou útočit. Hra by se dala upravit a rozšířit tak, že by si nepřátele nevybírali náhodně ze všech hráčů, ale jen z některých, tím by se dali vytvořit týmy hráčů.

Seznam použité literatury

[1] **Stacey Haffner** – Blog o programování v .NETu [webová stránka] [Citace: 11. 05. 2017.]:

https://blogs.msdn.microsoft.com/dotnet/2016/07/26/the-week-in-net-7262016

[2] Webová stránka Unity [webová stránka] [Citace: 11. 05. 2017.]:

https://unity3d.com/public-relations

[3] Porovnání 3D Enginů [webová stránka] [Citace: 11. 05. 2017.]:

https://www.slant.co/topics/1495/~3d-game-engines

[4] **Igor Galochkin** –Implementation of a cross-platform strategy multiplayer game based on Unity3D [diplomová práce] [Citace: 11. 05. 2017.]:
https://www.os.in.tum.de/fileadmin/w00bdp/www/Lehre/Abschlussarbeiten/Thesis_2013.01.10_eVersion_Galochkin.pdf

[5] Manuál Unity [webový dokument] [Citace: 11. 05. 2017.]:

https://docs.unity3d.com/Manual/

[6] **Alan Thorn** – Mastering Unity Scripting, PACKT Publishing, 2015

[5] MSDN Library [webová stránka] [Citace: 11. 05. 2017.]:

https://msdn.microsoft.com/library

7 Příloha A – Uživatelská dokumentace

V této dokumentaci si popíšeme, jak hrát hru Zničte zámek.

7.1 Instalace a otevření hry pro další rozšíření

Pro spuštění hry je na přiloženém USB flash disku potřeba spustit soubor znictezamek.exe, který se nachází ve složce Bin/ZnicteZamek.

Hra vyžaduje 64bitový operační systém Windows a DirectX9 nebo DitectX11, který je však automaticky naistalován se všemi 64bitovými operačními systémy Windows.

Pro otevření projektu v Unity k umožnění dalších úprav a rozšíření je nutné mít nainstalovanou verzi Unity 5.2. nebo verzi, která je s touto verzí zpětně kompatibilní, .NET verzi 4 (v našem projektu používáme verzi 4.6.) a program pro úpravu skriptů (v našem projektu používáme Visual Studio 2015).

Programátor otevře Unity projekt tak, že otevře Unity, poté klikne na File – Open Project, v novém okně klikne na tlačítko open a přejde na složku projektu ve složce Projekt na přiloženém USB flash disku.

7.2 Ovládání hry

Po spuštění hry hráč ovládá svou postavičku, která může vyvolávat kouzla. Cílem hry je zničit zámky všech nepřátel. K tomu slouží ve hře kouzla. Kouzla jsou různého typu (více v kapitole 7.2.2), ale stojí nějakou manu. Manu je možné získat zabitím monster (viz 7.2.4).

Hráčova postavička je obnovitelná a pokaždé, když zemře, zrodí se znovu ve svém zámku s plným životem. Pokud je zámek hráče zničen, nastává konec hry.

Pro zapnutí základní varianty hry bez úprav pomocí editoru, je třeba hru spustit, poté v hlavním menu (obrázek 23) zvolit tlačítko Play pak na obrazovce vkládání spawnerů/úpravy terénu (obrázek 29) kliknout na tlačítko Play vpravo nahoře.

Kliknutím na tlačítko Instructions v hlavním menu, se objeví popis ovládání hráče.

[Obrázek 23 – Hlavní menu]

7.2.1 Hráč

Vlevo nahoře jsou umístěny dva základní indikátory života a many, jak je vidět na obrázku 24. Zelený indikátor naznačuje, kolik má hráč života. Fialový indikátor ukazuje, kolik má many.

[Obrázek 24 -Indikátory]

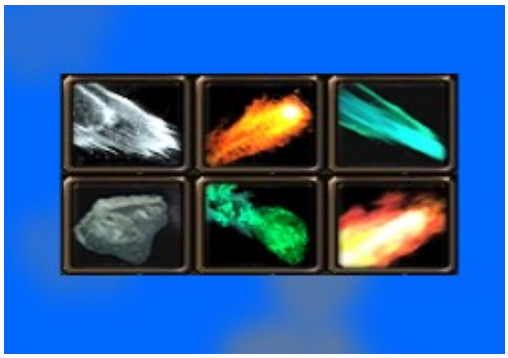
Pohyb

Pro pohyb hráče slouží šipky nebo kombinace WASD. Pro otočení doleva nebo doprava slouží šipky doleva/doprava nebo tlačítka A a D. pro pohyb dopředu a nahoru slouží šipka nahoru či tlačítko W a pro pohyb dolů a dozadu slouží šipky dolů či tlačítko S.

Pokud je postavička hráče ve vzduchu, stačí krátce zmáčknout tlačítko dolů a postavička se snese k zemi.

7.2.2 Kouzla

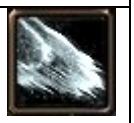
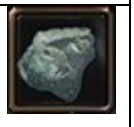
Pro vyvolávání kouzel má hráč na pravé straně obrazovky seznam kouzel, (viz obrázek 25). Pro výběr kouzla stačí, když hráč klikne na jedno z kouzel. Poté hráč klikne na cíl. Kouzla jsou vyslána k cíli, pouze pokud je cíl shodný s typem kouzla (viz níže). Pokud se hráč rozhodne vyslat raději jiné kouzlo, stačí kliknout na jinou ikonu v menu. Pravým tlačítkem se zruší výběr kouzel.



[Obrázek 25 – Menu kouzel]

V tabulce 3 je seznam kouzel v základní hře a jejich vlastnosti. Do hry je možné přidat nová kouzla (viz kapitola 7.3.7). Kromě těchto kouzel se ve hře objevují ještě speciální kouzla: uzdravovací kouzlo, vylepšující kouzlo a kouzlo obarvení.

Kouzlo léčení lze seslat na hráče, zámky nebo nepřátele, které následně vyléčí. Implicitní kouzlo stojí 12 jednotek many a vyléčí cíli 10 jednotek života.

Název	Popis	Poškození	Cena	Ikona
Fire (oheň)	Kouzlo typu Flying a On Impact poletí k cíli od zdroje a zraní ho pouze jednou.	40	17	
Firewall (ohnivá zeď)	Kouzlo typu On The spot a While Touch. se objeví přímo v cíli a zraní opakovaně kohokoli, kdo s ním přijde do kontaktu. Po následujících 10 sekund každých 5 vteřin zraní cíle o 4.	20	10	
Poison Spell (kouzlo jedu)	Kouzlo typu On the spot a Poison se objeví přímo v cíli a cíl otráví. (I když poté cíl přestane být v kontaktu s kouzlem, stále bude přicházet o zdraví). Zraní ho ještě 3x každé 2 sekundy o 3 jednotky.	5	8	
Kouzlo obarvení	Obarvuje manu.	0	0	
Vylepšující kouzlo	Vylepší zámek na vyšší úroveň.	0	Dle úrovně zámku.	
Kouzlo léčení	Kouzlo vyléčí cíli 10 jednotek života.	-10	12	

[Tabulka 4 –Kouzla]

7.2.3 Boj na blízko

Kromě vyvolávání kouzel má hráč ještě druhou možnost, jak bojovat s monstry. Když přiletí do těsné blízkosti monstra, má možnost ho udeřit. To udělá zmáčknutím klávesy mezerník. Boj na blízko způsobí velmi malé poškození a hráč tím riskuje, že ho monstrum zraní, ale pokud hráč nemá manu, je to způsob, jak ji získat.

7.2.4 Sbírání many

Na to, aby hráč mohl používat kouzla, potřebuje manu. Tu lze získat zničením monster. Poté, co monstrum zemře, vznikne neutrální mana (bílá mana). Tu je třeba obarvit kouzlem (které žádnou manu nestojí).

Následně z hráčova zámku vyletí balon, který manu posbírá a přenese ji zpět do zámku. Tam se promění v použitelnou manu a hráčův mana bar se zvětší (fialový indikátor vlevo nahoře na obrazovce).

Manu nelze zničit – může přes ni proletět neomezený počet kouzel.

7.2.5 Balony

Balon ve hře slouží pro přenos many, kterou si hráč obarvil, do zámku. Balony fungují automaticky. Pokaždé, když je nějaká obarvená mana k dispozici, balon pro ni vyletí. V okamžiku, kdy balon dorazí zpět do zámku, se mana promění v použitelnou manu a přičte se ke hráčově maně (jak ukazuje indikátor many).

Balon má určitou kapacitu a je schopen pojmout pouze určité množství mana balls. To se dá zvětšit vylepšením zámku (viz kapitola 7.2.6).

Hráč i nepřátelé mohou ničit balony ostatních hráčů. Stačí na balon seslat kouzlo. Když je balon zničen, mana z něj vypadne volně do prostoru. Potom začne běžet obnovovací perioda, po které hráč/nepřítel, který nemá balon, získá nový. Není tedy možné zabránit hráči/nepříteli navždy, aby získával manu, ale pouze dočasně.

7.2.6 Zámek

Zámek ve hře slouží k proměně sebraných mana balls v použitelnou manu hráče. Dále je základnou pro hráče, ve které je možné hráče regenerovat. Když je zámek zničen, hráč prohraje celou hru, a naopak když zničí zámky všech nepřátel, vyhraje.

Zámek je schopný se bránit a střílet do nepřátel. Zámek nikdy nestřelí zpět do vlastního hráče, i kdyby se stalo, že na něj vlastní hráč sešle útočné kouzlo. Zámky, stejně jako hráče, je možné uzdravovat léčivým kouzlem.

Zámky je možné vylepšovat. K tomu slouží speciální kouzlo pro vylepšení zámku, které stojí určité množství many. Když ho hráč/nepřítel k zámku vyšle, zámek se automaticky vylepší. Pokud nepřítel zámek zničí a zámek je na vyšší než první úrovni, zámek pouze degraduje na nižší úroveň a bude mít maximální život předchozí úrovně. Aby nepřítel mohl zámek zničit, musí zničit zámky všech úrovní,

dokud se nedostane k první úrovni, a teprve poté je zámek doopravdy zničen a hráč prohrál.

Pokud na hráčův zámek útočí nepřítel, hráč bude upozorněn červeným zabarvením obrazovky.

Implicitní hra má připraveno 5 typů zámků a 3 úrovně. Pokud designer vytvoří více úrovní než 5, zámky se budou zvětšovat, aby naznačily své vylepšení. Maximální počet úrovní je 10.

Vzhledem k tomu, že balón se vrací k zámku, až při dosažení své kapacity (nebo když není žádný další mana ball k dispozici), je vhodné mít kapacitu, co nejmenší a tím docílit častějšího doplňování many.

Úroveň	Bonus života (Life Bonus)	Poškození (Damage)	Kapacita balonu (Balloon Capacity)	Cena vylepšení (Upgrade Cost)	Maximální zdraví (Max Health)	Prodleva mezi útoky (Time Between Attacks)	Maximální mana (Max Mana)
1	50	5	1	20	200	1	100
2	50	10	1	30	300	1	130
3	50	10	1	40	400	1	170

[Tabulka 5 - Zámek]

7.2.7 Monstra

Monstra ve hře budou na hráče reagovat podle toho, v jaké jsou oblasti. Některé budou hráče ignorovat, jiné zaútočí hned, jak ho uvidí. Žádné monstrum ale nebude pronásledovat hráče mimo svou oblast. V oblasti, kde monstra hráče ignorují, na něj zaútočí, pokud hráč zaútočí na jakékoli monstrum v této oblasti.

Když hráč zničí monstrum, získá mana balls, které lze proměnit na manu.

Monstra se rodí neustále nová a pouze v některých oblastech lze zničením všech monster zabránit rození nových.

7.3 Ovládání editoru

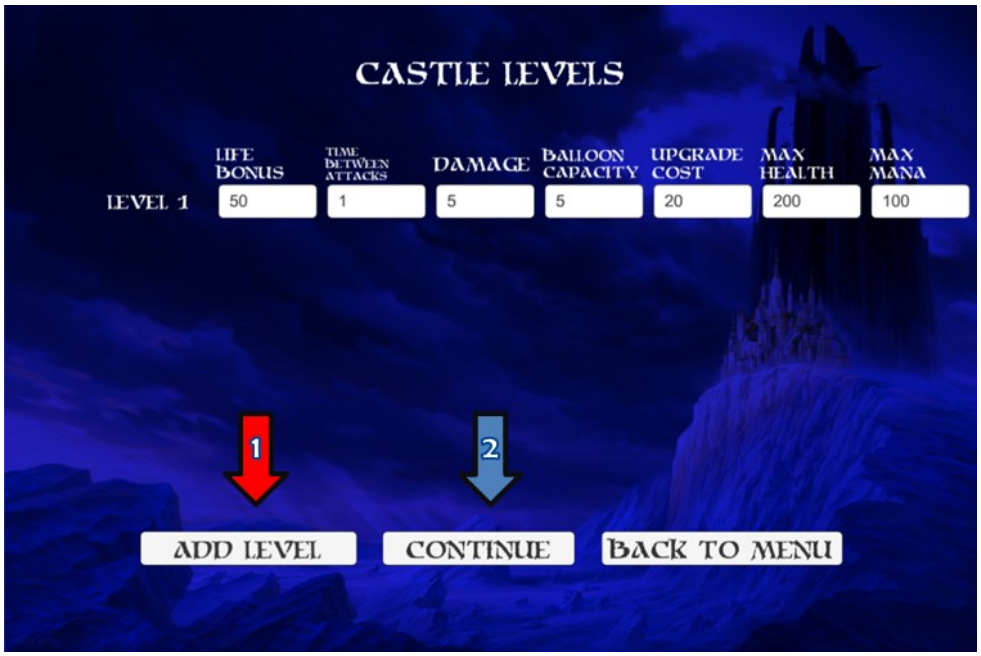
Pro spuštění editoru stačí kliknout v hlavním menu na tlačítko editor (viz obrázek 26). Zde se objeví obrazovka s některými částmi editoru. Pokud chce hráč upravit vše v editoru, měl by kliknout nejprve na Spell Creation a poté ho hra provede ostatními částmi. Jinak lze některé kroky vynechat (např. tvorbu nových kouzel) a přejít přímo na tvorbu monster či ještě dál na tvorbu nepřátel, nebo rovnou na tvorbu oblastí monster a terénu. Pokud se hráč dostane na obrazovku nepřátel, již musí vytvořit nepřátele a úrovně zámku.

[Obrázek 26 – Menu editoru]

7.3.1 Úrovně zámku

Pro úpravu úrovní zámku je třeba v hlavním menu kliknout editor a poté na Level Creation. Druhou variantou je projít některým z předchozích kroků editoru a postupně se přesunout k tvorbě zámku.

Poté se objeví obrazovka, kterou znázorňuje obrázek 27. Pro vytvoření další úrovně stačí zmáčknout Add Level (na obrázku vyznačeno červenou šipkou s číslem 1 na obrázku). Po zmáčknutí se objeví další řádek (stejný jako řádek pro Level 1) s popisem Level 2. Výchozí hodnoty všech úrovní jsou stejné. Pokud designer pouze přidá úrovně a nezmění jejich hodnoty, vylepšení zámku bude znamenat pouze jeho vyléčení. Také bude těžší zámek zničit, neboť po zničení se jen degraduje na nižší úroveň. V rámci určité úrovně lze změnit hodnoty zmíněné v kapitole 7.2.6. Když je designer spokojen s volbou úrovní, měl by kliknout continue (označeo modrou šipkou s číslem 2). Poté bude přesunut na obrazovku, kde může vložit zámky na planetu.



[Obrázek 27 – Úrovně zámku]

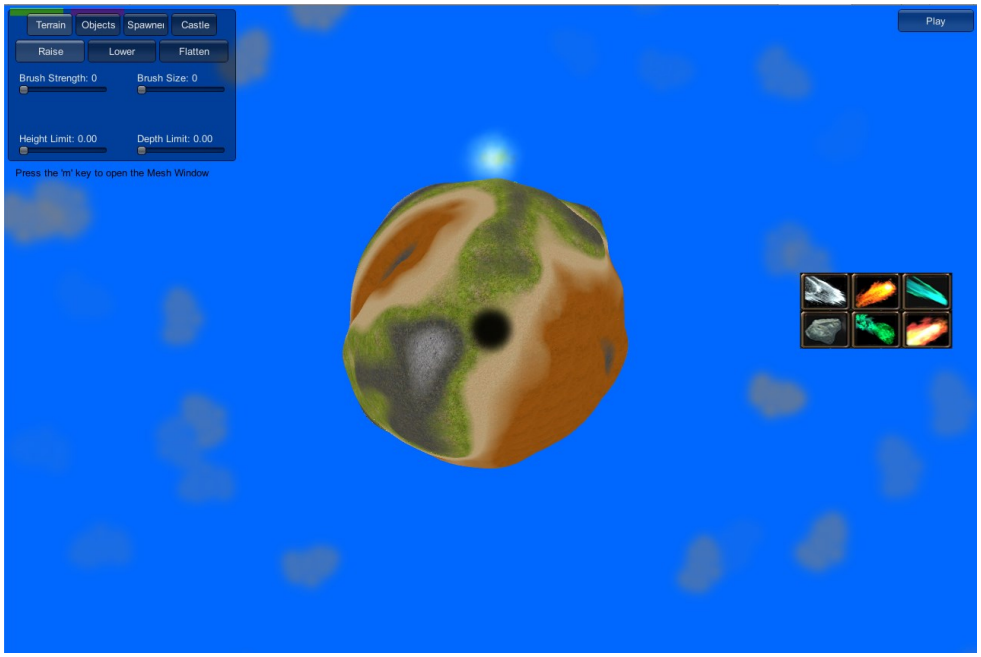
7.3.2 Vložení zámků na planetu

Pro vložení zámků na planetu se musí designer dostat na obrazovku úpravy terénu (viz obrázek 28) a pak kliknout položku Castle (vyznačeno modrou šipkou). Poté může kliknout kamkoli na planetu a postaví se zámek. První postavený zámek patří vždy hráči. Ostatní zámky jsou pak přiřazené nepřátelům v takovém pořadí, v jakém nepřátelé vznikali. Až designer postaví všechny zámky, hra automaticky přestane stavět další po kliknutí myši. Poté designer může vytvořit nové spawnery nebo rovnou začít hrát.

[Obrázek 28 – Vložení zámku na planetu]

7.3.3 Úprava terénu

Aby hráč mohl upravit terén, musí se nejdříve dostat na obrazovku, která je znázorněna na obrázku 29. Toho je možné docílit několika způsoby – buď hráč klikne z hlavního menu na Play, nebo klikne na editor a projde tvorbou kouzel, monster, nepřátel a úpravou úrovní zámku.



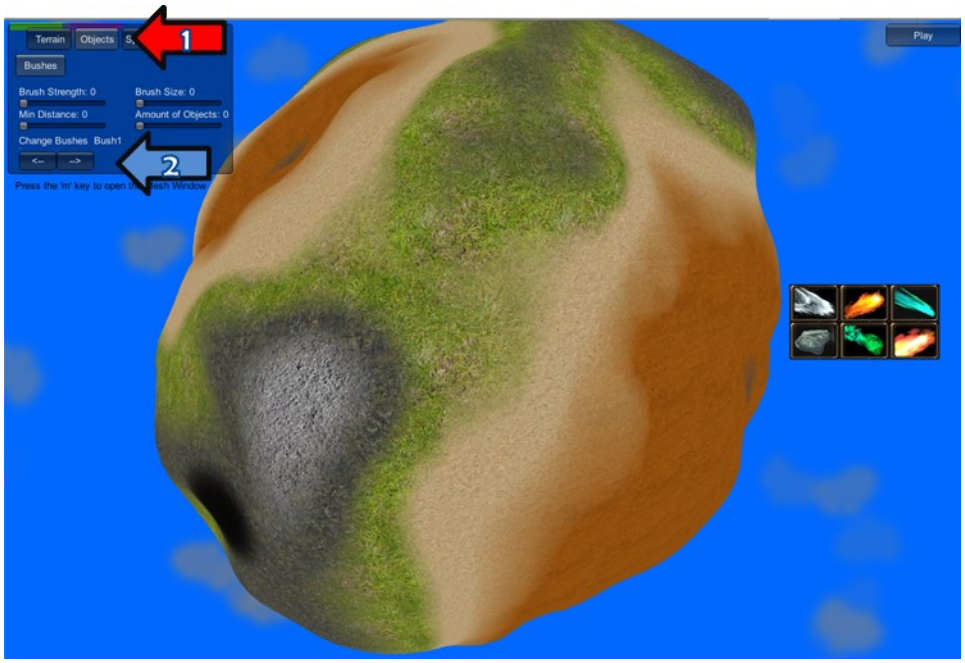
[Obrázek 29 – Úprava terénu]

Vlevo nahoře je již předem zvolené tlačítko Terrain. Pod ním se nacházejí tři další tlačítka Raise (pro zvýšení terénu), Lower (pro snížení terénu) a Flatten (pro zploštění terénu). Nejdříve je potřeba si vybrat jednu z těchto možností. Pro ilustraci si zvolíme Raise (zvýšení terénu). Poté nastavíme sílu štětce (Brush Strength) a velikost štětce (Brush Size). Štětec se zobrazuje jako černý kulatý stín na planetě. Pokud štětec není vidět, je potřeba se více přiblížit k planetě (kolečkem myši). Dále je možné nastavit výšková omezení, a to maximální výšku (Height limit) a limit hloubky (Depth limit).

Pro pohyb vlevo a vpravo kolem planety je třeba kliknout pravé tlačítko myši a pohybovat myší.

7.3.4 Vklad objektů na planetu

Kromě úpravy terénu, lze také do hry vložit rostliny, které jsou průchozí. K tomu stačí přejít na stejnou obrazovka jako pro úpravu terénu (viz obrázek 29). Poté je potřeba kliknout na tlačítko Objects (vyznačeno červenou šipkou s číslem 1 na obrázku 30).



[Obrázek 30 - Objekty]

Pro vklad rostlin na planetu je potřeba nejdříve vybrat typ rostliny pomocí šipek doleva a doprava (na obrázku 30 vyznačeno modrou šipkou s číslem 2). Poté designer zadá sílu štětce (Brush Strength), velikost štětce (Brush Size), minimální vzdálenost mezi objekty (Min Distance) a kolik objektů, se má vytvořit při kliknutí (Amount of Objects). Je důležité, aby minimální vzdálenost mezi objekty byla menší než velikost štětce, jinak se žádný objekt nevytvoří. Následným kliknutím na planetu se rostliny vytvoří a zobrazí na planetě.

7.3.5 Tvorba monster

Pro tvorbu monster je třeba se nejdříve dostat na obrazovku tvorby monster. To uděláme, tak že z menu vybereme položku editor a poté Monster Creation. Objeví se obrazovka na obrázku 31.

Pro každé monstrum je možné upravit poškození, život, obranu, rychlost a počet mana balls, které se objeví, když monstrum umře. Dále půjde upravit, jak daleko musí být hráč, aby ho monstrum ještě začalo pronásledovat. Pak lze nastavit chaselock (zda při pronásledovaní monstrum změní cíl) a úhel pohledu monstra.

Vzhled monstra jde samozřejmě měnit také. Designer zadá název monstra, které hra načte, více v kapitole 7.3.8. V rámci tvorby jednotlivých monster se ještě neurčuje, kde monstra na mapě budou, to se určí až v editoru úrovně.

[Obrázek 31 – Tvorba monster]

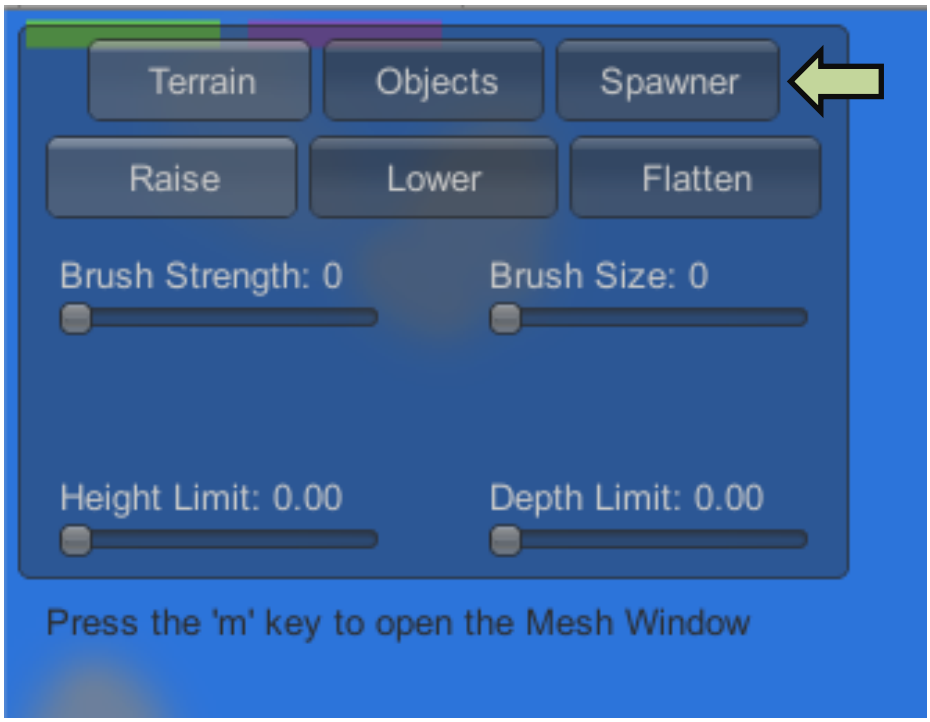
Název vlastnosti	Popis	Jednotky
Damage	poškození, které monstrum udělí hráči/nepříteli, pokud hráče/nepřítele trefí.	Přirozené číslo. Doporučený interval 1 – 15. Implicitní monstrum má nastavenou hodnotu 10.
Life	Život monstra.	Přirozené číslo Doporučený interval 5 – 20. Implicitní monstrum má nastavenou hodnotu 10 .
Mana balls on death	Kolik mana balls se objeví, když monstrum zemře. Každý mana ball obsahuje 100 many.	Přirozené číslo Doporučený interval 1 – 4. Implicitní monstrum má nastavenou hodnotu 3.
Defence	Pravděpodobnost, se kterou se monstrum ubrání útoku. Více v kapitole 5.3.3.	Přirozené číslo Doporučený interval 0 - 100. 0= Hráč se vždy trefí. 100=Hráč se netrefí nikdy. Implicitní monstrum má nastavenou hodnotu 20.

Speed	Rychlost s jakou se monstrum pohybuje	Přirozené číslo. Doporučený interval 1 – 4. Implicitní monstrum má nastavenou hodnotu 2.
Object Name	Název monstra ve složce Resources Více v kapitole 7.2.8.	Název objektu.
Chasing Distance	Maximální vzdálenost v jaké může být nepřítel, aby ho monstrum pronásledovalo.	1 – Radius Monstrum při pronásledování nevyběhne mimo radius. Doporučený interval 5 – 15 Implicitní monstrum má nastavenou hodnotu 10.
Attacking Distance	Vzdálenost v jaké monstrum útočí na nepřítele.	Přirozené číslo. Doporučený interval 1 – 3. Když bude příliš velký, monstrum bude útočit, ale nic netrefí, protože hráč bude daleko.
Chaselock	Při zapnutí platí, že pokud se monstrum rozhodně někoho pronásledovat bude ho pronásledovat, až dokud nedobojuje nebo se nedostane mimo svou oblast. Při vypnutí monstrum mění cíle a pronásleduje toho, kdo je nejblíže.	True/False
View Angle	Úhel pohledu monstra.	1 – 360 Nula nejde použít, protože by monstrum nic nevidělo a nemohlo na nikoho útočit.

[Tabulka 6 – Vlastnosti monster]

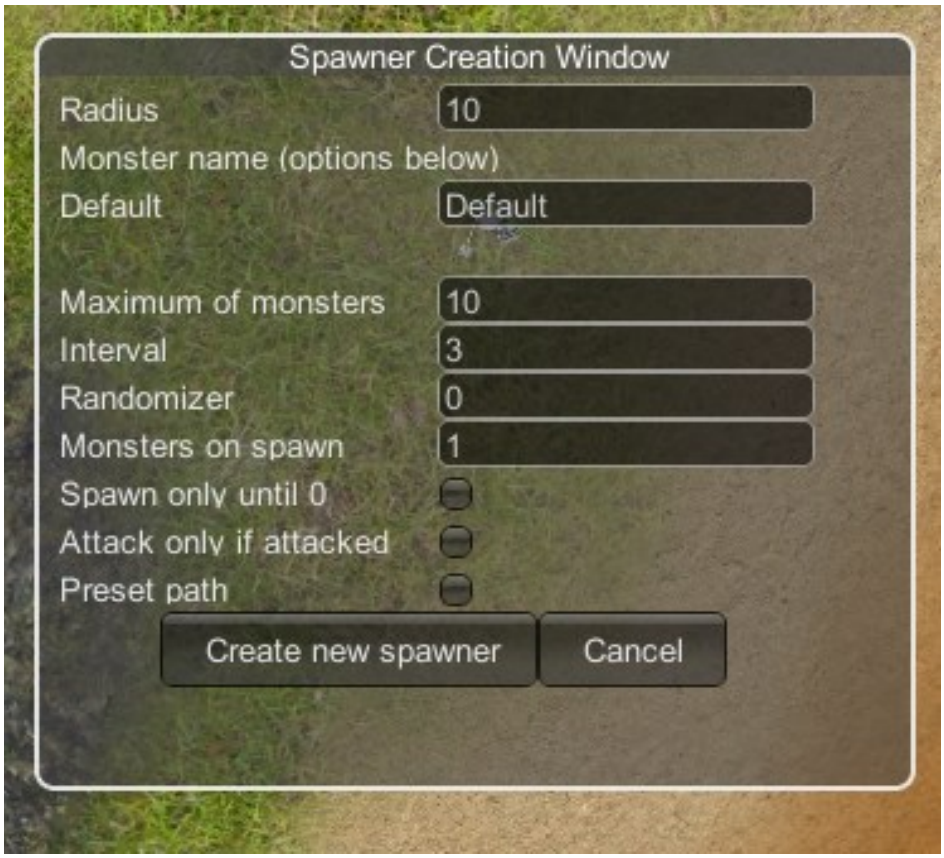
7.3.6 Tvorba Spawneru

Pro vložení monster do hry je potřeba vybrat, v jaké oblasti se budou vyskytovat. K tomu slouží spawner. Při zapnutí hry má hráč na levé straně obrazovky možnost kliknout na tlačítko spawner (viz obrázek 32).



[Obrázek 32 – Menu se spawnerem]

Poté se na obrazovce objeví menu s různými vlastnostmi spawneru.



[Obrázek 33 – Vlastnosti spawneru]

Pod tímto menu se na planetě objeví vždy jeden exemplář ze všech monster, které si hráč může vybrat.

Název vlastnosti	Popis	Jednotky
Radius	Určuje, jak daleko se mohou monstra v oblasti pohybovat.	0 – 40. 0= Monstra stojí na místě 40= Radius zabere polovinu planety.
Monstrum	Monstrum, které se bude ve spawneru pohybovat.	Výběr ze seznamu
Maximum of monsters	Maximální počet monster, která se v jednom okamžiku mohou nacházet ve spawneru. Pokud hráč jedno monstrum zabije, nové se zrodí v nejbližším intervalu, aby se udržel jejich počet.	Přirozené číslo. Doporučený interval (1 – 10)
Interval	Frekvence rození nových monster.	Sekundy. Doporučený interval (5 – 15)

Randomizer	Pravděpodobnost, s jakou se zrození monstra opozdí. Samotné opoždění se počítá samo.	0 – 1 0= Nikdy se neopozdí. 1= Vždy se opozdí.
Monsters on Spawn	Počet monster zrozených současně při vypršení intervalu.	0 – Maximum of monsters Pokud hráč zadá větší číslo, než je maximum monster, hra si s tím poradí a zrodí pouze tolik monster, kolik je jejich maximální limit.
Spawn only until 0	Monstra se budou rodit pouze v případě, že je ještě nějaké monstrum v oblasti naživu.	True/False
Attack only if attacked	Monstra nebudou na hráče útočit, pokud on na ně nezaútočí jako první.	True/False
Preset path	Monstrum se bude pohybovat po předem určené cestě.	True/False

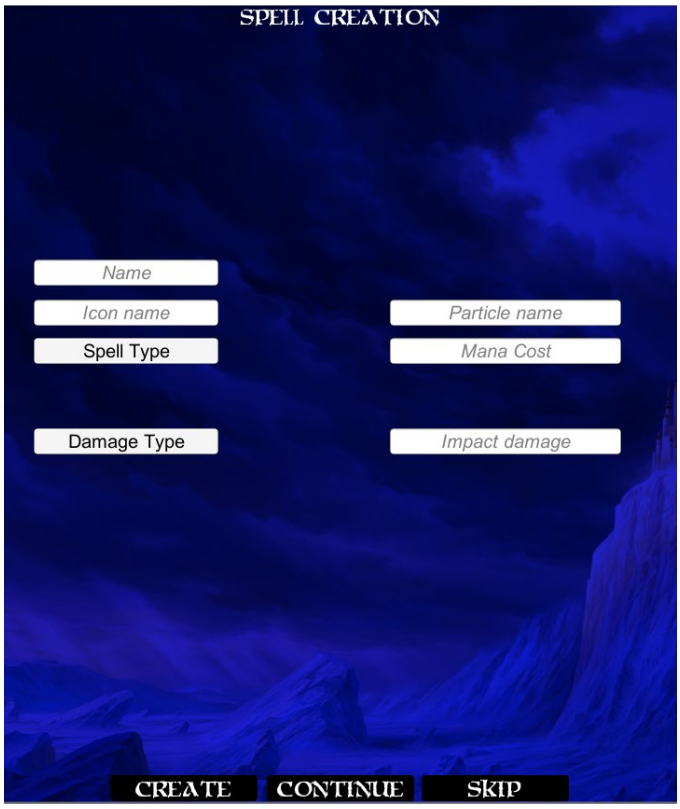
[Tabulka 7 – Vlastnosti spawneru]

V rámci tvorby spawneru je potřeba rozhodnout, zda se monstrum bude pohybovat náhodně nebo po předem dané cestě. Pokud se hráč rozhodne, že monstra budou chodit po předem dané cestě, tak po dokončení tvorby spawneru může hráč začít cestu tvořit. Kliknutím myši na planetu určuje body, po kterých monstra půjdou. Cestu je třeba potvrdit kliknutím na tlačítko Done, když je cesta dokončená.

Monstra se budou pohybovat v daném pořadí a vždy půjdou nejkratší cestou z předchozí destinace do následující.

7.3.7 Tvorba kouzel

Kouzla lze vytvářet na obrazovce kouzel, ke které je možné se dostat z hlavního menu kliknutím na položku editor a poté Spell Creation. Objeví se obrazovka znázorněná na obrázku 34.



[Obrázek 34 – Spell Creation]

Nejdříve designer zvolí název kouzla, jež chce vložit do hry a jež se nachází ve složce Resources (viz kapitola 7.3.8). Dále zadá absolutní cestu k ikonce kouzla. Poté je třeba vybrat typ kouzel a typ poškození. To nám vytvoří dodatečné položky na vyplnění.

Typ kouzla	Typ poškození	Popis	Nové položky
On The Spot	On Impact	Kouzlo, které se objeví na cílovém místě a zraní cíl jednou při kontaktu.	Spell Time
	While Touch	Kouzlo, které se objeví na cílovém místě a opakovaně zraní objekty, jež se ho dotýkají.	Spell Time, Additional Damage, Time Between Hits
	Poison	Kouzlo, které se objeví na cílovém místě a cíl otráví.	Spell Time, Additional Damage, Time Between Hits, Number of Additional Hits.

Flying	On Impact	Kouzlo, které poletí k cíli od zdroje a zraní cíl jednou při kontaktu.	
	While Touch	Kouzlo, které poletí k cíli od zdroje a opakovaně zraní objekty, jež se ho dotýkají.	Additional Damage, Time Between Hits
	Poison	Kouzlo, které poletí k cíli od zdroje a cíl otráví.	Additional Damage, Time Between Hits, Number of Additional Hits.

[Tabulka 8 – Typy kouzel]

Po vytvoření kouzel hráč potvrdí svou volbu kliknutím na tlačítko Create v dolní části obrazovky. Pokud si hráč tvorbu kouzel rozmyslí, stačí kliknout tlačítko skip a program ho pustí do hry.

Název položky	Popis	Jednotky
Name	Název kouzla, jež se nachází ve složce Resources (Více v kapitole 7.3.8)	Název souboru.
Icon Name	Absolutní cesta k ikoně kouzla.	Absolutní cesta.
Particle Name	Název kouzla, tak jak je v balíčku.	
Mana Cost	Kolik many kouzlo spotřebuje.	Přirozené číslo. Doporučený interval 1 – mana hráče/nepřítele. Implicitní kouzla mají 8 – 17
Impact Damage	Zranění při doteku prvního cíle.	Přirozené číslo. Doporučený interval 1 – 70. Implicitní kouzla mají 5 – 40.
Spell Time	Jak dlouho kouzlo zůstane na místě .(Jen pro kouzla typu On The Spot)	Přirozené číslo. Doporučený interval 1 – 10. Implicitní kouzla mají 2 – 10.
Additional Damage	Kolik kouzlo cíl zraní při dodatečných zraněních. (U jedu a While Touch)	Přirozené číslo. Doporučený interval 1 – 15. Implicitní kouzla mají 2 – 4.

Time Between Hits	Čas mezi dodatečnými zraněními.	Sekundy Doporučený interval 1 – 5. Implicitní kouzla mají 1 – 5.
Number of Additional Hits	Počet dodatečných zranění jedem.	Přirozené číslo. Doporučený interval 1 – 5. Implicitní kouzla mají 3.

[Tabulka 9- Vlastnosti kouzel]

7.3.8 Tvorba objektu a jeho načtení

Pro vytvoření nového kouzla nebo monstra je potřeba otevřít Unity projekt z přiloženého USB flash disku. K tomu je nutné mít nainstalovanou verzi Unity 5.2. nebo verzi, která je s ní zpětně kompatibilní. Následně se do Unity vloží objekt, který designer vytvořil, nebo je možné ho stáhnout z Unity obchodu (Asset Store). Pokud je objektem monstrum, je potřeba mu ještě dodat animátor s požadovanými proměnnými (více v kapitolách 3.1.6 a 4.6.2). Pokud je objektem kouzlo, je potřeba, aby na sobě mělo připojený collider. Poté se objekt musí přenést do složky Resources a v rámci ní kouzla do složky spells a monstra do složky monsters. Poté designer vytvoří spustitelný .exe soubor (File – Build & Run) a hra se spustí. Když bude designer vytvářet nové monstrum či kouzlo v jejich scéně, stačí zadat název objektu (u kouzel v položce name, u monster v položce Object name).

7.3.9 Tvorba nepřátel

Pro tvorbu nepřátel je potřeba v hlavním menu kliknout na položku editor a pak na AI Creation. Druhou variantou je projít některým z předchozích kroků editoru a postupně se přesunout k tvorbě nepřátel. Objeví se obrazovka na obrázku 35.

[Obrázek 35 – Tvorba nepřátel]

Pro přidání dalšího hráče stačí kliknout Add Player (na obrázku vyznačeno modrou šipkou s číslem 2). Objeví se nový řádek označený slovem Enemy a jeho číslem (na obrázku označené červenou šipkou s číslem 1). Nepříteli je možné upravit život, manu, poškození bez kouzel a barvu. Také je možné mu přiřadit Approach,což je přístup jakým se chová. Nepřítel má různé mise které bude vykonávat pokud má na ně dostatečné zdraví nebo manu. Vždy vykonává misi, která má největší priroritu. Aby se mohli různí nepřátele chovat jinak, může designer vytvořit nový Approach kliknutím na Add Approach (vyznačené fialovou šikpu s číslem 3).

[Obrázek 36 - Approach]

Objeví se okno na obrázku 36, kde jsou vypsané hlavní mise nepřítele. Nejdříve je vhodné přiřadit každé misi prioritu a poté misi vyléčení se maximální život, při kterém se nepřítel začne léčit. U Mise Protect castle, jaký život musí mít zámek, aby ho nepřítel šel bránit a u mise Get mana jaké maximální množství many

musí mít nepřítel, aby začal shánět novou manu. Poté musí deisgner potvrdit svůj výběr kliknutím na Create.

Potom co designer vytvoří nový Approach je třeba ho přiřadit nepříteli, tak že ho vybere v položce Approach v řádku nepřítele.

Výběr všech nepřátel designer ukončí kliknutím na tlačítko Continue (vyznačené bílou šipkou s číslem 4 na obrázku 35).

8 Příloha B – Struktura přiloženého USB flash disku

Na přiloženém USB flash disku se nacházejí čtyři složky: Projekt, Bin, Dokumentace a videa. Dále jsou zde nahrány dva soubory – ZnicteZamek.pdf, což je elektronická podoba této práce, a soubor readme.txt, kde je popsána struktura USB flash disku.

Složka Projekt obsahuje Unity projekt, který se dá otevřít v Unity.

Ve složce Bin se nachází spustitelný soubor znictezamek.exe, který spustí hru.

Ve složce Dokumentace je vygenerovaná dokumentace.

Ve složce videa se nachází videa hraní hry a úpravy hry v editoru.