VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HERNÍ DEMO V UNITY

GAME DEMO IN UNITY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **KRYŠTOF KOCIÁN**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ MILET**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Kocián Kryštof**
Program: Informační technologie
Název: **Herní demo v Unity**
Game Demo in Unity
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte herní engine Unity a techniky tvorby her.
2. Navrhněte herní demo.
3. Implementujte navrženou hru a vytvořte několik levelů.
4. Zhodnoťte a vytvořte demonstrační video.

Literatura:

- dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a kostra aplikace.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Milet Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 13. listopadu 2019

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací hry s procedurálně generovanými prvky v Unity. V práci lze najít shrnutí informací o videohrách, některé metody procedurální generace a popis návrhu a implementace hry samotné.

Abstract

The aim of this bachelor's thesis is to design and implement a game containing procedurally generated content in Unity. Inside you can find a summary of information about videogames, selected methods of procedural generation and a description of the design and implementation process of the game itself.

Klíčová slova

Unity, Unity3D, C#, počítačová hra, hra, 3D hra, tahová hra, herní engine, herní mechaniky, hledání cesty, procedurální generování, Perlinův šum, herní design, L-systém, uživatelské rozhraní, Shader Graph

Keywords

Unity, Unity3D, C#, computer game, game, 3D game, turn-based game, game engine, game mechanics, path finding, procedural generation, Perlin noise, game design, L-system, user interface, Shader Graph

Citace

KOCIÁN, Kryštof. *Herní demo v Unity.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Milet

Herní demo v Unity

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Mileta. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Kryštof Kocián
7. června 2020

Poděkování

Chtěl bych poděkovat panu Ing. Tomáši Miletovi, za odborné vedení mé práce a toleranci k mým nedokonalým časovým plánům. Dále bych chtěl poděkovat Petře Fedrselové a Iloně Dománek, za cenné rady a Danielu Kunstovi za jeho čas a práci na grafické stránce práce.

Obsah

Kapitola 1

Úvod

Hraní her se pomalu, ale jistě stává jednou z hlavních kratochvílí mnoha lidí na světě, například ve Spojených státech amerických hraje 60 % lidí videohry denně. Také proto měl herní průmysl v roce 2019 hodnotu více než 150 miliard amerických dolarů [13].

S větším zájmem a vidinou výdělku roste množství lidí, kteří vyvíjejí hry, ať už ve velkých herních studiích, v týmech čítajících stovky lidí nebo malých nezávislých studiích o několika málo lidech. V kombinaci s tím, že existuje čím dál větší množství veřejně dostupných nástrojů, je tvorba videoher jednodušší než kdy dřív, a proto vznikají masy her, z nichž některé jsou z dílen týmů složených jen z několika málo jedinců. Existují dokonce i úspěšné hry, které vyvinul jeden člověk, jako například hra Minecraft, kterou zpočátku vyvíjel pouze Markus „Notch“ Persson.

I když se od útlého věku stýkám s videohrami, jejichž hraní by se dalo považovat za můj hlavní koníček a moje akademická kariéra již dlouho zahrnuje programování, ještě nikdy jsem žádnou hru nevytvořil. A proto když přišel čas na výběr témat bakalářských prací, bylo velmi lákavé tyto dva aspekty svého života spojit a pokusit se o vytvoření počítačové hry.

Každý hráč počítačových her má někde v hlavě představu o své ideální hře, která by splňovala všechna jeho očekávání. Cílem této práce je vytvořit hru v Unity, která by se co nejvíce přiblížila mojí představě.

Hry jsou velmi unikátní kombinací technologií a umění, ale protože já jsem technicky zaměřený člověk, jsem rád že jsem některé umělecké aspekty vývoje této hry mohl přenechat na svém umělecky založeném kolegovi a věnovat se pouze technické stránce věci.

Součástí této práce budou:

všeobecné informace o hrách a jejich žánrech v kapitole 2,

vývoj her a iterativní přístup k němu v kapitole 3,

kategorie herních prvků budou popsány v kapitole 3.4,

v kapitole 4 budou představeny různé metody a principy používané při procedurálním generování, jako jsou generátory pseudonáhodných čísel, Perlinův šum nebo L-systémy

a v poslední teoretické kapitole 5 budou uvedeny systémy pro hledání nejkratších cest využívané v umělé inteligenci a hrách.

V kapitole 6 bude nastíněn proces návrhu hry, která je objektem této práce.

V následující kapitole 7 budou představeny zajímavější pasáže z implementace hry

a v poslední kapitole 8 bude zhodnocen průběh celé práce.

Kapitola 2

Počítačové hry

V této kapitole bude popsáno co je to hra, a do jakých žánrů se počítačové hry nejčastěji kategorizují.

Co je to hra?

Hry a herní návrhářství celkově je obor s nepříliš rozvinutým a standardizovaným slovníkem pojmů a definic. To je důvod, proč lze taky v literatuře nalézt mnoho rozdílných formulací.

Avedon a Sutton-Smith hru definují následovně „Hra je dobrovolné konání činnosti, která sestává ze střetu schopností, je omezená pravidly, a dosahuje nerovnovážného výsledku.“ [2, str. 7]. Chris Crawford ji definuje jako „Uzavřený formální systém, který subjektivně reprezentuje podmnožinu reality.“ [6, str. 7], podobně Tracy Fullerton říká, že hra je „Uzavřený formální systém, který zapojuje hráče do strukturovaného konfliktu. Tento ne zcela předvídatelný konflikt potom vrcholí nerovnovážným výsledkem.“ [12, str. 43].

Jednodušší popis hry nabízí například Jesse Schell „Hra je činnost zaměřená na řešení problémů, s cílem pobavit se.“ [25, str. 37].

A něco takového by mělo být výstupem této práce. Zábavná výzva.

2.1 Herní žánry

Jednou z nejdůležitějších charakteristik počítačové hry je její zařazení do herních žánrů. Některé hry se charakterizují kombinací více žánrů anebo vůbec nemusejí zapadat do tradičních kategorií, ale většinu her lze, alespoň zhruba, charakterizovat jedním ze základních žánrových označení.

Akční

Akční hry většinou probíhají v reálném čase a kladou důraz na reakční dobu a koordinaci zraku a ruky [12, str. 416]. Často obsahují násilí a mají rychlejší tempo, aby hráči navodily adrenalinový pocit z akce.

Sub-žánry:

- Akčně-dobrodružné hry - The Legend of Zelda, God of War 2.1a, ...
- Bojové hry - Tekken, Mortal Kombat, ...
- Plošinové hry - Super Mario Bros. 2.1b, Crash Bandicoot, ...
- Rytmické hry - PaRappa the Rapper, Beat Saber, ...
- Střílečky - Doom 2.1c, Call of Duty, Max Payne, ...

(a) God of War (2018)

(b) Super Mario Bros. (1985)

(c) Doom (1993)

Obrázek 2.1: Akční hry

Simulace

Simulační hry se zaměřují na reprodukci reálné činnosti. Čím vážnější je daná simulační hra, tím více pozornosti je kladeno na realističnost a přesnost [3, str. 9].

Sub-žánry:

- Budovatelské / Managementové - SimCity, Cities: Skyline, Theme Park, ...
- Sportovní - NHL 2.2a, FIFA, PES, UFC, NBA2K, ...
- Závodní - Project Cars, DiRT Rally, Gran Turismo, ...
- Ostatní - Farming Simulator, Euro Truck Simulator, The Sims, Goat Simulator 2.2b, Sim Ant, ...

(a) NHL 20 (2019) (b) Goat Simulator (2014)

Obrázek 2.2: Simulační hry

RPG

Neboli role-playing games, což v překladu znamená hry na hrdiny. Jedním z nejdůležitějších prvků RPG je silný příběh a dominantní mechanikou je zlepšování svého hrdiny nebo družiny hrdinů a jejich vybavení [3, str. 7]. RPG hry často před hráče kladou morálně náročné volby a mnohdy obsahují systém morality, kde se hráč svými akcemi stává kladným nebo naopak záporným hrdinou.

Moderní RPG často nesledují lineární příběh, ale nechávají hráče volně prozkoumávat otevřený svět obsahující množství spletitých a prolínajících se příběhů. Do tohoto žánru spadají například následující hry: Mass Effect, The Witcher, Divinity: Original Sin 2.3a, ...

Všudypřítomným pod-žánrem jsou MMORPG - massively multiplayer online RPG, což lze volně přeložit jako online hry na hrdiny s masivním množstvím hráčů. Momentálně nejhranějšími MMORPG jsou World of Warcraft 2.3b, RuneScape, Final Fantasy XIV, ... [4]

Pro nás důležitým podžánrem jsou rogue-like hry, které se vyznačují procedurálně generovaným prostředím, tahovým systémem a permanentní ztrátou svojí postavy po její smrti. Často se odehrávají v jeskyních a kobkách, které lze jednoduše generovat. Svůj název získaly svou podobností nejstarší hře tohoto žánru *Rogue*.

(a) Divinity: Original Sin II (2017)

(b) World of Warcraft (2004)

Obrázek 2.3: RPG hry

Strategie

Strategické hry, jak už jejich název napovídá, kladou důraz na strategické a taktické myšlení, a řízení jednotek a zdrojů [12, str. 416]. Důležitou součástí je umělá inteligence (AI - Artificial Intelligence), která hraje proti hráči. Samotná herní mapa je často složena z menších herních dílů, a proto ji lze jednoduše procedurálně generovat [10, str. 182].

Sub-žánry:

- RTS[1] - Warcraft, Age of Empires, Starcraft, Dune 2.4a, ...
- Turn-based[2] - XCOM 2.4b, Mutant Year Zero: Road to Eden, Darkest Dungeon, ...
- 4X[3] - Sid Meier's Civilization, Galactic Civilizations, Master of Orion, ...

(a) Dune 2000 (1998)

(b) XCOM: Enemy Unknown (2012)

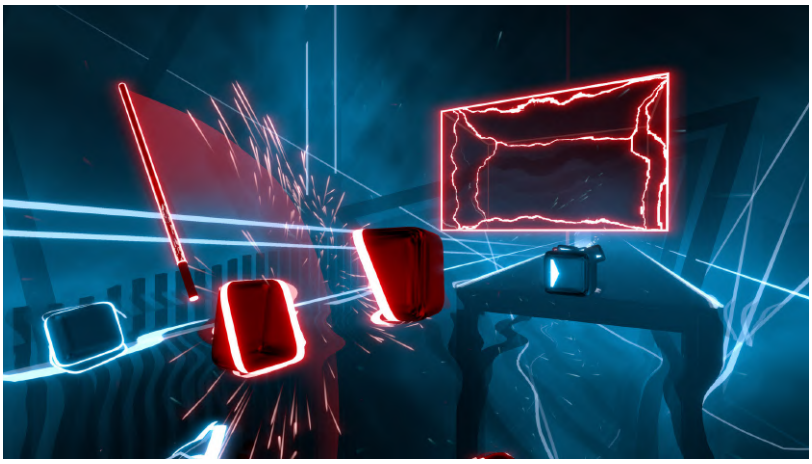
Obrázek 2.4: Strategické hry

[1] Real-time Strategy - strategické hry probíhající v reálném čase
[2] Tahové strategické hry
[3] "eXplore, eXpand, eXploit, and eXterminate", neboli "průzkum, expanze, využití a vyhlazení"

Ostatní

Kromě zmíněných základních žánrů existují další méně prominentní, které jsem zařadil do kategorie „Ostatní“.

- Oddechové - Angry Birds, Candy Crush, Mountain, ...
- Hudební / rytmické - Rocksmith, Audiosurf, Guitar Hero, ...
- Puzzle - The Room, Scarlett Mysteries, ...
- VR[4] - Beat Saber 2.5, Half-Life: Alyx, ...
- Vzdělávací
- ...



Obrázek 2.5: Beat Saber (2018)

[4]Virtuální realita

Kapitola 3

Postup herního vývoje

Návrh her je činnost tvorby hry, která se skládá z velkého množství malých rozhodnutí. Zahrnuje nastavení pravidel, mechanik, příběhu, grafické stránky, hudby a mnoha dalších aspektů hry.

Vývoj hry je samotná implementace tohoto návrhu pro libovolnou platformu. Od karetních a deskových her, až po ty počítačové.

Celý postup vývoje počítačové hry se skládá z následujících kroků: konceptualizace, návrh, tvorba prototypu, programování a testování. Obecně se doporučuje iterativní přístup k tvorbě her, tedy tvorba jednoduchých prototypů, jejich vyzkoušení, a v případě spokojenosti, jejich kompletní implementace [12, str. 148].

3.1 Konceptualizace a návrh

Hra začíná nějakým nápadem, nebo zajímavým konceptem, jako třeba: „realistický simulátor pirátství“, „souboj narvalů na život a na smrt“, nebo „moderní slovanský čert“, takhle jednoduše může začít tvorba hry.

Po nalezení zajímavé myšlenky je potřeba navrhnout základní charakteristiky hry. Tyto prvky lze rozdělit do čtyř kategorií: mechaniky, příběh, vzhled a technologie, které budou více probrány v kapitole 3.4 [25, str. 41].

3.2 Prototypování a programování

Jakmile má vývojářský tým jasně vytyčený směr, může začít pracovat na implementaci. Ať už jde o tvorbu modelů, psaní příběhových linek nebo programování funkcionality hry, doporučuje se vždy a často tvořit prototypy zamýšlených výtvorů.

Prototyp je funkční model prvku (nové mechaniky, hudební skladby, herní úrovně, ...), který slouží k vyzkoušení proveditelnosti a nalezení možných nedostatků dané součásti. Hlavním cílem tvorby prototypu je hrubý, ale funkční nástin potenciální součásti hry. Tvorba her není jednoduchá a není vždy jasné, co se bude hodit do komplexního celku mnoha prvků. Z tohoto důvodu je lepší rychle vytvořit jednoduchou verzi modelu nebo mechaniky, vyzkoušet ji a rozhodnout, zda stojí za to dotáhnout daný prvek do konce, nebo se ho zbavit [12, str. 175].

Sid Meier je mezi herními návrháři známý tvrzením, že jádro nové hry vždy rozběhne co nejdříve, i kdyby v něm měli být jen hůlkové postavičky[1], aby hru mohl v průběhu celého vývoje hrát [3, str. 226].

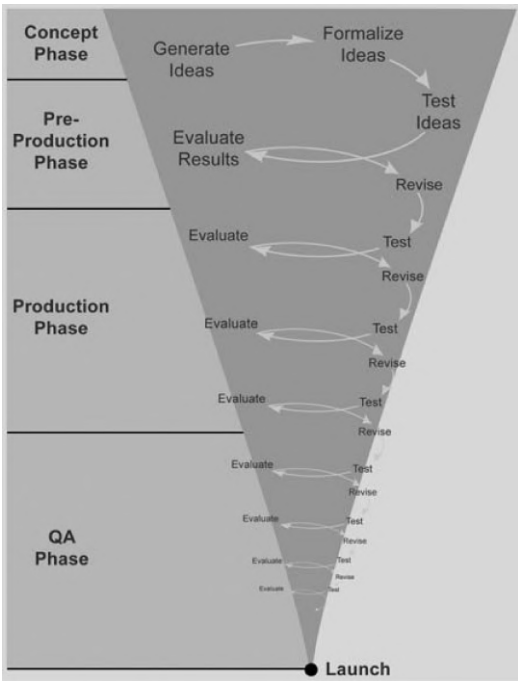
Obrázek 3.1: Na horním obrázku vidíme prototyp jedné z úrovní hry Spore z roku 2008, na spodním obrázku je potom hotová úroveň

[1] v anglickém originále „stickmen“

3.3 Testování hraním

Hraní vlastní hry, v průběhu jejího vývoje, je extrémně důležité, protože herní návrhář musí neustále kontrolovat, zda hra funguje kýženým způsobem, a že je vyvážená a zábavná. Může se zdát, že testovat hru na začátku jejího vývoje je zbytečné a bylo by lepší počkat, až bude hotový nějaký konkrétní produkt. Ale je skutečně žádoucí hru testovat v každé iteraci vývojového cyklu.

Kontinuální iterativní proces testování, vyhodnocování a revidování, je jedním ze způsobů, jak udržet vývoj hry na cestě k úspěšnému cíli. Na obrázku 3.2 je vidět, jak se testovací cyklus postupně zužuje, jak produkce postupuje kupředu. Největší změny se logicky odehrávají na začátku a postupem času jsou změny menší a specifičtější.



Obrázek 3.2: Iterativní vývojový cyklus [12, str. 249]

Samo-testování

Od první fungující verze svojí hry ji návrhář nebo celý tým opakovaně zkouší, aby pochopili jak hra funguje, našli chyby v existujících součástech a vymýšleli nové prvky. Samotestování je užitečné v průběhu celého životního cyklu hry, jak projekt roste, je vhodné nechat hru otestovat lidmi, kteří nebyli zahrnuti v procesu její tvorby.

Testování známými

Po utvoření stabilního základu hry a funkčního prototypu je přirozené jej sdílet s rodinou, přáteli nebo jinými známými mimo vývojářský tým. V této fázi není hra ještě nutně jednoduše pochopitelná, a proto bývá toto testování doprovázeno někým z týmu, kdo může například pravidla hry vysvětlit. Toto testování nabízí nový pohled na hru, a tím pádem si mohou všimnout nedostatků, které doposud nebyly vyneseny na světlo. Jak ale vývoj postupuje dál, je dobré prozkoumat další možnosti.

Testování cizími lidmi a cílovou skupinou

Jakmile se hra stane hratelnou a pochopitelnou se vším všudy, lze do procesu testování zahrnout cizí lidi. Tito noví lidé přinesou další čerstvý pohled na současný stav a vzhledem k tomu, že nemají co ztratit, tím že vyjádří svůj upřímný názor, může vývojářský tým získat úplně nové poznatky. Malé vývojářské týmy v dnešní době často využívají možnosti svůj produkt sdílet na sociálních platformách, jako je například Reddit, k získání lidí, kteří by potenciálně měli zájem o jejich produkt, a jejich zanesení do procesu testování demo verzí svých her [12, str. 248].

3.4 Čtyři základní prvky

Existuje mnoho různých způsobů jak rozřadit herní prvky do kategorií. Dělení na čtyři základní prvky je jeden z nich. Toto členění používá Jesse Schnell a bude použito i v této práci.

Herní mechaniky

Mechaniky jsou stavební bloky každé hry a tvoří její jádro. Řadíme do nich prvky jako jsou: pravidla hry, její prostředí a pohyb v něm, interakce hráče s hrou a stanovení herních cílů. Herní mechaniky jsou často asociovány s určitými žánry, například:

- Strategické hry využívají diskretizovaný prostor (rozdělený rovnoměrně na pole určitého tvaru) častěji, než třeba akční nebo sportovní hry.
- Sportovní hry mají časové limity častěji, než například RPG.
- RPG a akční hry mají, na rozdíl od závodních her, propracovaný bojový systém.
- ...

Příběh a umělecká stránka

Příběh může ve videohrách dostat několik odlišných podob, může být lineární a fungovat jako hlavní motivace k hráčovu postupu. Na druhou stranu v otevřenějších hrách je často použit nelineární příběh s větvícími se linkami, které nabádají hráče k vícenásobnému dohrání hry. V některých hrách příběh může úplně (nebo téměř úplně) chybět. Takové hry jsou většinou tzv. sandboxové hry (například Minecraft) nebo hry, které jsou zamýšlené k vlastní interpretaci hráči a nabízí pouze malé kousky informací o herním světě, jako to dělá například série Souls od společnosti From Software.

Umělecká stránka hry zahrnuje vzhled, zvuky, hudbu, atmosféru, ... Tyto prvky jsou velmi důležité a výrazně ovlivňují hráčův zážitek ze hry [25, str. 42]. Estetická stránka hry musí souznít s příběhem, mechanikami a může utvářet hlavní podstatu tvořeného herního světa.

Technologie – herní enginy

Technologiemi zde myslíme prostředky, které nám umožní hru hrát. Může jít o tužku a papír (Dračí doupě), plastové postavičky nebo speciální světelné vybavení (Laser tag). V našem případě jde o počítače, programovací jazyky, knihovny a herní enginy[2].

Herní engine je pojem, který se poprvé objevil v devadesátých letech v souvislosti s FPS hrami, jako je například Doom od společnosti id Software. Doom měl dobře definovanou hranici mezi základní funkcionalitou (trojrozměrné vykreslování, systém detekování kolizí, systém pro práci se zvukem, ...) a grafickou stránkou, rozložením úrovní a herními pravidly, která utvářela hráčův zážitek ze hry. Tato separace se ukázala být výhodnou, když vývojáři začali prodávat licence ke svým hrám a začali tvořit nové hry, pouze vytvořením nových assetů[3] a minimální úpravou svých původních her [15, str. 11].

[2] Game engines
[3] uměleckých prvků

Asi jako je zbytečné vytvářet si vlastní programovací jazyk, je zbytečné vytvářet svůj vlastní herní engine. Samozřejmě to není zbytečné vždy, ale je to velmi komplexní program a jeho tvorba nemusí stát za to, ledaže od svého herního enginu potřebujete speciální funkcionalitu, kterou žádný na trhu dostupný produkt nepodporuje. Použitím některého z veřejně dostupných herních enginů může malý vývojářský tým ušetřit velké množství času a pustit se rovnou do vývoje své hry.

Vybrat si správný herní engine může být rozhodující, protože každý s sebou nese určitá omezení – některé enginy jsou optimalizované na určitý žánr her, některé jsou Open Source a jiné jsou vhodnější pro menší týmy.

Žánrově zaměřené enginy:

- Adventure Game Studio – Point & Click adventury.
- Doom engine, Quake engine
  - 3D FPS hry na jednopodlažních úrovních.
  - Quake, Hexen, Doom, Strife
- M.U.G.E.N – 2D bojové hry.
- RPG Maker
  - RPG a JRPG[4] hry s pohledem z vrchu.
  - To the Moon, Doom & Destiny

Herní enginy pro ne-programátory: Existuje relativně široká nabídka herních enginů, které se propagují jako vhodné pro začátečníky a většinou v nich lze vytvořit hru bez nutnosti napsat jediný řádek kódu. Jako náhražka programování se používá tzv. vizuální nebo taky „Drag and Drop“ programování (například vizuální programovací jazyk Scratch) [11].

- Buildbox – Mobilní hry, převážně pro zařízení Apple.
- Construct – 2D hry v HTML5, vhodné pro prohlížeče.
- GameMaker
  - Asi nejznámější z této kategorie.
  - Používá vizuální programování, doplněné vlastním skriptovacím jazykem GML.
  - Hyper Light Drifter, Hotline Miami, Minit, Gunpoint, Risk of Rain

Proprietární herní enginy: Větší společnosti někdy při tvorbě hry vytvoří i herní engine a pracují s ním i na dalších svých hrách. Následuje ilustrativní výčet několika z nich:

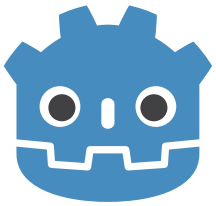
- REDEngine – CD Projekt – The Witcher 2 a 3
- RAGE[5] – Rockstar – Grand Theft Auto IV a V, Red Dead Redemption 1 a 2, Max Payne 3, ...

[4] RPG v japonském stylu
[5] Rockstar Advanced Game Engine

- Frostbite – DICE – Battlefield, FIFA, Need for Speed, Star Wars: Battlefront, ...
- id Tech – id Software – Doom, Doom Eternal, Wolfenstein, ...
- AnvilNext – Ubisoft – Série Assassin's Creed, For Honor, ...
- Fox Engine – Konami – Metal Gear Solid, Pro Evolution Soccer, P.T.[6]
- Source Engine – Valve – Half-Life 2, Left 4 Dead, Portal, CS: GO, DotA 2, ...
- Creation Engine – Bethesda – TES V: Skyrim, Fallout 4, Fallout 76

Pro veřejnost dostupné herní enginy: Většina lidí se před zahájením vývoje svojí hry rozhoduje nad jedním z následujících enginů. Nejpopulárnější možnosti jsou Unity a Unreal Engine, ale existují i jiné možnosti a několik z nich bude v následujících odstavcích představeno.

Godot



Herní engine na který jste čekali. Godot nabízí řadu nástrojů, abyste se mohli soustředit na tvorbu Vaší hry, a ne na znovuobjevování kola. Godot je úplně zadarmo a open-source pod velmi liberální MIT licencí. Bez závazků, bez poplatků, nic. Vaše hra je jen Vaše, až do posledního řádku kódu herního enginu [14].

Klíčové informace

Cena:	Zdarma, bez licenčních poplatků a bez procentuální platby z výdělku.
Zdrojový kód:	Dostupný na `https://github.com/godotengine/godot`
Programovací jazyky:	GDScript, VisualScript, C#, C++, ...
Vydané hry:	The Interactive Adventures of Dog Mendonça & Pizzaboy, RPG in a Box, ...

[6] playable teaser k později zrušené hře Silent Hills

CryEngine

Dosáhněte Vaší vize. Nejvýkonnější platforma pro vývoj her, která Vám a Vašemu týmu umožní tvořit hry na světové úrovni [7].

Klíčové informace

Cena:	Při roční tržbě větší než 5 000 $ si Crytek nárokuje 5 % z vydělané částky.
Zdrojový kód:	Dostupný na https://github.com/CRYTEK/CRYENGINE
Programovací jazyky:	C++
Vydané hry:	Far Cry, série Crysis, Kingdom Come: Deliverance, Hunt: Showdown, ...

Amazon Lumberyard

Dostupný zdarma. Výkonný. Plně přizpůsobitelný. V Amazonu věříme, že si herní vývojáři zaslouží další alternativu. Proto vyvíjíme Lumberyard: herní engine bez poplatků, bez nároku na procentuální podíl z prodeje, s bezproblémovou integrací s AWS[7] a Twitch, a ještě mnohem víc [1].

Klíčové informace

Cena:	Zdarma, platíte pouze za využití služeb AWS.
Zdrojový kód:	Dostupný na https://github.com/aws/lumberyard
Programovací jazyky:	Script Canvas, Lua
Vydané hry:	Star Citizen, New World, ...

Unreal Engine

Svět plný možností. Vytvořte nejmodernější interaktivní a pohlcující herní zážitky s nejotevřenější a nejpokročilejší real-time 3D vývojovou platformou na světě. Navrženo pro týmy všech velikostí a hry všech žánrů, Unreal Engine stojí za některými z nejúspěšnějších her všech dob [28].

Klíčové informace

Cena:	Platíte 5 % z vydělané částky nad 1 000 000 $, ale pouze pokud v daném čtvrtletí Vaše hra vydělá alespoň 10 000 $. Jinak je Unreal Engine zdarma.
Zdrojový kód:	Dostupný na https://github.com/EpicGames/UnrealEngine
Programovací jazyky:	C++
Vydané hry:	Borderlands Series, Batman Arkham Series, XCOM, Bioshock, Deus Ex, Mass Effect 1 a 2, Lineage II, Gears of War, Fortnite, PUBG, ...

[7] Amazon Web Services – on-demand cloudové služby

Unity

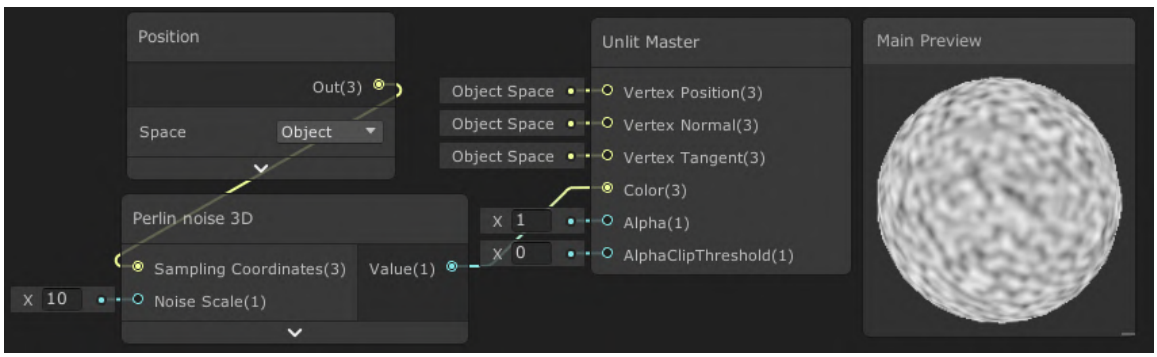
Vytvořte svět plný života. Přiveďte svou hru k životu s nejkompletnější a nejflexibilnější real-time vývojovou platformou. Unity Vám umožní rychle vytvořit, lehce ovládat, a monetizovat Vaši hru [27].

Klíčové informace

Cena:	Při výdělku nižším než 100 000 $ ročně je možné využívat Unity zcela zdarma. Při vyšších příjmech je potřeba zakoupit Unity licenci pro každého člena týmu, která stojí 40 $ měsíčně nebo 150 $ měsíčně, pokud výdělek přesahuje 200 000 $ za měsíc.
Zdrojový kód:	Vybrané sekce dostupné na `https://github.com/Unity-Technologies`
Programovací jazyky:	C#
Vydané hry:	Hollow Knight, Cuphead, Escape from Tarkov, Monument Valley 2, The Long Dark, ...

Tato bakalářská práce je vytvořena s pomocí Unity. Tento herní engine byl zvolen kvůli jeho podpoře C#, zdarma dostupným assetům na Unity Asset Store a množství videí a online příspěvků týkajících se Unity.

Shader Graph V této práci je využita Universal Render Pipeline (URP), což je jedna z grafických pipeline, pro kterou je dostupný vizuální nástroj pro tvorbu shaderů jménem Shader Graph. Shadery se v něm tvoří spojováním různých uzlů do grafu. Nástroj sám o sobě obsahuje velké množství uzlů od sčítání přes šumy až po barevné filtry, ale lze jej rozšířit o vlastnoručně vyrobené nebo stažené uzly napsané ve formátu *.hlsl*. V této práci jsou využity Shader Graph uzly z repozitáře Noisy-Nodes[8], který obsahuje implementace prostorových šumů.



Obrázek 3.3: Příklad jednoduchého shaderu, který na objekt aplikuje barvu podle hodnoty trojrozměrného Perlinova šumu vytvořeného v Shader Graphu

[8] `https://github.com/JimmyCushnie/Noisy-Nodes`

Kapitola 4

Metody procedurálního generování

Procedurální generování je alternativa k ruční tvorbě herních prvků. Místo herního návrháře provede práci počítačem provedený algoritmus. Abychom ale úplně neztratili kontrolu nad tímto procesem, bývá žádoucí do něj zavést parametry, jimiž jej návrháři mohou ovlivňovat [18].

Jde taky o úsporu místa, protože místo velkého množství textur, hudby, animací nebo map úrovní, můžete mít jen několik algoritmů, které mohou vygenerovat velké množství daných prvků. Samozřejmě, namísto zabrání místa na nosiči, je ale zatíženo CPU, které musí algoritmus provést a vygenerovat co je potřeba.

Aby byl algoritmus schopný tvořit velké množství unikátních výstupů, musí využívat generátory náhodných čísel, které, když jsou správně aplikovány, ovlivní výstup tak, aby byl unikátní, ale v blízkosti očekávaného výstupu. Generátory skutečně náhodných čísel jsou ale pomalé, a proto se ve většině počítačových aplikací používají generátory pseudonáhodných čísel.

4.1 Generátory pseudonáhodných čísel

Generátory pseudonáhodných čísel generují zdánlivě náhodná čísla, ale ve skutečnosti jde o deterministický proces, který lze reprodukovat pokud známe současný stav generátoru.

Nejčastěji používaným algoritmem PRNG[1] je takzvaný lineární kongruentní generátor[2]:

$$x_{x+1} = (ax_i + b) \bmod m \tag{4.1}$$

kde

- operace *mod* znamená zbytek po celočíselném dělení,
- a, b a m jsou vhodně zvolené konstanty. [22, str. 18]

LCG nejsou příliš kvalitní, pro většinu aplikací jsou však dostačující. Pokud je ale potřeba kvalitnější generátor, existují alternativy jako například: WELL[3], Mersenne Twister nebo Xorshift.

[1] Pseudo-random number generator, neboli generátor pseudonáhodných čísel
[2] LCG – Linear Congruential Generator
[3] Well Equidistributed Long-period Linear

Xorshift

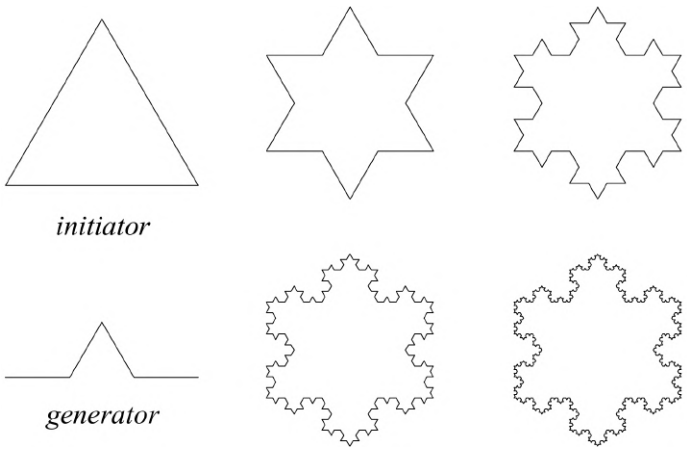
Jak už název této skupiny generátorů napovídá, využívají ke generování čísel operaci *xor*[4] mezi číslem a jeho další verzí, na kterou byla aplikována operace *shift*[5] [19]. Tento druh generátoru navrhl George Marsaglia a patří mezi nejrychlejší generátory, které nejsou kryptograficky bezpečné. Unity využívá Xorshift128+, což je upravená verze, která se sčítáním vyhýbá slabině xorshiftu v nízkých bitech a uchovává stav o délce 128 bitů.

4.2 L-systémy

L-systémy, nebo taky Lindenmayerovy systémy, jsou paralelní přepisovací systémy a varianta formálních gramatik, které v roce 1968 vyvinul maďarský biolog Aristid Lindenmayer k popisu růstu rostlin.

L-systém se skládá z neprázdné abecedy symbolů, konečné množiny přepisovacích pravidel a počátečního stavu, zvaného *axiom*. Ústředním konceptem je přepisování, pomocí něhož generují komplexní objekty opakovaným nahrazováním částí řetězce podle daných přepisovacích pravidel. Výsledné řetězce lze potom libovolně interpretovat [24, str. 1].

Na obrázku 4.1 je Kochova vločka, kterou lze s pomocí L-systémů jednoduše reprodukovat.



Obrázek 4.1: Kochova vločka, vlevo nahoře je počáteční stav a vlevo dole je prvek, kterým se v každém kroku nahradí všechny úsečky. Na obrázku potom vidíme taky výsledky prvních čtyř kroků. Převzato z [24, str. 2]

Typy L-systémů

Základní L-systémy jsou typu D0L, což znamená, že jsou deterministické a bezkontextové. Existují ale i další typy L-systémů a jsou charakterizovány kombinací následujících vlastností:

- Závorkové – Umožňují větvení do podřetězců, většinou značeno pomocí znaků [a].
- Kontextové – IL systémy – Pravidla berou v potaz také okolí, což znamená že pro přepsání určitého řetězce musí být splněny další podmínky aplikované na jeho okolí.

[4] „exclusive or“, do češtiny překládáno jako „bitová nonekvivalence“
[5] bitový posun

Existuje levý a pravý kontext. Levý kontext jsou předcházející řetězce a pravý kontext jsou řetězce následující.

- Stochastické – Zahrnují prvek náhody ve svých pravidlech
- Parametrické – Umožňuje asociaci parametrů k jednotlivým znakům, jejich změny a čtení jejich hodnot k určování dalšího postupu [17].

4.3 Šumové funkce

Šumy se v počítačové grafice používají, protože přidávají prvek náhody. K tomuto účelu lze použít i generátory náhodných čísel, ale jejich výsledky nevypadají moc přirozeně, a proto se používají šumové funkce, které tuto nedokonalost potlačují. Šumové funkce jsou v podstatě generátory náhodných čísel, které se volají s parametrem, ale když použijeme stejný parametr, dostaneme vždy stejný výsledek [9].

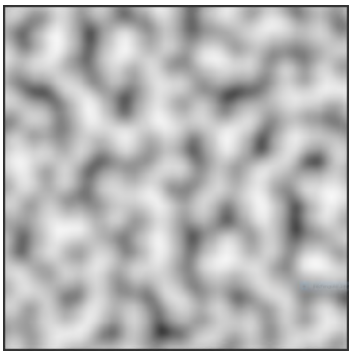
Existuje několik kategorií šumových funkcí, ale v této práci zmíníme pouze dvě gradientní funkce.

Perlinův šum

Ken Perlin jej vyvinul v roce 1983 a zveřejnil v roce 1985. V roce 1997 za něj dostal „Cenu za vědecký a technický přínos“ od americké Akademie filmového umění a věd, protože umožnil generovat přirozeně vypadající textury pro filmové plátno [16].

Perlinův šum může být implementován v libovolném množství dimenzí. Postup je typicky následující:

1. vytvořit n-rozměrnou mřížku s náhodnými vektory,
2. vypočítat skalární součin pro každý vektor a jeho sousedy,
3. interpolovat jejich hodnoty.

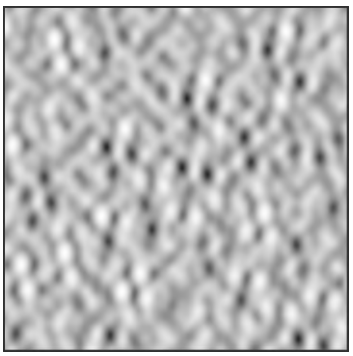


Obrázek 4.2: Perlinův šum, vygenerovaný v Unity Shader Graph

Simplex noise / OpenSimplex noise

V roce 2001 Ken Perlin přišel s lepší šumovou funkcí, která měla vyřešit nedostatky té původní. Šumová funkce Simplex je oproti Perlinově šumové funkci izotropní, efektivnější v počítání pseudonáhodných gradientů, neprodukuje viditelné artefakty, je jednodušší na výpočet, ... [23].

Ken Perlin si tuto šumovou funkci nechal patentovat[6] a k obejití tohoto problému vznikla funkce OpenSimplex, která není identická s funkcí Simplex, ale taky řeší většinu problémů původního Perlinova šumu [26].



Obrázek 4.3: Šum OpenSimplex, vygenerovaný v Unity Shader Graph

[6] https://patents.google.com/patent/US6867776

Kapitola 5

Pathfinding

Neboli hledání nejkratší cesty je důležitá součást velkého množství videoher, ať už jde o nepřátele s umělou inteligencí nebo hráči „nepřímo“ ovládanou postavu (tím se myslí, že hráč nekontroluje každý pohyb, ale pouze zvolí cílovou pozici a postava se na ni sama přemístí).

Problém hledání nejkratší cesty ze startu do cíle je stále aktuální, protože pořád rostou nároky na kvalitu a rychlost algoritmů řešících tyto úlohy [5].

Hledání nejkratší cesty znamená najít na mapě „nejlepší“ cestu z bodu A do bodu B, kde ale nejlepší cesta může znamenat více věcí: nejkratší, nejrychlejší, nejjednodušší, nejbezpečnější, ... a vyhnout se překážkám, které po cestě potká.

Algoritmy na hledání cest nepracují přímo s herní geometrií, ale s nějakou její reprezentací, kterou může být pole hodnot s předem danými cenami přechodů nebo graf s uzly a hranami.

Konvenční algoritmy jako Breadth First Search (prohledávání do šířky), Depth First Search (prohledávání do hloubky) nebo Iterative Deepening Search (prohledávání do hloubky s postupným zanořováním) byly ve hrách převážně nahrazeny metodou A*, která vylepšuje Dijkstrův Algoritmus o heuristickou funkci jako je použita v algoritmu Greedy Best First Search (hladové prohledávání nejlepšího) [8].

5.1 Dijkstrův algoritmus

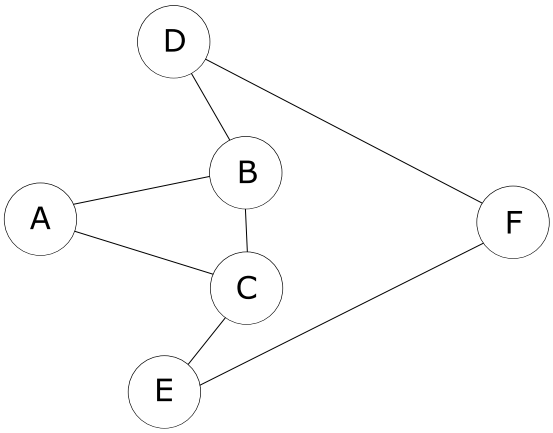
Dijkstrův algoritmus je úplný a optimální. Úplný znamená že jestli existuje cesta z A do B, tak ji vždy najde. Optimální znamená že vždy najde nejlepší možnou cestu.
Postup Dijkstrova algoritmu je následující:

1. Přidá všechny uzly do seznamu otevřených a nastaví jim nekonečnou hodnotu, kromě začátečního uzlu, kterému nastaví hodnotu 0.
2. Vezme ze seznamu uzel s nejnižší hodnotou, odstraní ho z něj a přidá ho do seznamu uzavřených. Pokud je tento uzel zároveň cílem úspěšně nalezl nejlepší cestu. Pokud je seznam otevřených prázdný, cesta k cíli neexistuje.
3. Postupně se přemístí do všech uzlů, které sousedí s tím aktuálním a změní jejich hodnotu na hodnotu toho současného + cenu daného přechodu, pokud ovšem uzel už nemá hodnotu nižší.
4. Opakovat od kroku 2.

5.2 Greedy Best First Search

GBFS algoritmus je úplný, ale není optimální, což znamená, že nemusí vždy najít tu nejlepší cestu. Funguje stejně jako Dijkstrův algoritmus, ale pro získání hodnoty jednotlivých uzlů, namísto vzdálenosti uražené od původní pozice, používá heuristickou funkci, která nějakým způsobem kvantifikuje vzdálenost od cíle. Kvalita heuristiky výrazně ovlivňuje kvalitu této metody.

Jestliže tuto metodu implementujeme bez schopnosti pamatovat si již objevené uzly, není algoritmus ani úplný, například v grafu na obrázku 5.1 nikdy nenajde cestu z A do F, za předpokladu, že heuristická funkce je vzdálenost vzdušnou čarou k cíli, z A přejde algoritmus do B a nejlepší cesta vede přes bod D, ale ten má vyšší heuristickou hodnotu a tak GBFS bude nekonečně přecházet mezi body B a C.



Obrázek 5.1: Příklad grafu, ve kterém GBFS nenajde cestu pokud si nepamatuje které uzly již navštívil

5.3 A*

A* je v podstatě synonymem pro hledání nejkratší cesty ve hrách. Je jednoduché A* implementovat, zároveň je velmi efektivní a lze ho jednoduše optimalizovat [20].

A* funguje podobně jako Dijkstrův algoritmus nebo GBFS, ale místo vzdálenosti od začátku nebo heuristické funkce vzdálenosti od konce, používá pro ocenění jednotlivých uzlů kombinaci těchto dvou hodnot.

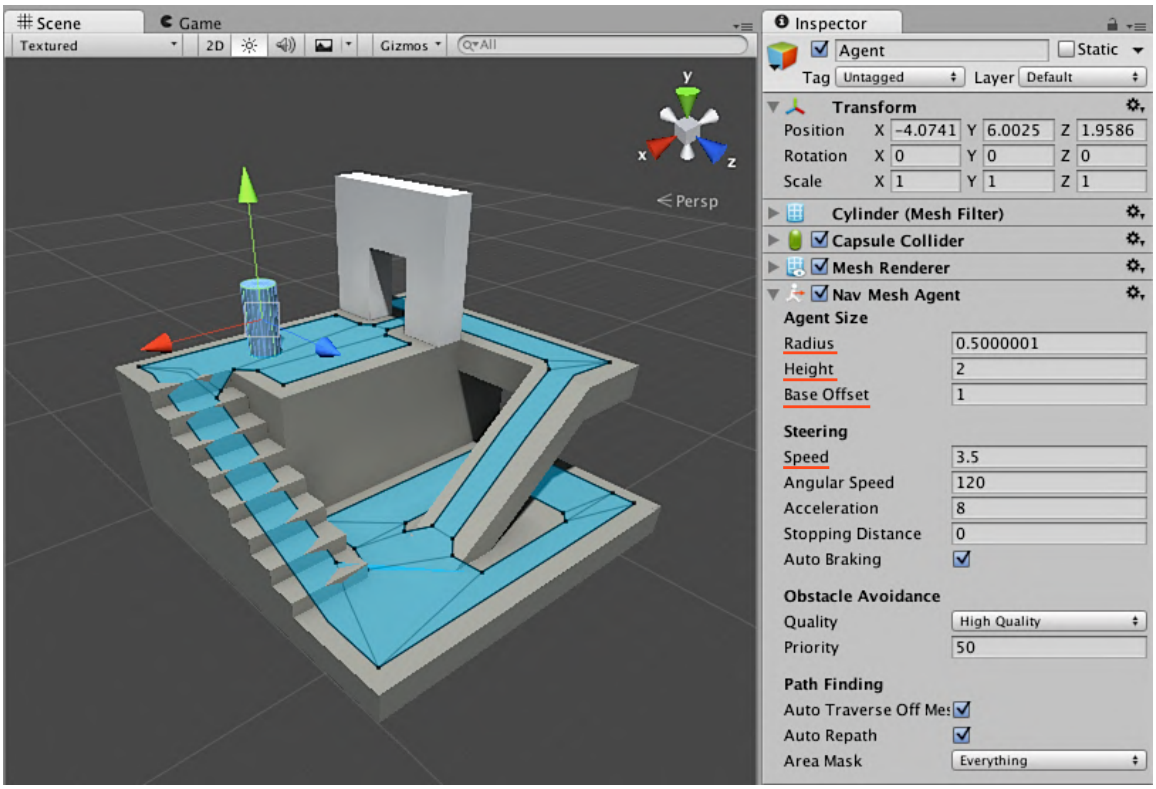
Použitá heuristická funkce musí být spodním odhadem skutečné ceny cesty, což znamená, že nesmí vracet větší odhadovanou cenu než jaká ve skutečnosti je – taková heuristika se nazývá přípustnou.

- Když je heuristika 0, A* degraduje na Dijkstrův algoritmus.
- Jestliže je heuristika nižší nebo rovna skutečné ceně cesty, najde A* nejkratší cestu, ale čím nižší je, tím více uzlů A* prohledá.
- Pokud má heuristika stejnou hodnotu jako skutečná cena, následuje A* ideální cestu a nenavštíví žádný zbytečný uzel.
- Heuristika, která je o trochu vyšší než skutečná cena, může ve skutečnosti zrychlit průběh A*, ale může způsobit že nebude nalezena ta nejlepší z cest.

- A pokud je heuristika příliš vysoká A* se začíná podobat GBFS [21].

5.4 Navmesh

Mnoho moderních her využívá k hledání cest přístup jménem navmesh, neboli navigační mesh, který využívá toho, že zem ve hrách je návrhářem předem specifikovaná a je jasné kde se jednotlivé části spojují a kde je tedy možné přejít z jedné části na druhou a podlaha samotná je tvořená z polygonů, které mohou být využity k navigaci. Každý polygon pak představuje bod v grafu a pokud měly původní polygony společnou hranu, mají jejich body mezi sebou hranu. Vyhledání cesty z polygonu A do polygonu B se potom provádí například algoritmem A*. Příklad navmeshe je vidět na obrázku 5.2.



Obrázek 5.2: Příklad navmeshe vygenerovaného na plošině v herním enginu Unity[1]

[1]Převzato z https://docs.unity3d.com/560/Documentation/Manual/nav-CreateNavMeshAgent.html

Kapitola 6

Návrh

V této kapitole si projdeme důležitá rozhodnutí, jejich odůvodnění a v některých případech jejich následky. Výsledná hra byla tvořena ve spolupráci s mým graficky zaměřeným kolegou a některá rozhodnutí byla výsledkem naší diskuze. Všechno modelování, texturování a animování, které nebylo procedurálně vytvořeno, měl na starosti on a všechno programování zase já.

6.1 Žánr

Kvůli našim preferencím jsme hned věděli, že půjde o hru žánru RPG nebo strategii. Začali jsme tedy pročesávat naše oblíbené hry (nejen) těchto dvou žánrů a hledat mechaniky, které se nám zdají zajímavé.

I když mnoho herních zkušeností a zážitků z minulosti ovlivnilo naše smýšlení, inspiraci jsme našli v hned několika titulech: John Wick Hex, Divinity: Original Sin 2 2.3a, Mutant Year Zero: Road to Eden, XCOM: Enemy Unknown 2.4b, Sleeping Dogs a původní česká hra Dračí Doupě II. A po grafické stránce byly inspirací například hry The Legend of Zelda: Skyward Sword nebo The Long Dark 6.1.

Nakonec jsme se rozhodli pro kombinaci rougelike a taktické RPG, která souzněla s mechanikami, které jsme vymysleli.

Cílem bylo do hry zakomponovat tahový systém plný akcí, které by se daly řetězit a odvíjely by se od okolí hráče a nepřítele a využít mechaniky z Dračího doupěte a dát hráči možnost přežít i ty nejhorší okolnosti s tím že musí nést trvalé následky.

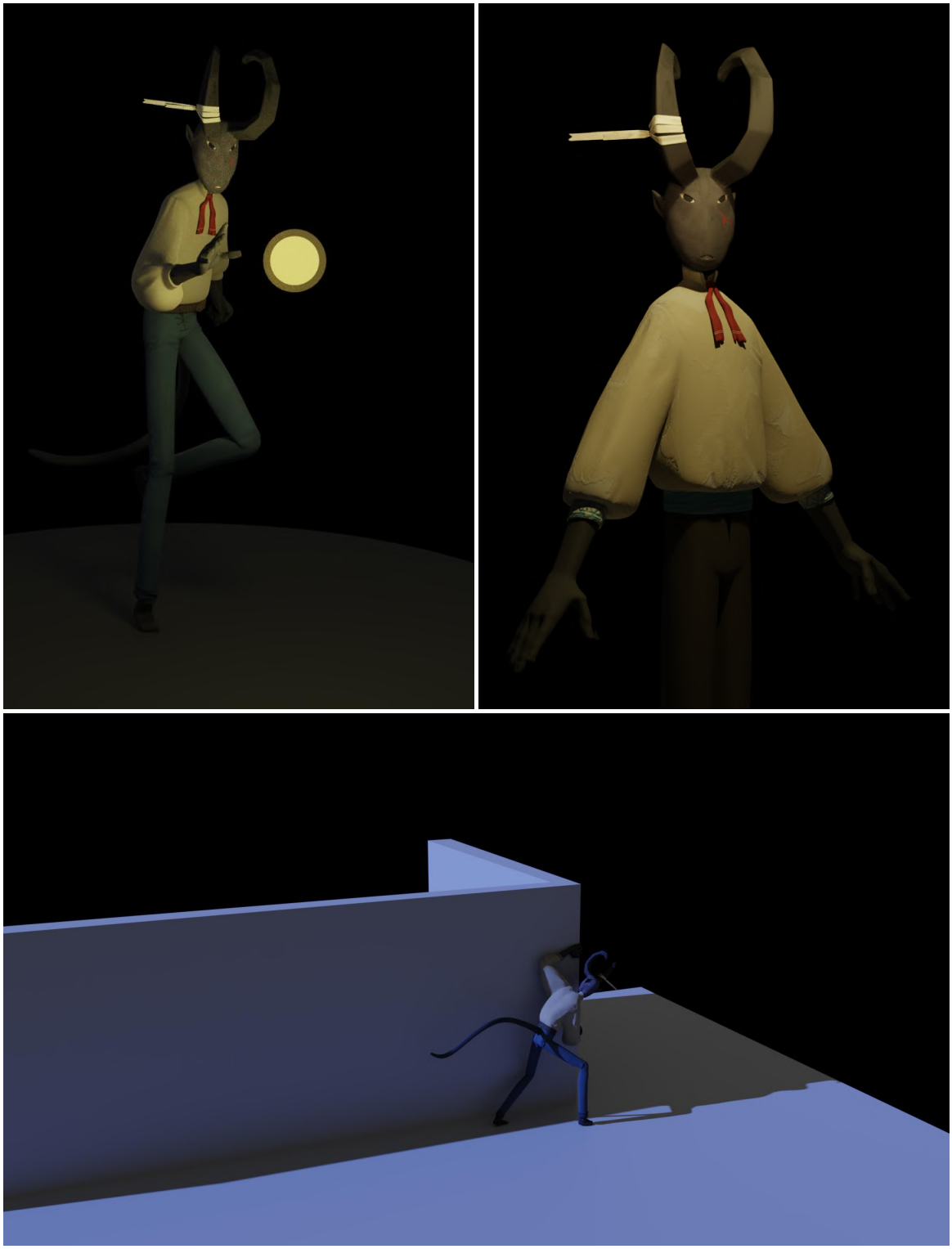
6.2 Prostředí a příběh

Hru jsme se rozhodli zasadit do prostředí inspirovaného skalními městy v České republice, jako jsou Prachovské skály nebo Tiské stěny 6.2 a celkově se inspirovat slovanskou tématikou, hudbou a prostředím. Hlavní postavou je čert, který je vidět na obrázku 6.3, a kolem kterého budeme později stavět příběh, ale ten zatím nemá prioritu.

Obrázek 6.1: The Long Dark (2014)

Obrázek 6.2: Tiské stěny[1]

[1] Autor: Frantakrivan – Vlastní dílo CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=30461618



Obrázek 6.3: Rendery hlavní postavy

6.3 Herní engine

V kapitole 3.4 bylo popsáno několik herních enginů a v této kapitole budou představeny naše volby a důvody, které k nim vedly.

Vzhledem k naší povrchové znalosti komerční nabídky herních enginů jsme ani nevěděli o většině nabízených produktů. Znali jsme enginy orientované na neprogramátory jako je RPG Maker a GameMaker, ale ty pro bakalářskou práci nepřipadaly v úvahu. Mezi našimi skutečnými možnostmi jsme znali pouze Unity a Unreal Engine.

Jedním z hlavních faktorů pro mě jako pro programátora byl programovací jazyk podporovaný daným enginem. **C++** versus **C#**. Ani jednomu z těchto jazyků jsem příliš nerozuměl, **C++** v mých rukách vždy degradovalo na **C**, které obsahuje vektory, ale více mě lákalo zkusit **C#** ať už svojí novotou, mírou abstrakce nebo absencí manuální alokace a dealokace paměti. Dalším důležitým faktorem byla všudypřítomnost všemožných návodů, diskuzních fór a videí s tématikou vývoje her v Unity, stejně jako dostupnost modelů, textur, hudby a dalších věcí na Unity Store.

Tyto argumenty vedly k volbě **Unity**, i když nainstalovat ho na Manjaro nebylo nejjednodušší a časem bylo stejně potřeba přejít na Windows, tak práce mohla začít. K synchronizaci našeho projektu jsme používali zabudované řešení od Unity, které se nazývá *Collaborate* a je velmi intuitivní, i když ne tak obsáhlé jako například Git.

Nakonec možná Unity nebyla ideální volba a při vývoji nastalo několik problémů se synchronizací, kompatibilitou s Linuxem, nekompatibilitou při použití různých grafických pipeline v Unity. Možná bych napodruhé zvolil Unreal Engine, ale chyby se pravděpodobně najdou všude a naše zkušenost by nebyla o tolik jiná.

6.4 Hlavní mechaniky

Postava

Důležité rozhodnutí se týká počtu ovládaných postav, existuje několik možností. Většinou hráč ovládá malou skupinku zhruba dvou až šesti hrdinů (Divinity: Original Sin, XCOM: Enemy Unknown, Mutant Year Zero: Road to Eden, ...) a nebo jednoho jediného hrdinu (John Wick Hex, Binding of Isaac, ...). My jsme měli od začátku na mysli jednu konkrétní postavu a proto jsme pracovali pouze s jedním hrdinou.

Další otázka je jak se bude postava pohybovat, na začátku jsme přemýšleli nad volným pohybem, ale kvůli kontextovým akcím byl systém převeden na mřížku, protože vyhledávání okolních předmětů se zdálo jednodušší.

Kamera

Jedním z definujících prvků každé hry je kamera a její ovládání (nebo jeho absence). Na začátku jsem nad hrou uvažoval jako nad dvourozměrnou hrou a představoval jsem si na ni pohled ze shora bez možnosti kameru ovládat, což by samovolně omezovalo dohled postavy. Po domluvě s kolegou grafikem, jsme ale dospěli k tomu, že bude lepší hru udělat ve 3D, a tak bylo potřeba přidat ovládání kamery.

Kamera bude vzhledem k jediné postavě fixovaná na jednu pozici a bude pouze orbitovat kolem hlavní postavy s tím že vzdálenost bude omezena.

Prostor

Jak už bylo řečeno, rozhodli jsme se pro 3D prostor. Jedním z důležitých prvků výsledné hry je procedurální generování, které mi navrhl pan inženýr Tomáš Milet, což je taky jeden z důvodů proč se hra odehrává na čtvercové mřížce, která usnadní generování herních úrovní.

Jako prostředí jsme chtěli využít skalních měst, která se nachází například i v České republice, které mají podobný tvar jako podzemní kobky (takže jdou lehce generovat), ale nejsou zdaleka tolik používané.

Tempo hry

Z taktických RPG jsme si vypůjčili tahový průběh hry a ze začátku jsme chtěli jít tradiční cestou střídání tahů podle hodnoty iniciativy nebo rychlosti dané postavy. Ale nakonec jsme v John Wick Hex našli systém, který využívá diskretizovanou časovou osu, všechny postavy plánují svoje akce když jsou na řadě a dostávají se znovu na řadu po uplynutí jejich aktuální akce.

K tomu abychom mohli zavést takovýto systém tahů, bylo potřeba vymyslet nějaký způsob jak uchovávat jednotlivé naplánované akce a provádět je ve správný čas. Rozumným řešením se zdál být kalendář, který by uchovával informace o tom kdo a kdy vykonává jakou akci.

Akce

Jako základní akce jsme zamýšleli útok, který bude možný pokud bude postava dostatečně blízko, a odstrčení, které by bylo prototypem pro kontrolování okolí, protože aby postava mohla někoho odstrčit, musí být za postavou prostor, do kterého bude odstrčena.

Ve hře bychom také chtěli mít interakci s předměty jako například hody kamením, které se postavám válí u nohou, nebo skrytí postavy v keři.

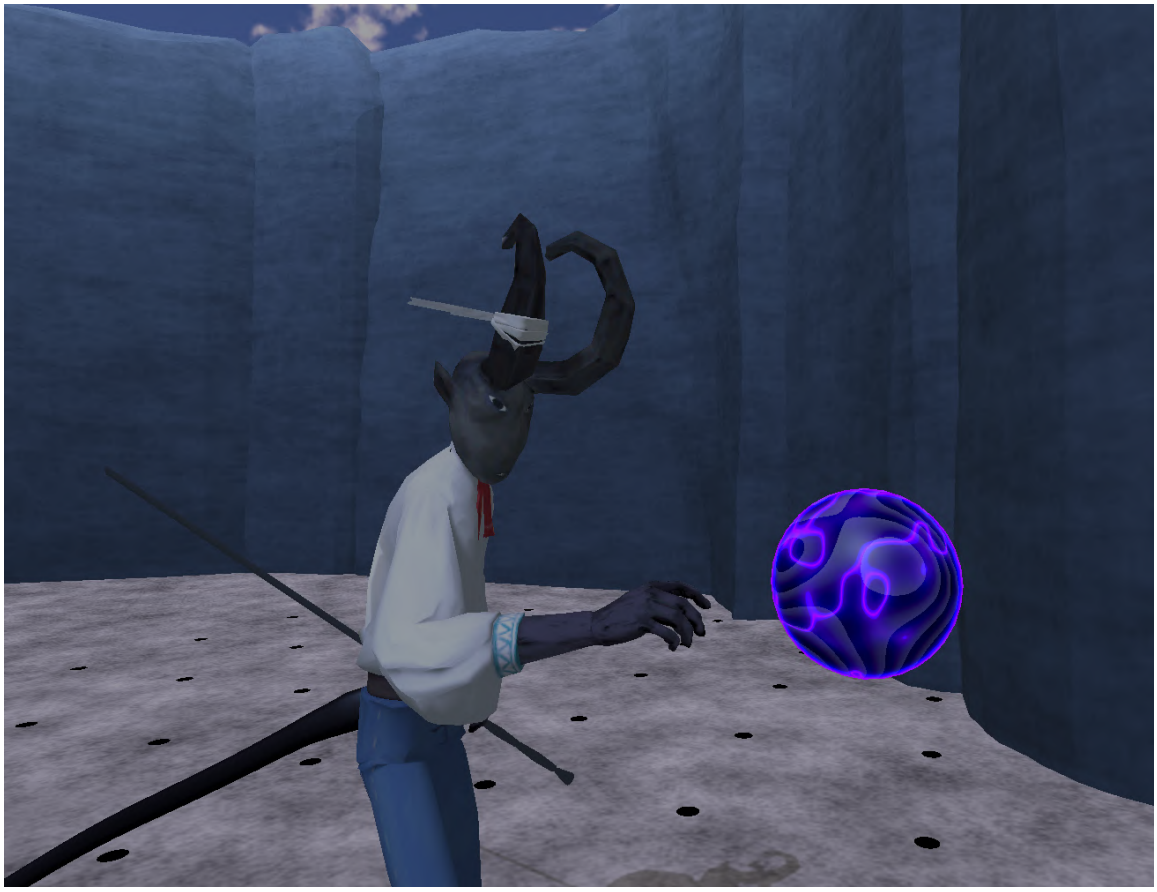
Zajímavou se nám zdála mechanika enviromentálních akcí, což znamená, že jako hráč můžete interagovat s ostatními charaktery a předměty, v závislosti na okolí. Například pokud hráč stojí s nepřítelem u propasti, může ho do ní strčit, nebo vykopnout hromadu písku a nepřítele oslepit.

Zároveň by akce mohli být řetězitelné. Například by hráč mohl po třech po sobě jdoucích útocích provést smrtící švih, který by nepřítele dorazil.

Úrovně

Jak už bylo zmíněno, herní úrovně jsou procedurálně generovány. Ale nestačilo hráče vypustit na jednu úroveň a tím končit, proto jsem navrhl systém nekonečného procházení generovanými úrovněmi. Na obrázku 6.4 je zachycen levitující orb, který je umístěn v každé úrovni a dotknutím se orbu, se hráč transportuje do další úrovně, která je větší a je v ní více nepřátel, hra je prozatím tedy nekonečný cyklus zvětšujících se map, který může skončit jen hráčovou prohrou.

V budoucnosti by mohlo být zajímavé hráče po X úrovních přesunout na úroveň, kde musí porazit bosse, což je speciální nepřítel, který je větší nebo silnější než obyčejní nepřátelé, aby mohl pokračovat dál. V dalších kapitolách by pak hráč mohl procházet novými úrovněmi s jinými vizuálními prvky, dokud by neprošel až na konec hry.



Obrázek 6.4: Orb, který funguje jako přenášedlo do další úrovně.
Jeho vzhled je tvořen shaderem, který je vytvořen v Shader Graphu.

Nepřátelé

Do hry nemáme navržené žádné konkrétní nepřátele, ale budou založeni na slovanských mýtech, nebo inspirováni slovanským prostředím. Hlavní myšlenkou, ale je že nepřátelé budou mít k dispozici v podstatě stejné akce jako samotný hráč, možná s nižším poškozením a delším trváním, ale na stejném principu.

Kapitola 7

Implementace

Celá hra je implementována v C# a shadery jsou implementovány s pomocí Shader Graphu. Všechen kód byl napsán od základu a hlavní pomocí byla dokumentace Unity[1], kromě tří případů: implementace OpenSimplexNoise.cs, Footsteps.cs a trojrozměrných šumů do Shader Graphu, všichni autoři jsou poznamenáni v příslušných zdrojových souborech.

7.1 Procedurálně generovaná mapa

Celé rozložení úrovní a stěn je generováno algoritmem podobným L-systému, který ale nepracuje s řetězci, které se později budou interpretovat, ale rovnou je interpretuje. Třídu je pojmenovaná jednoduše `MapGenerator` a je parametrizována, aby umožňovala ladění algoritmu a tvoření úrovní různých velikostí bez nutnosti zásahu do kódu.

Třída `MapGenerator` pracuje se záznamy typu `Point`, které uchovávají informace o svých x a y souřadnicích, směru (včetně diagonálních směrů) a vzdálenosti od poslední odbočky-/rozdělení chodby.

Výstup celého generování je vidět na obrázku 7.1

V následujících několika částech budou podrobněji vysvětleny její podčásti.

Chodby a místnosti

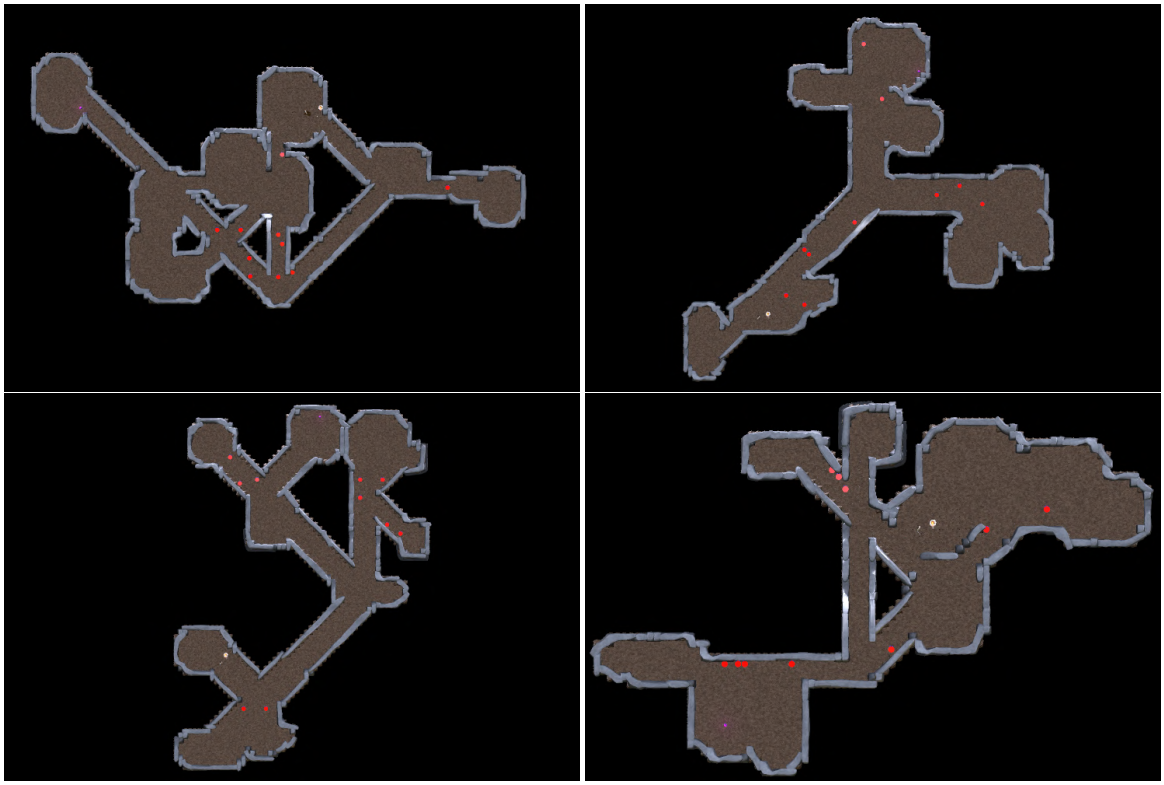
Celá třída má jen jednu veřejnou funkci a to funkci `generateAll()`, která se volá s 10 parametry: nejkratší povolenou délkou rovných úseků chodby, šance na provedení daného pravidla (pokračování ve stejném směru, zatáčka, rozdvojení a roztrojení chodby), maximální a minimální velikost místností, kýžený počet místností, maximální celkovou délku chodeb a míru přiblížení Perlinova šumu.

Funkce potom zavolá postupně tři další funkce `createCorridors()`, `widenCorridors()` a `createRooms()`, které jak už jejich názvy napovídají vytvoří chodby, rozšíří je a vytvoří místnosti.

`createCorridors()` První co funkce udělá je, že vybere náhodný bod ve střední části pole 1000 krát 1000 a přidá ho do fronty bodů čekajících na zpracování.

Poté následuje hlavní smyčka funkce, ta generuje další body ve stejném směru dokud nepřekročí minimální délku rovné chodby, danou příslušným parametrem a potom začne podle zadaných hodnot šancí generovat buď další bod ve stejném směru, zatáčku, rozdvojení

[1] https://docs.unity3d.com/2019.3/Documentation/ScriptReference/index.html



Obrázek 7.1: Čtyři vygenerované úrovně z ptačího pohledu. V jednotlivých obrázcích je vidět jeden bílý bod, což je ohniště u kterého se hráč na začátku kola objeví, fialový bod, což je orb, ke kterému se hráč musí dostat a červené body, což jsou nepřátelé.

nebo roztrojení do náhodných směrů, kde směry jsou ošetřeny tak, aby se například ze zatáčky nestala jen další rovná cesta.

Tato funkce simuluje paralelní vyhodnocování L-systémů, není sice skutečně paralelní, ale protože se body ukládají do fronty, střídají se body jednotlivých odrostů hlavní linie v prodlužování.

Funkce se ukončí pokud dosáhne zadané celkové maximální délky chodeb, nebo pokud se vyprázdní fronta bodů čekajících na zpracování.

V průběhu si metoda udržuje tři struktury typu `List`: první je seznam všech vygenerovaných bodů, druhý je seznam slepých konců chodeb a třetí je seznam všech bodů, ve kterých se chodby větví. Poslední dva seznamy budou důležité při generování místností.

V tomto bodě se taky zjistí, jestli je vygenerovaná mapa dostatečná na umístění požadovaného množství místností, což se zjistí sečtením množství slepých konců a větvících bodů a pokud je jejich součet menší než požadované množství místností, funkce se ukončí a je zavolána znovu.

`widenCorridors()` Tato funkce projde postupně celý seznam všech vygenerovaných bodů a z každého z nich vytvoří kruh, jehož velikost je určena hodnotou Perlinova šumu v daném bodě. K rozšiřování jednotlivých bodů používá funkci `widenPoint()`, kterou můžete ve výpisu 7.1.

```
void widenPoint(Point point, float xradius, float yradius){
```

```
    for(int y = point.y - (int)yradius; y < point.y + yradius; y++){
        for(int x = point.x - (int)xradius; x < point.x + xradius; x++){
            float ellipse =
                Mathf.Pow(x - point.x, 2) / Mathf.Pow(xradius, 2) +
                Mathf.Pow(y - point.y, 2) / Mathf.Pow(yradius, 2);
            if(ellipse < 1) instantiateObject(x, y);
        }
    }
}
```

Výpis 7.1: Kód pro rozšíření daného bodu

`createRoooms()` Funkce `createRooms()` v prvním kroku vezme seznam všech slepých míst a na každém z nich vytvoří místnost, čímž zamezí existenci slepých míst na mapě.

Pokud počet místností po provedení prvního kroku není alespoň takový jaký byl zadaný parametrem, začne brát ze seznamu body, kde se chodby dělí a generovat místnosti tam, dokud jich není tolik, aby splnili zadanou podmínku.

Místnosti jsou oválného tvaru a stejně jako `widenCorridors()` používá funkci `widenPoint()`, která umí pracovat i s ovály.

`trimLevel()` Předtím než se začnou umísťovat stěny, je zavolána funkce `trimLevel()`, která jak název může napovědět, ořízne mapu úrovně z tisíc krát tisíc na nejnižší možnou velikost, hlavně proto, aby se urychlila další práce s mapou.

Stěny

Generování stěn se ukázalo být jedním z největších problémů tvorby mapy, protože jsme se rozhodli použít předem vytvořené modely jednotlivých stěn. Nebylo žádoucí programovat natvrdo rozměry každého modelu přímo do kódu, aby grafik mohl vyzkoušet velké množství modelů.

Řešením se ukázalo být umístění kvádrových *colliderů*, které představují pole, které má stěna zabírat, do každého modelu stěny, potom je po jedné instanciovat a pomocí *Raycastingu*, neboli vrháním paprsků zjišťovat jejich tvar, z tohoto tvaru se vytvořilo pole boolovských hodnot, které slouží jako šablona pro jeho umístění.

Odteď tedy bylo možné do generátoru přímo v Unity přidat modely stěn a pravděpodobnosti jejich výskytu. Z praktických důvodů jsme vytvořili dvě kategorie stěn, prioritní a obyčejné, protože i přes vysoké pravděpodobnostní hodnoty se na výsledné mapě objevovalo málo stěn velkých rozměrů a proto jsou prvně vygenerovány všechny stěny z prioritní kategorie a potom se funkce opakuje pro obyčejné stěny. A když jsme měli seznam modelů a způsob zjištění jejich tvarů, mohli jsme je začít umisťovat na mapu.

Na začátku se označí body okolo chodeb jako místa, kde je žádoucí mít stěnu. Potom se vybere náhodná stěna a boolovské pole definující její tvar se porovná s oblastmi, kde mají být stěny. Neporovnává se ale pouze základní tvar stěny, ale taky otočený a zrcadlený tvar (celkem tedy osm variací) a ze všech validních možností se vybere jedna, kam se stěna umístí.

Pro práci s boolovskými poli vzniklo několik funkcí pro jejich rotaci 7.2, zrcadlení 7.3, ořezávání, porovnávaní a vyhledávání.

```
bool[,] rotateArray(bool[,] inputArray){
    int inputX = inputArray.GetLength(0);
    int inputY = inputArray.GetLength(1);
    bool[,] outputArray = new bool[inputY, inputX];

    for (int x = 0; x < inputX; x++){
        for (int y = 0; y < inputY; y++){
            outputArray[inputY - y - 1, x] = inputArray[x, y];
        }
    }
    return outputArray;
}
```

Výpis 7.2: Kód pro rotaci pole typu bool

```
bool[,] mirrorArray(bool[,] inputArray){
    int inputX = inputArray.GetLength(0);
    int inputY = inputArray.GetLength(1);
    bool[,] outputArray = new bool[inputX, inputY];

    for (int x = 0; x < inputX; x++){
        for (int y = 0; y < inputY; y++){
            outputArray[inputX - x - 1, y] = inputArray[x, y];
        }
    }
    return outputArray;
}
```

Výpis 7.3: Kód pro zrcadlení pole typu bool

Pokud nelze stěnu nikam umístit, je odstraněna ze seznamu a dále není brána v potaz. Potom co je odstraněna poslední stěna ze seznamu, je zavolána funkce, která pod zdi umístí zem, aby zacelila drobné mezery mezi stěnami a chodbami.

7.2 Herní mechaniky

Celý průběh hry ovládá třída `GameMaster`, která volá `MapGenerator`, spawnuje[2] hlavní postavu, nepřátele a cíl úrovně, reaguje na hráčovy akce a obsahuje funkce herních akcí a všeobecně ovládá hlavní herní smyčku.

Po spuštění hry `GameMaster` vytvoří nový kalendář třídy `Calendar`, zavolá `MapGenerator.generateAll()` a pokračuje do hlavní smyčky.

Hlavní smyčka obsahuje malý stavový automat, který obsahuje pouze tři stavy `inGame`, který znamená, že hráč je právě ve hře, `showingMap`, který indikuje, že hráč zrovna zobrazuje mapu a třetí stav znamená, že je hráč v herní nabídce, která se zobrazí stiskem klávesy *escape*.

V následujících sekcích budou jednotlivé části vysvětleny více podrobně.

Spawning

Do této kategorie spadají tři funkce: `spawnPlayer()`, `spawnOrb()` a `spawnEnemies()`.

`spawnPlayer()` Pozice pro hráče je vybrána zvolením náhodné místnosti a náhodné pozice v ní.

`spawnOrb()` Tato funkce umisťuje do herní mapy orb, který je cílem každé úrovně, a proto je žádoucí, aby se nenacházel na místě blízko hráče. Tato funkce stejně jako `spawnPlayer()` vybírá náhodné místnosti a pozice v nich, ale tento postup provádí několikrát a orb umístí na tu nejvzdálenější pozici.

`spawnEnemies()` Stejně jako předchozí metody této kategorie i tato využívá jednoduché náhody a vytvořené nepřátele ukládá do seznamu, který slouží k aktivaci AI po jednotlivých úsecích kalendáře.

Čas

Celá hra probíhá v tazích, každý o délce jedné desetiny sekundy. Čas udržuje a zvyšuje třída `Calendar` a běh kola ovládájí funkce ve třídě `GameMaster`.

`Calendar` Obsahuje seznam záznamů typu `CalendarEntry`, který je vidět ve výpisu 7.4 a funkce pro práci s nimi. Tento seznam tvoří ústřední prvek celého tahového systému hry.

Třída `Calendar` obsahuje následující funkce:

`addAction()` vytvoří nový `CalendarEntry` a přidá jej do kalendáře.

`removeActionsForId()` odstraní z kalendáře všechny (nebo parametrem zadaný druh) akce, které přidala postava s daným identifikačním číslem – volá se po smrti postavy, nebo když je zacílena a je ji znemožněna chůze.

`removeAllActions()` smaže všechny záznamy z kalendáře voláním. `calendar.Clear()`, tato funkce se volá vždy po ukončení úrovně.

`getTime()` vrací aktuální čas, používá se pro jeho zobrazení v uživatelském rozhraní.

`getCurrentActions()` je asi nejdůležitější funkcí, která postupně projde svým seznamem akcí a všechny akce, které se mají odehrát v aktuálním čase, z něj vyjme a vrátí jako výstup funkce. Pokud nejsou žádné akce aktuální, inkrementuje čas.

[2] zjevení herní postavy ve světě

`isCharacterBusy()` vrací boolovskou hodnotu, `true` pokud má postava s daným identifikačním číslem již naplánovanou nějakou akci, jinak `false`.

`isCharacterTarget()` vrací boolovskou hodnotu, `true` pokud je postava s daným identifikačním číslem cílem nějaké naplánované akce, jinak `false`.

```
public class CalendarEntry{
    public int id;
    public int targetId;
    public int time;
    public UnityAction action;

    public CalendarEntry(int id, int targetId, int time, UnityAction action){
        this.id = id;
        this.targetId = targetId;
        this.time = time;
        this.action = action;
    }
}
```

Výpis 7.4: Třída CalendarEntry

Tahový systém Pokud hráč nemá v kalendáři zapsanou žádnou akci pro aktuální kolo, běh času se pozastaví a čeká se na hráče. Když je hlavní stavový automat ve stavu `inGame` spouští se funkce `checkClick()`, která monitoruje hráčovy pohyby myši.

`checkClick()`: Na jaký objekt ve hře hráč ukazuje myší, je zjišťováno vrháním paprsků pomocí funkce `ScreenPointToRay()`, která od kamery směrem ke kurzoru vyšle paprsek a vrátí informace o všech objektech, které paprsek zasáhl.

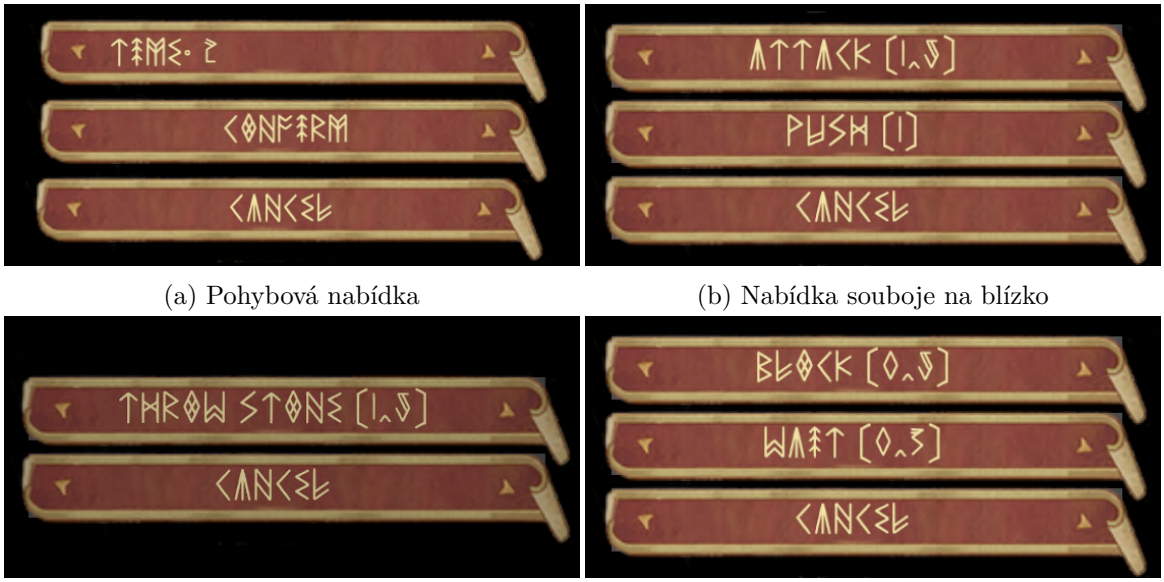
Je potřeba pamatovat, že paprsek pokračuje skrz objekty a může trefit i všechny objekty v zákrytu a proto musí být ustanoven určitý žebříček priorit, který je v naší implementaci následující: *Enemy* > *Orb* > *Player* > *Ground*. Pokud paprsek trefí více nepřátel, za platný cíl je považován ten, který je blíže ke kameře, stejná myšlenka je aplikována i u jednotlivých polí herního plánu. Hráč a orb jsou ve hře pouze po jednom kusu a proto není takováto kontrola potřeba.

Jakmile je zaznamenán stisk levého tlačítka myši je ze stavových proměnných, které `checkClick()` mění podle toho na jakém objektu aktuálně spočívá kurzor myši, vyčteno jaká nabídka má být hráči zobrazena.

Pokud je kurzor na herním poli, zobrazí se hráči předem vytvořená tabulka (viz obrázek 7.2a), která obsahuje časovou délku běhu a tlačítka *confirm*, neboli potvrdit a *cancel*, neboli zrušit. Jestliže je kurzor na některé z postav nebo orbu, volá se funkce `getPossibleActions()`, která podle současné situace určí, které akce jsou proveditelné a dynamicky vytvoří tabulku tlačítek, která po stisknutí naplánují příslušné akce, příklady jsou vidět na obrázku 7.2.

Jakmile hráč potvrdí svůj tah, voláním funkce `endTurn()` se ukončí jeho kolo a následuje cyklus volání `runCurrentActions()`, dokud není hráč znovu na řadě.

`runCurrentActions` získá pomocí volání funkce `getCurrentActions()`, která je implementována ve třídě `Calendar`, seznam akcí, které je potřeba provést a postupně je provádí voláním `UnityAction`, která je součástí každého `CalendarEntry`.



(a) Pohybová nabídka

(b) Nabídka souboje na blízko

(c) Nabídka vzdáleného útoku

(d) Defensivní nabídka

Obrázek 7.2: Vyskakovací okna, pohybová nabídka (7.2a) je jako jediná předvytvořená a ostatní jsou za běhu poskládány z předvytvořeného generického tlačítka.

Souboje

Hlavní myšlenka soubojů v této hře je, že akce, které jsou hráči dostupné, závisí na okolních podmínkách, je tedy potřeba mít vytvořenou sadu akcí, podmínek vedoucích k možnosti jejich použití a způsob rozpoznání kdy jsou všechny podmínky splněny.

Jednotlivé akce jsou definovány ve třídě `GameMaster` a jsou parametrizovány veřejnými proměnnými, které jsou součástí vlastností každé postavy ve hře, což vede k možnosti pozměnit například trvání nebo výši poškození dané akce pro každou postavu zvlášť.

`getPossibleActions()` je funkce, která stojí ve středu celého soubojového systému a obsahuje podmínky pro umožnění všech akcí. Například pro obyčejný útok na postavu stačí, aby jste byli dostatečně blízko, ale pro odstrčení nepřítele musí navíc ve směru odstrčení být volné místo. Aby bylo možné zahájit obranu a blokovat útok, musí nejdřív postava být cílem nějakého útoku. A aby bylo možné po nepříteli vrhnout kamenem, musí být nepřítel na dohled a na poli, na kterém stojí postava musí být kámen, který lze vrhnout.

7.3 AI – umělá inteligence

Nepřátelé mají prozatím velmi jednoduchou umělou inteligenci, která čeká, dokud se hráč nepřiblíží dostatečně blízko. Jestliže je hráč v jejich potenciálním dohledu, zkontroluje daný nepřítel jestli na hráče skutečně vidí, a to tak, že vrhne paprsek směrem k hráči, a pokud nezasáhne zeď, hráče si všimne. Jakmile hráče jednou spatří, snaží se k němu dojít a když je dostatečně blízko, začne útočit.

Pathfinding

Původně byl ve hře použit v Unity zabudovaný systém NavMesh. NavMesh byl nakonec nahrazen implementací A*, protože nebyl žádný jednoduchý způsob jak ho donutit držet se herní mřížky. Systém NavMesh se stále stará o přesun mezi uzly na cestě nalezené A*, ale nehledá celkovou cestu z bodu A do bodu B.

Implementace A* je celkem jednoduchá, i když ani ta se neobešla bez drobné chyby, která srazila počet snímků za sekundu na třetinu, ale problémy s výkonem se ve studentské verzi Unity řeší lehce díky zabudovanému profileru, který monitoruje průběh programu a zobrazuje jak dlouho trvalo vykonávání funkcí v jednotlivých snímcích.

Zajímavý byl vliv heuristické funkce na výkon hledání cesty, jako první heuristickou funkci jsem bezmyšlenkovitě napsal Manhattanskou vzdálenost, s ní probíhalo hledání rychle, ale někdy vracelo podivné cesty, které byly mnohem delší než by měly být. Při pokusu tento jev odstranit jsem změnil heuristickou funkci na Euklidovskou vzdálenost, která dělala příliš nízké odhady a nutila algoritmus A* projít větší množství uzlů, než bylo potřeba. Ale nakonec jsem změnil heuristickou funkci na oktalovou vzdálenost, která je ideální na osmi-spojenou mřížku a A* začala fungovat podle mých představ, kód je k vidění ve výpisu 7.5.

```
int getHeuristicCost(int x, int y){
    int xDiff = Mathf.Abs(x - endX);
    int yDiff = Mathf.Abs(y - endY);

    if(xDiff < yDiff){
        return xDiff * diagonalMoveCost + (yDiff - xDiff) * straightMoveCost;
    }
    else{
        return yDiff * diagonalMoveCost + (xDiff - yDiff) * straightMoveCost;
    }
}
```

Výpis 7.5: Kód pro výpočet oktalové vzdálenosti

V první verzi cest hledajícího algoritmu, mohli postavy procházet skrz rohy stěn, což není úplně žádoucí, a proto jsem do algoritmu zabudoval omezení, které zamezí přechodu na diagonálně sousedící plošinu, pokud nelze dojít na obě ortogonálně sousedící plošiny, tedy pokud je vlevo nebo nahoře stěna nemohu jít diagonálně doleva-nahoru.

7.4 UI – uživatelské rozhraní

Uživatelské rozhraní už obsahuje většinu prvků, které byly v plánu. Jak momentálně UI vypadá je vidět na obrázku 7.3.
Dá se kategorizovat do dvou skupin:

Neustále zobrazené informace (HUD – Heads Up Display) jsou jeden z nich a obsahují stav hráčových životů (a zdrojů), jak je vidět na obrázku 7.4 a herní čas, který je provizorně zobrazen jako na obrázku 7.6a, ale ideálně by byl zobrazen s pomocí časové osy, podobné té na obrázku 7.6b.

Druhou kategorií jsou dynamicky zobrazované informace, do které by se daly zařadit tabulky akcí (viz obrázek 7.2), dosažitelné herní pole označená tmavými body a čára zná-

Obrázek 7.3: Rozložení uživatelského rozhraní. Vlevo nahoře je zobrazen aktuální čas, vlevo dole se zobrazují nabídky pro výběr akcí a dole je vidět hráčův aktuální stav životů a zdrojů.

Obrázek 7.4: Ukazatel životů a placeholder pro další zdroje

zorňující aktuální trasu běhu (viditelné na obrázku 7.3) a ukazatele vyskytující se nad nepřáteli, které jsou vidět na obrázku 7.5.

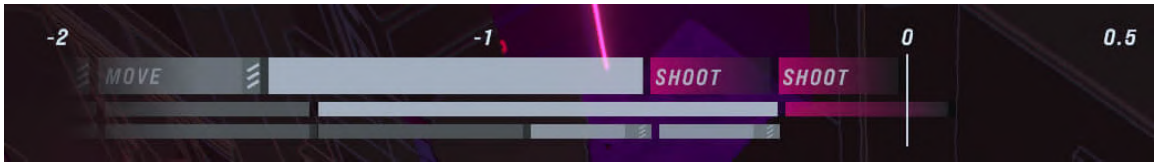
Obrázek 7.5: Ukazatele nad nepřáteli. Modrá barva znamená že nepřítel momentálně necílí žádnou akci na hráče, jakmile naplánuje útok, změní se barva na červenou. Všichni nepřátelé nad sebou mají nevyplněný ukazatel a jedině nepřítel na kterého hráč klikne má ukazatel, aby bylo jasně vidět, se kterým nepřítelem právě interaguje.

7.5 Shader Graph

Pro tvorbu textur na stěny a zem byly použity shadery vytvořené pomocí Shader Graphu, který byl zmíněn v kapitole 3.4. Efekt vytvořených shaderů je vidět na obrázku 7.7.

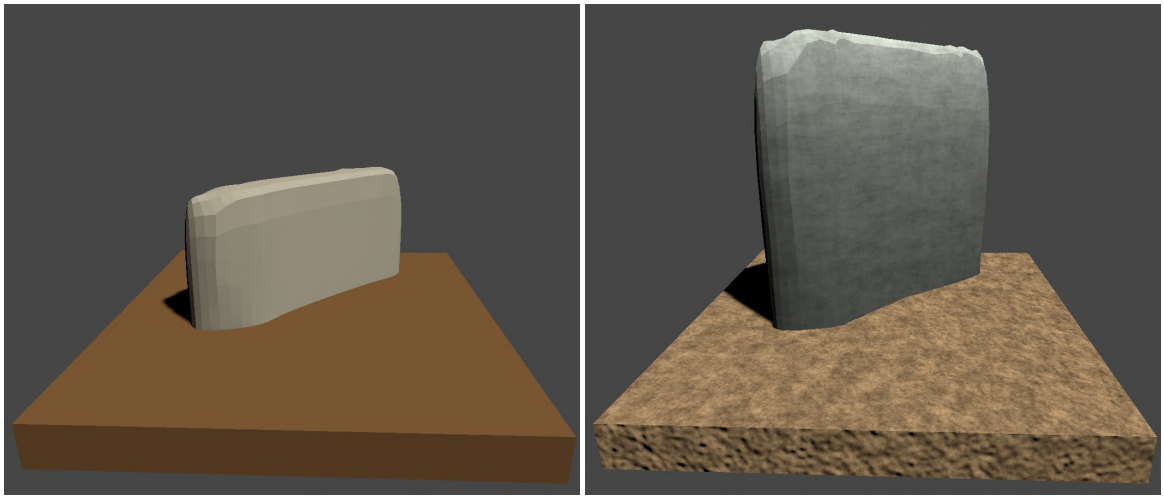
Shader, který je aplikován na zem je v podstatě součtem několika šumů různých měřítek vynásobený barvou a podobně jako barva, i normálová mapa je generována součtem šumů.

(a) Současný ukazatel času



(b) Vzorová časová osa ze hry John Wick Hex

Obrázek 7.6: Zobrazení času



Obrázek 7.7: Před a po použití shaderů na zdi a zem

Výhodou generování textury na zem je že na sebe dokonale navazuje, ale neopakuje se, což je velmi žádoucí u textury, která je použita na procedurálně generovanou mapu.

Jak je na obrázku 7.7 na první pohled vidět, je stěna po použití shaderu mnohem vyšší, tohoto efektu je docíleno pomocí manipulace pozic vertexů, což Shader Graph také umožňuje. Zdi bylo potřeba zvýšit k dosažení cíleného efektu uzavřenosti mezi stěnami, ale modely stěn byly příliš nízké a změna všech modelů se zdála zbytečná, když stejného efektu lze docílit s použitím Shader Graphu velmi snadno. Stěny mají složitější shader, ale v základu jde o použití množství šumů a jejich vhodné kombinace. Pro dosažení horizontálních čar je použitý šum se zhuštěnou osou y a podle výšky daného pixelu ve světě je obarvený na daný stupeň šedé (dole tmavší a nahoře světlejší).

Vzhledem k tomu že celá mapa je obklopená stěnami, je často problém, že kamera a postava se vyskytují na opačných stranách. I tento problém nakonec vyřešil Shader Graph, aplikovali jsme trojrozměrný Simplex Noise na alpha kanál stěny a pak parametrem určovali AlphaClipThreshold, což je hodnota, která určuje při jaké alpha hodnotě se daný pixel ještě zobrazuje a jakmile se nachází stěna mezi kamerou a hráčem, zavolá se funkce, která tuto hodnotu zvyšuje dokud zeď není průhledná. Na obrázku 7.8 je vidět zeď, která zrovna mizí.

Obrázek 7.8: Mizející zeď, která blokovala pohled na hráče.

Další aplikací Shader Graphu je cílový orb (k vidění na obrázku 6.4), který zobrazuje výřez z několika tří dimenzionálních šumů, které jsou plynule posouvány jedním směrem, aby se v kombinaci s levitací samotného orbu vytvořil zajímavý vizuální efekt. Levitace orbu je uskutečněna změnou pozic na všech třech osách o hodnotu OpenSimplex šumu.

Kapitola 8

Závěr

V rámci této práce byla vytvořena funkční demoverze hry, která obsahuje většinu základních systémů a splňuje moje očekávání.

Vzhledem k současné fázi hry jsme se nepokoušeli kontaktovat cizí lidi a hru jsme vždy v průběhu implementace testovali sami. Když už se ale hra začínala rýsovat a byla hratelná sdíleli jsme videa, obrázky a stabilní verze hry s blízkou skupinou přátel, jejichž zpětná vazba měla zcela jistě vliv na současný vzhled hry, například ukazatele nad nepřáteli vznikly po žádosti jednoho z nich, o indikátor toho, jestli nepřítel útočí nebo ne.

V rámci práce na této bakalářské práci jsem získal velké množství zkušeností s Unity, C# a celkově s programováním, ostatně jako vždy, když člověk uplatní svoje vědomosti ve větším projektu, dostane často úplně nový pohled na věc a kdybych začínal znovu, možná bych postupoval jinak nebo s větším rozmyslem a více plánoval do budoucnosti. I když většinou nebylo potřeba přepisovat části kódu, které jsem napsal ze začátku, aby celá hra fungovala, byla na nich jejich omezenost často poznat.

Jedna z věcí, která mě překvapila je, jak náročné bylo komunikovat se svým grafickým kolegou, synchronizovat náš postup a navzájem se držet v obraze, o tom co děláme na projektu. Nakonec jsme tento problém, alespoň částečně, vyřešili pomocí soukromého kanálu na Discordu a občasnými telefonáty.

Co se týče budoucnosti, je v plánu ještě mnoho věcí. V textu bylo zmíněno několik z nich:

- více různých kontextuálních akcí – jako například propasti, do kterých lze strčit nepřátele, nebo keře, ve kterých se lze schovat a nepřítele přepadnout ze zálohy,
- řetězení akcí – například po dvou po sobě jdoucích úderech by se zpřístupnil silný dokončovací útok,
- pokročilejší umělá inteligence a více druhů nepřátel,
- různá prostředí,
- zobrazení dění na časové ose a
- využívání zdrojů po vzoru Dračího Doupěte.

Literatura

[1] Amazon. *Fully Customizable Game Engine - Amazon Lumberyard -Amazon Web Services* [online]. 2020 [cit. 2020-05-20]. Dostupné z: https://aws.amazon.com/lumberyard/.

[2] Avedon, E. M. a Sutton Smith, B. *The Study of Games*. John Wiley & Sons, Inc., 1971. ISBN 0471038393.

[3] Bates, B. *Game Design*. 2. vyd. Thomson Course Technology PTR, 2004. ISBN 1-59200-493-8.

[4] Blakey, D. *Most Played MMOs - MMO Populations & Player Counts* [online]. 2020 [cit. 2020-05-04]. Dostupné z: https://mmo-population.com/activity.

[5] Botea, A., Bouzy, B., Buro, M., Bauckhage, C. a Nau, D. Pathfinding in Games. 2012.

[6] Crawford, C. *The Art of Computer Game Design*. 1984. ISBN 0-88134-117-7.

[7] CRYENGINE. *CRYENGINE | The complete solution for next generation game development by Crytek* [online]. 2020 [cit. 2020-05-20]. Dostupné z: https://www.cryengine.com/.

[8] Cui, X. a Shi, H. A*-based Pathfinding in Modern Computer Games. 2010.

[9] Elias, H. *Perlin Noise* [online]. 1998 [cit. 2020-05-28]. Dostupné z: https://web.archive.org/web/20160530124230/http://freespace.virgin.net/hugo.elias/models/m_perlin.htm.

[10] Feil, J. a Scattergood, M. *Beginning Game Level Design*. Thomson Course Technology PTR, 2005. ISBN 1-59200-434-2.

[11] Florian. *Best Game Engines For Beginners: 14 Alternatives* [online]. 2018 [cit. 2020-05-19]. Dostupné z: https://www.websitetooltester.com/en/blog/best-game-engine/.

[12] Fullerton, T., Swain, C. a Hoffman, S. S. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*. 2. vyd. Elsevier Inc., 2008. ISBN 978-0-240-80974-8.

[13] GamingScan. *2020 Gaming Industry Statistics, Trends & Data (Biggest Study)* [online]. 2020. Dostupné z: https://www.gamingscan.com/gaming-statistics/.

[14] Godot Engine. *Godot Engine - Free and open source 2D and 3D game engine* [online]. 2020 [cit. 2020-05-20]. Dostupné z: https://godotengine.org/.

[15] Gregory, J. *Game Engine Architecture*. 2. vyd. Taylor & Francis Group, LLC, 2015. ISBN 978-1-4665-6006-2.

[16] Gustavson, S. Simplex noise demystified. 2005.

[17] Hanan, J. S. Parametric L-systems and their application to the moddeling and visualization of plants. 1992.

[18] Hendrikx, M., Meijer, S., Velden, J. van der a Iosup, A. Procedural Content Generation for Games: A Survey. 2013.

[19] Marsaglia, G. Xorshift RNGs. 2003.

[20] Millington, I. a Funge, J. *Artificial Intelligence for Games*. 2. vyd. Elsevier Inc., 2009. ISBN 978-0-12-374731-0.

[21] Patel, A. *Amit's A* Pages* [online]. 2010 [cit. 2020-05-31]. Dostupné z: http://theory.stanford.edu/~amitp/GameProgramming/index.html.

[22] Peringer, P. *Modelování a simulace: IMS Studijní opora* [online]. Fakulta informačních technologií VUT v Brně, 2012. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FIMS-IT%2Ftexts%2Fopora-ims.pdf.

[23] Perlin, K. Noise Hardware. 2001.

[24] Prusinkiewicz, P. a Lindenmayer, A. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990. ISBN 978-0-387-97297-8.

[25] Schell, J. *The Art of Game Design: A Book of Lenses*. Elsevier Inc., 2008. ISBN 978-0-12-369496-6.

[26] Uniblock. *Noise!* [online]. 2014 [cit. 2020-05-28]. Dostupné z: https://uniblock.tumblr.com/post/97868843242/noise.

[27] Unity Technologies. *Unity Real-Time Development Platform | 3D, 2D, VR & AR Visualisations* [online]. 2020 [cit. 2020-05-20]. Dostupné z: https://unity.com/.

[28] Unreal Engine. *Unreal Engine | The most powerful real-time 3D creation platform* [online]. 2020 [cit. 2020-05-20]. Dostupné z: https://www.unrealengine.com/.

[29] Zbořil, F. *Základy umělé inteligence* [online]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FIZU-IT%2Flectures%2FIZU%2F1819izu_2.pdf&cid=12216.