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 MECHANIKY ZALOŽENÉ NA RŮZNÝCH ÚHLECH POHLEDU

VIEW-DEPENDANT GAME MECHANICS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAN ZDENĚK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ MILET, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

153439

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Zdeněk Jan**
Program:	Informační technologie
Název:	**Herní mechaniky založené na různých úhlech pohledu**
Kategorie:	Počítačová grafika
Akademický rok:	2023/24

Zadání:

1. Nastudujte herní vývoj, herní enginy, Unity a herní mechaniky. Soustřeďte se na fyziku a možné způsoby zobrazování scény. Prozkoumejte hry s podobnými herními mechanikami.
2. Navrhněte hru a herní mechaniky, které ovlivňuje úhel pohledu kamery. Přepínání úhlu kamery bude hlavní herní mechanika.
3. Implementujte hru podle návrhu. Průběžně testujte.
4. Ověřte hru na uživatelích a změřte její charakteristiky.
5. Hru zveřejněte, sepište závěry a možnosti budoucího vývoje a vytvořte demonstrační video.

Literatura:

- Jeremy Gibson Bond. Introduction to Game Design, Prototyping, and Development: From Concept to Playable Game with Unity and C#. Addison-Wesley Professional 2017. ISBN-100134659864.
- Ernest Adams. Fundamentals of Game Design 3rd Edition. New Riders 2013. ISBN-100321929675.

Při obhajobě semestrální části projektu je požadováno:

Funkční prototyp hry. Tři herní mechaniky založené na změně pohledu kamery. Třicet stran technické dokumentace.

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

Vedoucí práce:	**Milet Tomáš, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	9.5.2024
Datum schválení:	9.11.2023

Abstrakt

Cílem této bakalářské práce je vytvořit jednoduchou počítačovou hru v Unity, která je založena na změně perspektivy. Dále toto řešení obsahuje procedurální generování úrovní, které je implementováno za využití celulárního automatu a algoritmu marching squares. Výsledkem je tedy hra spadající do žánru roguelike, která uživateli poskytuje možnost změny perspektivy z pohledu shora na pohled z profilu. Hlavním úkolem hráče je dosáhnout co nejvyšší úrovně a postupným vylepšováním postavy získat výhodu do následujícího běhu.

Abstract

The aim of this bachelor thesis is to create a simple computer game in Unity that is based on a change of perspective. Furthermore, this solution includes procedural level generation, which is implemented using the cellular automaton and the marching squares algorithm. The result is thus a game that falls into the roguelike genre, providing the user with the possibility to change perspective from a top-down view to a profile view. The main task of the player is to reach the highest level and, by gradually upgrading the character, gain an advantage for the next run.

Klíčová slova

Počítačová hra, Unity, roguelike, změna perspektivy, procedurální generování, pohled shora, pohled z profilu, 3D, celulární automaty, marching squares

Keywords

Computer game, Unity, roguelike, perspective change, procedural generation, top-down view, profile view, 3D, cellular automata, marching squares

Citace

ZDENĚK, Jan. *Herní mechaniky založené na různých úhlech pohledu.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Milet, Ph.D.

Herní mechaniky založené na různých úhlech pohledu

Prohlášení

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

.....
Jan Zdeněk
28. dubna 2024

Poděkování

V této sekci bych chtěl poděkovat všem lidem, kteří se podíleli na testování hry MageFlip a hlavně vedoucímu mé práce Ing. Tomáši Miletovi, Ph.D, který mi vždy ochotně poradil a pomohl při psaní i implementaci tohoto díla.

Obsah

Kapitola 1

Úvod

Videohry každodenně přitahují pozornost desítek milionů uživatelů, a proto není překvapivé, že nové tituly, vyvíjené jak mezinárodními korporacemi, tak malými nezávislými studii, jsou uváděny na trh velmi často. Tento fakt však neimplikuje nemožnost vytvoření něčeho originálního – a právě toto je hlavním cílem mé bakalářské práce.

Výstupem je tedy hra, která se vyznačuje unikátní herní mechanikou – změnou perspektivy z pohledu shora na pohled z profilu. Tato interaktivní videohra, která spadá do žánru roguelike, umožňuje hráčům po spuštění procházet jednotlivé úrovně a snažit se překonat své předchozí výsledky. Kromě průchodu úrovněmi hra rovněž nabízí metaprogresi, což zahrnuje vylepšování schopností vlastní postavy pro jednodušší průběh budoucích pokusů.

Dalším prvkem, který je začleněn do hry MageFlip, je procedurální generování jednotlivých úrovní za účelem zvýšení znovuhratelnosti. Toto generování se provádí pomocí celulárních automatů, které definují tvar dané úrovně, a algoritmu marching squares, jenž následně úroveň vytváří podle výstupu automatu. Kromě již zmíněných aspektů hra také představuje široké spektrum nepřátel, kteří mění svůj vzhled a vlastnosti v závislosti na hráčem zvolené perspektivě. Dále hra obsahuje pasti, jež se snaží hráči znemožnit postup do vyšších úrovní.

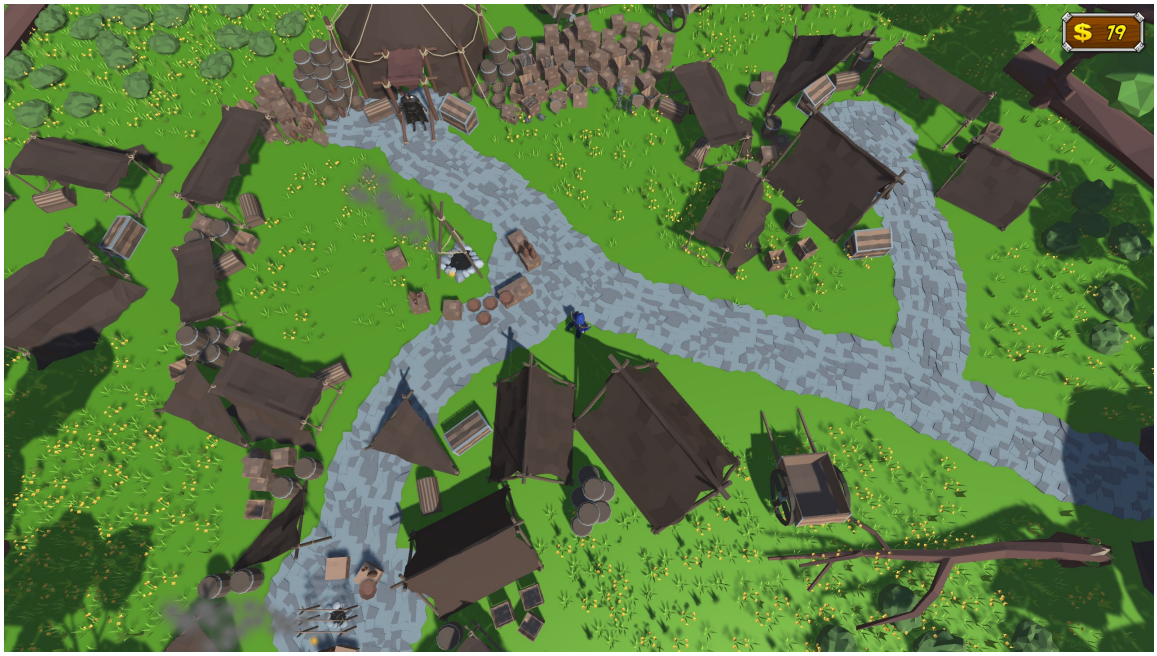
Výsledná hra je publikována na platformě OneDrive[1] pod názvem MageFlip, přičemž ukázky hry jsou dostupné na obrázcích 1.1 a 1.2 uvedených níže. Kromě samotné aplikace je dílo podrobně popsáno v této zprávě v následujících kapitolách. Kapitola 2 se zabývá stanovením cílů a jejich porovnáním s ostatními existujícími hrami. Další kapitola (3) poskytuje obecný úvod do světa videoher, včetně přehledu her, herního vývoje a nástrojů používaných při jejich vývoji. Tato kapitola rovněž obsahuje teoretický popis algoritmů použitých při realizaci hry. Kapitola 4 se věnuje návrhu herních mechanik a grafického uživatelského rozhraní. Závěrečné dvě kapitoly — implementace (5) a testování (6) -– dokumentují proces vývoje samotné hry.

[1] Dostupné z: https://1drv.ms/f/s!Ao8SOuS2wrlYkaoh9s_Xmszw-tccPA?e=u9eUde

(a) Pohled shora

(b) Pohled z profilu

Obrázek 1.1: Obrázky výše porovnávají dva pohledy, pohled shora a pohled z profilu, a slouží k představení hry MageFlip.



(a) Středověký kemp

(b) Souboj se silnějším nepřítelem

Obrázek 1.2: Obrázek (a) představuje podobu středověkého kempu, který slouží k vylepšování atributů hráče. Druhý obrázek obsahuje souboj s těžším nepřítelem.

Kapitola 2

Současná řešení a stanovení cílů

V současné době existuje na trhu široká paleta her, zahrnující rozmanité žánry s ohledem na různé charakteristiky. Z tohoto důvodu je obtížné vytvořit zcela originální hru. Nicméně podobného efektu lze dosáhnout kombinací prvků z jiných her, čímž je zajištěn jedinečný herní zážitek. Před začátkem vývoje bylo pro tuto práci stanoveno několik cílů, které musí splňovat.

Klíčovým požadavkem bylo vytvořit hru, kterou je možné zapnout i po dobu několika minut a která nebude hráče zahlcovat různými zbytečně komplikovanými mechanikami. Z tohoto důvodu a s ohledem na další faktory byl zvolen žánr **roguelike**. Tato volba je vhodná nejen z hlediska herního zážitku, ale také z perspektivy vývoje, protože roguelike žánr nevyžaduje přílišnou náročnost při tvorbě.

Dalším kritériem byla opakovatelná hratelnost výsledné hry. Tohoto lze docílit několika způsoby, kde nejpopulárnějším je vytvořit internetovou hru a nechat samotné hráče vytvářet obsah a herní situace. Pokud je však cílem hra, která není závislá na připojení k internetu, popularitě hry a vhodně doplňuje roguelike žánr, pak musí samotná hra ony nečekané situace vytvářet za využití **procedurálního generování** prostředí a objektů. Tímto způsobem hráč nikdy nebude vědět, co jej čeká, když hru zapne a může ji hrát opakovaně.

Produktů využívajících stejné herní prvky jako jsou uvedeny výše, je na trhu nespočet, a proto bylo rozhodnuto zakomponovat alespoň jednu zcela unikátní mechaniku. Byla zvolena možnost změny perspektivy, která funguje na bázi otočení světa o devadesát stupňů a tím umožní hráči přepínat mezi **pohledem shora a pohledem z profilu**. Tato mechanika vytváří pak zcela odlišné herní prostředí, což otevírá dveře dalším zajímavým herním prvkům, které změnu perspektivy využívají.

2.1 Roguelike hry

Celý žánr roguelike je pojmenován podle hry Rogue: Exploring the Dungeons of Doom (1980) a jak již název napovídá, jedná se o hry podobné právě této. Roguelike hry se obecně vyznačují několika klíčovými herními aspekty, mezi něž patří především existence jednoho života. Pokud chce hráč ve hře pokračovat i po smrti, musí začít od začátku a postupně se dostat tam, kde skončil dříve. Toto na straně hráče vyžaduje opatrnost a minimalizaci chyb.

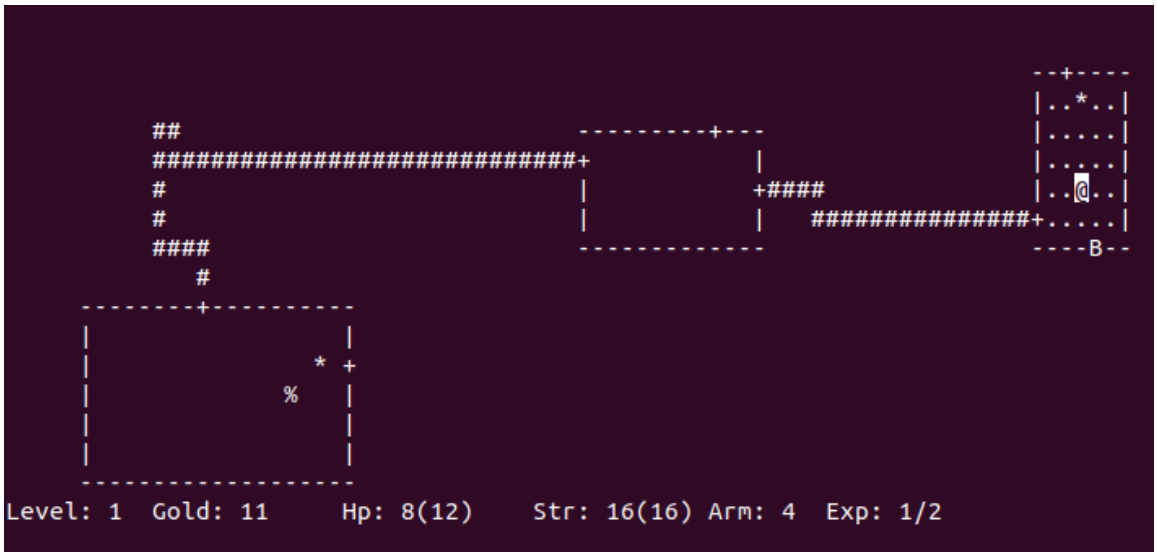
V současné době je však ve valné většině roguelike her zakomponována i metaprogrese v podobě vylepšování atributů mimo hlavní herní běh. Toto pak dovoluje hráči začít nový pokus v lepší pozici než minule, což přináší element postupného posunu vpřed. Dalším

indikátorem roguelike her je procedurální generování prostředí, bez kterého by nebylo možné docílit neomezené znovuhratelnosti.

Pevná definice tohoto herního žánru neexistuje, ale v rámci společného konsensu můžeme označit hry podobné, ale nesplňující některé podmínky pro herní žánr rogulike, jako roguelite hry.

2.1.1 Rogue: Exploring the Dungeons of Doom

Tento prapředek moderních roguelike a roguelite her je na první pohled velice odlišný od moderních titulů, což lze pozorovat na obrázku 2.1 níže. Rogue je hrán výhradně pomocí terminálu a k jeho ovládání je k dispozici pouze klávesnice. Hra generuje úrovně, z nichž každá obsahuje několik prozkoumatelných místností. I když se to na první pohled může jevit jednoduše, už jen zapamatování všech klávesových zkratek lze v dnešní době považovat za náročnou přípravu před samotným hraním.



Obrázek 2.1: Ukázka ze hry Rogue: Exploring the Dungeons of Doom, kde lze pozorovat jednotlivé místnosti ohraničeny svislými a vodorovnými čárami, chodby mezi jednotlivými místnostmi (znak #), hráče tvořeného zavináčem a jiné předměty nebo nepřátele.

2.1.2 The Binding of Issac

Jedná se o opět na povrch jednoduše vypadající hru, která nabízí hodně pokročilých herních mechanik. Hra The Binding of Issac je podobná této bakalářské práci a ze začátku vývoje byla největší inspirací. Mezi podobné mechaniky patří pohled shora a soubojový systém střílení projektilu[1].

2.2 Hry využívající procedurální generování

Procedurální generování je častým jevem u her nejen typu roguelike, ale i v jiných žánrech, například ve hrách o přežití, kde je však svět vygenerován pouze u jeho vytváření. Na

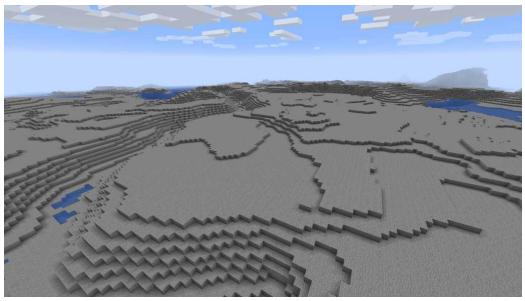
[1]Soubojový systém ve hře The Binding of Issac funguje na bázi míření pomocí šipek, kdežto v rámci této bakalářské práce je míření řešeno pomocí myši.

rozdíl od tohoto typu je u roguelike her svět tvořen při každém běhu (po smrti hráče) nebo i častěji. Vzhledem k popularitě tohoto způsobu existuje široká škála různých metod, jak prostředí generovat.

Tyto metody zahrnují tvorbu místností nebo dokonce celých světů za běhu hry a to podle předem stanovených pravidel pro generování. Všechny hry, které jsou uvedeny výše jako příklady roguelike her, využívají procedurální generování, avšak v této sekci je představeno i her, které do uvedeného žánru nespadají.

2.2.1 Minecraft

Minecraft je nejen jednou z nejpopulárnějších her s procedurálním generováním, ale i jednou z nejpopulárnějších her obecně. Procedurální generování terénu je v této hře od jejího začátku a stále zůstává jedním z nejvýznamnějších aspektů hry. Generování terénu dovoluje hráči celý svět prozkoumávat a přetvářet jej dle libosti. Svět generovaný v této hře je obrovský a mnohem větší než u ostatních roguelike her. Dle článku The World Generation of Minecraft [13] generování probíhá v několika krocích, kde v prvním kroku je za využití Perlinova šumu s předem určeným seedem (3.1) vygenerován hrubý terén a za využití dalších metod je svět obohacen o různé biomy(3.1), struktury a další objekty. Toto postupné generování lze vidět na obrázcích 2.2 níže.



(a) Svět v 1. fázi generování

(b) Kompletně vygenerovaný svět

Obrázek 2.2: Na obrázcích ze článku uvedeného výše [13] lze pozorovat postupné generování světa ze hry Minecraft, kde obrázek (a) je pouze kostra světa a obrázek (b) ukazuje již finální produkt s vygenerovanými strukturami a objekty.

2.2.2 No man's sky

Co se generování obrovských světů týče, No man's sky nemá ve světě herního vývoje konkurenci. V této počítačové hře se totiž negeneruje pouze jeden svět, ale hned několik světů v podobě velkého množství odlišných planet. Hráč má možnost všechny tyto planety prozkoumávat a stejně jako u hry Minecraft je schopen je i přetvářet k obrazu svému.

2.2.3 Valheim

Valheim byl na tento seznam zařazen nejen díky generování samotného světa, ale především proto, že se v této hře nachází i procedurálně generované podzemní místnosti. Hlavním rozdílem je však způsob a vzhled, kterým jsou tyto místnosti generovány. U hry Valheim je generování realizováno pomocí předem vytvořených místností, které jsou následně spojovány a tím tvoří celé podzemí. Cílem generování v rámci této práce by měly být místnosti, které

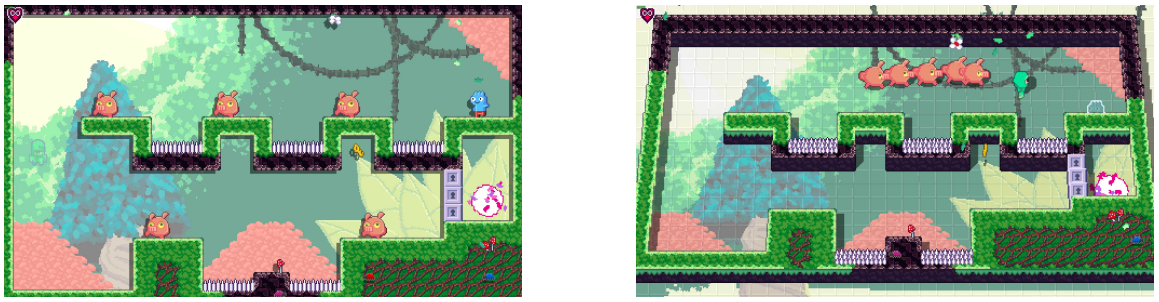
jsou generovány v celé své podobě a to pouze na základě předem stanovených jednoduchých pravidel.

2.3 Hry se změnou perspektivy

Her, které si hrají s perspektivou, je na trhu podstatně méně než roguelike her s procedurálním generováním. Navíc velká část z nich jsou hádankové hry, kde změna perspektivy ovlivňuje různé hádanky a jejich řešení. Ještě méně je pak her využívajících změnu z pohledu shora na pohled z profilu.

2.3.1 Toodee and Topdee

Jedná se o jeden z mála titulů, který využívá zrovna tento styl změny perspektivy (tato změna je ukázána na obrázcích 2.3). Hra tuto mechaniku využívá především k řešení hádanek. Spadá však mimo žánr roguelike a nevyužívá procedurální generování úrovní.



(a) Pohled z profilu

(b) Pohled shora

Obrázek 2.3: Obrázky obsahující porovnání dvou perspektiv mezi kterými lze přepínat a tak řešit připravené hádanky ve hře Toodee and Topdee.

2.4 Hry kombinující žánr roguelike a změnu perspektivy

V současné době se na trhu nevyskytuje hra, která by vyhověla všem uvedeným požadavkům. Hry obvykle spadají pouze do jedné z kategorií. Tento fakt je způsoben především absencí her, které kombinují změnu perspektivy a zároveň se neřadí mezi hádankové hry. Vývoj hádankové hry s procedurálním generováním představuje komplexní úkol, zvláště vzhledem k tomu, že stejně jako hráč ani vývojář nemá předem informaci o podobě herního prostředí.

Kapitola 3

Teorie

Cílem této kapitoly je čtenáře seznámit s herním průmyslem a vývojem počítačových her a následně mu představit hlavní koncepty vývoje tohoto díla. Těmito hlavními koncepty je generování světa při spuštění a v průběhu hry a vysvětlení mechaniky změny perspektivy. Ke každému způsobu řešení jsou dále představeny alternativy nebo i jiné hry, které těchto či podobných mechanik využívají. Celá kapitola je soustředěna pouze na teoretické aspekty metod a algoritmů pouze s minimálním nastíněním implementace.

3.1 Seznam pojmů

Níže je výčet některých pojmů, které jsou používány v anglickém jazyce a nemají překlad nebo se jejich překlad běžně v českém jazyce nepoužívá.

Seed

Seed (česky „semínko“) je hodnota často v podobě řetězce a slouží jako vstup pro generátor pseudonáhodných čísel. Stěžejní vlastností seedu musí být dosažení totožného výsledku při jeho opakovaném použití, jinak seed ztrácí smysl. V praxi se se seedem může setkat i hráč a to v případě, kdy daná hra jeho použití dovoluje.

Biom

Biom slouží jako označení různých geografických oblastí a při vývoji her dovoluje definovat pro zvolené oblasti různé atributy. Mezi tyto atributy patří například teplota, vlhkost a tvar terénu. Jednotlivé atributy dále definují, jaká vegetace nebo fauna se bude v daném biomu nacházet. Rozdělení světa na biomy tedy umožňuje tvořit realisticky vypadající hry.

Mesh

Mesh neboli síť slouží k definici tvaru modelů pomocí trojúhelníků, které jsou tvořeny spojením vždy tří bodů v prostoru. Pomocí meshe dále můžeme definovat jak se od daných modelů bude odrážet světlo (normálová mapa) nebo jak se na daný model bude „nanášet“ textura (UV mapa).

Boss

Boss je ve videoherním průmyslu označení pro obzvláště silného nepřítele, který se však vyskytuje jen zřídka. Jeho poražení často vyžaduje více času a to i díky mechanice více etap, které hráč musí absolvovat. Tyto etapy se nazývají fáze.

Spawnování

Spawnování je proces instanciace objektů do prostoru. V rámci tohoto procesu dochází k umístění nepřátel nebo jiných objektů do scény. Místo nebo objekt, který je „zodpovědný“ za tento proces, se nazývá spawner.

3.2 Počítačová hra

Jedná se o program spustitelný na libovolném elektronickém zařízení, především na počítačích, mobilních telefonech a herních konzolích. Dále by tato aplikace měla splňovat několik základních kritérií. Primárním účelem každé hry by měla být zábava, neslouží tedy k produktivitě. S výjimkou některých sandboxových her, musí hra definovat cíl a postup k tomuto cíli. Cíle může být dosaženo poražením nepřítele, dokončením příběhu nebo dosažením největšího počtu bodů. Hry obvykle disponují možností prohry, která se často projevuje smrtí. Počítačové hry lze rozdělit do několika žánrů, přičemž jedna hra může spadat do více kategorií.

3.2.1 Roguelike/roguelite

Tento žánr byl již představen výše v kapitole 2 – Současná řešení a stanovení cílů a tudíž bude jen uvedeno několik další titulů jako např.:

- Spelunky 2;
- Kingdom.

3.2.2 Sandboxové hry

Cílem sandboxových her je nechat hráči prostor pro experimentování a hraní si s různými prvky, které daná hra nabízí. Tento typ her se vyznačuje absencí striktně daného cíle a naopak podporuje svobodu projevu hráče. Hráči sandboxových her často vytvářejí vlastní stavby, vozidla nebo jiné objekty a následně je jim umožněno s nimi experimentovat. Toto může být doprovázeno možností sdílení výtvorů s ostatními hráči. Mezi příklady sanboxových her patří:

- Raft;
- Garry’s mod.

3.2.3 Hry o přežití

Cílem v rámci her o přežití je, jak název naznačuje, přežít. Velké procento těchto her klade důraz na podobu s realitou a tudíž obsahují mechaniky jako hlad, žízeň nebo únava. Tyto hry jsou obvykle těžké na začátku a s postupným získáváním objektů a zkušeností se stávají

lehčími pro přežití a hráč se tak může soustředit na jiné herní mechaniky jako je stavění nebo souboj. Příklady některých takových her jsou:

- The Forest;
- Don't starve.

3.2.4 Hádankové hry

U hádankových her musí hráč zapojit logické myšlení, vyřešit předem připravené hádanky a dostat se tak k cíli. Hry se skládají často z kolekce úrovní, přičemž hráč musí vyřešit každou z nich. Kromě logických úloh může být do hádanek začleněna i mechanika pohybu, kterou se hráč musí naučit. Mimo hry Topdee and Toodee představené v sekci 2.3.1 do této kategorie patří například:

- Unravel;
- It Takes Two.

3.3 Herní vývoj

Vývoj her je ve svém jádru podobný vývoji jiných softwarových programů, avšak pro vývoj celé hry je potřeba více než jen programátorů. Níže jsou uvedeny hlavní části produkce videoher (tento seznam a doplňující informace byly převzaty z knihy Game Development and Production [2]):

- **Návrh:** Tato část se stará o širší obrázek celé hry a určuje, o čem hra bude, jakého bude žánru a jak bude výsledný produkt vypadat. Mezi hlavní role této části patří například vedoucí návrhář, který je zodpovědný za obecnou vizi projektu, návrhář úrovní nebo autor dialogů a příběhu.
- **Programování:** Programátoři společně s umělci tvoří samotnou hru podle představ návrhářů a starají se tedy o implementaci herních mechanik, grafiky, umělé inteligence, úrovní a sítě. Kromě těchto úkolů je také důležitý i tým starající se o vývoj nástrojů, které dále usnadňují proces vývoje.
- **Umění:** Artistické oddělení zodpovídá za tvorbu modelů, textur, animací a uživatelských rozhraní. Do této části lze dále zařadit i hlasový doprovod (angl. voice-over), tvůrce zvukových efektů nebo skladatele hudby.
- **Aktéři nepodílející se na vývoji:** Obzvláště velká herní studia obsahují i oddělení, která se přímo na vývoji hry neúčastní. Mezi tato oddělení patří řízení herního studia, kontrola kvality v podobě testerů nebo role týkající se průzkumu trhu a zajištění potřebných licencí. Dále je potřeba zajistit vydavatele a distributora samotné hry, aby se produkt dostal k zákazníkovi.
- **Podpora po vydání:** V současné době se herní průmysl posouvá stále víc k vývoji tzv. *live service* her (hry, které často nejsou kompletní při vydání a vyžadují časté aktualizace). Z tohoto důvodu potřebují hry zahrnovat i podporu po vydání. Tým podpory se stará o hru někdy i dlouhé roky a pravidelně upravuje nebo přidává herní mechaniky a opravuje chyby, které hra obsahovala při vydání.

Na trhu existují indie hry, které jsou vyvíjeny malými herními studii nebo i jednotlivci. Indie studia často nemají všechny části produkce uvedené výše a zaměstnanci těchto studií plní více rolí.

3.3.1 Problémy vývoje her

Všechna herní studia se často potýkají s problémy při herním vývoji. Tyto problémy mohou být interní (například technické problémy při vývoji) nebo externí (potíže ze strany zákazníka po vydání). Zvláště externí problémy mohou být existenční vzhledem k často křehké důvěře mezi zákazníkem a studiem. Níže je pak výčet a popis nejčastějších problému, které mohou při nebo i po vydání nastat.

Nedostatek financí

Nedostatek finančních prostředků je častým důvodem ukončení vývoje her. Tento problém vzniká především z důvodu nedostatečného plánování a nerealistických předpokladů ohledně financování nebo v případě prodloužení vývoje hry. Jako řešení se poté nabízí redukce obsahu dané hry nebo získání potřebných financí za pomocí tzv. *crowdfunding* (financování ze strany komunity).

Změny v plánu a rozsahu

Změna původního plánu může být iniciována nejen již zmíněným nedostatkem finančních prostředků, ale i změnou technologií nebo zpětnou vazbou hráčů. V případě, kdy herní studio na tyto popudy nereaguje, může tato skutečnost vést ke zhoršení spokojenosti uživatelů.

Zpoždění

Další problém, se kterým se potýká většina titulů, je zpoždění v rámci vývoje hry. Důvodem může být špatný odhad času a zdrojů, nečekané technické problémy nebo neschopnost reagovat na vzniklé překážky.

Konkurence a tržní změny

Vzhledem ke skutečnosti, že vývoj her je dlouhý proces, se může trh během vývoje drasticky změnit. Tyto externí vlivy pak mohou mít negativní dopad na samotnou úspěšnost hry. Mezi nejčastější změny patří uvedení podobné konkurenční hry na trh nebo změna preference mezi uživateli.

Legislativní a licenční problémy

Dalšími, i když méně zastoupenými, problémy mohou být ty z oblasti práva. Do této kategorie spadá například porušení autorských práv nebo změna v legislativě. Tento problém nastává zejména u menších herních studií, které nemají k dispozici rozsáhlý expertní tým.

3.3.2 Proces vývoje her

Vývoj videoher je komplexní proces, který obsahuje několik fází a vyžaduje různé odborné dovednosti. Tento proces může vypadat následovně:

Plánování a vize

Tato fáze zahrnuje formování základního konceptu hry, včetně návrhu, příběhu a identifikace cílového uživatele. Cílem této fáze je nasměrovat vývojáře a definovat hlavní rysy a technické specifikace hry.

Prototypování

V této fázi se vytvářejí prototypy herních mechanik, což umožňuje jednotlivé prvky testovat a postupným iterováním vybírat způsob následného vývoje. Prototypování dovoluje validovat koncepty a rozhodnout o jejich přínosu do hry.

Vývoj

Vývoj představuje jádro celého procesu tvorby hry. Mimo implementace mechanik v této fázi probíhá i tvorba textur, modelů a zvuků.

Testování

Testovací fáze je potřebná pro identifikaci chyb ještě před vydáním samotného titulu a dovoluje se vrátit do fáze vývoje pokud je to potřeba. Testování nemusí probíhat pouze interně, ale může zahrnovat i hráče (tzv. *beta testování*).

Optimalizace

Ještě před vydáním je potřebné hru dostatečně optimalizovat v podobě zlepšení výkonu, oprav chyb nebo vylepšení některých herních mechanik.

Vydání

V tomto kroku proběhne zhodnocení výsledného produktu z pohledu veřejnosti a často se zde rozhodne o dalším směřování studia. Mimo samotného vydání hry může tato fáze zahrnovat i propagační aktivity.

Podpora po vydání

Po vydání hry může tým pokračovat v práci na aktualizacích, opravách chyb, rozšíření, nebo stahovatelném obsahu (DLC).

3.4 Herní engine

Herní engine je softwarový nástroj, jehož hlavním cílem je zjednodušit vývojářům proces vytváření interaktivních videoher. Vývojář, který daný herní engine využívá, nemusí hru programovat od základů, ale může využít předdefinované komponenty poskytované zvoleným herním enginem, což značně usnadňuje vývojový proces. Multifunkčnost herních enginů vývojářům dovoluje použít jeden engine pro vývoj několika různých her. Některé enginy dovolují vytvářet hry všech žánrů, různých počtů hráčů nebo i světy ve 2D i 3D.

Engine nadále slouží jako předloha pro vývoj her a psaní kódu a umožňuje práci s editorem, což je grafické uživatelské rozhraní dovolující měnit hru bez nutnosti navštívení

kódu. Mezi hlavní komponenty patří například grafický engine, fyzikální engine a umělá inteligence.

Vlastnosti a nástroje poskytované herními enginy však nejsou limitované pouze pro vývoj her a lze jich využít i v jiných odvětvích. Jak je popsáno v práci Game Engines In Scientific Research [8], lze herních enginů využít i pro vědecké účely. Mezi další využití patří například architektonická vizualizace, umění, zdravotnictví nebo výcvik ve vojenství, leteckém průmyslu a dalších oblastí vyžadující jinak nákladný výcvik personálu.

3.4.1 Grafický engine

Grafický engine představuje klíčovou složku každého herního enginu, je zodpovědný za vykreslování 2D nebo 3D grafiky na obrazovku a umožňuje vývojářům vykreslovat modely a textury bez složitého programování. Výhodou využití grafického enginu je také možnost hru vytvořit pro různé operační systémy a různá zařízení. Mimo vykreslování modelů a textur se grafický engine také stará o správné osvětlení a stíny ve scéně.

3.4.2 Fyzikální engine

Fyzikální engine je komponentou, která simuluje fyzikální interakce mezi jednotlivými objekty ve virtuálním prostoru. Stará se tudíž o pohyb, kolize a obecné chování objektů ve světě. Lze jej využít pro simulaci gravitace, fyzikálních sil, kolizí a celkové dynamiky objektů. Mezi pokročilejší funkce fyzikálního enginu patří například simulace tekutin nebo plynů.

3.4.3 Umělá inteligence

Herní engine nám dále dovoluje použít umělou inteligenci, neboli definovat chování postav, které nejsou ovládány hráčem. S využitím umělé inteligence můžeme určovat navigaci a pohyb postav v herním světě, definovat jejich rozhodovací procesy nebo postavy obohaceny o umělou inteligenci nechat reagovat na své okolí.

3.4.4 Editor hry

Editor je uživatelské rozhraní, které vývojáři umožní vytvářet, upravovat a spravovat obsah hry, slouží tedy hlavně k úpravě úrovní nebo scén, vkládání objektů do scény, definování herních pravidel a nastavování různých parametrů. Mezi další vlastnosti editoru patří manipulace s objekty, úprava animací a další.

3.4.5 Příklady herních enginů

Mezi nejznámější herní enginy patří Unity, Unreal a Godot, které jsou k dispozici každému. Některá velká herní studia však používají své vlastní herní enginy, které nejsou zpřístupněny pro veřejnost. Tato bakalářská práce byla zhotovena v enginu Unity, který je popsán detailněji v sekci 3.5.

Unreal Engine, vyvinutý společností Epic Games, je hlavním konkurentem Unity. Vyznačuje se jeho grafickými schopnostmi a vizuálním skriptovacím systémem, který umožňuje vývoj her bez programování. Unreal je velice multifunkčí a obsahuje všechny potřebné komponenty jako grafický engine, fyzikální engine a editor a navíc podporuje i virtuální realitu. Unreal umožňuje programování v jazyce C++ nebo za využití již zmíněného vizuálního skriptování.

Dalším významným hráčem v této oblasti je **Godot**, který se vyznačuje zpřístupněným zdrojovým kódem (angl. opensource). Godot využívá svůj vlastní skriptovací jazyk GDScript, který je podobný jazykům jako Python. Vše ve scéně je reprezentováno pomocí uzlů, které představují objekty. Tabulka 3.1 níže pak obsahuje porovnání všech zmíněných herních enginů.

Funkce/Aspekt	**Unity**	**Unreal**	**Godot**
Grafika	Dobré	Vynikající	Dobré
Licence	Bezplatná i placená	Licenční poplatky	Bezplatná
Komunita	Velká	Velká	Rostoucí
Knihovna aktiv	Velká	Kvalitní aktiva	Limitovaná
Skriptovací jazyk	C#	C++	GDScript
Více platforem	Ano	Ano	Ano
Podpora VR	Ano	Ano	Experimentální
Podpora 2D	Ano	Horší optimalizace	Dedikovaný 2D
Fyzikální engine	NVIDIA PhysX	Chaos Physics	Godot Physics
Vzdělávací křivka	Střední	Strmá	Střední
Dokumentace	Rozsáhlá	Rozsáhlá	Rostoucí
Návody	Hojné	Hojné	Rostoucí

Tabulka 3.1: Tabulka obsahuje porovnání herních enginů dle článku Unity vs Unreal Engine vs Godot: Which Is Best for Indie Developers? [7]

3.5 Herní engine Unity

Unity je komplexní herní engine, který poskytuje vývojářům široké spektrum nástrojů pro tvorbu interaktivních her a simulací. Jeho grafický editor umožňuje uživatelům snadno vytvářet a upravovat herní scény. Díky konceptu komponentní architektury mohou vývojáři přidávat různé komponenty k herním objektům, což umožňuje flexibilní definici chování a vlastností. Níže jsou uvedeny některé hlavní vlastnosti tohoto enginu:

- Jedním z klíčových prvků Unity enginu je jeho podpora různých platforem. Unity nabízí možnost distribuce her na široké škále zařízení a operačních systémů jako například Windows, MacOS, Linux, Android, iOS nebo také zařízení pro virtuální nebo augmentovanou realitu.
- Programování v Unity je převážně založeno na jazyku C#, což umožňuje programátorům snadno implementovat herní logiku, ovládání postav, umělou inteligenci a další klíčové aspekty.
- Tento engine rovněž obsahuje vestavěný systém pro simulaci fyziky, což přispívá k realističnosti pohybu objektů a interakcí ve virtuálním prostředí. Tímto způsobem mohou vývojáři vytvářet hry s autentickým vzhledem a pocitem.
- Unity dále podporuje komunitní rozšíření a balíčky, což umožňuje sdílení nástrojů mezi vývojáři.
- V tomto enginu je vše reprezentováno pomocí herních objektů, kterým je poté pomocí skriptů nebo dalších objektů přiřazeno chování.

3.5.1 Herní objekt v Unity

Herní objekt v Unity je základní stavební prvek pro všechny entity ve scéně a může obsahovat různé komponenty, které společně definují jeho chování a vzhled. Každý objekt má několik základních vlastností. Nejzákladnější vlastností je pozice ve scéně (řešená pomocí komponenty *Transform*), která je reprezentována trojrozměrným vektorovým bodem. Kromě pozice má objekt ještě rotaci a škálování, což umožňuje objektu měnit orientaci a velikost.

Mezi nejběžnější komponenty objektu dále patří *Mesh Renderer*, který zajišťuje viditelnost objektu s použitím modelů a materiálů. Jako další komponenta může být přidán skript, který definuje jeho chování pomocí kódu. Objekty v Unity mohou být použity nejen pro různé reálné objekty ve hře, ale mohou obsahovat i abstraktní funkcionality bez potřebné vizualizace.

Z objektů vytvořených v Unity lze vytvořit tzv. **prefab**, který slouží jako vzor pro další objekty. Z prefabu je později možné vytvořit samostatnou instanci (např. pomocí skriptu), které může vývojář měnit vlastnosti pomocí změny interních atributů nebo i přidáváním a odděláváním komponentů.

3.5.2 Herní smyčka v Unity

Herní smyčka je hlavním prvkem, který řídí průběh hry. V rámci této smyčky se provádějí fyzické simulace, vstupy od uživatele a vše potřebné pro běh samotné hry včetně vykreslování na obrazovku. V průběhu se invokují různé metody, které vývojáři umožňují definovat chování jednotlivých herních objektů. Níže jsou některé tyto metody popsány.

Awake

Awake je metoda, která je volána vždy jednou – při načtení skriptu. Tato metoda může být využita například pro nastavení referencí mezi skripty nebo pro definici proměnných, které budou použity před voláním metody *Start*.

OnEnable a OnDisable

Tato dvojice metod je invokována při aktivaci a deaktivaci daného objektu s daným skriptem (tedy mohou být volány i vícekrát). Často je těchto metod využíváno pro odběr a zrušení odběru různých událostí nebo pro inicializace proměnných, které je potřeba provádět vždy po aktivaci objektu.

Start

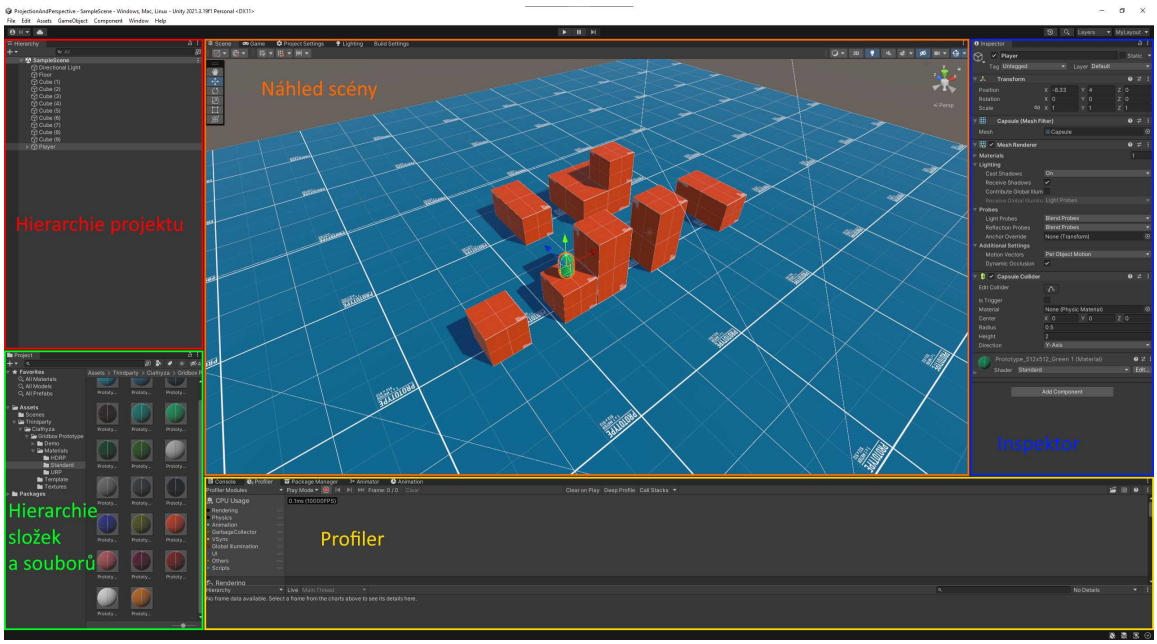
Metoda *Start* je volána před prvním voláním metody *Update* a slouží tedy jako poslední bod pro inicializaci proměnných, které jsou používány při běhu hry. Dále lze využít faktu, že u všech jiných skriptů proběhlo invokace metody *Awake*.

Update a FixedUpdate

Tyto metody jsou volány opakovaně (metoda *Update* pro každý snímek a metoda *FixedUpdate* vždy se stejným a předem určeným časovým rozestupem) a slouží k definici chování herních objektů v čase.

3.5.3 Unity editor

Jedná se o, jak již bylo zmíněno výše v sekci 3.4.4 – Editor hry, grafické uživatelské rozhraní, které dovoluje část herního vývoje provádět bez nutnosti úpravy samotného kódu. Mimo jiné lze v editoru Unity přemístit objekty ve scéně, měnit jejich velikost nebo rotaci a přidávat jim komponenty, které ovlivní jejich chování. Příklad vzhledu editoru Unity jako celku je k dispozici na obrázku 3.1, přičemž podrobnější popis jednotlivých nástrojů je uveden níže.



Obrázek 3.1: Ukázka celého editoru Unity, kde je vidět hierarchie projektu, hierarchie složek a souborů, úprava objektu (tzv. *inspektor*), profiler a náhled scény.

Úprava objektů a jejich vlastností

Úprava jednotlivých objektů v Unity editoru probíhá pomocí samostatného okna *Inspector*. Tento nástroj dovoluje každému hernímu objektu přidávat jednotlivé komponenty, které pak definují jeho vlastnosti a chování. Tyto komponenty mohou být doprovázeny množstvím parametrů, které dovolují chování měnit přímo v editoru a za běhu programu. Níže jsou popsány nejčastější komponenty:

- *Transform* je komponenta, která definuje polohu, rotaci a velikost daného objektu ve scéně pomocí níž můžeme dosáhnout přesnějších hodnot, než při využití úpravy scény.
- Další častou komponentou je *RigidBody* v kombinaci s některým komponentem typu *Collider*. Tato dvojice umožňuje definovat chování objektu s ohledem na fyzikální zákony. Pomocí první komponenty je vývojář schopen určit hmotnost, odpor a úhlový odpor a zda je objekt zatížen gravitací. Druhá komponenta pak definuje meze, kde se objekt stává hmotným a je tudíž ovlivněn srážkou s jinými objekty.
- Stěžejní komponentou je pak *Script*, který dovoluje objektu přiřadit skripty psané v jazyce C#. Parametry jednotlivých skriptů je možné měnit také přímo v editoru. Přidáváním skriptů je vývojář schopen definovat vlastní chování objektu.

K objektu můžeme přiřadit i více komponent stejného druhu, které je poté možno aktivovat nebo deaktivovat. Přidávat a definovat komponenty je možné provádět i za běhu programu a to třeba přes skript.

Náhled a tvorba animací a animátorů

Editor Unity poskytuje nástroj pro tvorbu animátorů. Animátor je komponenta, která slouží pro řízení animací, tedy určuje, jaká animace se má v daný okamžik přehrávat a definuje podmínky přechodů mezi nimi. Každý animátor může obsahovat více vrstev s různou váhou. Uzly animátoru jsou reprezentovány jednotlivými animacemi nebo pomocí tzv. *BlendTree*, který umožňuje plynulé přechody mezi animacemi.

Nástroj, který upravuje samotné animace, může vývojář najít v okně *Animation*. Každá animace obsahuje několik animačních snímků, které se zde mohou upravovat. Právě v okně *Animation* je možné i přidávat události, které jsou později invokovány v případě dosažení daného snímku animace.

Modely, textury a materiály

Unity editor bohužel nemá zabudovaný nástroj pro tvorbu či úpravu modelů ani textur, avšak je možné je zobrazit, upravit některé parametry pro jejich použití a manipulovat s nimi. Materiály je možné vytvářet a konfigurovat přímo v editoru. Všechny úpravy a zobrazení probíhají za pomoci inspektoru stejně jako je tomu s objekty.

V Unity dále existují ještě fyzické materiály, které je možno zcela vytvářet. Fyzické materiály slouží pro úpravu fyzických vlastností objektů a je možné pomocí nich měnit tření a skákavost.

Náhled hry

K ulehčení testování hry přímo za běhu aplikace vývojářům slouží okno *Game*, které zpřístupní hru přímo v editoru bez nutnosti pokaždé program přeložit a spustit samostatně. Toto okno zpřístupňuje plnou hru a navíc umožňuje měnit parametry herních objektů pomocí inspektoru a přesouvat je pomocí náhledu scény.

Správa balíčků

Unity se pyšní širokou nabídkou balíčků, které jsou dostupné přímo z oficiálního tržiště Unity Asset Store. Balíčky jsou často v podobě modelů, textur nebo ikon, které si vývojář může koupit přímo na tomto tržišti nebo je mu umožněno svoje vlastní výtvory zde sdílet a prodávat ostatním. Balíčky však mohou obsahovat i dodatkové nástroje pro tvorbu her. Správa balíčků probíhá pomocí nástroje *Package Manager*.

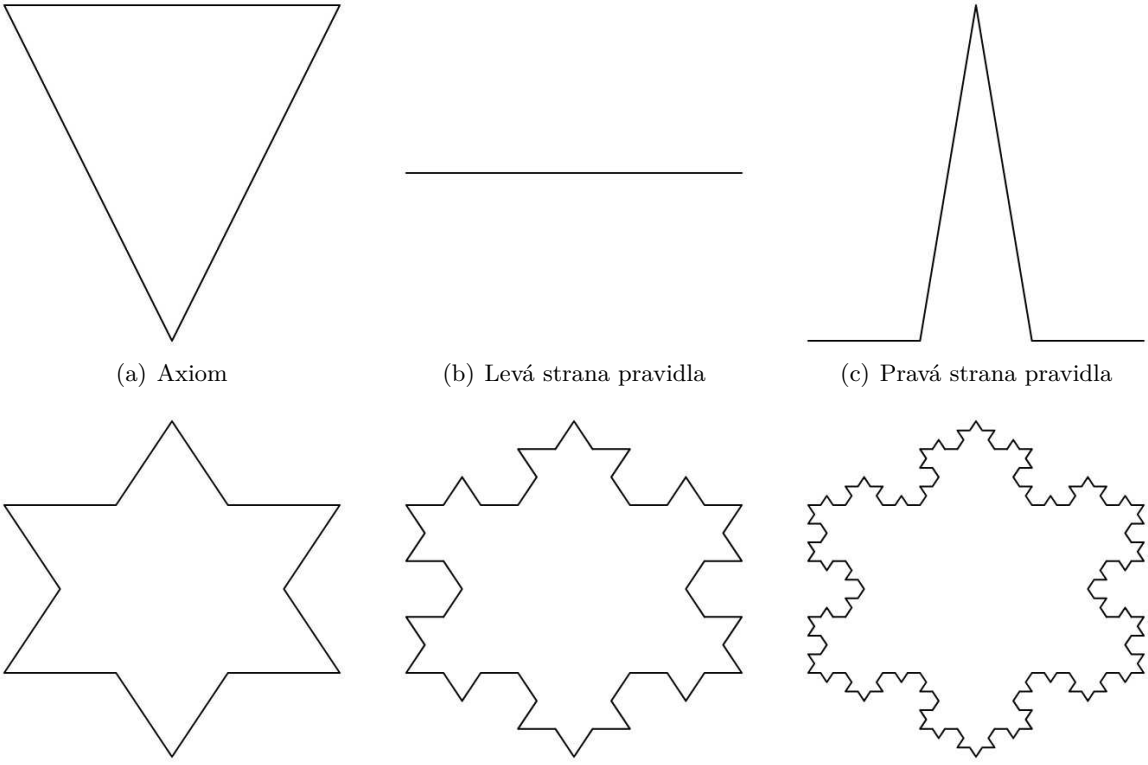
3.6 Procedurální generování

Procedurální generování je obecně proces vytváření dat pomocí algoritmů, často založených na náhodě nebo na matematických pravidlech. Obsah se tedy nevytváří manuálně (staticky), ale je vytvářen dynamicky za běhu programu.

V herním průmyslu můžeme procedurálního generování využít hlavně pro tvorbu terénu nebo jednotlivých úrovní, ale v určitých případech nám může umožňovat procedurálně vytvářet i postavy nebo objekty, zvuky (podle článku Procedural Sound Generation for Soft

Bodies in Video Games [5]) nebo jiné aspekty naší hry. V kinematografickém průmyslu lze využít technik procedurálního generování pro vytváření realistických kulis nebo speciálních efektů. Další uplatnění tohoto způsobu můžeme najít i v umění, kde je procedurální generování někdy zodpovědné za vytváření obrazů nebo i soch na základě pravidel definovaných umělcem. Mezi základní typy procedurálního generování (v oblasti herního vývoje) patří:

- **Celulární automaty** 3.6.1
- **Perlinův šum** 3.6.3
- **L–systémy (Lindenmayerovy systémy):** Toto jsou matematické formální systémy vyvinuté jako model pro růst rostlin. Těchto systémů lze využít i při procedurálním generování jiných objektů, které jsou reprezentovány řetězcem vytvořeným iterativní aplikací jednoduchých pravidel. Mezi základní složky patří **axiom** (základní symbolický řetězec reprezentující počáteční stav), **pravidla** (soubor jednoduchých pravidel pro iterativní úpravu řetězce) a **abeceda** (jednotlivé symboly obsažené v řetězci). Příklad jednoduchého L–systému je ukázán na obrázku 3.2.



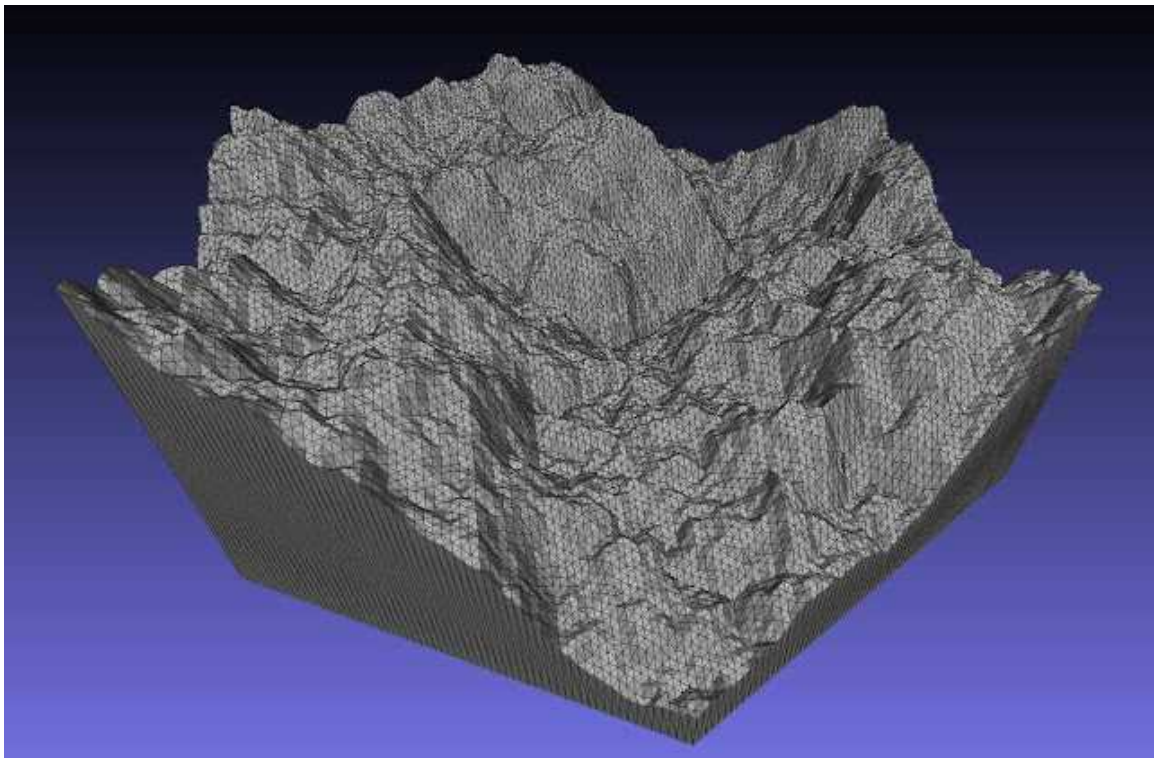
(a) Axiom (b) Levá strana pravidla (c) Pravá strana pravidla

(d) 1. iterace (e) 2. iterace (f) 3. iterace

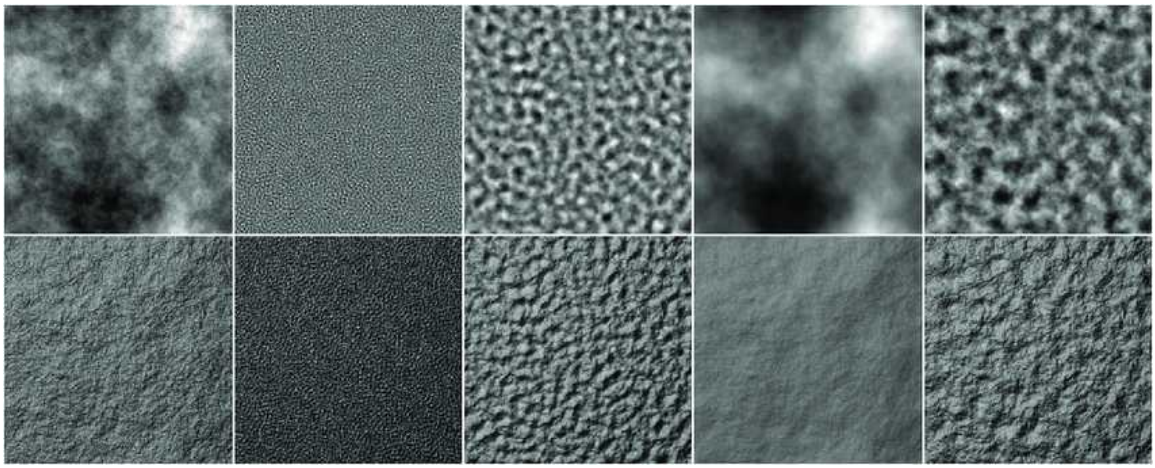
Obrázek 3.2: Výše lze vidět vizuální příklad L–systému převzatý z diplomové práce Marka Pasičnyka [9]. Jedná se o Kochovu vločku, jeden z prvních fraktálů vytvořený pomocí L–systémů. Na podobrázku (a) je vidět počáteční stav systému, vedle se nachází přepisovací pravidlo a níže je příklad jednotlivých iterací.

- **Fraktály:** Tato metoda využívá matematické struktury, které mají vlastnost soběpodobnosti. V procedurálním generování jsou využívány k vytváření složitých vzorů

jako jsou textury nebo terén. Příklad generování terénu pomocí metody fraktálů je obsahem kapitoly 2 ve knize Focus on 3D Terrain Programming [10]. Při generování terénu můžeme využít fraktálů pro vytváření detailních hor, údolí, rostlin nebo jiných prvků s přirozeným vzhledem. Použití této metody jsou předvedeny na obrázcích 3.3 a 3.4.



Obrázek 3.3: Obrázek obsahuje příklad terénu vygenerovaného pomocí fraktálů [11].



Obrázek 3.4: Dalším významným využitím fraktálů je tvorba textur, příklady takto stvořených textur jsou na obrázcích výše [4].

3.6.1 Celulární automaty

Pro generování jednotlivých úrovní bylo zvoleno v rámci této práce právě metody celulárních automatů. Jedná se o matematický model, který popisuje systém skládající se z buněk, které mohou nabývat určitých stavů a podléhají pravidlům, která definují jejich chování na základě jiných buněk v okolí. Základní prvky celulárního automatu jsou:

- **Buňky:** Jedná se o základní stavební jednotky, které mohou nabývat různých stavů.
- **Stavy:** Hodnoty, kterých mohou buňky nabývat. Stavy mohou být jednoduché (*živý*, *mrtvý*) nebo složitější (více hodnot nebo i hodnoty spojité).
- **Pravidla:** Množina jednoduchých pravidel, na základě nichž se buňky chovají. Pravidla pak definují, jak se buňky chovají v čase (jednotlivých iteracích) a v závislosti na okolních buňkách.

Proces evoluce celého modelu probíhá iterativně a v diskrétním čase. V každém kroku jsou aplikována pravidla na každou buňku a tím se stav dané buňky aktualizuje. Tento jednoduchý proces se pak opakuje a simuluje tak i komplexní a dynamický vývoj celého systému.

Conwayova Hra života (Game of Life)

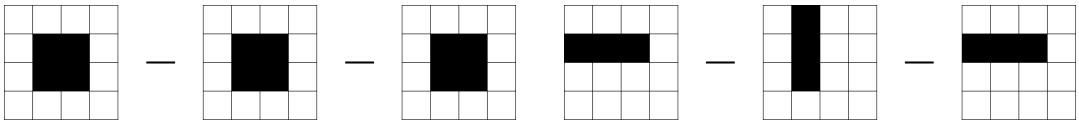
Jedná se o nejpopulárnější případ celulárního automatu, který z celého konceptu dělá hru. Herní svět je reprezentován pomocí dvourozměrné čtvercové sítě jejíž buňky (jednotlivá políčka sítě) mohou nabývat stavu *živý* nebo *mrtvý*. Stav buňky v další iteraci je pak určen přechodovou funkcí (pravidly). Pravidel pro tuto hru je několik a podle výběru se celý model bude chovat. Pravidla, jak je vymyslel John Conway, podle knihy Game of Life Cellular Automata [1][1]:

- buňka ve stavu *živý* v tomto stavu zůstane pouze tehdy když se kolem ní nachází 2 nebo 3 buňky ve stavu *živý*;
- buňka ve stavu *mrtvý* změní stav na *živý* pouze tehdy když se kolem ní nachází právě 3 buňky ve stavu *živý*.

I když se tato pravidla mohou zdát jednoduchá, lze s nimi vytvořit rozsáhlý systém. Tvary vytvořené simulací lze pak rozdělit do několika kategorií (příklady některých tvarů, které patří do zmíněných kategorií jsou k dispozici na obrázku 3.5) podle jejich chování:[2]

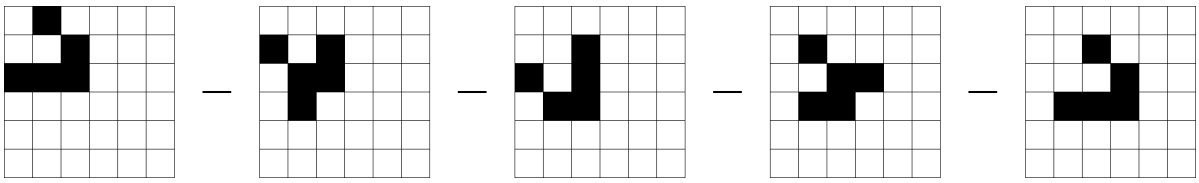
- **Zátiší:** Tyto tvary jsou stabilní a s následující iterací se nemění.
- **Oscilátory:** Tvary v této kategorii oscilují s periodou, neboli v určité iteraci se jejich tvar bude opakovat.
- **Děla:** Stacionární tvary, které vytvářejí jiné tvary jako například hvězdné lodě.
- **Hvězdné lodě:** Tvary, které se budou pohybovat, dokud nebudou přerušeny.

[1] Buňka reaguje vždy pouze na sousedící buňky (i diagonálně sousedící) a celkem jich tedy „vidí“ 8.
[2] Nejedná se o výčet všech kategorií.



(a) Zátiší (*block*)

(b) Oscilátor (*blinker*)



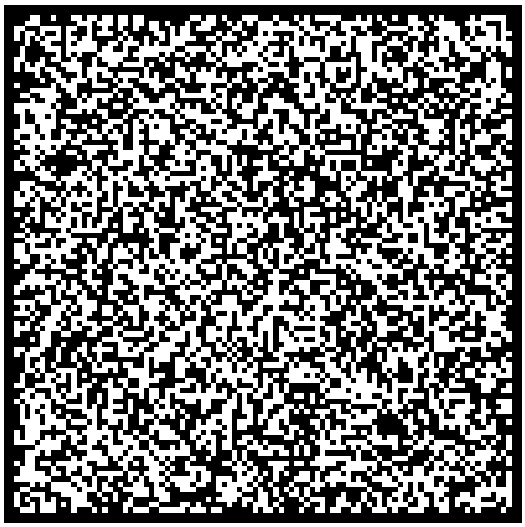
(c) Hvězdná loď (*glider*)

Obrázek 3.5: Každý z obrázků obsahuje představitele jedné kategorie. U každé kategorie je v závorkách uveden název konkrétního tvaru.

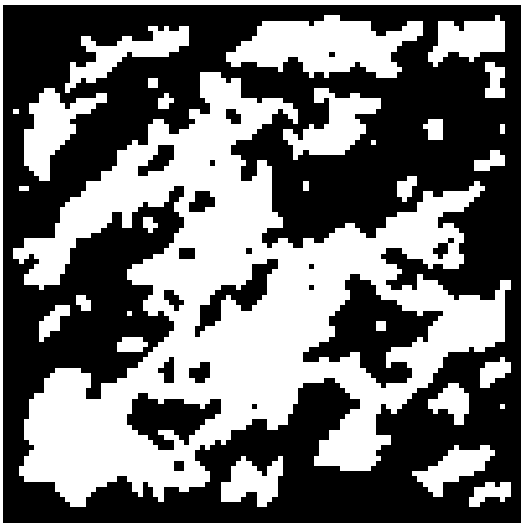
Celulární automaty jakožto generátory terénu

Buněčné automaty můžeme při vytváření terénu využít několika způsoby podle toho, zda generování probíhá jednou nebo opakovaně:

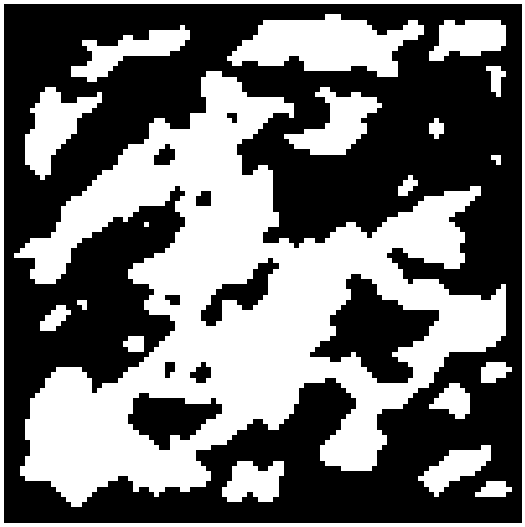
- **Dynamicky:** Tento způsob se týká především modelování chování terénu v čase. Dynamicky můžeme podle článku Automata model for soil erosion by water [3] modelovat například erozi půdy pod vlivem vody.
- **Staticky:** Způsob generování staticky byl vybrán pro tuto bakalářskou práce a jeho konkrétní implementace je popsána později v sekci 5.1.1. Pro lepší představu se jedná o generování terénu před zahájením samotné hry, tudíž bude potřebný pouze výsledek simulace, podle kterého bude terén generován. Bude tedy zvolena výchozí konfigurace modelu a následně proběhne několik iterací, kdy se bude model měnit pomocí stanovených pravidel. Příklad generování terénu pomocí této metody je ukázán na obrázcích 3.6.



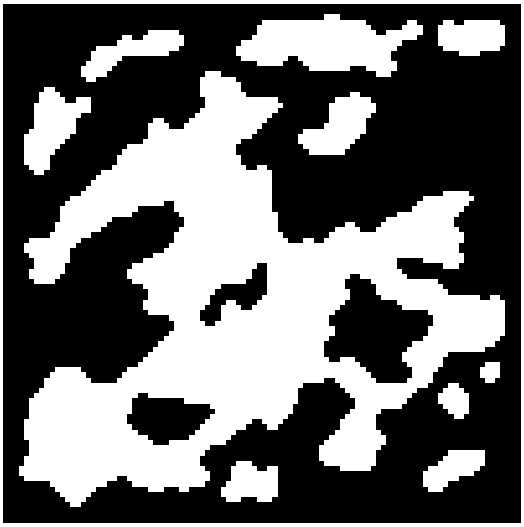
(a) 0. iterace



(b) 1. iterace



(c) 2. iterace



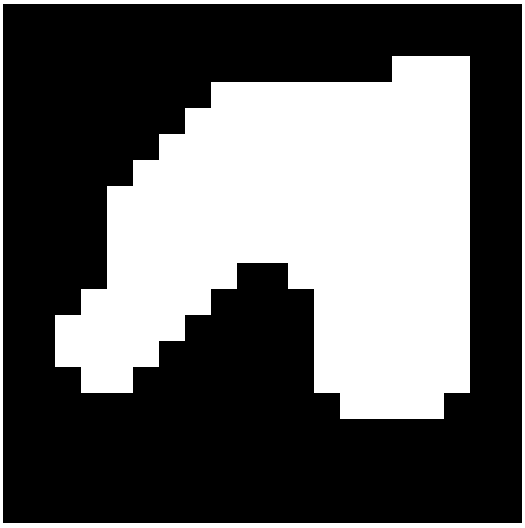
(d) 5. iterace

Obrázek 3.6: Obrázek představuje čtyři stádia vývoje celulárního automatu, kde podobraz (a) představuje náhodný počáteční stav, (b) první iteraci s formujícími se shluky a podobrazy (c) a (d) iterace s většími vzory.

3.6.2 Marching squares

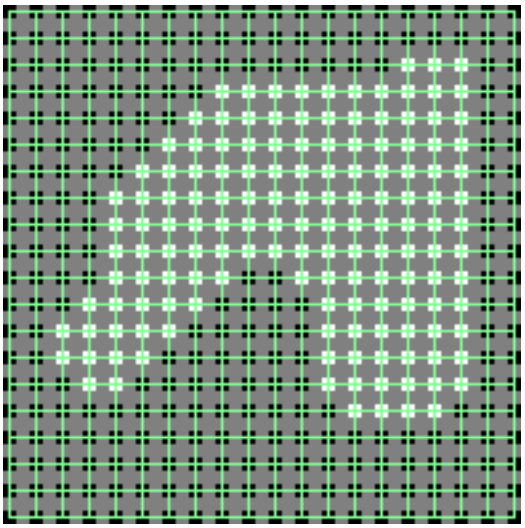
Tento algoritmus nic sám o sobě negeneruje a je zcela deterministický, ale může nám pomoci při vizualizaci dat, která jsme vygenerovali pomocí jiné metody. Marching squares slouží k transformaci dvourozměrného pole hodnot na aproximace hranic mezi danými hodnotami pole. Postup může vypadat následovně:

- Nejprve je potřeba získat dvourozměrné pole (příklad takového pole je na obrázku 3.7), kde každý prvek nabývá jedné ze dvou hodnot. Tohoto lze docílit za pomoci výše zmíněných celulárních automatů, Perlinova šumu nebo dalších algoritmů pro procedurální generování.



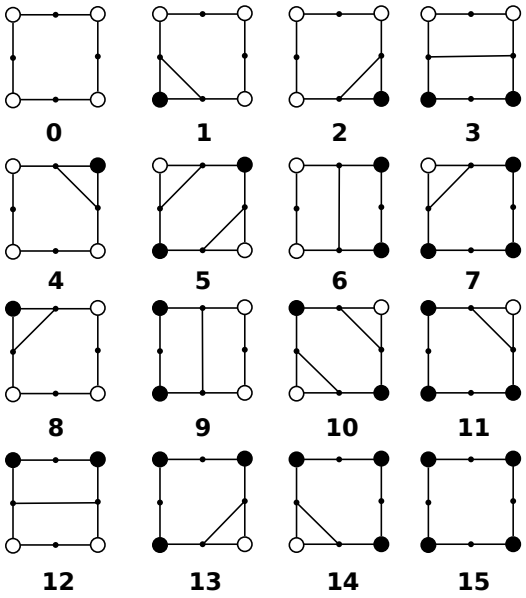
Obrázek 3.7: Obrázek obsahuje ukázku vstupu, který byl vygenerován pomocí celulárního automatu.

- Dalším krokem algoritmu marching squares je rozdělení vstupního pole na čtverce (příklad tohoto rozdělení je ukázán na obrázku 3.8), kde každá hodnota pole představuje jeden vrchol.



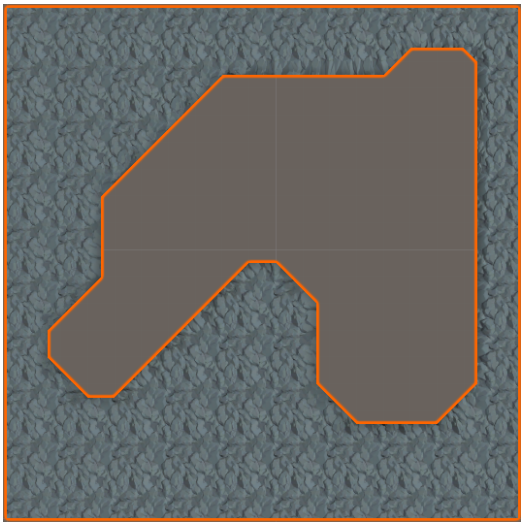
Obrázek 3.8: Ukázka rozdělení vstupního pole z celulárního automatu, kde zelené čáry představují ohraničení daného čtverce. Pro účely prezentace byly buňky vstupního pole zmenšeny.

- Vzniklé čtverce lze poté zařadit do jedné z 16 konfigurací (tyto konfigurace jsou na obrázku 3.9). Každá konfigurace má předem definováno, jak vypadá aproximace hrany, která má být vykreslena.



Obrázek 3.9: Obrázek obsahuje vizuální výčet 16 konfigurací čtverců, které mohou vzniknout při tvorbě hranic v algoritmu marching squares. Jednotlivé vrcholy reprezentují hodnoty 0 nebo 1 ze vstupního pole a vnitřní čáry ukazují, jak by mělo vypadat výsledné ohraničení.

- Následně podle zvoleného algoritmu lze vykreslit vzniklé hrany nebo i výplně čtverců (ukázka tohoto vykreslení je k dispozici na obrázku 3.10). Samotné vykreslení hran není předmětem tohoto algoritmu, ale bude popsáno níže v sekci 3.6.4 – Generování meshe.



Obrázek 3.10: Obrázek představuje vizualizaci algoritmu marching squares na využitém příkladu. Lze zde pozorovat nové hrany, které vznikly aplikováním konfigurací na jednotlivé čtverce.

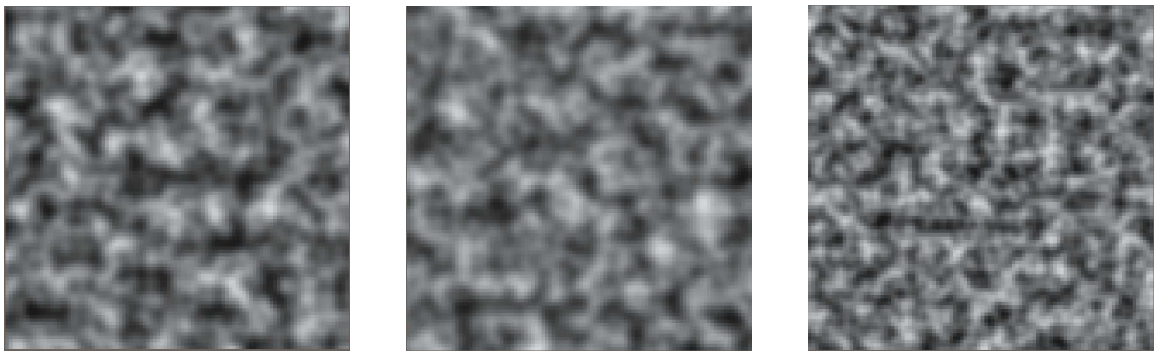
3.6.3 Perlinův šum

Perlinův šum, nazvaný po jeho tvůrci Kenovi Perlinovi, je typ gradientového šumu často používaný v počítačové grafice. Tato technika je využívána především pro generaci přirozeně vypadajících textur a modelů, které mohou ve výsledné aplikaci představovat terén, mraky a další efekty. Tento způsob je založen na matematickém algoritmu, který produkuje pseudonáhodné vzory s konzistentní strukturou.

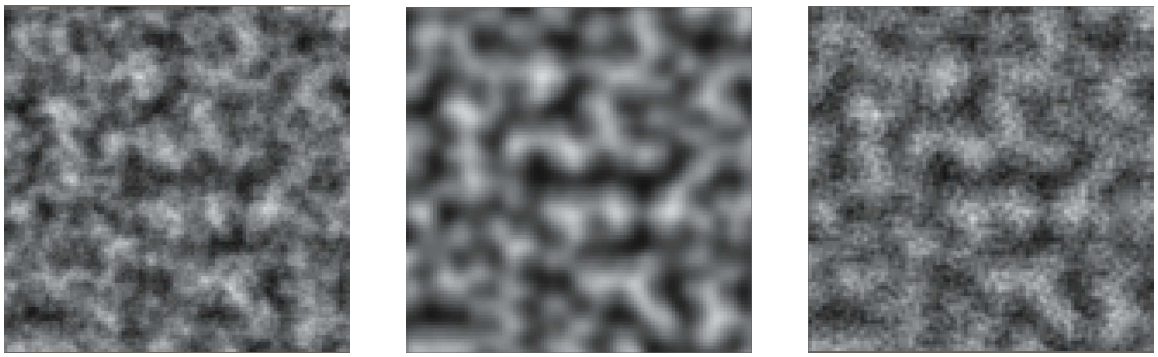
Mezi základní vlastnosti Perlinova šumu patří **kontinuita** a **hladkost**, které zaručují přirozený realistický vzhled a jemné přechody mezi hodnotami. Perlinova šumu lze využít nejen pro generování textur a modelů v herním průmyslu, ale také pro vizuální efekty ve filmech nebo pro vědecké účely. Perlinův šum není stěžejní součástí této práce a je tudíž popsán jen ve stručnosti a spíše z pohledu jeho použití. Mezi základní vstupní parametry pro tento algoritmus patří:

- **Škála:** Tento parametr určuje hustotu bodů mřížky, ve které se šum generuje.
- **Počet oktáv:** Oktávy definují počet vrstev, které jsou překládány přes sebe. Každá další vrstva je generována s vyšší frekvencí a nižší amplitudou.
- **Persistence:** Určuje rychlost snižování amplitudy v další oktávě.
- **Lakunarita:** Ovlivňuje míru změny frekvence pro následující oktávu.
- **Seed:** Hodnota využita pro generování pseudonáhodných čísel, kterých je využito při inicializaci generátoru. Stejný seed zaručí stejný výsledný šum.

Změnou výše uvedených vstupní parametrů vznikají rozdílné vzory s jinými vlastnostmi (tento fenomén je možno pozorovat na příkladu níže – obrázek 3.11). Dalšími důležitými pojmy, které byly již zmíněny výše, jsou **amplituda** a **frekvence**, které jsou ovlivněny persistencí a lakunaritou. Tyto hodnoty se mění postupně v každé následující oktávě a zajišťují tak detailnější vzhled generované mapy.



(a) výchozí Perlinův šum (b) změna seedu (c) změna škály



(d) změna počtu oktáv (e) změna persistence (f) změna lakunarity

Obrázek 3.11: Na obrázcích výše lze pozorovat vliv jednotlivých parametrů na výslednou mapu Perlinova šumu. Změnou těchto parametrů lze docílit požadovaného výstupu a změnou seedu zajistíme jeho jedinečnost.

3.6.4 Generování meshe

V poslední fázi procedurálního generování je potřebné vytvořit objekt, který bude vizualizovat vygenerovaný výsledek. Této vizualizace lze docílit vytvořením textury pro 2D hry nebo meshe v případě vývoje hry ve 3D. Dále je v rámci této bakalářské práce uvažováno pouze 3D varianty.

Mesh jako celek je struktura, která reprezentuje objekt ve fyzickém prostoru. Každý mesh se v počítačové grafice skládá z několika trojúhelníků, které tvoří trojrozměrný model. V případě velkého počtu trojúhelníků lze reprezentovat i složité tvary. Základní struktury pro generování meshe jsou:

- **Vrcholy:** Vrcholy jsou body ve 3D prostoru, které definují ohraničení meshe.
- **Trojúhelníky:** Pomocí trojúhelníků lze definovat plochy, které budou daný mesh reprezentovat. Trojúhelník je definovaný vždy pomocí 3 bodů (vrcholů) v prostoru. V Unity enginu jsou trojúhelníky reprezentovány polem čísel, kde každá tři čísla odkazují na indexy vrcholů, ze kterých se daný trojúhelník skládá.
- **Normály:** Pro určení, jak se bude světlo odrážet od povrchu modelu, se využívají normály. Normály jsou definovány pro každý bod meshe a jsou klíčové pro realistické odrážení světla.

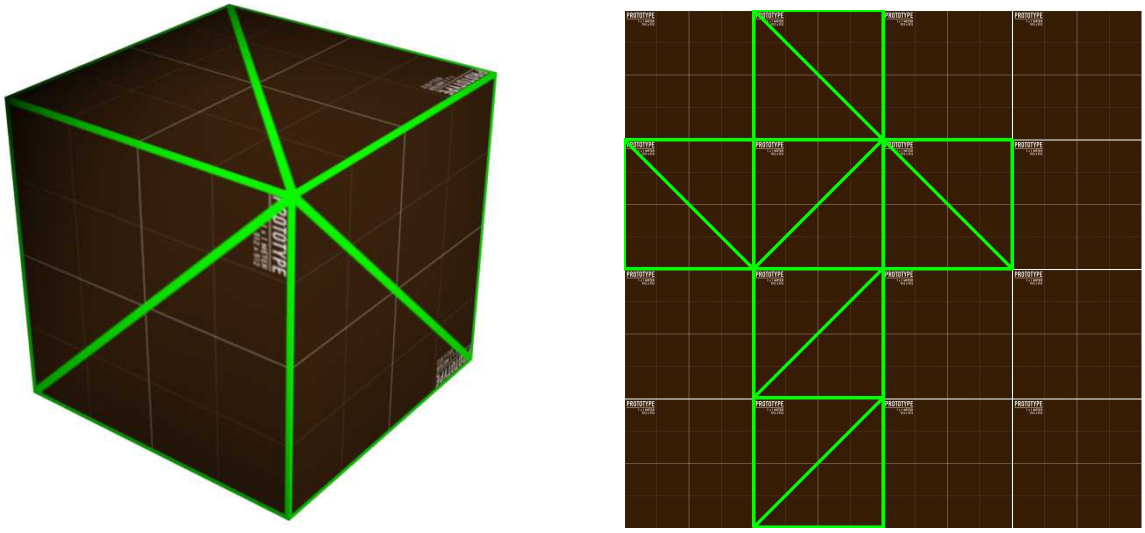
- **UV mapa:** Každý vrchol má dále přiřazenou UV koordinátu, která odpovídá bodu na textuře.
- **Barvy:** Zbarvení meshe je řešeno pomocí textury, která definuje barvy. Tato textura je „nanášena“ na model pomocí UV mapy.

Tvorba UV mapy

UV mapování je způsob, jakým se textura aplikuje na model a je klíčovou součástí procesu texturování v 3D grafice. Mezi základní koncepty patří **UV koordináty** a proces **rozvinutí**.

UV koordináty jsou velmi podobné těm tradičním ve 3D grafice (X, Y a Z) s rozdílem, že jsou definovány na 2D textuře. Rozvinutí je proces, kdy se 3D objekt „rozbalí“ a „existuje“ již pouze ve 2D prostoru. Proces tvorby UV mapy může vypadat následovně:

- **Výběr plochy:** Nejprve je nutno vybrat plochy nebo strany modelu, které budou mapovány.
- **Rozvinutí ploch:** Vybrané plochy jsou poté rozvinuty do 2D prostoru jak je ukázáno na obrázku 3.12. Tento proces bývá při složitějších modelech náročný.
- **Úprava švů:** Švy jsou místa, kde bude plocha přerušena (model při rozvinutí bude v těchto hranách „roztržen“). Právě švy mohou být důvodem pro definici více vrcholů, než by bylo jinak nutno.
- **Přizpůsobení UV:** Dále lze UV koordináty upravit, aby textura lépe seděla na model. Za využití UV mapování můžeme texturu na modelu škálovat, rotovat nebo i posouvat.
- **Další kroky:** Dalšími kroky může být i optimalizace v podobě zamezení viditelnosti švů, maximalizování využití prostoru textury a další.



(a) Složená výsledná krychle s vyznačenými trojúhelníky

(b) Rozvinutá krychle přiložená k textuře

Obrázek 3.12: Obrázky představují rozvinutí krychle a lze na nich pozorovat jednotlivé mapování vrcholů na texturu včetně vzniklých švů v místech, kde byly vytvořeny přídavné vrcholy (reálná krychle má přesně 8 vrcholů, ale mesh z příkladu má 14).

3.7 Perspektiva ve videoherním průmyslu

Vzhledem k nutnosti reprezentovat 3D svět pomocí 2D monitoru je zapotřebí využít perspektivy, která určuje jak budou trojrozměrné herní objekty vypadat. Perspektiva tedy určuje, z jakého úhlu a s jakou hloubkou je herní prostředí reprezentováno. Výběr správné perspektivy určuje i způsob interakce hráče s virtuálním světem. Před představením různých typů perspektiv je ještě nutné popsat pojem projekce a její kategorie.

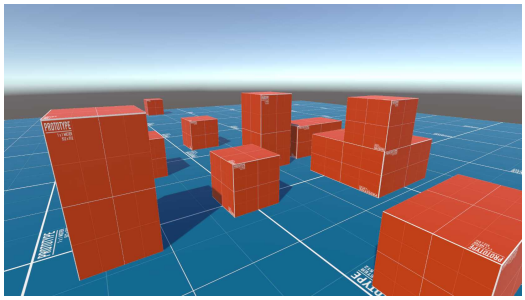
3.7.1 Projekce

V oblasti videoher existuje několik typů projekcí, které určují, jak jsou trojrozměrné objekty reprezentovány na dvourozměrném povrchu. Vybraná projekce tedy určuje metodu transformace 3D objektu do 2D. Na rozdíl od toho perspektiva definuje i úhel pohledu a umístění objektů do scény. Při vývoji her lze poté vybrat libovolnou kombinaci těchto metod[3]. Mezi základní typy projekce patří (tyto typy projekcí jsou pak i na obrázcích 3.13):

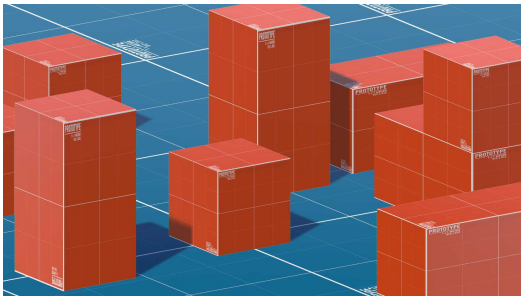
- **Perspektivní:** Nejčastější projekce, která vytváří iluzi hloubky a vzdálenosti. Paralelní linie se sbíhají a vytváří tak realistický trojrozměrný efekt.
- **Ortografická:** V této projekci rovnoběžné linie zůstávají paralelními a nevytváří tak hloubkový efekt. Objekty zůstávají ve stejné velikosti bez ohledu na vzdálenost od pozorovatele. Tento typ se běžně využívá při tvorbě 2D her nebo her využívající pohled z profilu.

[3]Existují nekompatabilní kombinace, které se v praxi nevyužívají.

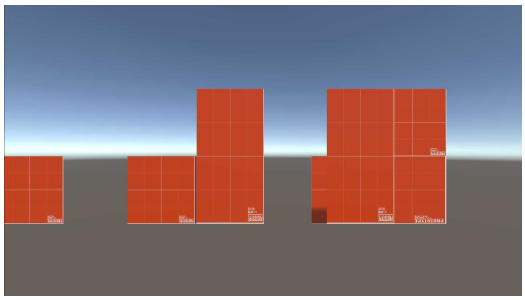
- **Izometrická:** Jedná se o typ ortografické projekce, kdy pozorujeme scénu pod úhlem 30 stupňů z obou stran. Tato projekce tedy vytváří 3D efekt, ale bez zkreslení perspektivní projekce.



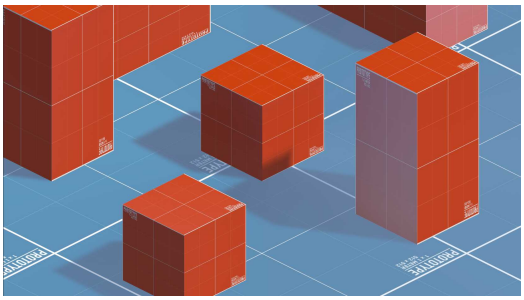
(a) Perspektivní projekce



(b) Ortografická projekce



(c) Ortografická projekce z profilu



(d) Izometrická projekce

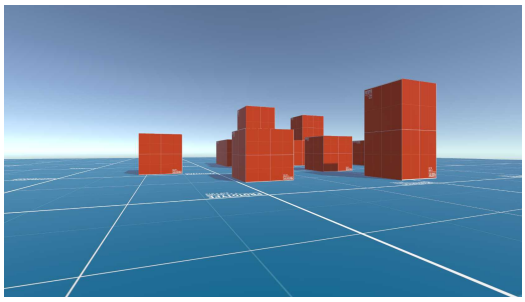
Obrázek 3.13: Obrázky výše ukazují rozdíly mezi jednotlivými jmenovanými projekcemi. Perspektivní i ortografická projekce byly zachyceny pomocí kamery na stejném místě. Pro dosažení efektu izometrické projekce byla kamera přesunuta na jiné místo. Podobraz (c) vystihuje hlavní rys ortografické projekce, neměnné velikosti objektů vzhledem k jejich vzdálenosti. Tento rys může být žádoucí právě pro 2D hry s pohledem z profilu.

3.7.2 Perspektiva

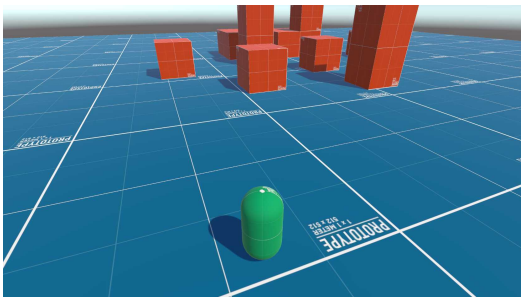
Perspektiva ve videohrách není jen o vizuálním zobrazení, ale také o interaktivitě. Rozhodnutí o perspektivě určuje, jak se hráč pohybuje a jak interaguje s objekty a postavami. Vzhledem k těmto faktorům je perspektiva klíčovým nástrojem pro tvůrce her. Mezi základní typy perspektiv, které jsou ukázány na obrázku 3.14, patří například:

- **Pohled z první osoby:** Kamera je umístěna přímo v oblasti očí ovládané postavy. Tento typ perspektivy je často používán u střílečeк nebo hororových her a umožňuje maximální ponoření do hry.
- **Pohled z třetí osoby:** Kamera se obvykle nachází za postavou a „sleduje“ hráče shora. Tento způsob umožňuje širší pohled na okolí a je využíván často ve hrách na hrdiny nebo v akčních adventurách.
- **Izometrická perspektiva:** Jedná se o typ perspektivy, který využívá často izometrické projekce a umožňuje tak hráči vidět velkou část scény. Této perspektivy je často využito ve strategických hrách.

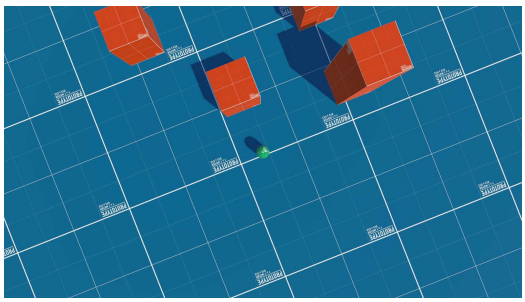
- **Pohled shora:** Kamera je umístěna přímo nad hráčem nebo v malém úhlu a stejně jako izometrická perspektiva umožňuje viditelnost a přehlednost scény.
- **Pohled z profilu:** Pohled z profilu je využit zejména u skákaček, které jsou často ve 2D. Tento způsob perspektivy často používá ortografickou projekci.



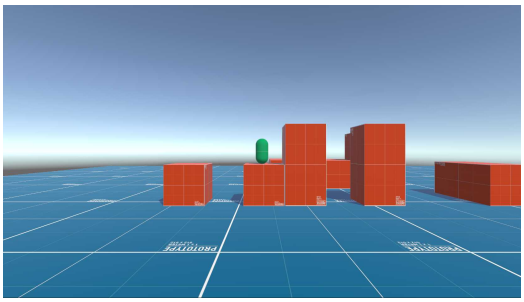
(a) Pohled z první osoby



(b) Pohled z třetí osoby



(c) Pohled shora



(d) Pohled z profilu

Obrázek 3.14: Na obrázcích lze pozorovat rozdíly mezi jednotlivými pohledy (perspektivami). Zelená kapsle na podobrázcích (b), (c) a (d) nahrazuje hráčovou postavu. Pro účely demonstrace byla projekce kamery ponechána na perspektivní a nebyl zahrnut izometrický pohled z důvodu jeho totožnosti s izometrickou projekcí. Po shlédnutí těchto obrázků a obrázků 3.13 lze pozorovat, že perspektiva je závislá na poloze a úhlu kamery a je nutné do herní scény zakomponovat postavu, kdežto projekce je čistě geometrická metoda transformace mezi 2D a 3D prostorem.

3.7.3 Perspektiva jako herní mechanika

Valná většina herních titulů má předem zvolenou perspektivu, která je poté neměnná. Některá herní studia ale využívají změny perspektivy jakožto herní mechaniky a tím hru obohacují o zajímavý prvek. Níže jsou uvedeny některé typy této změny společně s tituly, které tuto mechaniku obsahují:

- **Přechod mezi první a třetí osobou:** Nejčastějším způsobem změny perspektivy je změna pohledu z první osoby na třetí osobu. Tato metoda často není stěžejní součástí hry a slouží pouze jako preference hráče. Jako příklad nejznámější hry využívající tohoto prvky lze uvést Grand Theft Auto V.
- **Změna velikosti objektů pomocí perspektivy:** Při projekci 3D objektů do 2D se mění jejich velikost v závislosti na vzdálenosti od kamery. Této vlastnosti lze využít

při vývoji her a tak přidat novou mechaniku, která hráči dovolí změnu reálné velikosti objektu v závislosti na jeho velikosti na obrazovce. Například Superliminal je hra, která této změny velikosti využívá k vyřešení různých hádanek.

- **Změna polohy zdroje světla:** Tento způsob je využit hrou Echochrome, kde je daná úroveň tvořena stínem objektu v popředí. Hráči je umožněno přesouvat zdroj světla a tím měnit stín, respektive tvar celé úrovně.
- **Změna pohledu shora na pohled z profilu:** Tento způsob byl již představen i s příkladem hry v sekci 2.3 – Hry se změnou perspektivy.

3.7.4 Změna perspektivy a problémy implementace

Změnou perspektivy je v této sekci myšlena změna pohledu shora na pohled z profilu. K docílení lze využít dvou hlavních metod a to pomocí změny gravitace a zachování pozic objektů ve scéně nebo „překlopením“ celé scény a zachováním gravitace. Porovnání těchto způsobů je obsaženo v tabulce 3.2.

Problém	**Metoda**
Transformace kamery	(2)
„Překlopení“ scény	(2)
Změna gravitace	(1)
Změna pozice objektů	(2)
Změna rotace objektů	(1)/(2) podle preference
Orientace v prostoru	(1)

Tabulka 3.2: Tabulka výše slouží pro porovnání dvou metod pro změnu pohledu shora na pohled z profilu. Metoda (1) představuje metodu změny gravitace a metoda (2) představuje metodu „překlopení“ celé scény. Druhý sloupec identifikuje metodu, která problém z prvního sloupce musí řešit.

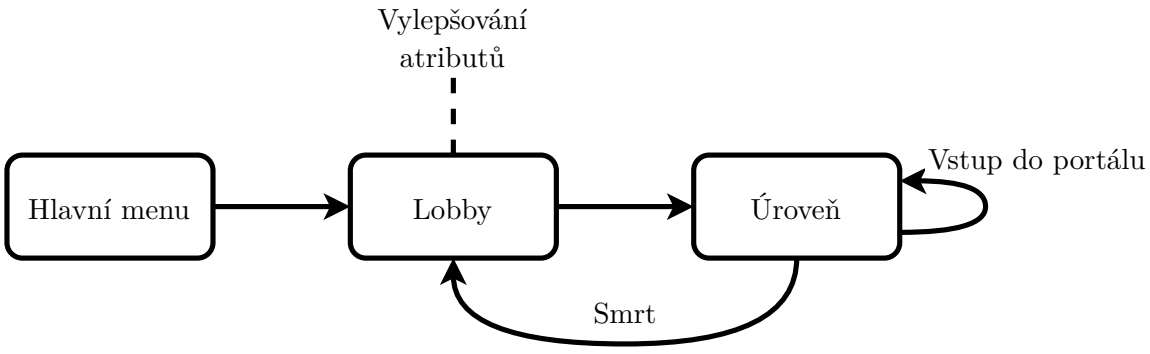
Kapitola 4

Návrh řešení

Tato kapitola je věnována obecnému popisu hry, která je předmětem této bakalářské práce a popisu jejich jednotlivých částí a herních mechanik. Nejprve je v krátkosti popsán herní běh obecně z pohledu hráče a je nastíněno, jak vypadají jednotlivé scény, které hra nabízí. Níže je celá hra postupně rozdělena na moduly a podmoduly, které jsou popisovány.

Hlavní herní běh videohry MageFlip, který je ukázán na obrázku 4.1 níže, obsahuje celkem tři samostatné scény, z nichž každá plní specifickou roli při interakci s hráčem. Mezi tyto scény patří následující:

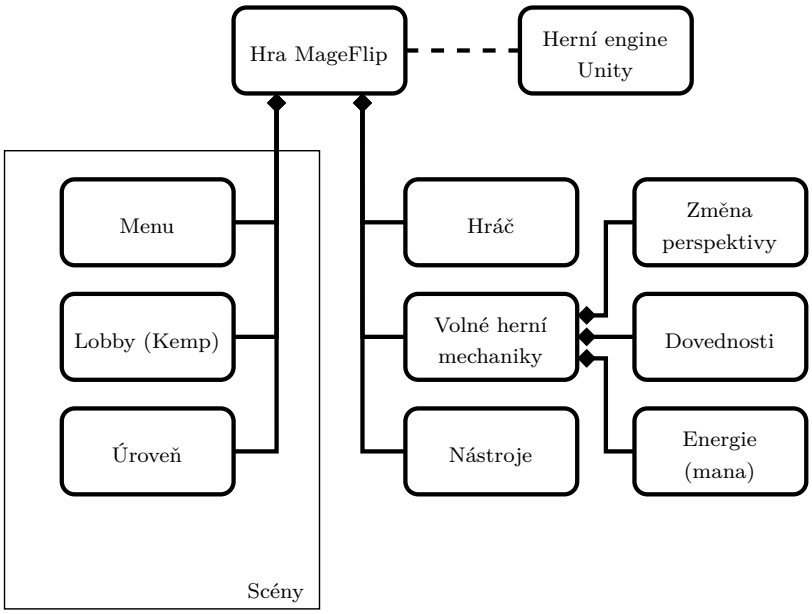
- **Menu:** Jedná se o první aktivní scénu po spuštění hry a tedy slouží primárně k uvítání hráče. Kromě toho obsahuje i jednoduché rozhraní pro změnu mapování jednotlivých kláves na akce ve hře a pro nastavení uživatelských preferencí.
- **Lobby:** Scéna s názvem *lobby*, která je reprezentována pomocí středověkého kempu, slouží jako prostor mezi jednotlivými běhy a tedy se zde hráč ocitá vždy po smrti a před zahájením dalšího pokusu. Mimo této funkce slouží i ke koupi vylepšení jednotlivých atributů, která jsou perzistentní napříč jednotlivými běhy a spuštěními hry.
- **Úroveň:** Tato scéna představuje základní součást samotné hry a reprezentuje valnou většinu všech mechanik v ní obsažených. Scéna tedy reprezentuje prostor, ve kterém dochází k vygenerování celé úrovně včetně všech nepřátel a objektů, k samotnému běhu a k postupu do vyšších úrovní.



Obrázek 4.1: Diagram představuje „pohyb“ hráče mezi herními scénami.

Hru MageFlip lze, jak je vidět na obrázku 4.2, rozdělit na moduly, kdy některé z nich je možné identifikovat jako scény. Scény jsou „fyzické“ moduly, které obsahují objekty nebo

kolekce objektů. Na rozdíl od toho jiné moduly[1] nelze přiřadit k jedné instanci a tedy existují mimo tyto scény.

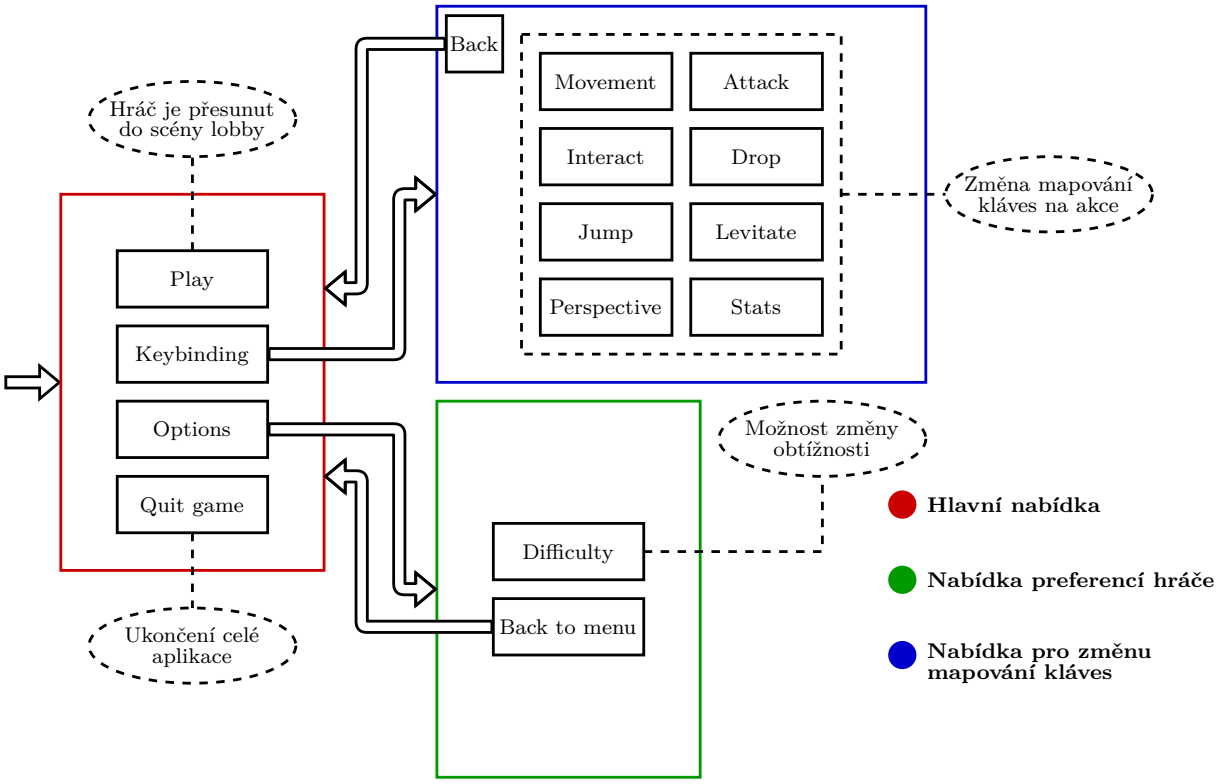


Obrázek 4.2: Tento diagram obsahuje nejhrubší rozdělení celé hry na moduly. Dále je zde rozlišeno mezi moduly představujícími scény a tedy i objekty v nich obsažené a moduly, které nejsou vázány na scény.

4.1 Menu

Scéna *menu* obsahuje pouze grafické uživatelské rozhraní a jedná se tedy o nejjednodušší scénu z výše zmíněných. Tato scéna může být vždy právě v jednom ze tří stavů podle toho, která z dostupných nabídek je právě aktivní (všechny dostupné nabídky a jejich společné interakce jsou ukázány níže na diagramu 4.3).

[1] Hráčem zde není myšlen pouze samotný herní objekt, ale i celková interakce uživatele s aplikací.



Obrázek 4.3: Diagram obsahuje všechny dostupné nabídky ve scéně *menu*. Na tomto diagramu lze vidět všechny akce, které hráč v jednotlivých oknech může vykonat včetně vzájemného přepínání mezi jednotlivými nabídkami.

4.2 Lobby

Hlavním účelem této scény je nahrazovat čekací prostor mezi jednotlivými pokusy v podobě nabídky a tím podpořit ponoření hráče do hry. Mimo této čistě pasivní funkce *lobby* slouží i jako místo k trvalému vylepšení některých atributů.

4.2.1 Vylepšování atributů

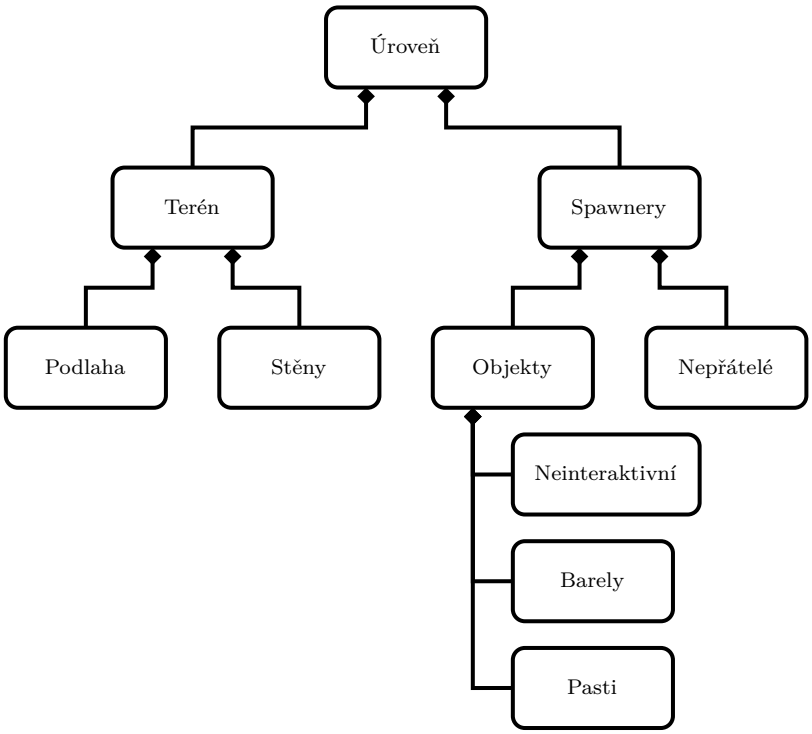
Metaprogrese v rámci hry MageFlip funguje na principu vylepšování jednotlivých atributů, což dále zjednodušuje následující průchody hrou. Vylepšit lze následující atributy:

- poškození;
- rychlost útoku;
- rychlost pohybu;
- síla skoku;
- síla levitace;
- maximální počet životů;
- maximální počet energie (mana).

K vylepšení jednotlivých atributů je potřebné vlastnit určitý počet mincí, které lze získat v rámci jednotlivých pokusů. Tyto mince jsou po zakoupení spotřebovány a dojde k plošnému navýšení cen všech atributů.[2] Pro toto vylepšení je využito nabídky, která se zobrazí po příchodu hráče k hlavnímu stanu.

4.3 Úroveň

Vzhledem ke komplexnímu charakteru této scény byl i tento modul rozdělen na podmoduly, kde jejich vzájemné vztahy lze pozorovat na diagramu 4.4 níže. Každá úroveň obsahuje terén, který je vytvářen pomocí procedurálního generování a je složen z podlahy, stěn a stropu. Zbylé objekty jsou tvořeny až později a za jejich instanciaci jsou „zodpovědné“ spawnery. Podrobný popis implementace procedurálního generování a spawnování je obsažen v sekci 5.1 a v sekci 5.3.



Obrázek 4.4: Diagram obsahuje rozdělení modulu *úroveň* na jednotlivé podmoduly. Lze zde pozorovat rozdělení na dvě hlavní části, kde část *terén* obsahuje objekty generované pomocí procedurálního generování a část *spawner* zaštiťuje objekty postupně instanciované do úrovně.

4.3.1 Terén

Terén je „kostrou“ celé úrovně, do které jsou umisťovány ostatní herní objekty, a tudíž je pozice nebo i existence těchto objektů závislá na jeho tvaru. Samotný terén je tvořen a následně instanciován hned po vyvolání požadavku na tvorbu nové úrovně, kde stěny jsou

[2] Všechny atributy nemusejí mít stejnou cenu, pouze dochází k násobení všech základních cen stejným multiplikátorem.

vytvářeny za pomoci celulárních automatů, Perlinova šumu a algoritmu marching squares (tyto algoritmu jsou popsány obecně v sekci 3.6 a jejich implementace v rámci této bakalářské práce v sekci 5.1).

4.3.2 Spawnery

Tento modul je zodpovědný za umístění všech potřebných objektů do scény, tedy musí pomocí algoritmu popsaného v sekci 5.3 – Spawnování vybrat objekt, následně pro něj vybrat náhodně pozici[3] a nakonec jej instanciovat. Objekty a nepřátelé disponují vlastními spawnery a jejich spawnování neprobíhá paralelně (nejprve jsou umístěny všechny objekty a až poté nepřátelé), ale v rámci této kapitoly spadají pod jeden modul z důvodu zachování jednoduchosti.

Objekty

Podle diagramu 4.4 lze rozdělit všechny objekty ve hře do tří skupin, kde objekty některých skupin jsou čistě pasivními aktéry kdežto jiné jsou stěžejními částmi hry.

Nejpočetnější skupinou jsou objekty **neinteraktivní**, které jsou čistě vizuálními prvky a ani nedisponují komponentou *collider* (komponenta *collider* je blíže popsána v sekci 3.5.3). Tyto objekty jsou rozmístěny po mapě pomocí příslušného spawneru a následně je každému z nich náhodně upravena rotace. Speciálními objekty jsou poté portály, které jsou do úrovně umístěny vždy po dvojici. Portály slouží pouze jako vizuál vstupního a výstupního místa pro pohyb mezi úrovněmi.

Objekty spadající do skupiny **barely** jsou již interaktivní a slouží hráči jako zdroj zbraní a mincí. Po vygenerování každé úrovně jsou barely rozmístěny po mapě stejným způsoben jako výše zmíněné neinteraktivní objekty. Barely se liší svojí interakcí s hráčem a okolním prostředím včetně gravitace při převrácení světa na pohled z profilu. Každý barel je instanciován jako nerozbitý a ke změně na stav *rozbitý* může dojít při překlopení světa v případě, že daný barel dopadl na zem v dostatečné předem definované rychlosti. Rozbitý barel zůstává rozbitý až do zničení instance a tedy z něj již nelze získat zbraň. Barel může mít podle jeho stavu definováno jedno z následujících chování:

- V případě, že je barel ve stavu *nerozbitý*, hráč je u něj dostatečně blízko a zároveň stiskne zvolenou klávesu pro interakci, tak bude zničena instance daného barelu a na jeho místě bude instanciován objekt zbraně, který bude následně možno si vybavit.
- Barel ve stavu *rozbitý* je možné zničit kolizí s hráčovou postavou. Každé zničení instance rozbitého barelu bude odměněno mincemi, které lze později uplatnit v obchodě ve scéně *lobby*.

Všechny výše zmíněné objekty disponují spíše pasivním charakterem a samostatně nic nedělají, avšak **pasti** plní spíše aktivní roli. Napříč všemi druhy pastí platí, že jejich primárním účelem je udělit poškození hráči a tak se mu pokusit zabránit v postupu do další úrovně. Mezi pasti patří:

- **Kanóny**: Instance kanónu jsou „připevněny“ na zdi s mírnou náhodnou rotací a střílí v určených intervalech (pro každý kanón je při jeho vytvoření tento interval definován a zůstává neměnným). V případě kolize mezi vystřeleným projektilem a hráčem hráč utrpí poškození.

[3] Pozice je vybírána pomocí předem stanoveného seedu a tedy není vybírána zcela náhodně.

- **Kyselinové generátory**: Tyto pasti se skládají z vícero objektů, kdy stěžejním je samotný generátor kyseliny a mezi další patří hromádka peněz a padací mříž. V případě pohledu shora generátor „střílí“ kyselinové kapky náhodně kolem sebe a pokud nějaká kapka dopadne na podlahu, vytvoří kaluž (v momentu kolize mezi kyselinovou kapkou nebo kaluží a hráčem dochází k udělení poškození). Pokud je hra v režimu pohledu z profilu, generátor „střílí“ kyselinové kapky přímo pod sebe. Pod samotný generátor je vždy umístěna i padací mříž, která po otočení perspektivy na pohled z profilu spadne a odkryje hromádku mincí, které bude následně možno posbírat kolizí s hráčovou postavou. Popis spawnování kyselinových generátorů je obsažen v sekci 5.3.1.

Nepřátelé

Tato skupina herních objektů představuje jednu z primárních součástí videohry MageFlip a slouží jako hlavní překážka mezi hráčem a postupem do vyšších úrovní. Každý nově spawnutý nepřátel je pasivní až do doby jeho aktivace, která může proběhnout přiblížením hráče do stanoveného poloměru[4] nebo v případě utržení poškození. V případě aktivace nepřítele dojde ke změně jeho chování a začne se přibližovat směrem ke hráči (aktivace je ireversibilní proces). Každý nepřítel se přibližuje maximálně na vzdálenost předem určeného útočného dosahu (angl. attack range), poté se nepřítel zastaví a započne jeho útok. V případě kolize mezi zbraní[5] nepřítele a hráčovou postavou hráč utrží poškození.

Všichni nepřátelé jsou párováni do dvojic reprezentovaných otcovským objektem, který je instanciován do úrovně pomocí spawneru (počet spawnovaných objektů je závislý na zvolené obtížnosti a popřípadě i na počtu nezabitých nepřátel v předchozí úrovni). Tento otcovský objekt je zodpovědný za přepínání mezi svými syny v závislosti na právě zvolené perspektivě. Každý syn má definované své chování pro pohyb a útok včetně vnitřních atributů pro rychlost, poškození a dalších. Mezi společné vlastnosti obou synů, a tedy vlastnosti otcovské, patří:

- aktuální počet životů;
- maximální počet životů;
- informace o aktivaci;
- informace o smrti.

Mezi speciální typ nepřítele patří **nepřátelé na dálku**, kteří disponují poupraveným chováním. Nepřátelé na dálku mají značně větší útočný dosah a místo kolize mezi zbraní a hráčem útok probíhá pomocí „vystřelení“ projektilu směrem ke hráči. Teto typ nepřátel při pohybu bere v potaz i stěny, které mu mohou překážet ve střelbě.

Nepřítelem, který, minimálně z pohledu implementace, stojí mimo ostatní, je tzv. **boss** (3.1). Boss disponuje vlastními mechanikami a je do scény umístěn vždy každou pátou úroveň (číslo úrovně je dělitelné pěti). V této úrovni jsou změněny následující mechaniky:

- úroveň je tvořena jednou velkou místností (jsou upraveny vstupní argumenty pro celulární automat, který vytváří úroveň);

[4] Každý druh nepřátele disponuje vlastním poloměrem aktivace (angl. aggro radius).
[5] Zbraní je, v některých případech, myšlena i část těla, která obsahuje *collider* pro útok.

- změna perspektivy není řízena hráčem (tato mechanika je využita při souboji s bossem);
- hráč nespotřebovává energii při levitaci.

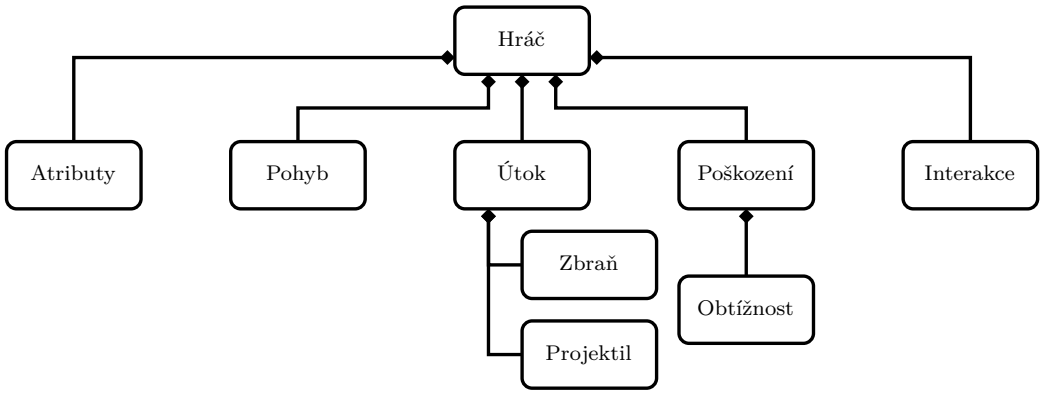
Souboj s bossem je fundamentálně odlišný od souboje s normálním nepřítelem. Tento souboj je rozdělen na několik částí (ve videoherním slangu se tyto části nazývají fáze), kde tento konkrétní boss disponuje následujícími třemi:

- **1. fáze:** Boss mění mezi třemi dostupnými útoky (útok na blízko, útok na dálku a plošný útok) a pohybuje se směrem ke hráči. Jakmile aktuální životy bosse klesnou pod poloviny, tak dojde k přechodu do 2. fáze. Alternativně se společně s bossem můžou spawnout i další nepřátelé, kdy je jejich počet závislý na zvolené obtížnosti.
- **2. fáze:** Na začátku 2. fáze dojde k deaktivaci bosse a naspawnování několika draků, kteří útočí z dálky a pronásledují hráče. Pro přepnutí do 3. fáze je potřeba všechny draky porazit.
- **3. fáze:** Tato fáze je identická jako první, jen s tím rozdílem že trvá až do smrti bosse, kdy se objeví výstupní portál a hráči bude umožněno projít do následující úrovně.

4.4 Hráč

Tato sekce se věnuje popisu hráče včetně jeho ovládání a přiblížení jednotlivých mechanik. Rozdělení tohoto modulu lze vidět na obrázku 4.5 níže. Hráčem je v této sekci myšlena veškerá interakce[6] uživatele se hrou a tedy je zde zahrnuto více než jen samotný herní objekt hráče.

Objekt hráče je „fyzický“ herní objekt v enginu Unity a je středem interakce s uživatelem. Tento objekt je na začátku každé nové úrovně přesunut na pozici vstupního portálu a následně je uživateli umožněno jej ovládat pomocí klávesnice a myši.



Obrázek 4.5: Diagram vizualizuje rozdělení modulu *hráč* na podmoduly, které jsou níže popsány do detailu.

[6] Je vynechána interakce přímo s grafickým uživatelským rozhraním.

4.4.1 Atributy

Atributy jsou ve hře MageFlip pouze číselné hodnoty, které ovlivňují některé interakce hráče s okolní scénou. Tyto hodnoty je možné měnit ve scéně *lobby* (atributy hráče) nebo i za běhu (atributy zbraně). Atributy hráče jsou perzistentní mezi jednotlivými běhy a to i po vypnutí celé aplikace, zatímco atributy zbraně se mění každou nově vybavenou zbraní nebo po započetí nového běhu. Výpis všech atributů hráče i zbraně je obsažen v tabulce 4.1.

Atribut		**Účel**	**Typ atributu**
		Atributy hráče	
Poškození	(damage)	míra poškozen	multiplikátor
Rychlost útoku	(attack speed)	doba mezi útoky	multiplikátor
Pohyb	(movement)	rychlost pohybu	hodnota
Skok	(jump)	síla skoku	hodnota
Levitace	(levitation)	síla levitace	hodnota
Životy	(health)	maximální počet životů	hodnota
Energie	(mana)	maximální počet energie	hodnota
		Atributy zbraně	
Poškození	(damage)	míra poškození	hodnota
Rychlost střelby	(fire rate)	doba mezi útoky	hodnota
Rychlost projektilu	(projectile speed)	rychlost projektilu	hodnota

Tabulka 4.1: Tabulka obsahuje všechny dostupné atributy hráče i atributy zbraně. V prvním sloupci je v závorkách přesné pojmenování atributu ve hře a mimo závorky jeho překlad do češtiny, druhý sloupec slouží jako vysvětlení atributu. Třetí sloupec poté rozděluje všechny atributy podle jejich typu. Atributy označené jako *multiplikátory* pouze násobí jinou hodnotu (jejich výchozí hodnota je 1) a atributy označeny jako *hodnoty* jsou těmito hodnotami (jejich výchozí hodnota je 100).

4.4.2 Pohyb

Pohyb hráčové postavy je závislý na zvolené perspektivě a tedy i v této sekci bude jeho popis podle toho rozdělen. V prvním paragrafu bude vysvětlen pohyb při zvoleném pohledu shora a ve druhém z profilu.

Při změně perspektivy na **pohled shora** je dostupný pouze pohyb po zemi a to pomocí čtyř zvolených kláves (výchozí použité klávesy jsou W/S/A/D). Pohyb je řešen pomocí zabudovaného fyzikálního systému dostupného v enginu Unity. Použití fyziky pro pohyb znamená plynulejší zrychlení i zpomalení postavy.

Pokud je aktivní **pohled z profilu**, pak je umožněn „normální“ pohyb pouze v horizontálních směrech. Vertikální pohyb je řešen skokem, který je dostupný jen po dopadu hráče na zem, a levitací, která je k dispozici i ve vzduchu, ale její použití snižuje aktuální množství energie.

4.4.3 Útok

Mechanika souboje ze strany hráče je neměnná v závislosti na aktivní perspektivě a funguje na bázi „kam uživatel klikne myší, tam vystřelí projektil“. Ve výchozím stavu není vystřelení

projektilu okamžité a „čeká“ se na animaci. Při vybavené schopnosti *okamžité vystřelení* (obecný popis schopností je v sekci 4.5.2) je projektil spawnut instantně. V obou případech je projektil instanciován v oblasti hrotu zbraně.

4.4.4 Poškození

Mezi další mechaniky souboje patří i poškození, které hráč obdrží několika způsoby. Mezi možné zdroje poškození patří:

- zbraň nepřítele (nebo část jeho těla);
- boss (útok z blízka/plošný útok/útok na dálku);
- projektil (od nepřítele/od kanónu);
- kyselina (kapka/kaluž).

Všechny výše zmíněné zdroje disponují vlastní hodnotou poškození, která po jejich interakci s hráčem bude odečtena od aktuálních životů. Dále je malá pravděpodobnost, že dané poškození bude zdvojnásobeno a tedy vyústí v kritické poškození. V případě, že aktuální životy klesnou na nulu, hráč zemře a uživateli bude zobrazena obrazovka konce hry (angl. game over screen).

4.4.5 Interakce

Nepočítaje soubojové interakce mezi hráčem, nepřáteli a pastmi, je jedinou interakcí ve hře MageFlip sbírání předmětů z barelů a interakce s obchodem ve scéně *lobby*. Obchod s vylepšením atributů lze aktivovat pomocí příchodu do předem definované zóny a jeho deaktivaci lze provést odejitím z této zóny nebo použitím klávesy Escape.

S barelem lze interagovat přiblížením se k němu a následným stisknutím uživatelem zvolené klávesy (výchozí klávesa pro tuto akci je E). Po provedení interakce je objekt barelu zničen a proběhne následná instanciace náhodné zbraně na jeho pozici.

4.5 Volné herní mechaniky

Hra MageFlip obsahuje i takové herní mechaniky, které nejsou závislé na jednotlivých herních objektech a tedy ani na daných scénách. Rozdělení těchto mechanik je vidět na obrázku 4.2, který rozlišuje tři hlavní mechaniky. Tyto mechaniky jsou popsány v následujících sekcích.

4.5.1 Změna perspektivy

Tato herní mechanika je hlavní součástí této práce a je zmíněna i v jejím zadání. Jedná se o změnu perspektivy z pohledu shora dolů na pohled z profilu, která na hráče působí stejným dojmem jako převrácení celého světa o 90 stupňů. Svět však zůstává nehybným a mění se pouze projekce (více o projekci a perspektivě v sekci 3.7) a chování ostatních herních objektů.

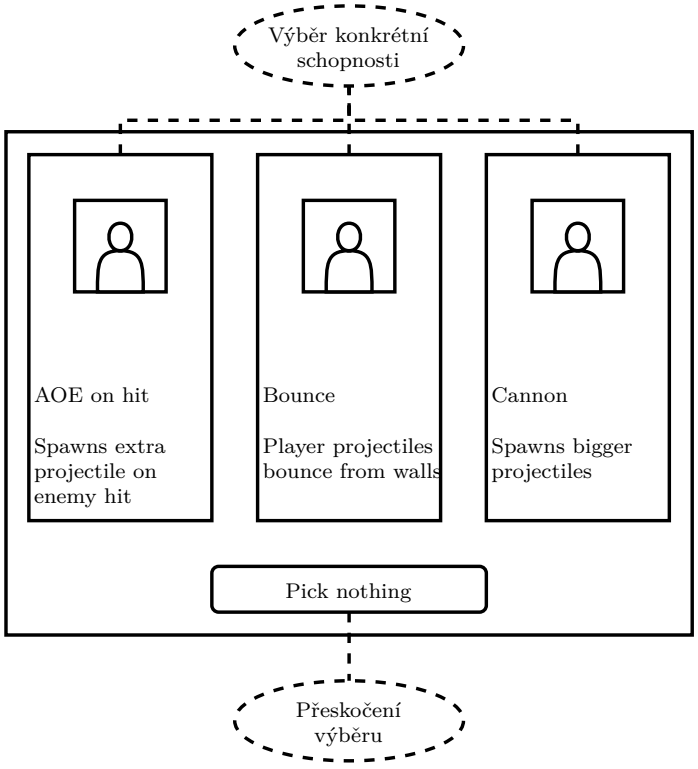
Změna perspektivy sama o sobě není nijak zajímavou mechanikou, ale dovoluje jiným objektům této skutečnosti využít a tím přinést něco nového. Mezi hlavní změny při změně perspektivy patří:

- změna pohybu hráče;
- zapnutí gravitace pro pohyblivé objekty;
- změna každého nepřítele včetně chování;
- možnost průchodu některými chodbami;
- odstranění kyselinových kaluží;
- možnost posbírání hromádek mincí pod mřížemi.

Mimo výše zmíněných je přepínání mezi pohledy využito i při souboji s bossem. Tato mechanika je dále využita při vyvažování hry z hlediska obtížnosti.

4.5.2 Schopnosti

Pro postup do další úrovně je zapotřebí projít výstupním portálem a při této události je hráči nabídnuto si vybrat jednu ze tří dostupných schopností, které bude moci využít při aktuálním běhu. Pro případ, že by si uživatel žádnou z dostupných schopností vybrat nechtěl, se v této nabídce – která je ukázána na obrázku 4.6 – vyskytuje i tlačítko pro přeskočení volby. Seznam všech možných schopností je obsažen v tabulce 4.2 níže.



Obrázek 4.6: Tento diagram přibližuje proces vybírání schopností, který probíhá vždy na konci každé úrovně.

Název schopnosti		Popis schopnosti
Plošný útok	(AOE on hit)	vytvoří čtyři projektily po zasažení nepřítele
Odraz	(Bounce)	projektily se odrážejí od stěn
Kanón	(Cannon)	větší projektily
Doplnění životů	(Health over time)	průběžné doplňování životů
Instantní výstřel	(Instashot)	instantní vystřelení projektilu
Nesmrtelnost	(Invincibility)	hráč je nesmrtelný po dobu levitace
Zpětný ráz	(Knockback)	projektily odrážejí nepřátele po směru výstřelu
Více projektilů	(Multishot)	jedna akce od uživatele vyústí ve více projektilů
Průraznost	(Piercing)	projektil není zničen po kolizi s nepřítelem
Oživení	(Reborn)	místo smrti se hráči doplní životy (jedno použití)
Štít	(Early shield)	nesmrtelnost na chvíli po započetí nové úrovně

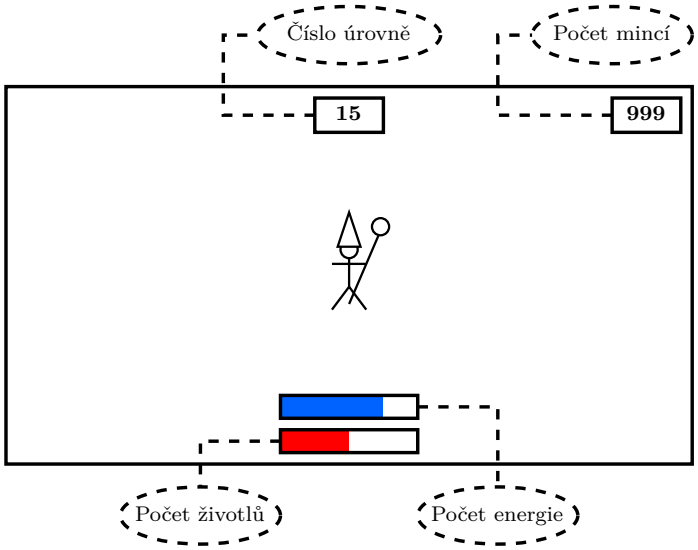
Tabulka 4.2: Tato tabulka obsahuje výčet všech dostupných schopností ve hře MageFlip včetně jejich podrobnějšího popisu v druhém sloupci. V závorkách je přesné pojmenování atributů.

4.5.3 Energie (mana)

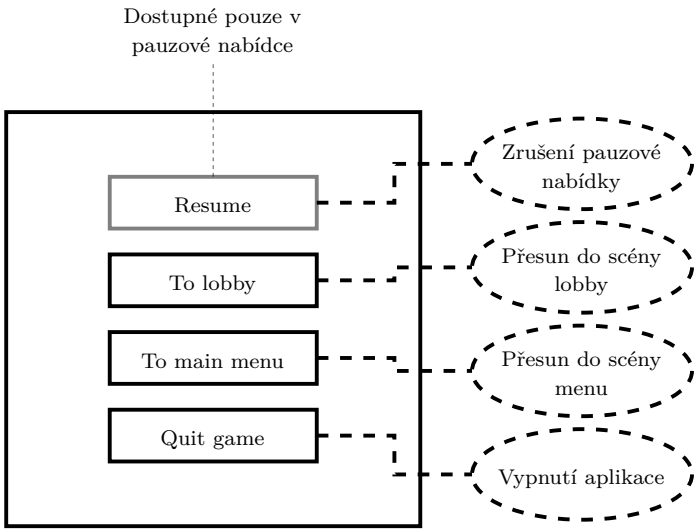
V situaci, kdy si uživatel nezvolí nejlehčí obtížnost, existují akce, které spotřebují určité množství energie. V současné době hra MageFlip obsahuje dvě takové akce – změna perspektivy a levitace. **Změna perspektivy** vždy spotřebuje 90 % maximálního množství energie, kdežto **levitace** má pevně daný odběr a tedy hráč profituje vylepšením atributu *maximální energie*. Spotřebovaná energie se pasivně dobíjí časem a rychlost tohoto dobíjení je závislá na několika faktorech (např. dobíjení probíhá jen při stání na zemi).

4.6 Návrh grafického uživatelského rozhraní

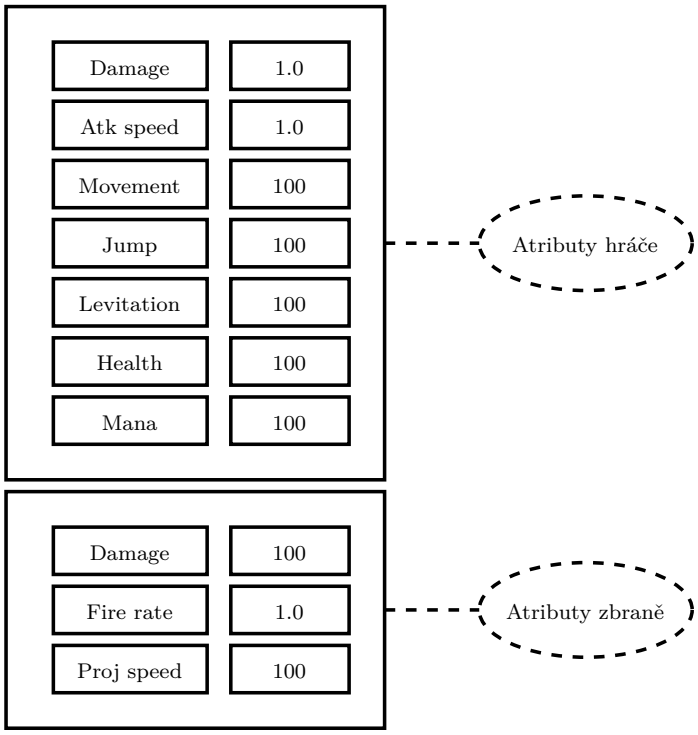
Tato sekce bude věnována návrhu grafického uživatelského rozhraní (dále jen GUI) v rámci samotného běhu a tedy pouze ve scéně *úroveň* (tuto část GUI lze také označit jako HUD). Stručný diagram GUI, které je viditelné v rámci běhu je níže na obrázku 4.7. Diagram 4.8 obsahuje vizualizaci pauzové nabídky a nabídky, které se zobrazí po smrti. Na posledním grafu (obrázek 4.9) jsou vidět nabídky vypisující jednotlivé aktuální atributy hráče.



Obrázek 4.7: Diagramu zobrazuje grafické rozhraní dostupné při celém průchodu úrovní.



Obrázek 4.8: Diagram obsahuje vizualizaci pauzové nabídky včetně možných akcí. Tlačítko *Resume* je zobrazeno pouze v případě aktivace pauzové nabídky. Pauzová nabídka i nabídka po smrti jsou až na toto tlačítko identické.



Obrázek 4.9: Na diagramu výše lze pozorovat grafický návrh panelu, který ukazuje jednotlivé atributy hráče a aktuálně vybavené zbraně.

Kapitola 5

Implementace

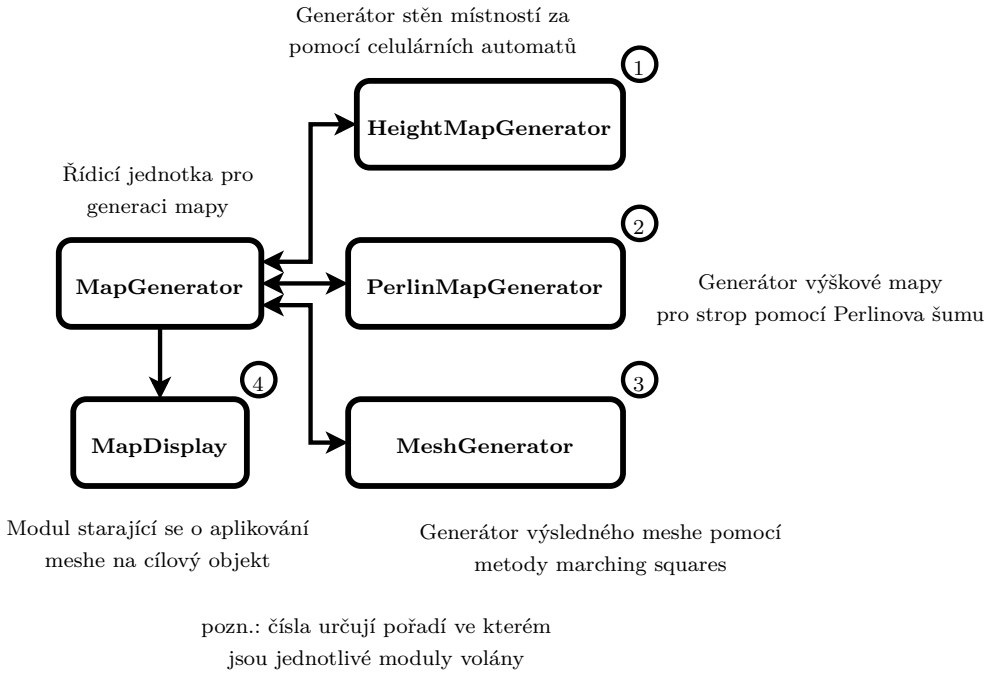
Tato kapitola obsahuje implementační detaily jednotlivých částí této bakalářské práce. Nejprve je popsán proces generování jednotlivých úrovní včetně celulárního automatu a algoritmu marching squares. V následující sekci je vysvětlena mechanika změny perspektivy a její realizace v různých částech hry. Zbytek této kapitoly je věnován popisu implementace spawnování objektů a nepřátel a představení dalších méně zajímavých herních částí jako například:

- pohyb a chování hráče a nepřátel;
- objekty řídící hru (objekty „neviditelné“ pro hráče);
- práce s daty.

5.1 Procedurální generování

Procedurální generování[1] je stěžejní částí a je mu věnována tato sekce. Je zde tedy popsán průběh generování a následného vytváření meshe ze získaných dat a přiblížení funkcionality jednotlivých modulů a algoritmů. Tento proces volání jednotlivých částí pro generování je ukázán na grafu 5.1 níže. Vzhledem ke komplexnosti byly z této sekce vypuštěny některé implementační detaily nebo i celé moduly (například generátor Perlinova šumu).

[1] Implementace níže popsaných algoritmů byla ze začátku vývoje inspirována videosérií od Sebastiana Lagua dostupnou z: https://www.youtube.com/watch?v=v7yyZZjF1z4&list=PLFt_AvWsXl0eZgMK_DT5_biRkWXftAOf9.

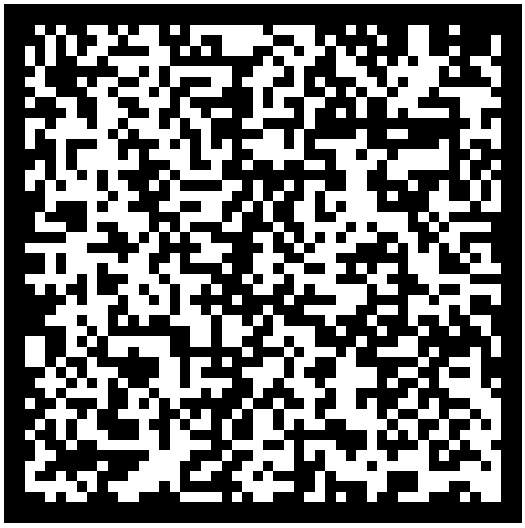


Obrázek 5.1: Graf popisuje proces volání jednotlivých modulů pro procedurální generování v rámci této bakalářské práce. Čísla u jednotlivých částí určují pořadí jejich volání.

5.1.1 Celulární automat

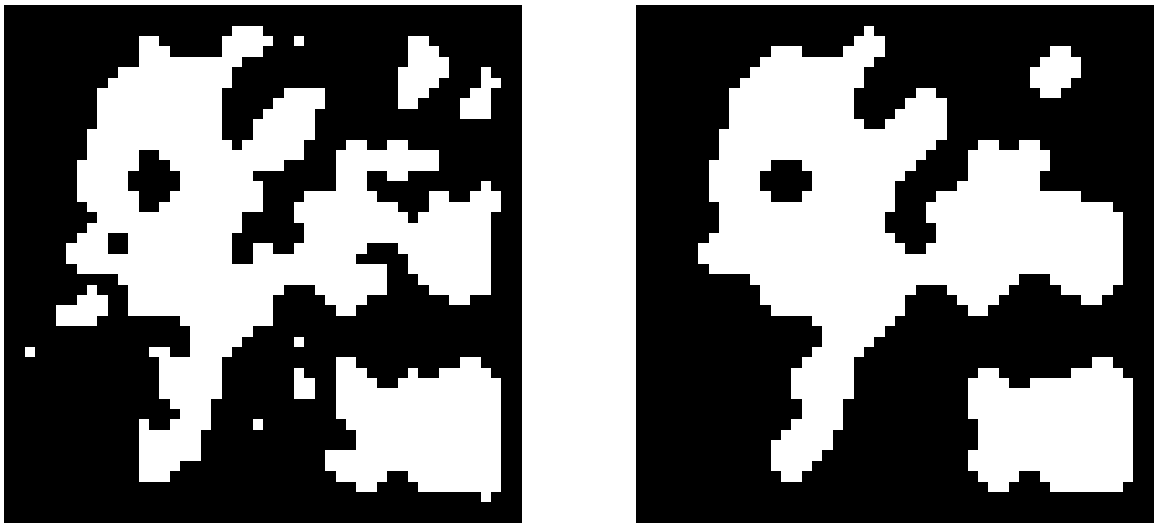
Implementace celulárního automatu se nachází v modulu `HeightMapGenerator` a probíhá ve dvou následujících krocích:

1. Vyplnění mapy hodnotami 0 nebo 1 v závislosti na seedu (ukázka tvorby mapy pomocí metody náhodného vyplnění je na obrázku 5.2);



Obrázek 5.2: Ukázka náhodně vyplněného pole hodnotami 0 nebo 1 v rámci celulárního automatu.

2. Iterativní zahlazování mapy přes její buňky jak je vidět na obrázku 5.3. Zahlazováním je myšleno projití celé mapy a postupné měnění jejich buněk v závislosti na okolí. V případě, že kolem dané buňky existují více než čtyři buňky totožné hodnoty, bude její hodnota změněna na jejich hodnotu. Pokud je v okolí buňky rovný počet obou hodnot (čtyři bílé a čtyři černé), tak se její stav nemění. Tento proces je variací morfologických operací nad 2D polem s použitím celulárních automatů. Detailnější popis morfologických operací je obsahem práce Training two dimensional cellular automata for some morphological operations [12].



(a) 1. iterace

(b) 5. iterace

Obrázek 5.3: Ukázka jednotlivých iterací při zahlazování mapy v rámci celulárního automatu. Jedná se o zahlazení mapy z obrázku 5.2.

5.1.2 Úprava výškové mapy

Mapa vytvořena pomocí celulárního automatu uvedeného v předchozí sekci však není dostačující pro tuto hru, protože obsahuje i místnosti, které nejsou dosažitelné. Implementace řešení tohoto a dalších problémů vypadá následovně:

1. **Získání všech regionů**[2] **pomocí průchodu mapy:** Prvním krokem pro nalezení následujícího regionu je identifikace libovolné dosud neprozkoumané buňky a pomocí postupného „rozšiřování“, které je ukázáno na obrázku 5.4, je daný region definován. Tento způsob identifikace regionů se nazývá semínkové vyplňování (angl. seed fill nebo flood fill) a je obsahem práce Scan flood Fill(SCAFF): an Efficient Automatic Precise Region Filling Algorithm for Complicated Regions [6], která popisuje i variace tohoto algoritmu.

[2]Region je oblast buněk mající stejnou hodnotu.

(a) Vyznačená libovolná neprozkoumaná buňka

(b) Proces identifikace v 1/3

(c) Proces identifikace ve 2/3

(d) Dokončený proces identifikace regionu

Obrázek 5.4: Obrázky výše vizualizují identifikaci regionu a postupné rozšiřování plochy. Na obrázku (a) lze vidět první buňka, obrázky (b) a (c) ukazují postupné rozšiřování zóny a obrázek (d) již představuje dokončený proces a lze na něm vidět celá místnost.

2. **Odstranění malých regionů:** V případě, že nějaký z regionů nesplňuje minimální velikostní požadavky, tak jsou všechny jeho hodnoty převráceny. Tento proces je zachycen na obrázcích 5.5.

(a) Před odstraněním malých regionů

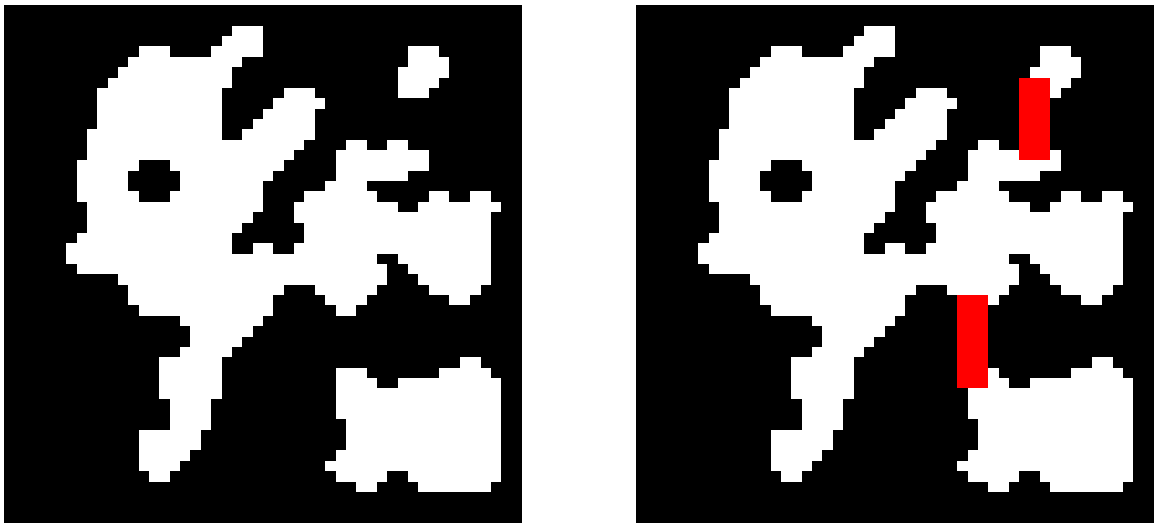
(b) Po odstranění malých regionů

Obrázek 5.5: Oba obrázky výše jsou vygenerovány s totožnými vstupními proměnnými. Lze pozorovat, že obrázek (a) obsahuje malé (a pro tuto hru nežádoucí) regiony, zatímco obrázek (b) již tyto regiony neobsahuje.

3. **Identifikace hlavní místnosti**[3]**:** Za hlavní místnost je označena ta s největší rozlohou. K hlavní místnosti jsou postupně připojovány ostatní pomocí koridorů.

4. **Postupné spojení všech místností:** Tohoto je docíleno nalezením nejbližší nespojené místnosti a jejím následným spojením s ostatními pomocí algoritmu pro vykreslování čar. Po vykreslení vznikají koridory, které jsou vidět na obrázku 5.6 červenou barvou. Tento krok je opakován až do doby, kdy neexistuje žádná nespojená místnost. V rámci tohoto algoritmu probíhá i nalezení dvou nejbližších buněk z obou spojovaných místností.

[3]Místnost je region ve kterém všechny buňky nabývají hodnoty 0.



(a) Před vytvořením koridorů

(b) Po vytvoření koridorů

Obrázek 5.6: Obrázky ukazují, jak byly jednotlivé místnosti v rámci generování výškové mapy pospojovány. Oblasti ovlivněné tímto spojováním jsou na obrázku (b) vyznačeny červenou barvou (toto vyznačení je zde pouze pro účely demonstrace).

5.1.3 Generování meshe

V tomto stádiu procedurálního generování je k dispozici výšková mapa vytvořena pomocí celulárního automatu a druhá mapa[4], která byla zhotovena za využití Perlinova šumu (nejedná se o unikátní implementaci, a tudíž proces tvorby sekundární mapy není popsán). Obě tyto mapy jsou zaobaleny společně s metadaty o rozměrech do struktury `MapData` (použití této struktury je ukázáno na obrázku 5.7).Popis výškových map všetně jejich jednoduchého použití se nachází ve knize Focus on 3D Terrain Programming [10]. Tato struktura je jediným vstupem pro modul `MeshGenerator`. Výstupem tohoto modulu je mesh, tedy pole trojrozměrných bodů (vrcholů) a pole reprezentující trojúhelníky.

Vytvoření dvourozměrného pole čtverců

Prvním krokem v rámci algoritmu marching squares je transformace vstupní výškové mapy na dvourozměrné pole čtverců, které budou každý obsahovat osm uzlů (čtyři řídící a čtyři vedlejší). Proces vytváření takového pole vypadá následovně:

1. Pro každou buňku vstupní výškové mapy je spočtena reálná pozice ve scéně. Pozice na osách x a z je závislá pouze na indexu dané buňky v mapě a rozměrech mapy. Pro výpočet pozice na ose y je využito mapy tvořené pomocí Perlinova šumu.

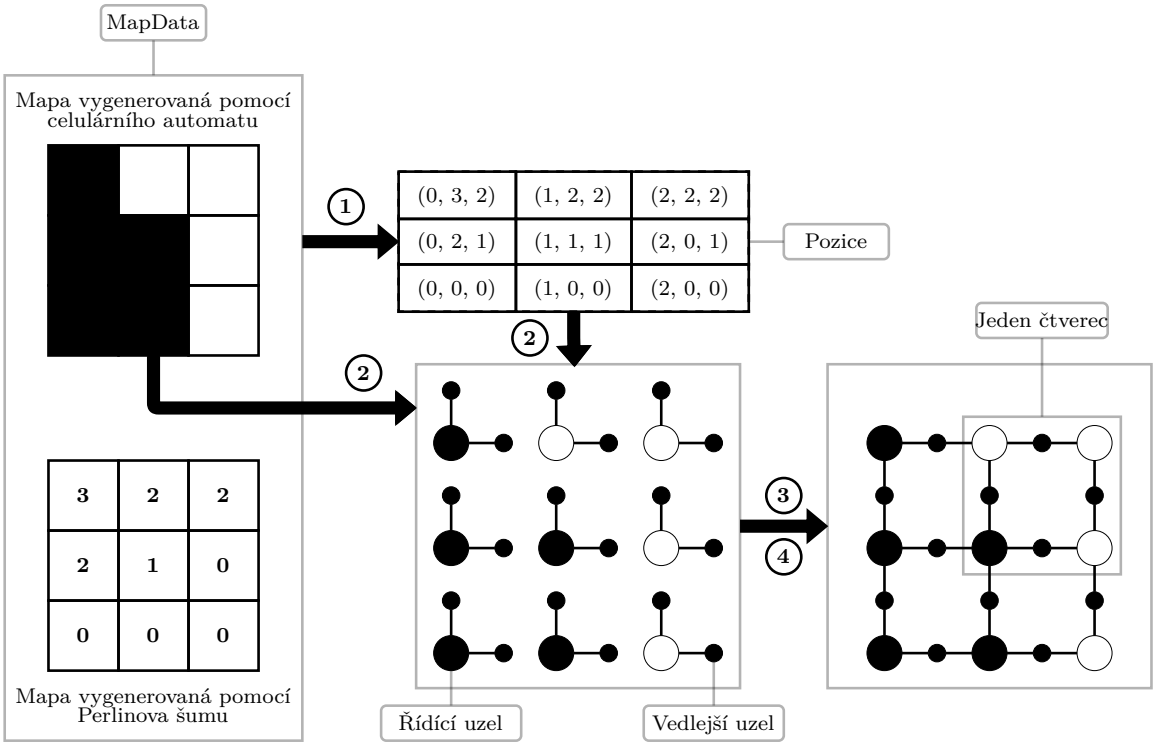
2. Pro každou spočtenou pozici je vytvořen řídící uzel, který mimo reálné pozice ve světě obsahuje i informaci o jeho hodnotě, kterou „zdědil“ z buňky obsažené ve výškové mapě. Společně s vytvořením řídícího uzlu jsou vytvořeny dva vedlejší uzly (horní a pravý). Vedlejší uzly mají pouze pozici, která je spočtena z pozice jejich řídícího uzlu.

[4] Také se jedná o výškovou mapu.

3. Postupnou iterací přes jednotlivé řídící uzly jsou tvořeny čtverce. Při konstrukci čtverce jsou přiřazovány čtyři řídící a čtyři vedlejší uzly. Každý čtverec má dále přiřazenou konfiguraci, která je určena z hodnot jeho řídících uzlů.

4. V poslední řadě jsou všechny čtverce přidány do výsledného dvourozměrného pole, které bude použito v následujících krocích.

Algoritmus popsaný výše lze také pozorovat ve vizuální formě na obrázku 5.7. Očíslování jednotlivých kroků je identické.



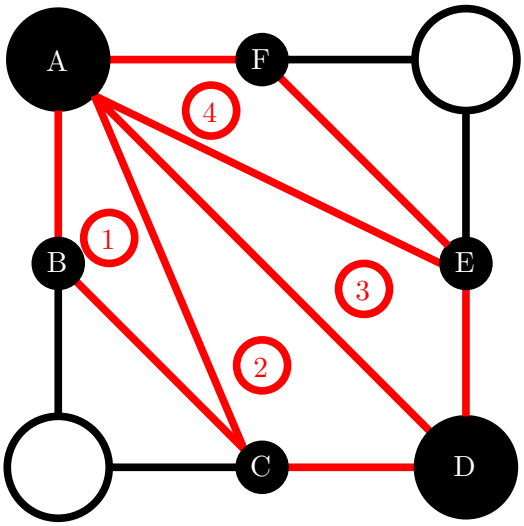
Obrázek 5.7: Diagram obsahuje celý proces vytváření dvourozměrného pole čtverců pro algoritmus marching squares. Jednotlivé kroky jsou očíslovány a všechny významné prvky pojmenovány.

Triangulace čtverců

Dalším krokem algoritmu marching squares je triangulace všech čtverců z pole. Triangulace probíhá postupným přidáním všech uzlů do pole vrcholů a následným definováním potřebných trojúhelníků. Proces triangulace jednoho čtverce probíhá v následujících krocích:

1. Postupnou iterací přes všechny uzly náležící danému čtverci jsou všechny přidány do pole vrcholů (do pole jsou vloženy reálné pozice uložené v uzlech).

2. Podle konfigurace daného čtverce probíhá triangulace postupným vkládáním trojúhelníků do pole (trojúhelník je reprezentován třemi uzly). Příklad triangulace čtverce s konfigurací číslo 10 je ukázán na obrázku 5.8. Tabulky 5.1 a 5.2 obsahují hodnoty, které by byly obsaženy v polích v případě uvažování tohoto čtverce.



Obrázek 5.8: Obrázek ilustruje triangulaci čtverce s konfigurací číslo 10. Čísla na obrázku značí jednotlivé vzniklé trojúhelníky, písmena pak slouží jako označení jednotlivých vrcholů.

index	0	1	2	3	4	5
pozice	A	B	C	D	E	F

Tabulka 5.1: Tabulka obsahuje příklad podoby pole vrcholů v případě triangulace čtverce s konfigurací 10 z obrázku 5.8. Jednotlivá velká písmena ve druhém řádku zastupují trojrozměrné body v prostoru ($\{A, B, C, D, E, F\} \rightarrow (x, y, z) \in \Re^3$).

trojúhelník	1			2			3			4		
index	0	1	2	3	4	5	6	7	8	9	10	11
hodnota	0	2	1	0	3	2	0	4	3	0	5	4

Tabulka 5.2: Tabulka představuje podobu pole trojúhelníků po triangulaci čtverce z obrázku 5.8. První řádek slouží jen pro ilustraci a odkazuje na jednotlivé trojúhelníky z obrázku. Třetí řádek obsahuje indexy vrcholů z tabulky 5.1, které jsou v trojúhelníku obsaženy.

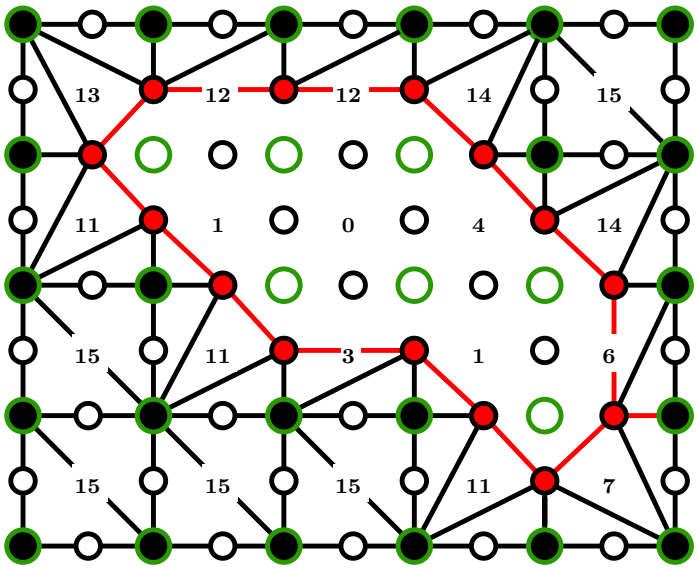
Přidání stěn

V této části již nebylo využito algoritmu marching squares, který je použitelný jen ve dvourozměrném prostoru. Přidání stěn do meshe probíhá ve dvou krocích – definice ohraničení a triangulace stěn. Ohraničení je potřebné definovat z důvodu určení kde mají stěny začínat (kde jsou spojeny se stropem). Celý tento proces probíhá následovně:

1. Nalezení libovolného uzlu, který není obsažen v konfiguraci 15 (tato konfigurace neobsahuje žádná ohraničení) a který je obsažen v nějakém trojúhelníku.

2. V následujícím kroku je nalezen jiný uzel, který s tím původním sdílí právě jeden trojúhelník (v tomto případě se jedná o hranu). Pokud takový uzel existuje, proběhne rekurzivní nalezení jeho následujícího uzlu.

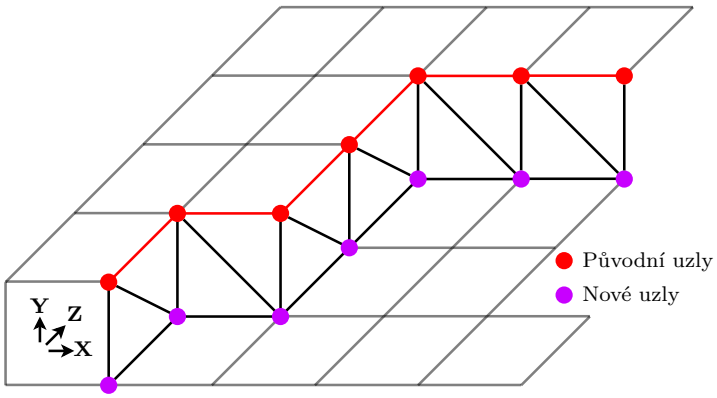
3. Jakmile neexistuje další uzel, který již nebyl předtím zkontrolován, tak se přidá ten první (z kroku 1) a tím se uzavře celé ohraničení. Příklad celého uzavřeného ohraničení lze pozorovat na obrázku 5.9 níže.



Obrázek 5.9: Diagram obsahuje příklad jednoho dokončeného ohraničení (červeně). Dále tento diagram znázorňuje i triangulaci všech obsažených čtverců podle jejich konfigurací (každý čtverec obsahuje číslo jeho konfigurace).

4. Celý proces výše se opakuje, dokud už neexistuje nezkontrolovaný uzel, který by tvořil ohraničení.

5. Nakonec proběhne iterace přes všechny uzly, které spolu sousedí v libovolném ohraničení. Pro tyto uzly jsou definovány další dva přesně pod nimi. Vzniklý obdélník je následně triangulován a přidán do meshe pro strop. Proces přidání nových uzlů a identifikace nových trojúhelníků je znázorněno na obrázku 5.10.



Obrázek 5.10: Graf naznačuje proces přidávání stěn do meshe pomocí vytváření nových uzlů. Nové uzly existují v meshi přímo pod těmi původními (sdílí stejné souřadnice x a z).

5.1.4 Instanciace objektu s příslušným meshem

Samotné vytvoření objektu, který obsahuje mesh (mesh je v Unity komponentou), probíhá v modulu `MeshCreator`. V rámci tohoto modulu je nejprve vytvořen prázdný objekt, kterému je poté přiřazen vytvořený mesh a v poslední řadě i příslušný materiál a fyzický materiál. Celý tento proces probíhá následně:

1. instanciace prázdného objektu;
2. vytvoření normálové mapy (řešeno pomocí funkcí obsažených v Unity);
3. vytvoření UV mapy (tento proces je popsán pomocí algoritmu 1);
4. přiřazení meshe k objektu;
5. definice dalších vlastností (materiál, fyzický materiál a komponenta *Collider*).

V rámci procesu popsaného výše probíhá i instanciace objektu, který představuje podlahu, avšak v tomto případě je objekt pouze tvořen z prefabu.

foreach $[i, vertex] \in vertices$ **do**
  $normal \leftarrow normals[i]$;
  `// výpočet posunu v obou horizontálních směrech`
  if $vertex \in wall$ **then**
    $offset.x \leftarrow vertex.y \times normal.x$;
    $offset.y \leftarrow vertex.y \times normal.z$;
  end
  `// výpočet kam na 2D UV mapu se bude daný 3D bod meshe mapovat`
  $percent.x \leftarrow \text{InverseLerp}(-width/2, width/2, vertex.x + offset.x)$;
  $percent.y \leftarrow \text{InverseLerp}(-height/2, height/2, vertex.z + offset.y)$;
  `// přidání nové UV souřadnice do meshe`
  $mesh.uvs[i] \leftarrow [percent.x, percent.y]$;
end

Algoritmus 1: Tento pseudokód reprezentuje proces mapování bodů meshe na body UV mapy ([x, y, z] $\in \Re^3 \rightarrow$ [u, v] $\in \Re^2$). Body patřící stropu jsou jen transformovány na souřadnice UV mapy pomocí výpočty v kolika procentech mapy se bod nachází. Tento způsob jednoduchého mapování je popsán i v kapitole 3 knihy od Trenta Polacka [10]. Ostatní body, náležící stěnám, jsou rozbaleny za využití jejich normál. Tento proces se nazývá unwrapping. V části, která se týká pouze nestropových bodů, `vertex.y` představuje konstantu a `normal.x` a `normal.z` představují „jako moc se stěna dívá jakým směrem“ (`normal.x` + `normal.z` = 1, `normal.y` = 0).

Přidělení materiálů

Po vytvoření objektu, který reprezentuje terén dané úrovně, je mu přidán materiál[5], který se mění vždy po porážce bosse. Ukázky dalších čtyř podob úrovně jsou na obrázcích 5.11.

[5] Specifický materiál je přidán i objektu podlahy.

(a) Ledová úroveň

(b) Lesní úroveň

(c) Písečná úroveň

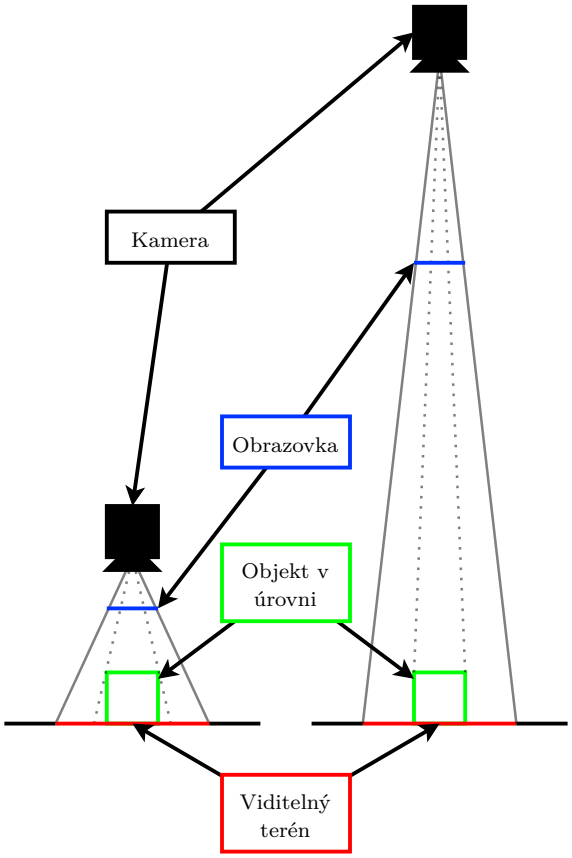
(d) Zimní úroveň

Obrázek 5.11: Obrázky výše představují jednotlivé podoby úrovně v závislosti na procházené úrovni (materiály jsou vybrány podle čísla úrovně).

5.2 Změna perspektivy

V jádru této mechaniky stojí modul *FlipManager*, který komunikuje s ostatními objekty prostřednictvím samostatného herního objektu. Tento objekt reaguje na akce hráče (stisknutí klávesy pro změnu pohledu) a uchovává informaci o právě aktivní perspektivě. Ostatní objekty ve hře buďto přímo reagují na invokované události nebo této informace využívají za běhu hry.

Objektem, reagujícím na překlopení, je kamera, která se skládá ze dvou kamer, mezi kterými se přepíná (vždy je aktivní právě jedna). Primární kamera (aktivní při pohledu shora) disponuje normálními vlastnostmi, avšak sekundární kamera (aktivní při pohledu z profilu) je umístěna daleko od hráče (ve vzdálenosti několika stovek jednotek) a její zorné pole (angl. FOV) je nastaveno na několikrát menší hodnotu, než by bylo běžné. Tato úprava vlastností sekundární kamery vede k iluzi, že je zvolena ortografická projekce.



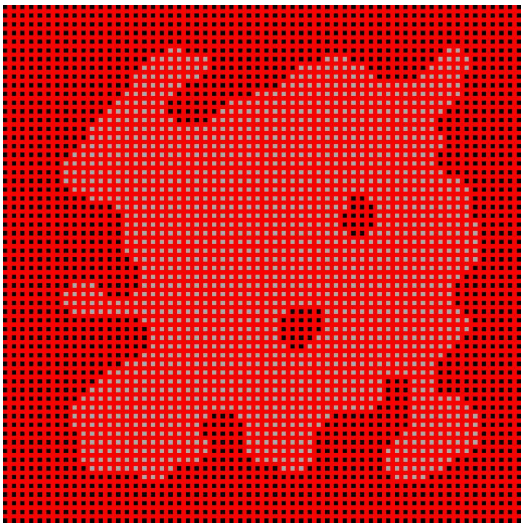
Obrázek 5.12: Obrázek vizualizuje efekt, kdy je sekundární kamera (vpravo) distancována od zobrazovaného objektu a je jí zmenšeno zorné pole. Jak lze vidět, tak obě kamery promítají stejnou část terénu (červeně) na obrazovku (modře) a velikost objektu (zeleně) je téměř totožná. Tento obrázek slouží pouze pro demonstraci, při implementaci jsou použity výrazně rozdílnější hodnoty a výsledný efekt je nerozlišitelný od efektu ortografické projekce.

5.3 Spawnování

Pro vytvoření libovolného objektu nebo nepřítele je nutné nejprve získat přesnou pozici, na které se má objekt instanciovat, a prefab, který slouží jako výchozí šablona pro daný objekt. Prefaby všech herních objektů jsou předem definovány, avšak určení pozice je založeno na náhodném výběru, který závisí na seedu definovaném během generování dané úrovně. Proces spawnování je realizován pomocí spawneru, který provádí instanciaci objektů následovně:

1. Inicializace procesu zahrnuje vytvoření 2D mřížky, která obsahuje hodnoty v rozsahu 0 až 255. Tyto hodnoty reprezentují pravděpodobnostní funkci sloužící k určení pozic pro objekty. Zdrojem dat pro mřížku je výšková mapa uložená ve struktuře `MapData`, jež představuje jediný vstupní bod pro tento algoritmus. Hodnoty z výškové mapy jsou transformovány do pravděpodobnostní mřížky podle definované sady pravidel: $\{(0 \rightarrow 255), (1 \rightarrow 0)\}$. Proces tohoto mapování je ukázán na obrázku 5.13.

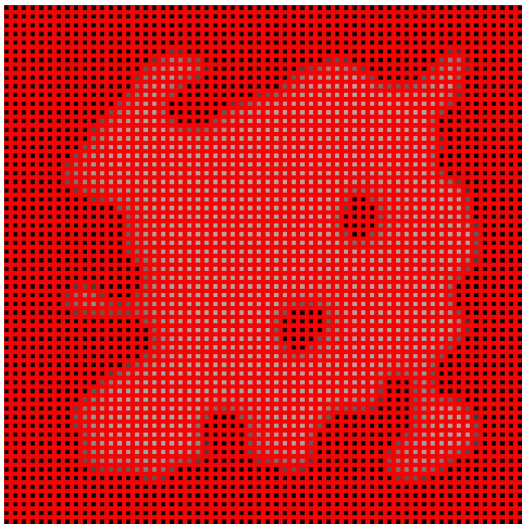
(a) Vstupní výšková mapa



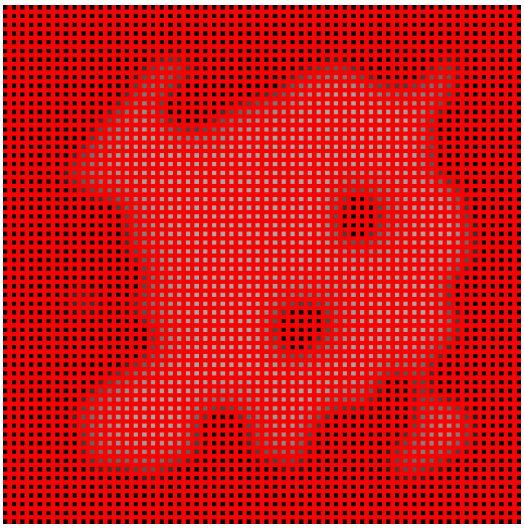
(b) Výsledná mřížka po transformaci

Obrázek 5.13: Obrázky zachycují vytvoření mřížky ze vstupní výškové mapy vygenerované pomocí celulárního automatu. Červená barva je zde pouze pro demonstraci.

2. Dalším krokem je průměrování hodnot v jednotlivých buňkách mřížky, které závisí na hodnotách v jejich okolí. Tento proces, ukázaný na obrázku 5.14, umožňuje vyhlazení pravděpodobnostní funkce tím, že se bere v úvahu lokální kontext každé buňky, což vede k realističtější distribuci pravděpodobností pro umístění objektů.



(a) Mřížka po 1. rekalkulaci



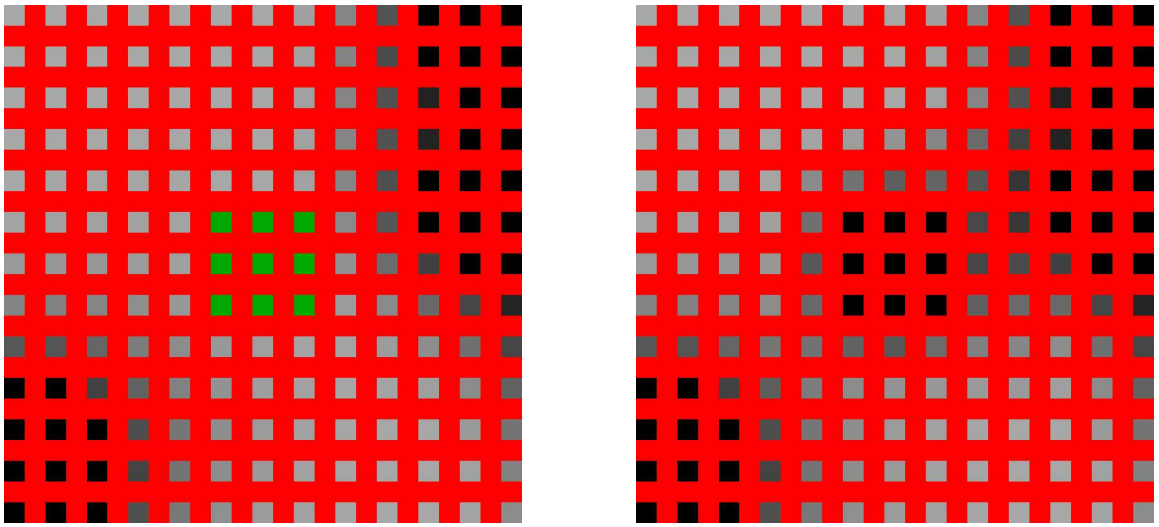
(b) Mřížka po 3. a konečné rekalkulaci

Obrázek 5.14: Obrázky výše znázorňují proces zahlazení mřížky pomocí průměrování hodnot kolem každé buňky. Červená barva je zde pouze pro demonstraci.

3. Pro určení reálné pozice každého objektu v herním světě je nejprve vybrána specifická buňka z mřížky, z níž je poté vypočtena její odpovídající pozice. Výběr buňky je realizován pomocí postupného kumulativního sčítání hodnot ve mřížce. Tento proces

pokračuje až do momentu, kdy součet dosáhne nebo překročí předem stanovenou náhodně zvolenou hodnotu (z intervalu <0, x>, kde $x = \sum buňky\ mřížky$). Jakmile je tento limit dosažen, je vybrána právě sčítaná buňka.

4. Po provedení tohoto výběru dochází k nastavení hodnot všech sousedních buněk na 0 a následnému průměrování buněk v daném rádiu. Ukázka vybrání buňky a následného průměrování je na obrázku 5.15.



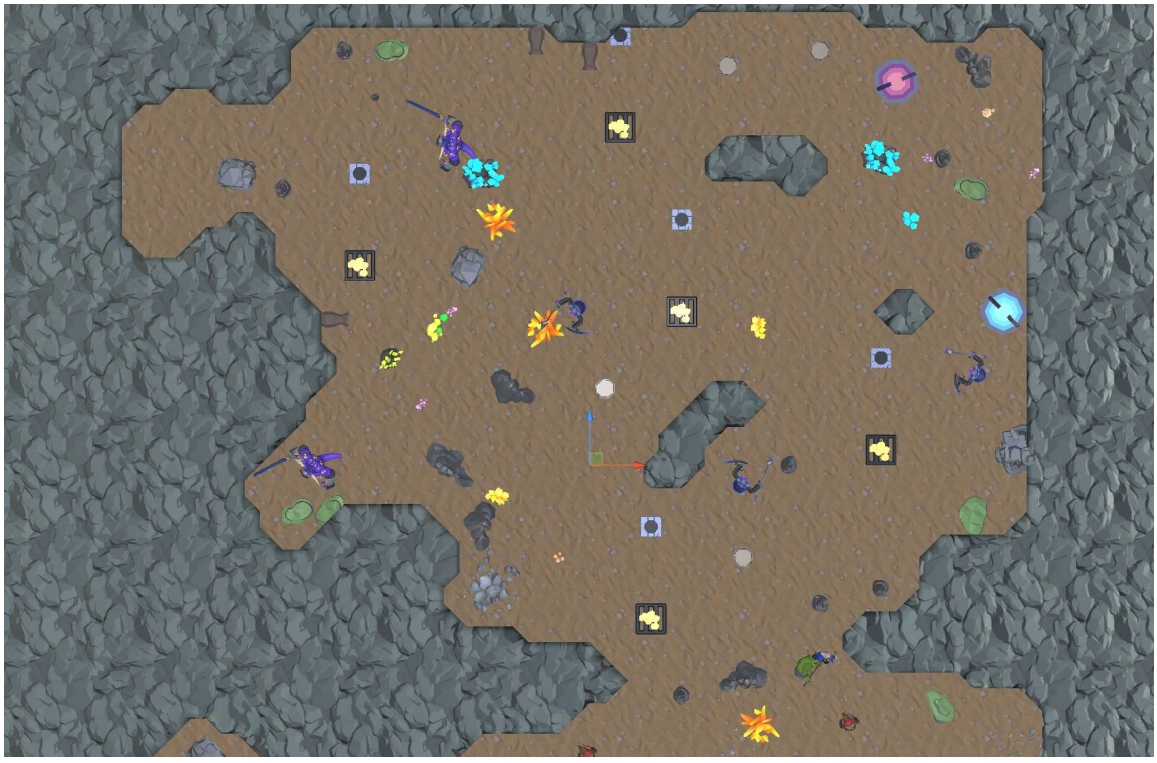
(a) Vybraná buňka (zeleně) ve mřížce

(b) Rekalkulace okolních buněk

Obrázek 5.15: Obrázek (a) obsahuje ukázku vybrání buňky pro instanciaci objektu. Druhý obrázek zachycuje stav okolí po zahlazení buněk.

5. Závěrečným krokem procesu je instanciace objektu a nastavení jeho náhodné rotace. Tento celý proces je opakován pro všechny objekty a nepřátele. Výsledná úroveň s již umístěnými objekty je na obrázku 5.16.

(a) Úroveň s objekty při perspektivní projekci



(b) Úroveň s objekty při ortografické projekci

Obrázek 5.16: Obrázky obsahují mapu s objekty a nepřáteli umístěnými pomocí algoritmu vysvětleného výše.

5.3.1 Spawnování generátorů kyseliny

Generátory kyselin se skládají z vícero objektů, a je proto nezbytné získat dvě pozice pro jejich instanciaci. Algoritmus použitý pro umístění těchto objektů je ve své podstatně identický s algoritmem pro generování těch ostatních, avšak s tím rozdílem, že v tomto případě probíhá celý proces paralelně pro obě vyžadované pozice (pozici pro generátor a pro padací dveře) a to s odstupem několika řádků v mřížce.

5.3.2 Spawnování děl

Speciálním případem je proces spawnování děl, kde pozice získaná pomocí mřížky není konečnou pozicí. Z této pozice je pouze vyslán paprsek v jednom z osmi možných směrů (čtyři ortogonální a čtyři diagonální) a místo, kde paprsek dopadne, se stává konečnou pozicí. Na tomto místě dochází k instanciaci prefabu děla, který je orientován ve směru normály k bodu dopadu paprsku. Celý tento proces je ukázán na obrázku 5.17.

Obrázek 5.17: Obrázek vizualizuje spawnování děla, kde červená kostka představuje místo vybrané pomocí mřížky, zelená čára kopíruje trajektorii vystřeleného paprsku a čára modrá znázorňuje směr rotace instanciovaného objektu.

5.4 Hráč

Herní postava představuje klíčový bod interakce mezi uživatelem a hrou, neboť je přímo ovládána hráčem. Aktivní mechaniky zahrnují pohyb postavy, útok a interakci s okolím. Pasivními prvky jsou ztráta životů a ukládání dat, nad kterými nemá uživatel přímou kontrolu.

Zachytávání uživatelského vstupu je realizováno prostřednictvím nového vstupního systému (angl. input system) platformy Unity, jenž využívá mechanismus zasílání a odebírání zpráv.

5.4.1 Pohyb

Pohyb postavy je realizován s využitím zabudovaného fyzikálního enginu a implementován je na principu aplikace síly. Chování postavy je ovlivněno vnějšími faktory, jako jsou kolize s ostatními objekty, a vnitřními vlastnostmi, jako jsou tření, odpor a úhlový odpor. Tohoto je dosaženo pomocí komponenty *RigidBody*. Kromě „standardního“ pohybu jsou implementovány také mechaniky levitace, která spotřebovává energii, a skákání včetně brzkého a pozdního skoku (angl. early jump a coyote jump).

5.4.2 Útok

Druhým způsobem, jak může hráč interagovat s herním prostředím, je soubojový systém, který umožňuje střílení projektilů směrem k pozici kurzoru myši na obrazovce. Proces vystřelení projektilu[6] probíhá ve třech fázích. Nejprve je zaznamenán stisk klávesy, následně je aktivována animace útoku, a konečně je v rámci animační události invokována instanciace projektilu, který je vystřelen směrem od hráče kam se hráč „dívá“.

5.4.3 Schopnosti

Všechny schopnosti, které může hráč sbírat, jsou implementovány prostřednictvím skriptovatelných objektů. Seznam aktuálně vybavených schopností je organizován jako fronta s pevně stanovenou maximální kapacitou. Při vybavení nové je tato schopnost zařazena do fronty a v případě, že byla kapacita fronty překročena, je nejdéle vybavená schopnost z fronty odstraněna. Seznam vybavených schopností je přístupný všem herním objektům, které mohou na základě stavu tohoto seznamu upravovat své chování.

5.4.4 Ukládání herních dat

Jediná data, která hra ukládá, zahrnují preference hráče a vylepšení atributů. Při jejich úpravě jsou tato data serializována a uložena ve formátu JSON. Při každém spuštění hry jsou data deserializována a načtena do herního objektu, který je uchovává po dobu běhu aplikace. Uložená data se nacházejí v perzistentním adresáři dat, který je při výchozím nastavení v operačním systému Windows umístěn ve složce AppData.

5.5 Nepřátelé

Jak je uvedeno v sekci 4.3.2, nepřátelé jsou organizováni do párů, přičemž každý pár je řízen jedním parentálním objektem. Tento objekt zajišťuje správu potomků v hierarchii scény a zodpovídá za aktivaci a deaktivaci těchto potomků v závislosti na změně perspektivy, při níž také dochází k přemístění neaktivního objektu na pozici aktivního. Tento přesun je vizuálně zamaskován částicovým systémem (angl. particle system). Kromě těchto úloh parentální objekt uchovává informace o celkovém $stavu \in (\check{z}ivý, mrtvý) \times (aktivní, neaktivní)$ a o aktuálním počtu životů.

[6] Za předpokladu, že hráč nemá aktivovanou žádnou schopnost.

Existuje několik druhů nepřátel, mezi kterými jsou některé mechaniky sdílené, zatímco jiné jsou specifické pro jednotlivé skupiny. Nepřátelé mohou být klasifikováni podle typu útoku (boj na blízko nebo na dálku) či podle perspektivy, ve které je definováno jejich chování. Sdílené mechaniky zahrnují celý bojový systém, včetně aktivace nepřátel a mechanismů pro utržení poškození. Specifické pro každou skupinu jsou pak mechaniky pohybu a rotace v prostoru.

5.5.1 Útok

Soubojový systém nepřátel funguje na podobném principu jako u hráče a závisí tedy na aktivaci útočné animace následované invokací útoku prostřednictvím zasílání zpráv. K aktivaci animace útoku dojde, pokud jsou splněny následující podmínky:

- Nepřítel je aktivní;
- Nepřítel se nachází v dostatečné blízkosti k hráči;
- Nepřítel není blokován stěnou.

Po zavolání metody pro útok je aktivována komponenta *Collider*, což dovoluje hráče zranit. Nepřátelé útočící na dálku jsou výjimkou – místo aktivace této komponenty vystřelí projektil.

5.5.2 Pohyb

Pohyb nepřátel je realizován s využitím navigační sítě (*navmesh*). Tato síť představuje prostor, který je pro postavy nepřátel schůdný, a je vytvářena během generování úrovně. Každý aktér, který se chce po této síti pohybovat, musí obsahovat komponentu *NavMesh Agent*. Proces hledání cesty a následný pohyb jsou řešeny interně prostřednictvím enginu Unity.

5.5.3 Boss

Pro bosse jsou specificky implementovány tři základní mechaniky: pohyb, útok a fázování souboje. Objekt bosse je aktivní během první a třetí fáze souboje. Ve druhé fázi jsou aktivní pouze draci, kteří střílí projektily, a po kolizi s hráčovým projektilem jsou zničeni. Po eliminaci posledního draka přechází souboj do třetí fáze, během které je boss znovu aktivován. Pohyb bosse je řešen stejným způsobem jako pohyb ostatních nepřátel. Obrázek 5.18 obsahuje jednotlivé útoky bosse včetně druhé fáze.

Boss může být v jedné chvíli právě v jenom ze stavů *roaming*, *attacking* a *dead*. Pokud je ve stavu *roaming*, pohybuje se boss pouze směrem ke hráči. Do stavu *attacking* se boss dostane, pokud je alespoň jeden jeho útok v dosahu. Proces útočení probíhá následovně:

1. zastavení agenta pro pohyb;
2. výběr jednoho z dostupných útoků (útok je po provedení na nějakou dobu deaktivován);
3. spuštění animace útoku;
4. aktivace kolizí pro zbraň (pokud nejde o plošný útok).

Během plošného útoku jsou před bossem postupně generovány exploze, přičemž každá následující exploze je umístěna ve větší vzdálenosti.

(a) Útok z blízka

(b) Útok na dálku

(c) Plošný útok

(d) Druhá fáze

Obrázek 5.18: Obrázky představují jednotlivé útoky v rámci souboje s bossem včetně druhé fáze, ve které jsou do prostoru umístěni draci střílející po hráči.

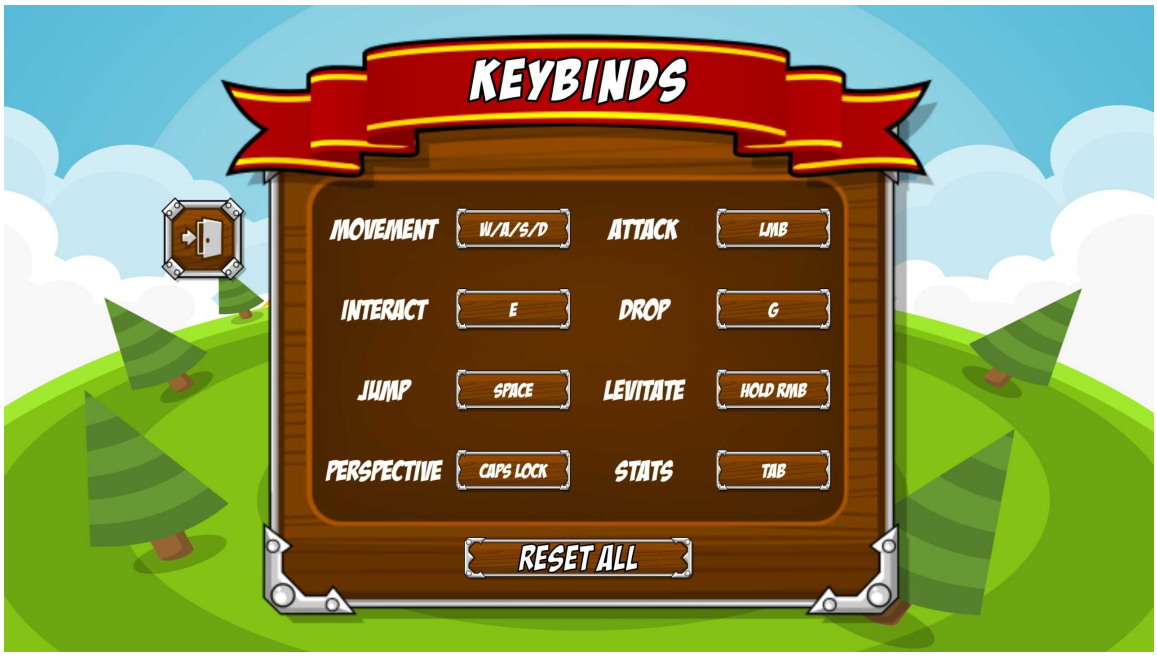
5.6 Grafické uživatelské rozhraní

Tato sekce je rozdělena na představení „klasického“ rozhraní (dále jen GUI) a na rozhraní, které překrývá část obrazovky při běhu hry (dále jen HUD). Vývoj obou těchto rozhraní proběhl v prostředí Unity s využitím předem definovaných herních objektů a s podporou balíčku TextMeshPro, který je integrován přímo v herním engine. Pro veškeré texty byl zvolen font BADABB.

5.6.1 GUI

Implementace GUI byla provedena v souladu s návrhem uvedeným v sekci 4. Většina uživatelského rozhraní je umístěna ve scéně *Menu*, která zahrnuje tři panely prezentované na obrázcích 5.19, 5.20 a 5.21. Funkcionalita jednotlivých tlačítek je realizována prostřednictvím specifických skriptů, které jsou následně zodpovědné za komunikaci s dalšími objekty ve hře.

Obrázek 5.19: Na obrázku je ukázka hlavní nabídky, která je zobrazena hráči po spuštění hry.



Obrázek 5.20: Nabídka, která je obsahem obrázku výše, slouží ke změně mapování kláves na akce ve hře.

Obrázek 5.21: Obrázek obsahuje náhled na podobu nabídky pro úpravu preferencí hráče. V současné době je zde pouze nastavení obtížnosti.

Mimo výše zmíněných hra obsahuje i nabídky, které nejsou obsaženy ve scéně *Menu*. Nabídka obchodu, ukázána na obrázku 5.22, slouží ke koupi vylepšení atributů hráče. V poslední řadě hra MageFlip obsahuje různé pauzové nabídky, které jsou ukázány na obrázcích 5.23.

Obrázek 5.22: Na obrázku je ukázka obchodu s vylepšeními, kterou je možno aktivovat ve scéně *Lobby*.

(a) Pauzová nabídka ve scéně *Lobby*

(b) Pauzová nabídka ve scéně *Level*

(c) Nabídka, která se zobrazí po smrti hráče

Obrázek 5.23: Na obrázcích lze viděl jednotlivé nabídky, které se zobrazí při stisku klávesy ESC nebo po smrti hráče.

5.6.2 HUD

HUD je specifický typ GUI, který často není interaktivní (nebo jen částečně) a který nenahrazuje obsah na obrazovce, ale jen jej částečně překrývá. Tento fenomén je vidět na obrázku 5.24, který detailněji ukazuje jednotlivé části tohoto rozhraní. Při návrhu nebylo uvažováno interaktivity těchto panelů a tedy tato funkce nebyla implementována.

Obrázek 5.24: Obrázek vizualizuje podobu obrazovky při průchodu úrovněmi, kde zeleně je označeno číslo úrovně, ve které se hráč právě nachází, modře aktuální počet mincí, oranžovou barvou stav životů a energie a fialově okno se statistikami o hráči a vybavené zbrani (toto okno není aktivní vždy, ale při jeho aktivaci hráčem). Červeně je vyznačen display aktivních schopností, které hráč posbíral v současném průchodu.

Kapitola 6

Testování

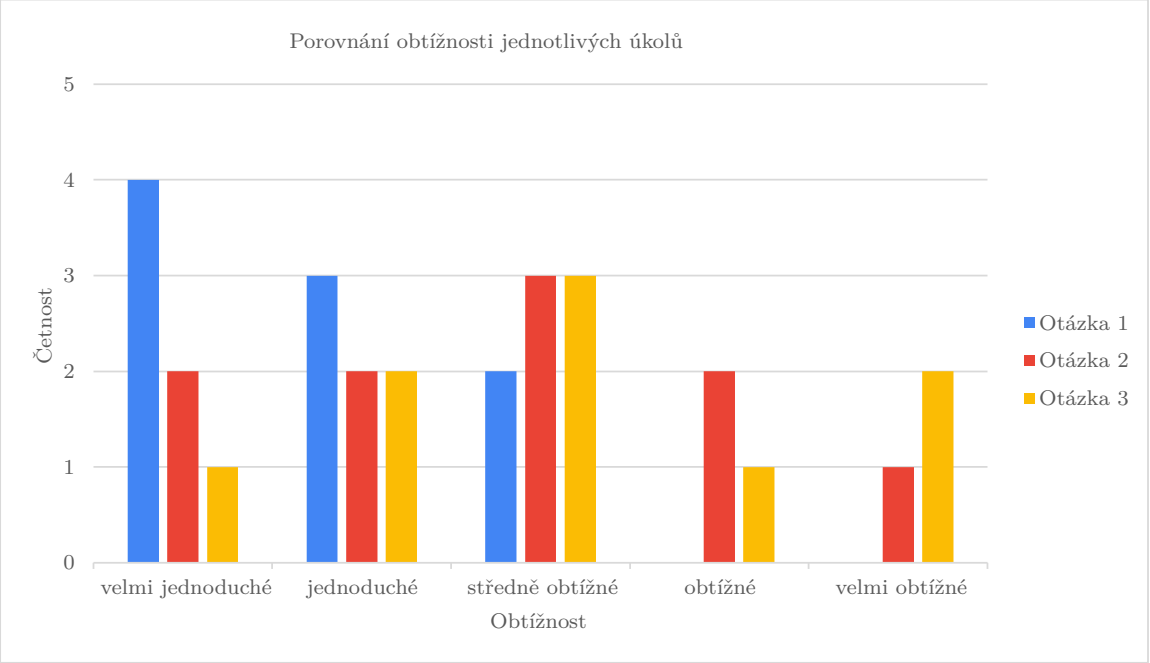
Tato kapitola je věnována uživatelskému testování, které bylo zhotoveno formou dotazníku. Dotazník byl rozeslán několika uživatelům, kteří postupně plnili úkoly a hodnotili jejich obtížnost a hru celkově. Mimo dotazníku byl zaslán i krátký návod ke hře a popis některých stěžejních mechanik.

Úkoly

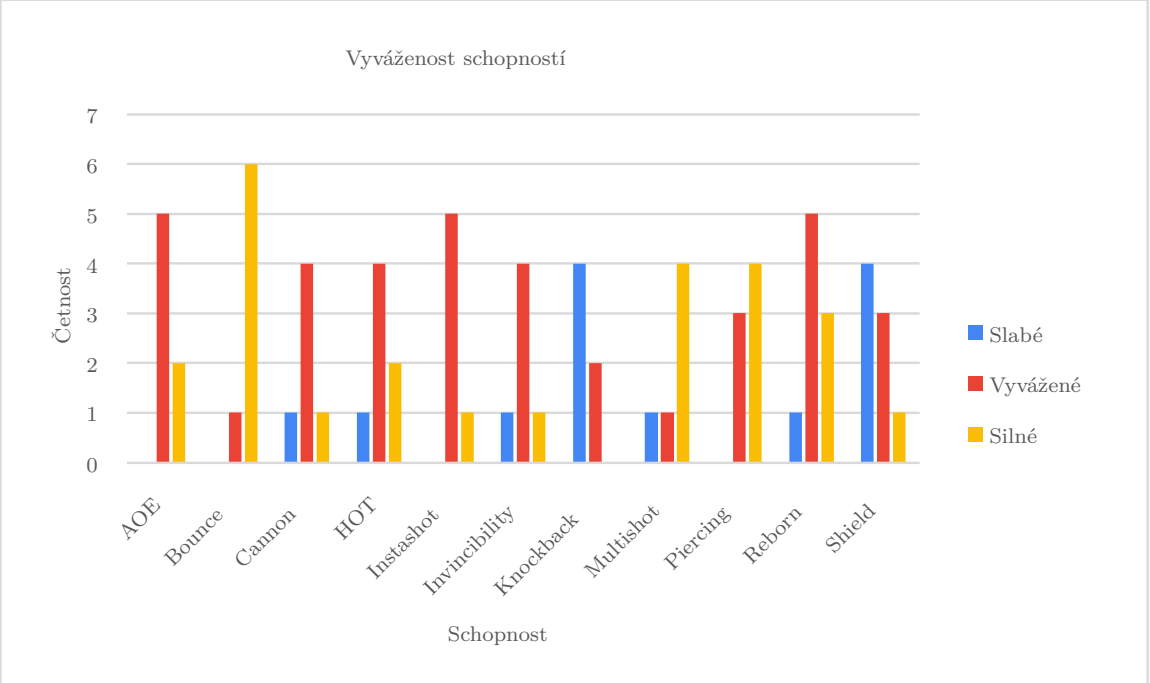
Dotazník, který byl rozeslán testerům, obsahuje následující čtyři úkoly:

1. Deaktivuj poškození a změň zbraň;
2. Vylepši libovolné atributy;
3. Poraž bosse a změň tím vizuál úrovně;
4. Vyzkoušej různé schopnosti.

Uživatelé měli za úkol toto splnit a následně ohodnotit obtížnost zadaných úkolů. Celkové vyhodnocení se nachází na grafu 6.1. Výjimkou je otázka číslo 4, která obsahovala ohodnocení vyváženosti jednotlivých schopností. Ohodnocení schopností je ukázáno na grafu 6.2.



Obrázek 6.1: Graf výše reprezentuje obtížnost jednotlivých zadaných úkolů, které museli testeři plnit. Na grafu lze pozorovat trendu zvyšování obtížnosti úkolů, ale žádný úkol nebyl nesplnitelný.

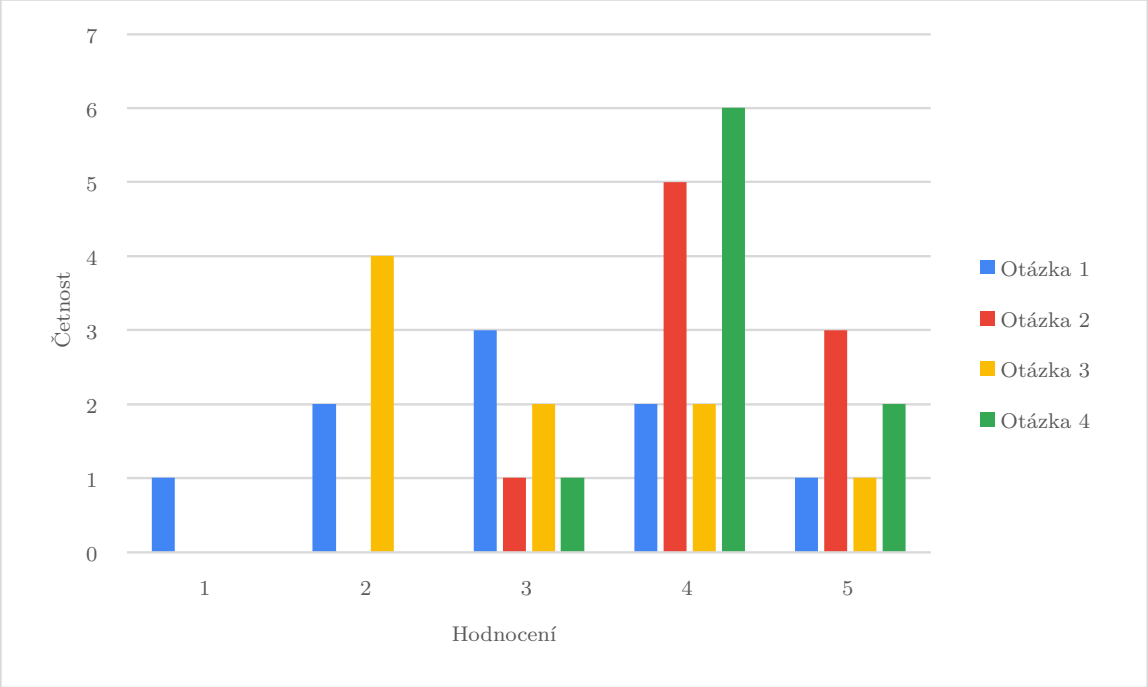


Obrázek 6.2: Graf ukazuje vyváženost jednotlivých schopností, které hra nabízí. Za nejsilnější schopnosti byly označeny *Bounce*, *Instashot* a *Piercing* a nejslabšími byly *Knockback* a *Shield*. Všechny schopnosti včetně jejich popisu jsou dostupné v tabulce 4.2. Žádné balancování po testování neproběhlo.

Obecné otázky

Ve druhé části dotazníky byly obecné otázky týkající se celé hry. Odpovědi na ně obsahuje graf 6.3. Tyto otázky byly položeny až po splnění všech úkolů a zní takto:

1. Jak je hra těžká?
2. Jak je hra zábavná?
3. Jak je hra vizuálně přívětivá?
4. Jaká je hra celkově?



Obrázek 6.3: Graf obsahuje odpovědi na otázky vypsané výše (1 je nejhorší, 5 je nejlepší).

Problémy a připomínky

Kromě výše uvedených otázek dotazník obsahoval i textová pole, kam bylo testerům umožněno psát různé připomínky nebo problémy. Jedním z problémů, který byl v dotazníku obsažen, je nedostatečná indikace po přiblížení se k barelu. Tento problém byl po ukončení testování vyřešen přidáním ikony v rámci HUD. Jedinou připomínkou byla nemožnost zobrazení informací ohledně právě aktivních schopností. Tento problém nebyl řešen.

Kapitola 7

Závěr

Cílem předkládané bakalářské práce je navrhnout a realizovat interaktivní videohru, která integruje prvky založené na změně perspektivy. Dalším bodem zadání je důkladná analýza oblasti herního vývoje a herních enginů, přičemž zvláštní pozornost bude věnována enginu Unity. Než dojde k samotnému vývoji hry, je nezbytné provést průzkum trhu s cílem identifikovat již existující podobné hry. Výsledky této práce musí být nakonec zveřejněny.

Všechny požadavky specifikované v zadání byly během realizace projektu úspěšně splněny. Aktuálně je hra plně implementována a obsahuje několik herních mechanik, jejichž fungování je přímo závislé na změnách perspektivy, které provádí hráč. Výsledná aplikace byla následně publikována na platformě OneDrive[1].

Kromě požadavků explicitně stanovených v zadání, hra zahrnuje i další mechaniky, které jsou nezbytné pro její kompletnost. Disponuje plně funkčním uživatelským rozhraním, procedurálně generovanými úrovněmi a systémem pro vylepšení postavy, a to jak během jednoho běhu, tak i mezi jednotlivými pokusy. Navíc hra nabízí uživatelům možnost přizpůsobit si nastavení dle vlastních preferencí, jako jsou úroveň obtížnosti nebo konfigurace klávesových zkratek pro různé akce ve hře. Celkově je tedy aplikace plně funkční a hratelná.

Přestože je hra již nyní funkční, její vývoj zatím není ukončen a plánuji v něm nadále pokračovat. V dalších fázích vývoje mám v úmyslu do hry implementovat zvukové efekty a hudbu, vytvořit žebříček nejvyšších skóre a začlenit mé vlastní modely pro určité herní objekty. Konečným cílem je zveřejnit hru na jedné z předních platforem pro distribuci her, jako je například Steam, což by umožnilo širší přístupnost a dostupnost mého díla.

[1] Dostupné z: https://1drv.ms/f/s!Ao8SOuS2wrlYkaoh9s_Xmszw-tccPA?e=u9eUde

Literatura

[1] Adamatzky, A. *Game of Life Cellular Automata*. 1. vyd. Springer London, 2010. ISBN 978-1-84996-217-9. Dostupné z: https://link.springer.com/book/10.1007/978-1-84996-217-9#bibliographic-information.

[2] Bethke, E. *Game Development and Production*. 1. vyd. Wordware Publishing, Inc., 2003. ISBN 1-55622-951-8. Dostupné z: https://books.google.cz/books?hl=cs&lr=&id=m5exIODbtqkC&oi=fnd&pg=PR3&dq=game+development&ots=oyyoUFdlPT&sig=ULQeHFAT-DB2gt8md6Xz7XRT8dc&redir_esc=y#v=onepage&q=game%20development&f=false.

[3] D'Ambrosio, D., Di Gregorio, S., Gabriele, S. a Gaudio, R. A Cellular Automata model for soil erosion by water. *Physics and Chemistry of the Earth, Part B: Hydrology, Oceans and Atmosphere*. 2001, sv. 26, č. 1, s. 33–39. DOI: https://doi.org/10.1016/S1464-1909(01)85011-5. ISSN 1464-1909. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1464190901850115.

[4] Dong, J., Liu, J., Yao, K., Chantler, M., Qi, L. et al. Survey of Procedural Methods for Two-Dimensional Texture Generation. *Sensors*. 2020, sv. 20, č. 4. DOI: 10.3390/s20041135. ISSN 1424-8220. Dostupné z: https://www.mdpi.com/1424-8220/20/4/1135.

[5] Feng Su, C. J. *Procedural Sound Generation for Soft Bodies in Video Games* [online]. 2019 [cit. 2023-11-27]. Dostupné z: https://dl.acm.org/doi/10.1145/3359566.3360068.

[6] He, Y., Hu, T. a Zeng, D. Scan-Flood Fill(SCAFF): An Efficient Automatic Precise Region Filling Algorithm for Complicated Regions. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2019.

[7] Hurley, S. *Unity vs Unreal Engine vs Godot: Which Is Best for Indie Developers?* [online]. 2023 [cit. 2023-11-27]. Dostupné z: https://indiegamecorner.com/unity-vs-unreal-engine-vs-godot/.

[8] Michael Lewis, J. J. *Game engines in scientific research* [online]. 2002 [cit. 2023-11-24]. Dostupné z: https://www.cse.unr.edu/~sushil/class/gas/papers/GameAIp27-lewis.pdf.

[9] Pasičnyk, M. *Generování rostlin pomocí L-systémů*. Brno, CZ, 2011. Diplomová práce. Masarykova univerzita Fakulta informatiky. Dostupné z: https://is.muni.cz/th/mmk0k/dp.pdf.

[10] Polack, T. *Focus on 3D Terrain Programming*. Premier Press, 2003. Focus on Game Development. ISBN 9781592000289. Dostupné z: https://books.google.cz/books?id=yrpgT_vHhqoC.

[11] Pruss, A. R. *Super-simple fractal terrain generator* [online]. 2017 [cit. 2023-11-27]. Dostupné z: http://alexanderpruss.blogspot.com/2017/03/super-simple-fractal-terrain-generator.html.

[12] Shukla, A. P. a Agarwal, S. Training two dimensional cellular automata for some morphological operations. In: *2014 Innovative Applications of Computational Intelligence on Power, Energy and Controls with their impact on Humanity (CIPECH)*. 2014, s. 121–126. DOI: 10.1109/CIPECH.2014.7019114.

[13] Zucconi, A. *The World Generation of Minecraft* [online]. 2022 [cit. 2023-11-24]. Dostupné z: https://www.alanzucconi.com/2022/06/05/minecraft-world-generation/.