

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

HRA V UNITY
THESIS TITLE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DAVID ZAHÁLK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ STARKA,

BRNO 2023

Zadání bakalářské práce



Ústav: Ústav počítačové grafiky a multimédií (UPGM)
Student: Zahálka David
Program: Informační technologie
Název: Hra v Unity
Kategorie: Počítačová grafika
Akademický rok: 2023/24

155959

Zadání:

1. Prostudujte herní engine Unity a techniky potřebné k tvorbě procedurálních her.
2. Prostudujte hry podobné hře Vampire Survivors, rozeberte klíčové herní mechaniky.
3. Navrhněte hru ve stylu Vampire Survivors. Můžete se inspirovat i v jiných žánrech.
4. Uvedenou hru implementujte v herním enginu Unity.
5. Vytvořte krátké demonstrační video práce.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:
1, a části 2-4.

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

Vedoucí práce: **Starka Tomáš, Ing.**

Vedoucí ústavu: Černocký Jan, prof. Dr. Ing.

Datum zadání: 1.11.2023

Termín pro odevzdání: 9.5.2024

Datum schválení: 9.11.2023

Abstrakt

Tato závěrečná práce se zaměřuje na vytvoření 2D rogue-like počítačové hry na styl Vampire Survivors s procedurálními prvky. Hra je implementovaná v herním enginu Unity. Mapa hry je procedurálně generovaná za využití Perlinova šumu a Celulárních automatů.

Abstract

This final thesis focuses on the development of a 2D rogue-like computer game in the style of Vampire Survivors, featuring procedural elements. The game is implemented using the Unity game engine. The game map is procedurally generated using Perlin noise and cellular automata.

Klíčová slova

rogue-like, pohled shora, procedurální generace, unity, c#, 2D, rpg, perlinův šum, celulární automaty, AI, počítačová hra, vampire survivors

Keywords

rogue-like, topdown, procedural generation, unity, c#, rpg, 2D, perlin noise, celular automata, context steering, AI, computer game, vampire survivors

Citace

ZAHÁLKA, David. *Hra v Unity*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Starka,

Hra v Unity

Prohlášení

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

.....
David Zahálka
6. května 2024

Poděkování

Poděkování patří vedoucímu bakalářské práce Ing. Tomaši Starkovy za nápady, rady a konzultace, které poskytl.

Obsah

1	Úvod	4
2	Teorie	5
2.1	Procedurální generování	5
2.2	Šum	6
2.3	Funkce pro generování procedurálního šumu	7
2.4	Celulární automaty	13
2.5	Whittakerovy Diagramy	16
3	Analýza a návrh	17
3.1	Analyzované hry	17
3.2	Popis hry	20
4	Implementace	23
4.1	Animace	23
4.2	Generování mapy	24
4.3	Efekty	28
4.4	Souboj	29
4.5	Pohyb nepřátel	31
4.6	Uživatelské rozhraní	33
4.7	Systém ukládání	34
4.8	Objevování nepřátel	35
4.9	Audio	36
4.10	Optimalizace	37
5	Závěr	39
	Literatura	40

Seznam obrázků

2.1	Procedurálně generovaná mapa hry Dwarf Fortress ¹	5
2.2	Porovnání náhodného šumu a Perlinova šumu	6
2.3	a) Mřížka s vyzobrazeným vstupním bodem 1.67, 1.33 b) Vektory v každém vrcholu mřížky a vektoru reprezentující pozici bodu vůči nejbližším vrcholům	7
2.4	fade funkce dána polynomem, uvedeným K. Perlinem [4]	8
2.5	Zobrazení výstupů funkcí $n(x)$ Perlinova šumu s: a) 1, 2 a 4 oktávami b) frekvencemi 2, 4 a 8 c) persistencí $\frac{3}{4}$, $\frac{1}{2}$ a $\frac{1}{4}$	9
2.6	Simplexová 2D mřížka složená z rovnostranných trojúhelníků	10
2.7	Porovnání Simplexového (a) a Perlinova šumu (b)	10
2.8	Waveletový šum. (a) Obrázek R náhodného šumu. (b) Obrázek R zmenšen na poloviční velikost. (c) Obrázek $R \downarrow$ rozšířen na původní velikost. (d) Šum $N = R - R \downarrow \uparrow$. (Obrázek převzat z [1])	11
2.9	Anizotropní gaborův šum, připomínající vzor kůže	12
2.10	okolí typu Von Neumann, Moore a Extended Moore	14
2.11	Mapy generované pomocí celulárních automatů	15
2.12	a) Příklad Whittakerova diagramu definujícího několik biomů na základě jejich průměrné teploty a počtu srážek. b) Výšková mapa, mapa teplot, mapa srážek a výsledná mapa biomů ²	16
3.1	Vampire survivors. Hráč bojuje s horou netopýrů ³	18
3.2	Hades. Výběr zbraní před vstupem do úrovně hry ⁴	19
3.3	Mapa (a) Don't Starve a obrázek ze hry (b) ⁵	19
3.4	Koláž obrázků ze hry	21
4.1	Všechny jednotlivé obrázky tvořící finální animaci běhu hráče	23
4.2	Stavy animací bosse v Animator Controlleru, při pohybu přechází do stavu Run, pokud je hráč ve vzdálenosti útoku, přechází z jakéhokoliv stavu do stavu Útoku, pokud je v klidu tak se nachází ve stavu Idle	24
4.3	Blend Tree představující pohyb hráče. Stavy jednotlivých animací se mění na základě změny souřadnic (x, y) hráčovy postavy	24
4.4	Mapa srážek, teplotní mapa, výšková mapa a finální mapa biomů	25
4.5	Vygenerovaná mapa kobky pomocí celulárních automatů	27
4.6	Příklady objektů nacházejících se na mapě	28
4.7	Animace krve po udeření nepřítele, realizované v Unity shader graph	28
4.8	Efekt krve při udeření nepřítele a efekt sněžení, který se aktivuje v zimním biomu	29
4.9	Celková animace útoku, složená z jednotlivých částí úderu	29
4.10	Čísla indikující udělené poškození. Šipka nad hráčem je jedno z pasivních vylepšení 4.12	30

4.11 Context steering - proces vybírání ideálního směru pohybu (Obrázky převzaty z [2]).	31
4.12 Příklady implementovaného herního UI.	33
4.13 Oblast, ve které se můžou objevit nepřátelé.	36
5.1 Obrázky z finální verze hry.	39

Kapitola 1

Úvod

Tato bakalářská práce je věnována návrhu a vývoji 2D rogue-like počítačové hry, jež klade důraz na procedurální generování obsahu, zejména pak herních map. Procedurální generování je klíčové pro tvorbu rozmanitých a proměnlivých herních světů, které jsou unikátní při každém novém herním sezení, čímž značně rozšiřuje opakovanou hratelnost her.

Cílem této práce není pouze konceptuálně navrhnut, ale také úspěšně implementovat hru, která se spoléhá na procedurální generování pro vytváření dynamických map. K vývoji byl zvolen herní engine Unity pro jeho flexibilitu a komplexní škálu nástrojů, které podporují rychlý a efektivní vývoj her. Unity představuje platformu pro tvůrčí práci s grafikou, fyzikou a interaktivitou, a je tak ideální volbou pro prototypování a iterace, které jsou nezbytné pro vývoj herního softwaru.

Theoretická část je podrobněji zaměřena na prozkoumání klíčových konceptů a metodik procedurálního generování. Tyto techniky v herním designu neznamenají pouze náhodné rozmístění prvků, ale především vytváření komplexních algoritmů, které dokážou simulovat realistické herní prostředí. Rovněž jsou analyzovány stávající hry žánru rogue-like, jejichž designové principy a inovace jsou pro vývoj uvedeného projektu inspirací.

Praktická část se poté podrobněji zaměřuje na konkrétní implementaci hry. Zde je rozepsán návrh a následná realizace procedurálně generovaných map, inteligentního systému nepřátelské AI, sofistikovaných technik správy herních objektů a metod optimalizace, které zajišťují plynulý běh hry.

Výsledkem této práce je tedy nejenom plně funkční a hratelná hra, ale také vhled do procesu jejího vývoje, od prvních teoretických kroků až po praktickou realizaci a finální testování.

Kapitola 2

Teorie

Tato kapitola pojednává o základech procedurálního generování ve světě videoher a technikách, které se k tomu běžně využívají, jako například Perlinův šum, Waveletový šum nebo celulární automaty.

2.1 Procedurální generování

Procedurální generování (angl. procedural content generation, PCG) je metoda vytváření dat počítačem pomocí předem definovaných pravidel a parametrů, jak je definováno v knize *Procedural Content Generation in Games* [6]. Využívá se na vytváření obsahu do videoher, jako například krajiny, objektů, postav či efektů. Důvodem pro použití procedurálního generování ve videohrách je schopnost vytvářet rozsáhlý a variabilní obsah bez potřeby ruční práce. Nevhodou PCG je jeho složitá implementace, jelikož špatně navržený algoritmus může způsobit potenciální repetitivnost.



Obrázek 2.1: Procedurálně generovaná mapa hry Dwarf Fortress¹.

¹Obrázek převzat z: <https://www.eurogamer.net/dwarf-fortress-updated-map-is-a-whole-new-world>

2.2 Šum

Šum je série náhodných nestrukturovaných čísel, obvykle seřazených do pole. K. Perlin a E. M. Hoffert [5] popsali šum následující definicí: *Šum je approximací bílého šumu, pásově omezeného na jedinou oktávu.*

Bílý šum má stejnou intenzitu ve všech frekvenčních pásmech. Omezení na jedinou oktávu znamená, že šum má tuto rovnoměrnou intenzitu pouze v určitém rozsahu frekvencí.

Každá funkce na vytvoření šumu by měla splňovat následující vlastnosti:

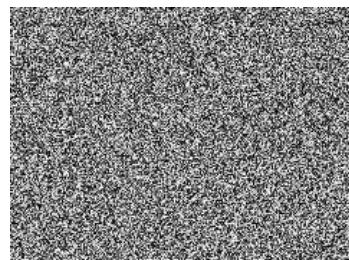
- je opakovatelná pseudonáhodná funkce svých vstupů
- má známý rozsah, zejména od -1 do 1
- je pásově omezená, obvykle s maximální frekvencí 1 Hz
- nevykazuje zjevné periodické nebo pravidelné vzory
- je stacionární - její vlastnosti se nemění s posunem
- je izotropní - její vlastnosti se nemění s rotací

Zatímco náhodný šum je založen na čistě náhodných procesech, procedurální šum je generován algoritmicky pomocí matematických funkcí nebo procedur, což umožňuje větší kontrolu nad výslednými daty a jejich strukturou. Není závislý na konkrétním vstupu, nýbrž na algoritmu, který generuje náhodné hodnoty na základě daných pravidel.

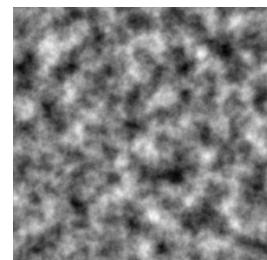
Funkci pro generování procedurálního šumu definujeme jako techniku na simulaci a vyhodnocení šumu.

Výhody funkce pro generování procedurálního šumu jsou následující:

- kompaktnost, obvykle zabírá několik kilobytů
- může být použita pro jakékoli rozlišení
- není periodická
- je náhodně přístupná, dokáže být vyhodnocena v konstantním čase nehledě na lokaci vyhodnocovaného místa



a) Náhodný šum



b) Perlinův šum

Obrázek 2.2: Porovnání náhodného šumu a Perlinova šumu

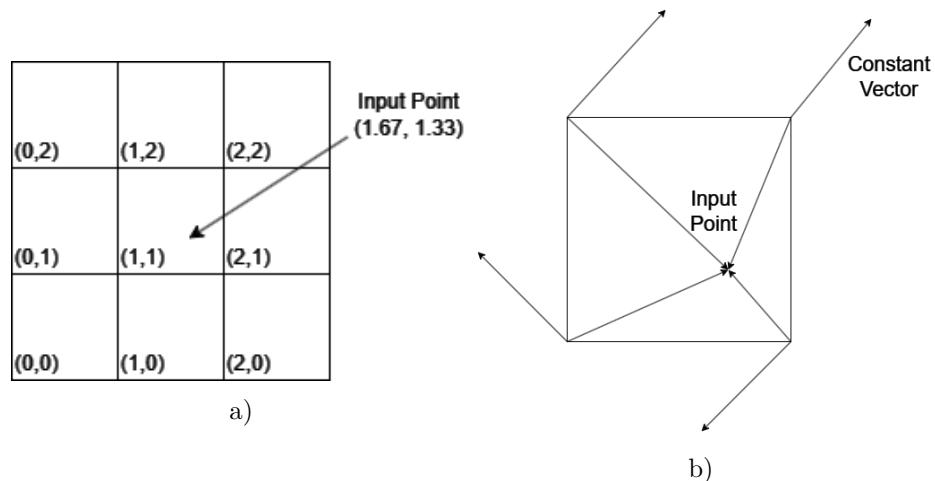
2.3 Funkce pro generování procedurálního šumu

Každá z těchto funkcí má své vlastní charakteristiky a využití v závislosti na konkrétním použití a požadovaném výsledku.

Perlinův šum

Perlinův šum je funkce pro generování procedurálního šumu, představena Kennethem Perlinem v roce 1983. Může být implementován v jedné nebo více dimenzích, což znamená variabilitu v počtu vstupních hodnot. V rámci dvoudimenzionálních aplikací je algoritmus strukturován tak, aby přijímal dvojici desetinných čísel jako vstupní hodnoty. Tyto číselné hodnoty reprezentují souřadnice v dvoudimenzionálním prostoru, na základě kterých algoritmus generuje odpovídající hodnotu šumu. Rozsah výstupních hodnot šumu je typicky definován v intervalu od -1 do 1. (viz 2.2).

V jeho 2D variantě začíná rozdělením prostoru na mřížku s celočíselnými souřadnicemi. Každý bod v prostoru je poté vyhodnocen vůči nejbližším čtyřem bodům této mřížky.



Obrázek 2.3: a) Mřížka s vyzobrazeným vstupním bodem 1.67, 1.33 b) Vektory v každém vrcholu mřížky a vektory reprezentující pozici bodu vůči nejbližším vrcholům

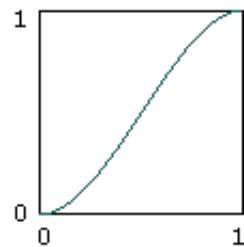
V každém z těchto vrcholů generuje algoritmus vektorový gradient, jejichž směry jsou náhodně určeny. Hodnota šumu v jakémkoliv bodě prostoru je pak určena skalárním součinem tohoto gradientu a vektoru, který reprezentuje relativní pozici tohoto bodu vůči vrcholu mřížky. Pro každý bod v prostoru se takto vypočte skalární součin s gradienty ve všech čtyřech nejbližších vrcholech mřížky. (obrázek 2.3)

I když v této fázi máme finální výsledek šumu v daném bodě, může tento šum působit velmi blokovitě a nerealisticky, protože je možné vidět přechody mezi hodnotami šumu v různých buňkách mřížky.

K dosažení hladšího a vizuálně přesvědčivějšího vzhledu šumu se uplatňuje technika lineární interpolace. Princip této techniky spočívá v počítání průměrů hodnot šumu mezi sousedními body v závislosti na jejich relativní poloze v mřížce, což vede k vytvoření hladkého a spojitého šumu bez zřetelných oddělení mezi segmenty mřížky.

Pro další zlepšení vizuální kvality je implementována tzv. fade funkce. Tato funkce transformuje lineární interpolanty na přirozenější křivky, čímž odstraňuje náhlé skoky v hodnotě šumu na hranách buňky. Fade funkce, běžně reprezentovaná polynomickým výrazem 2.1, zajistí, že první derivace šumu je na okrajích mřížkových buněk nulová, což vede k plynulému přechodu bez náhlých změn gradientu, jak je ilustrováno na obrázku 2.2.

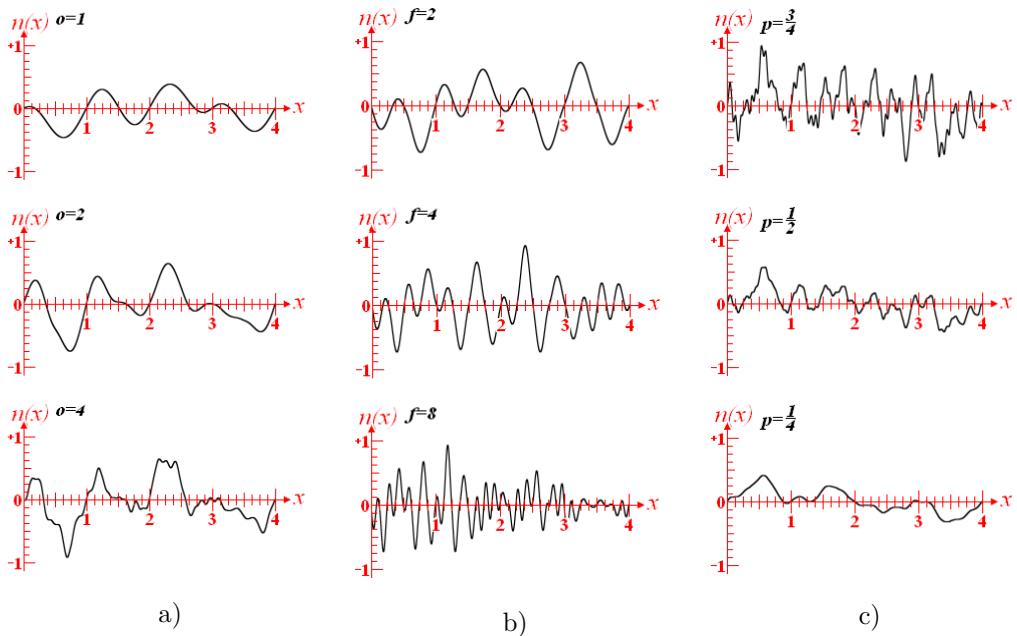
$$6t^5 - 15t^4 + 10t^3 \quad (2.1)$$



Obrázek 2.4: fade funkce dána polynomem, uvedeným K. Perlinem [4]

Mezi klíčové parametry Perlinova šumu patří následující:

- **Oktáva:** Každá z šumových funkcí v rámci Perlinova šumu se označuje jako oktáva. Každá následující oktáva má obvykle dvojnásobnou frekvenci ve srovnání s předchozí, což zvyšuje celkovou hustotu a detail šumu. Počet oktáv je přitom klíčovým faktorem určujícím, jak detailně budou výsledné textury vypadat, přičemž vyšší počet oktáv znamená složitější a časově náročnější výpočet.
- **Amplituda:** Definuje maximální možnou hodnotu, kterou může šum dosáhnout v každé oktávě. Vyšší amplituda znamená, že šum je intenzivnější a má silnější vliv na výslednou texturu.
- **Frekvence:** Určuje, kolik cyklů šumu se objeví na jednotku délky. Vyšší frekvence vytváří textury s více detaily na menších plochách.
- **Lacunarita:** Určuje, jak rychle frekvence šumu narůstá s každou novou oktávou. Vyšší hodnoty lacunarity vedou k vytváření složitějších a vizuálně bohatších vzorů.
- **Persistence:** Jde o koeficient, který ovlivňuje, jak rychle amplitudy jednotlivých oktáv klesají. Nižší hodnota persistencie způsobuje, že vyšší oktávy mají menší vliv na výsledný šum.

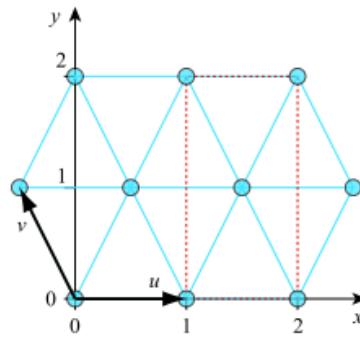


Obrázek 2.5: Zobrazení výstupů funkcí $n(x)$ Perlinova šumu s: a) 1, 2 a 4 oktávami
b) frekvencemi 2, 4 a 8 c) persistencí $\frac{3}{4}$, $\frac{1}{2}$ a $\frac{1}{4}$

Simplexový šum

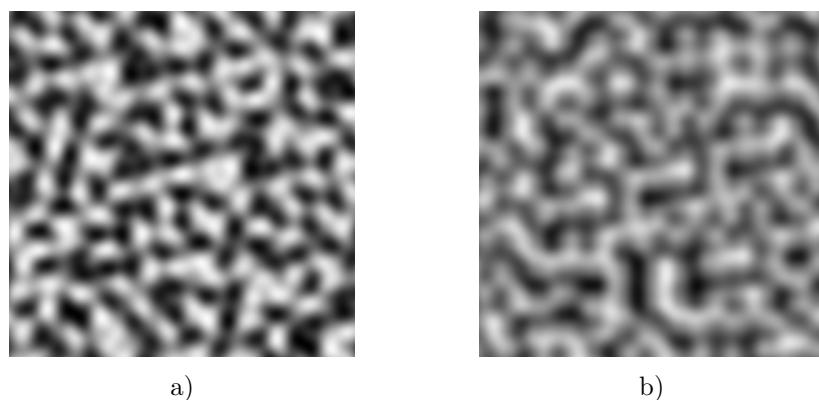
Simplexový šum, také vytvořen Kennethem Perlinem v roce 2001, představuje vylepšenou verzi Perlinova šumu. Tato metoda je efektivnější a často eliminuje některé z problémů spojených s Perlinovým šumem, zejména ve vyšších dimenzích.

Zatímco Perlinův šum využívá čtvercovou mřížku, simplexový šum se spoléhá na simplexovou mřížku. V 2D je tato simplexová mřížka tvořena rovnostrannými trojúhelníky (obrázek 2.6), což vede k menšímu počtu vyhodnocovaných bodů, a tudíž k rychlejším výpočtům. Simplexový šum také minimalizuje vizuální artefakty, které jsou často přítomné v Perlinově šumu, a dokáže tak dosáhnout vyšší kvality vygenerovaného šumu.



Obrázek 2.6: Simplexová 2D mřížka složená z rovnostranných trojúhelníků

V důsledku nižších nároků na výpočetní kapacitu je simplexový šum preferován ve vyšších dimenzích, zejména 3D a 4D prostředích, kde je jeho výkon výrazně vyšší než u Perlinova šumu. Nicméně kvůli komplexnějšímu algoritmu zůstává pro jednodušší 2D aplikace často používanější původní Perlinův šum, přestože má menší efektivitu při minimalizaci artefaktů.



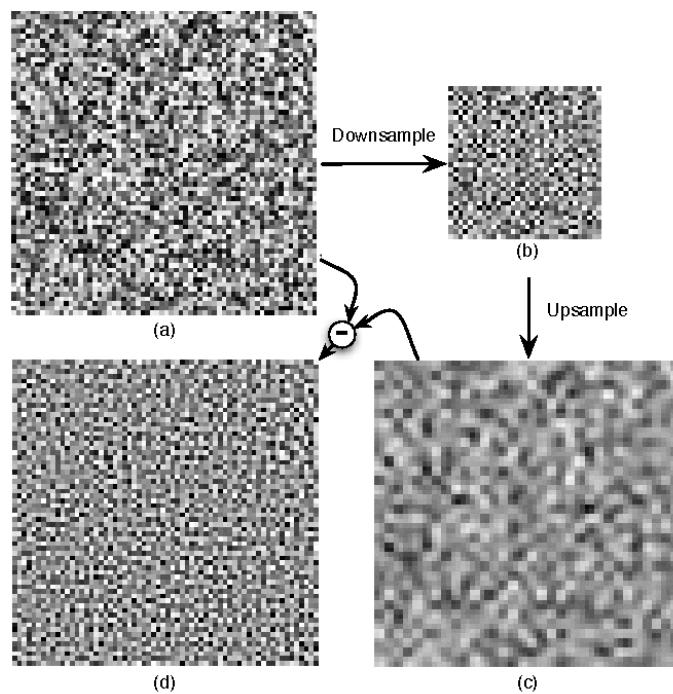
Obrázek 2.7: Porovnání Simplexového (a) a Perlinova šumu (b)

Waveletový šum

Waveletový šum představuje vylepšený typ procedurálního šumu, který byl vyvinut jako alternativa k Perlinovu šumu. Jeho klíčovou výhodou je schopnost redukovat aliasing² a zachovat detailey šumu. Používá se především v situacích, kde je potřeba větší kvalita a detail šumu.

Proces vytvoření waveletového šumu:

- Inicializace náhodného šumu (R): Proces začíná vytvořením základního obrázku R , naplněného náhodným bílým šumem.
- Postup snižování a zvyšování vzorkování: Tento základní obrázek je nejprve redukován na poloviční velikost, označený jako $R\downarrow$, a poté znova rozšířen zpět na původní rozměry, vytvářející obrázek $R\downarrow\uparrow$.
- Generování šumu N : Klíčovým krokem je následné odčítání upraveného $R\downarrow\uparrow$ od původního R . Výsledný obrázek, označený jako N , představuje finální šum. Tento šum zahrnuje především ty části původního šumu, které nebyly zachyceny v procesu snižování vzorkování, tedy ty detailey, které jsou relevantní pro vyšší frekvence.



Obrázek 2.8: Waveletový šum. (a) Obrázek R náhodného šumu. (b) Obrázek R zmenšen na poloviční velikost. (c) Obrázek $R\downarrow$ rozšířen na původní velikost. (d) Šum $N = R - R\downarrow\uparrow$. (Obrázek převzat z [1])

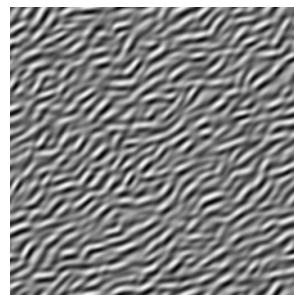
²Aliasing se objevuje, když je vyšší frekvence signálu zkreslena nedostatečným vzorkovacím procesem, což vede k nepřesnostem ve vizuálním výstupu, jako jsou zubaté okraje a ztráta detailů.

Gaborův šum

Gaborův šum, představený A. Lagaem v roce 2009 v článku *Procedural noise using sparse Gabor convolution* [3], je metoda pro tvorbu procedurálního šumu, která využívá speciální techniku zvanou řídká konvoluce s Gaborovým filtrem. Tato metoda pracuje s projekcí náhodně rozmístěných bodů přes Gaborův filtr. Tento filtr reprezentuje kombinaci Gaussovy funkce a kosinové funkce. Hlavní výhodou šumu je možnost manipulace s texturami a jejich vlastnostmi, jako jsou frekvence a orientace.

Využití Gaborova filtru ve výpočtu šumu umožňuje kontrolu nad vzhledem výsledné textury. Například, lze generovat izotropní šum, který je uniformně rozložený ve všech směrech, nebo anizotropní šum, který preferuje určitý směr.

Gaborův šum najde své uplatnění především v aplikacích, kde je potřeba přesného ovládání spektra, jako je vytváření realistických textur pro povrchové aplikace, například generování textur napodobující přírodní materiály, jako jsou kůže či tráva.



Obrázek 2.9: Anizotropní gaborův šum, připomínající vzor kůže

Porovnání funkcí pro generování šumu

Perlinův šum, s jeho fundamentálním přínosem pro tvorbu terénních variací, je ceněn pro jeho vizuální kvality a lehkou implementaci. Simplexový šum, navržen jako vylepšení Perlinova šumu, je efektivnější ve vyšších dimenzích a nabízí méně viditelných artefaktů. Gaborův šum, s jeho schopností modelovat anizotropní vlastnosti materiálů, poskytuje sofistikované nástroje pro tvorbu více strukturovaných textur. Naproti tomu waveletový šum se vyznačuje svými anti-aliasingovými charakteristikami a je vhodný pro aplikace, kde je potřeba zachování vysokého rozlišení detailů při různých měřítkách.

2.4 Celulární automaty

Celulární automaty (anglicky *Cellular Automaton*, CA) jsou matematické modely, původně navržené John von Neumannem [7], které simuluji komplexní dynamiku prostřednictvím interakcí jednoduchých buněk. Základem celulárního automatu je mřížka buněk, které mohou nabývat různých stavů. Stav každé buňky se určuje na základě stavů sousedních buněk podle předem definovaných pravidel. V herním designu umožňují tyto automaty vytváret dynamické, procedurálně generované prostředí.

Definice a vlastnosti CA

CA jsou složeny z následujících komponent:

- Mřížka (Lattice): Prostor, ve kterém buňky existují a interagují. Nejčastější používané jsou 1D (lineární řada buněk) a 2D mřížky.
- Okolí buňky (Neighbourhood): Okolí buňky definuje, které sousední buňky ovlivňují její stav.
- Množina stavů buňky: Každá buňka může nabývat počtu stavů, které jsou typicky reprezentovány jako sada diskrétních hodnot. V jednoduchém binárním CA jsou to například hodnoty 0 (neaktivní) a 1 (aktivní).
- Množina pravidel chování buňky: Pravidla definují, jak se stavy buněk mění v čase v závislosti na jejich aktuálním stavu a stavu okolí.

Typy celulárních automatů

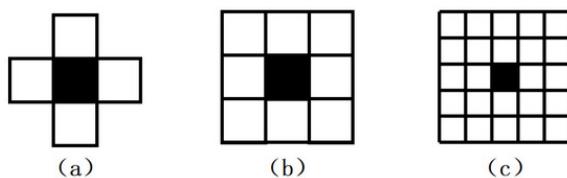
Celulární automaty mohou být klasifikovány na základě jejich dimenzionality a pravidel, která určují jejich chování. Zatímco jednorozměrné automaty jsou nejjednodušší, existují také vícedimenzionální varianty, které poskytují širší spektrum možností a komplexnější chování. Nejznámější klasifikace celulárních automatů podle jejich dynamického chování pochází od Stephena Wolframa [8], který je rozdělil do čtyř tříd:

- Třída 1: Automaty s jednoduchým a předvídatelným vývojem, po konečném počtu kroků dosáhnou jednoho konkrétního ustáleného stavu.
- Třída 2: Automat skončí v periodicky se opakující posloupnosti stavů nebo zůstane stabilně v některém ze stavů.
- Třída 3: Automaty, vykazující zdánlivě náhodné chování, jsou obtížně předvídatelné.
- Třída 4: Automaty kombinující běžné a chaotické chování.

Typy okolí

Volba okolí závisí na rozměru prostoru a tvaru buněk.

- Von Neumann: Buňka interahuje pouze se svými bezprostředními sousedy v jednotlivých směrech.
- Moore: Zahrnuje všechny buňky přímo sousedící s danou buňkou.
- Extended Moore: Zahrnuje větší počet sousedních buněk, specificky ty, které jsou umístěny v rozšířeném dosahu, konkrétně ve vzdálenosti dvou kroků od centrální buňky.



Obrázek 2.10: okolí typu Von Neumann, Moore a Extended Moore

Implementace CA

Implementace CA začíná definicí mřížky, což je prostor, ve kterém buňky existují a interagují. Tato mřížka může mít různé dimenze – od jednoduché lineární řady až po složitější dvou nebo vícedimenzionální struktury. Mřížka může být navržena jako konečná s pevnými hranicemi nebo jako nekonečná, umožňující rozšíření bez omezení.

Dalším krokem je inicializace stavů buněk, která může být provedena náhodně nebo podle specifického vzoru. V jednoduchém systému jsou stavy buněk 0 (neaktivní) nebo 1 (aktivní). Klíčovou rolí hráje definice okolí, které určuje, jaké sousední buňky ovlivňují stav dané buňky.

Zásadním aspektem CA jsou pravidla chování buňky, která určují, jak se stav buňky mění v závislosti na jejím aktuálním stavu a stavech sousedních buněk v jejím okolí. Jeden z nejznámějších příkladů pravidel v celulárních automatech je ten v Conwayově Hře života, která je dvoudimenzionálním celulárním automatem. Pravidla Hry života jsou následující:

- Narození: Prázdná buňka se stane živou, pokud má právě tři živé sousedy.
- Přežití: Živá buňka zůstává naživu v dalším kroku, pokud má dva nebo tři živé sousedy.
- Úmrtí: Živá buňka umírá, pokud má méně než dva nebo více než tři živé sousedy.

Proces simulace CA probíhá v diskrétních časových krocích, přičemž v každém kroku se aktualizují stavy všech buněk podle definovaných pravidel.

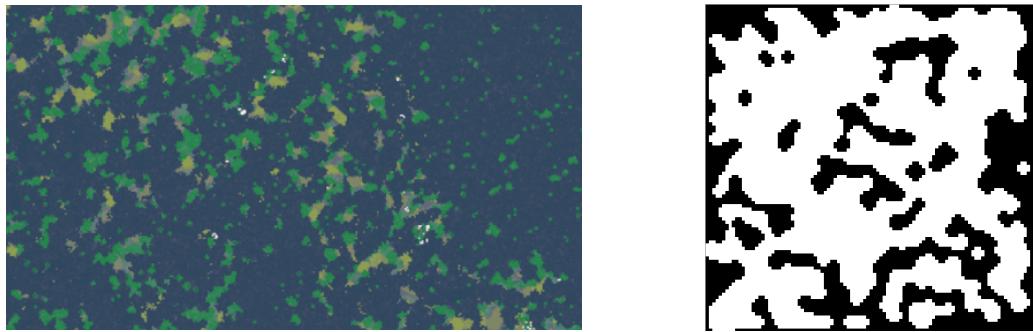
Příklady použití

Použití CA v oblasti procedurálního generování v počítačových hrách je například simulace požáru a šíření ohně. V takovém modelu může každá buňka reprezentovat část lesa nebo krajiny a její stav určuje, zda je oblast nehořící, hořící nebo již vyhořelá.

Dalším příkladem může být využití CA pro generování terénu nebo tvorbu komplexních jeskynních systémů (obrázek 2.11), kde stavy buněk mohou určovat, zda daná oblast je pevná zem, voda, nebo prázdný prostor.

Podobně jako simulace požáru mohou CA simulovat další živelné události jako je šíření záplav, lavin, nebo i šíření nemoci. Pravidla mohou odrážet různé faktory ovlivňující rychlosť a směr šíření.

CA lze také použít k dynamické změně úrovně hry podle postupu a chování hráče. NPC (non-player characters) chování může být řízeno celulárními automaty, kde každá buňka představuje potenciální akci nebo stav NPC. Podle stavů sousedních buněk a vstupů z prostředí (např. hráčova přítomnost nebo nedávné akce) se NPC mohou rozhodnout pro určité chování, jako je útok, ústup, sběr zdrojů nebo vyhledávání spojenců.

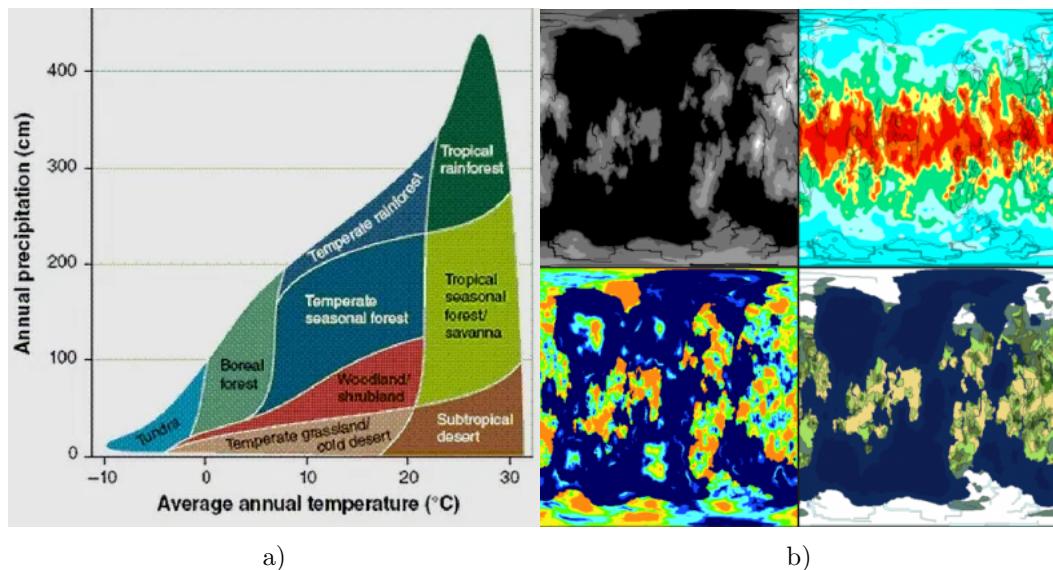


Obrázek 2.11: Mapy generované pomocí celulárních automatů

2.5 Whittakerovy Diagramy

Whittakerovy diagramy, vytvořené ekologem Robertem H. Whittakerem, jsou klíčové v ekologii pro zobrazení vztahů mezi klimatem a rozložením biotopů. Tyto diagramy vizualizují, jak teplota a srážky ovlivňují typy vegetace a ekosystémů v různých regionech, jak lze vidět na obrázku 2.12.

V herním designu najdou Whittakerovy diagramy uplatnění při modelování přírodních prostředí, zvláště ve hrách s procedurálně generovanými světy. Informace z těchto diagramů pomáhají rozhodovat o distribuci vegetace a terénu v závislosti na klimatických podmínkách.



Obrázek 2.12: a) Příklad Whittakerova diagramu definujícího několik biomů na základě jejich průměrné teploty a počtu srážek. b) Výšková mapa, mapa teplot, mapa srážek a výsledná mapa biomů³.

³Obrázek převzat z: <https://www.gamedeveloper.com/programming/procedurally-generating-wrapping-world-maps-in-unity-c-part-4>

Kapitola 3

Analýza a návrh

Tato kapitola se zaměřuje na klíčové prvky a mechaniky, které definují žánr vytvořené hry, a na konkrétní příklady her, které sloužily jako hlavní inspirace pro návrh vlastního herního projektu.

3.1 Analyzované hry

Tato část je věnovaná důkladné analýze a přehledu herních titulů, které byly významným zdrojem inspirace pro design a herní systémy navrhovaného projektu. Každá vybraná hra je představena prostřednictvím stručné charakteristiky jejího herního stylu, následované podrobným rozborem specifických mechanik a prvků.

Vampire Survivors

Vampire Survivors, vyvinutý společností Poncle v roce 2021, představuje typický příklad 2D rogue-like hry s výrazným důrazem na procedurální generování obsahu a systémy postupného zlepšování hráče v průběhu úrovně. Hra nabízí širokou škálu map, které jsou generovány náhodně a poskytují různé herní prostředí a výzvy. Každá postava ve hře má jedinečné schopnosti a atributy, což umožňuje hráčům přizpůsobit svůj herní styl a strategie.

Hlavní herní mechanika zahrnuje boj proti vlnám nepřátel, přičemž hráči sbírají body za každého poraženého nepřítele. Za zabité nepřátele získává hráč zkušenosti, díky kterým odemyká nové zbraně, ulehčující jeho souboj s monstry. Systém vylepšení je zásadní pro přežití ve stále náročnějších úrovních hry. Po smrti postavy mohou hráči využít získané mince k odemknutí nových postav a map, což podporuje opakovou hratelnost hry.

Z Vampire Survivors byly převzaty klíčové prvky pro návrh vlastní hry, jako je procedurální generování map, které zajišťuje unikátní herní zážitek při každém spuštění. Tento princip byl implementován s cílem zvýšit dynamiku a variabilitu v herním prostředí. Dále byl adaptován systém postupného získávání vylepšení, který umožňuje hráčům rozvíjet své schopnosti a zvyšovat šance na přežití proti zesilujícím nepřátelům. Systém odměn po smrti postavy byl rovněž integrován do navrhované hry, což motivuje hráče k dalším pokusům.

Tento přístup k návrhu hry má za cíl nejen zlepšit herní zážitek, ale také prodloužit celkovou dobu hraní díky vyšší opakování hratelnosti a motivaci hráčů. Průběžné vylepšování schopností postavy a možnosti odemknout nový obsah podporují hráče v kontinuálním průzkumu herních možností a strategií.



Obrázek 3.1: **Vampire survivors**. Hráč bojuje s hordou netopýrů¹.

Hades

Hades, vyvinutý společností Supergiant Games v roce 2020, je rogue-like hra, která překračuje tradiční hranice žánru díky svému jedinečnému přístupu ke storytellingu a propracovanému systému postupného vylepšování. Hra je zasazena do světa řecké mytologie, kde hráč ovládá Zagrea, syna Háda, který se snaží uniknout z podsvětí.

Jedním z klíčových prvků, které Hades využívá, je detailně propracovaný systém odemykání a vylepšování zbraní. Každá zbraň ve hře nabízí odlišné herní styly a schopnosti, což hráčům umožňuje experimentovat a najít strategie, které nejlépe vyhovují jejich způsobu hraní. V průběhu hry hráči odemykají nové zbraně a schopnosti pomocí speciálních měn získaných během jejich pokusů o útěk z podsvětí.

Procedurální generování úrovní a dynamický bojový systém zajišťují, že každý nový pokus o průchod hrou je jedinečný, s různorodými výzvami a nepřáteli. S každým novým průchodem se otevírá příležitost použít různé kombinace zbraní a schopností, což dodává hře vysokou míru opakování hratelnosti.

Inspirace pro tento konkrétní herní projekt byla převzata právě z tohoto systému odemykání a nasazování zbraní do dalších úrovní hry. V projektu je implementován podobný systém, kde hráči mohou odemknout a vybavit své postavy různými zbraněmi, které následně ovlivňují jejich sílu v následujících hrách. Tento přístup poskytuje hráčům motivaci pokračovat ve hře a odemykat nové zbraně, aby dosáhli lepších výsledků.

Dalším klíčovým aspektem, který byl převzat do projektu, je koncept generování mapy. Na rozdíl od Vampire Survivors, kde je herní prostor jedna velká otevřená mapa, Hades přistupuje k tvorbě herního prostoru odlišně. Hra je strukturována do série menších místností a prostorů, které jsou generovány procedurálně a jsou spojeny do lineárnějšího, avšak rozvětveného průchodu.

Ve hře byl tento princip adaptován do systému velké mapy, která je generována pomocí celulárních automatů, což přispívá k efektivnímu vytváření struktury rozdělené do oblastí připomínajících místnosti.

¹Obrázek převzat z: <https://www.polygon.com/reviews/22927447/vampire-survivors-review-roguelike-bullet-hell-steam-pc-browser-8-bit-characters-power-ups>

²Obrázek převzat z: <https://www.thegamer.com/hades-weapons-ranked/>



Obrázek 3.2: **Hades**. Výběr zbraní před vstupem do úrovně hry².

Don't Starve

Don't Starve je survival hra vyvinutá společností Klei Entertainment. Hra kombinuje prvky dobrodružství a boje o přežití ve světě plném nebezpečí. Po spuštění hry je hráč náhodně umístěn do procedurálně generovaného světa, který je plný divoké přírody, nepřátelských tvorů a různých přírodních zdrojů. Hráčova hlavní úloha je přežít co nejdéle tím, že sbírá suroviny, vyrábí nástroje a zbraně, staví přístřeší a udržuje si dobrý psychický a fyzický stav.

Herní mechaniky zahrnují komplexní systém sběru a využití zdrojů, kde hráč musí efektivně spravovat své zásoby jídla, materiálů a dalších klíčových předmětů pro přežití.

Inspirace pro generování herního světa byla čerpána jak z Vampire Survivors, tak ze hry Don't Starve, která využívá procedurálního generování k vytváření rozmanitých 2D map rozdelených do různých biomů. Na rozdíl od her jako například Minecraft, kde je krajina trojrozměrná s různými výškovými úrovněmi, Don't Starve se soustředí na plochý terén s jasně definovanými oblastmi. Tento přístup byl adaptován ve vytvořené hře, aby i přes absenci výškových rozdílů poskytovala každá mapa vizuální i funkční rozmanitost, od lesů až po pustiny, což podněcuje hráče k průzkumu prostředí. V Don't Starve jsou vodní plochy nepřístupné, v návrhu jsou rovněž zachovány jako nepřístupný terén, který slouží jen jako vnější bariéra a vyznačuje konec mapy, aniž by přidával další interaktivní možnosti.



Obrázek 3.3: Mapa (a) Don't Starve a obrázek ze hry (b)³.

³Obrázek (a) převzat z: <https://interfaceingame.com/screenshots/dont-starve-map/>
Obrázek (b) převzat z: <https://www.zestolu.cz/deskovky/dalsi-stolni-adaptaci-videohry-bude-survival-dont-starve-575630>

3.2 Popis hry

Hráč začíná své dobrodružství načtením se do náhodně vygenerovaného světa, který je při každém novém startu hry odlišný, a to díky implementovanému systému procedurálně generované mapy. Tento svět je rozdelen do několika odlišných biomů, které jsou distribuovány po mapě podle předem definovaných pravidel, jak je podrobněji popsáno v sekci týkající se generování mapy (viz 4.2). Hlavním úkolem, který na hráče čeká, je co nejdéle přežít proti hordám nepřátel, jejichž síla a počet narůstá s postupem času. Hráč má za úkol shromažďovat co nejvíce bodů, přičemž tyto body jsou mu přidělovány za úspěšné zneškodnění nepřátel, stejně jako za každou sekundu, kterou dokáže ve hře přežít.

V každém z vygenerovaných biomů se objevují různé typy monster, která se liší svými unikátními schopnostmi a charakteristickými atributy. Některá monstra se mohou pohybovat s vyšší rychlostí, jiná naopak disponují větší odolností vůči úderům hráče. Další skupina monster je schopna vypouštět silné, avšak pomalu letící ohnivé koule, jiné střílí mnoho rychlých koulí slizu, které ale udělují nižší poškození. Kromě získávání bodů hráč po zabití nepřátel získává také zkušenosti a mince, které mohou být použity k vylepšení různých atributů jeho postavy. Mezi tyto atributy patří například síla, která zvyšuje poškození způsobené nepřátelům, štěstí, které zvyšuje pravděpodobnost získání dalších mincí z poražených nepřátel, nebo inteligence, která zvyšuje počet získaných zkušeností ze zabitých monster.

Začátek každé nové hry znamená pro hráče reset úrovně a jeho atributů. Co ale zůstává, jsou mince, které si hráči mohou uchovat a použít k nákupu silnějších zbraní. Každá dražší zbraň nabízí vyšší poškození, delší dosah útoku a rychlejší rychlosť útoku. Za mince lze také zakoupit posilnění, která hráčům poskytují různé pasivní bonusy v následujících hrách, jako například magnet na mince, nebo regenerace ubraných životů. Kromě nepřátel a biomů je na mapě umístěn i vstup do podzemí, kde se nacházejí ještě silnější nepřátelé, kteří však za své poražení nabízejí výrazně vyšší odměny v podobě bodů. Po uplynutí určitého času se dále na mapě objevují bossové, kteří jsou oproti běžným nepřátelům mnohem silnější a jejich přítomnost výrazně ztěžuje hráčův pokus o co nejdéle trvající přežití, avšak zároveň poskytují nejvyšší možné odměny v bodovém hodnocení.



(a) Hráč bojuje s nejslabším typem nepřátele, lesními gobliny.

(b) Hráče se v pouštním biomu snaží zabít píseční goblini.



(c) Souboj s bossem na pokraji zimního biomu.

(d) Hráč bojující s rytíři na sněžných pláních.



(e) Souboj v podzemí s přízraky rytířů a létajícíma očima.

Obrázek 3.4: Koláž obrázků ze hry.

Seznam nepřátel

Ve hře se objevuje několik typů nepřátel. Následující tabulka obsahuje seznam jednotlivých nepřátel, místa jejich výskytu a jejich klíčové atributy.

Tabulka 3.1: Přehled nepřátel a jejich atributů

Jméno	Životy	Poškození	Dosah	Rychlosť	Zk	Skóre	Mince	Oblast
Lesní goblin	110	25	0.65 m	2.5 m/s	10 zk	8	0.5%	les/louka
Písečný goblin	125	25	0.65 m	2.5 m/s	10 zk	10	0.5%	poušt
Mág goblin	60	40	8 m	2 m/s	15 zk	10	0.5%	les/louka
Slimák	150	25	0.8 m	1.5 m/s	15 zk	15	0.6%	les/podzemí
Zlý rytíř	180	40	0.65 m	2.5 m/s	20 zk	15	0.6%	sněžná plán
Přízrak rytíře	150	50	0.65 m	3 m/s	20 zk	20	0.6%	podzemí
Létající oko	80	15	6 m	3.5 m/s	10 zk	10	0.5%	podzemí

Seznam vylepšení

V obchodu může hráč zakoupit následující vylepšení:

- **Magnet na mince** – Přitahuje mince z určité délky, tím pádem není potřeba přes ně přebíhat za účelem sebrání.
- **Pasivní regenerace životů** – Životy se doplňují při každé nově dosažené úrovni zkušeností hráče, při vyšších úrovních a pomalejším postupování na další úroveň se hodí pasivní doplňování životů.
- **Hledač podzemí** – Vchod do podzemí se na mapě objevuje pouze jeden a to na náhodném místě. Hráči, kteří chtějí dosáhnout rychleji většího skóre se musí vydat do podzemí. Aby jej lehčejí nalezli, slouží k tomu hledač podzemí, který hráče naviguje ke vchodu.

Přehled atributů

Při každé nově dosažené úrovni je hráč odměněn třemi body vylepšení, za jejich útratu může vylepšovat následující atributy:

- Síla – Zvyšuje poškození, které hráč uděluje nepřátelům.
- Obratnost – Zvyšuje obranu hráče, tím pádem snižuje přicházející poškození.
- Inteligence – Zvyšuje množství zkušeností, které nepřátelé odměňují za zabití.
- Vitalita – Zvyšuje životy hráče.
- Rychlosť – Zvyšuje rychlosť pohybu hráče.
- Štěstí – Zvyšuje šanci na získání mincí z nepřátel. Při překročení šance 100% se zvyšuje množství odměněných mincí za zabití nepřítele.

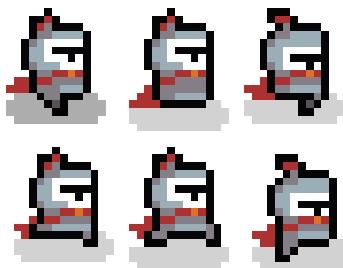
Kapitola 4

Implementace

Tato kapitola se podrobně věnuje popisu implementace všech klíčových herních mechanik a systémů, které byly vytvořeny pro hru. Každá podkapitola detailně rozebírá specifické aspekty a technické řešení jednotlivých mechanik.

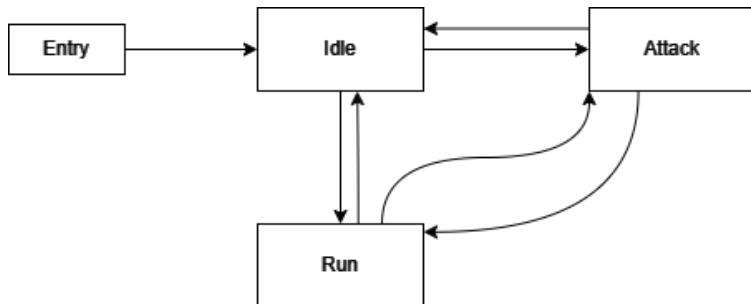
4.1 Animace

Animace postav a jejich akce, jako je sekání mečem, byly v herním engine Unity realizovány s využitím nástroje zvaného Animator Controller, který funguje jako centrální prvek pro správu a koordinaci různých animačních stavů. Tyto stavy zahrnují například chůzi, běh, nebo různé útoky, jako je zmíněné sekání mečem, či sesílání kouzel. Animator Controller umožňuje definovat podmínky pro přechody mezi jednotlivými stavami, jako je například zmáčknutí tlačítka uživatelem, nebo splnění podmínky pro zaútočení přímo ve hře.



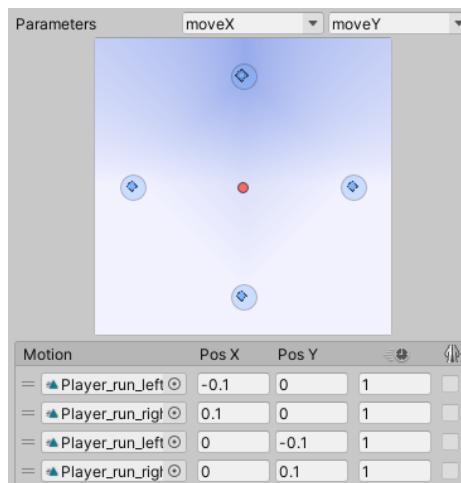
Obrázek 4.1: Všechny jednotlivé obrázky tvořící finální animaci běhu hráče.

Pro realizaci plynulých a realistických animací bylo nezbytné nejprve vytvořit klíčové snímky pro každý základní pohyb postavy (viz obrázek 4.1). Jednoduché animace jako je seknutí mečem byly vytvořeny přímo v Unity pomocí rotace zbraně, složitější animace, jako je běh postavy byly vytvořeny pomocí předem definovaných assetů, stažených z volně dostupných zdrojů, jako je například Unity Asset Store. Všechny potřebné stavy byly poté vloženy do Animator Controlleru. V tomto nástroji pak každý animační stav koresponduje s konkrétním pohybem nebo akcí postavy, a tvoří plynulé přechody mezi těmito stavy v reakci na herní události a vstupy od hráčů.



Obrázek 4.2: Stavy animací bosse v Animator Controlleru, při pohybu přechází do stavu Run, pokud je hráč ve vzdálenosti útoku, přechází z jakéhokoliv stavu do stavu Útoku, pokud je v klidu tak se nachází ve stavu Idle.

Dále je v Animator Controlleru využita funkce Blend Trees. Uvnitř Blend Tree mohou být přidány různé animace, které mají být mezi sebou kombinovány nebo přechodně smíchány. Každá animace je přitom spojena s konkrétními hodnotami parametrů, které určují, za jakých podmínek a v jaké míře bude daná animace aktivována.



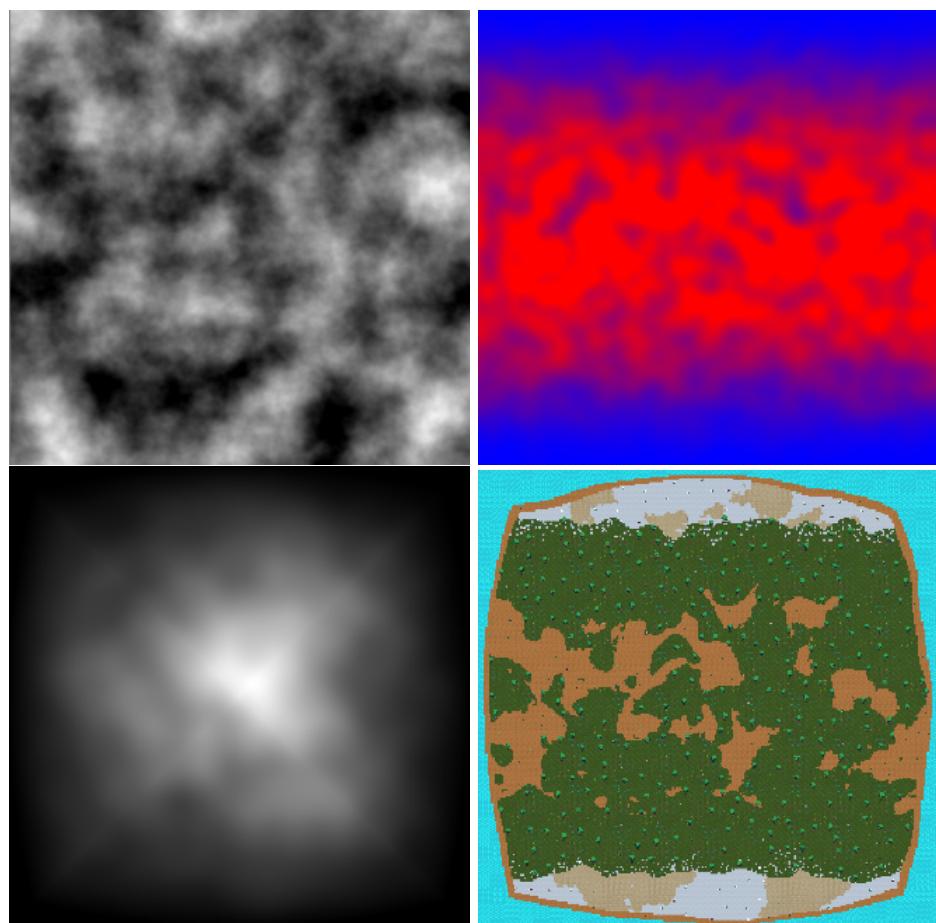
Obrázek 4.3: Blend Tree představující pohyb hráče. Stavy jednotlivých animací se mění na základě změny souřadnic (x, y) hráčovy postavy.

4.2 Generování mapy

Generování mapy ve hře je realizováno pomocí dvou rozdílných metod. První metoda využívá Perlinův šum 2.3 spolu s Whittakerovým diagramem 2.5 pro rozdělení biomů, zatímco druhá metoda je založena na principu celulárních automatů 2.4.

První typ mapy je sestaven z kombinace tří základních map: srážkové, výškové a tepelné. Tepelná mapa je navržena tak, aby co nejvíce odpovídala reálné teplotní mapě Země. Na pólech jsou simulovány nejnižší teploty, které postupně rostou směrem k rovníku. Pro dodání realistické náhodnosti a přirozeného vzhledu teplotních gradientů byl do mapy zakomponován Perlinův šum, který zabraňuje vzniku příliš ostrých přechodů mezi různými teplotními zónami.

Výšková mapa je vytvořena s cílem imitovat ostrovní geografii. Na okrajích mapy, kde se nachází voda, jsou nejnižší nadmořské výšky. Směrem ke středu mapy se výška postupně zvyšuje pomocí Perlinova šumu, což vytváří dojem centrálního pohoří nebo vyvýšeniny. Tento efekt umožňuje simulovat různé typy terénu od nížinných oblastí po horské vrcholy. Mapa srážek, vytvořená také pomocí Perlinova šumu, simuluje rozložení srážek v rámci generovaného světa hry. Při jejím vytváření se používá více oktav šumu, což zvyšuje komplexitu a detailnost mapy.



Obrázek 4.4: Mapa srážek, teplotní mapa, výšková mapa a finální mapa biomů.

Výsledná kombinace všech těchto map poté určuje biomy dle Whittakerova diagramu. Nízké oblasti představují vodní plochy, oblasti s nízkými teplotami a vysokými srážkami formují sněžné pláně, zatímco vysoké a chladné oblasti jsou prezentovány jako skalnaté hory. Pouště se nacházejí v oblastech s vysokými teplotami a nízkým počtem srážek, a nakonec, regiony s průměrnými hodnotami srážek a teplot tvoří lesy a louky (viz algoritmus 1).

Algoritmus 1: Výběr biomu na základě hodnot prostředí

Data: Výška, Srážky, Teplota, souřadnice x, souřadnice y

Result: Pole s odpovídajícím birom

1 Stanovení hodnot pro každý biom;

2 **Function** ChooseBiome(*výška, srážky, teplota, x, y*):

3 Oblasti s nejnižší výškou vyhodnoceny jako voda;

4 Teplé a suché oblasti vyhodnoceny jako poušt;

5 Studené oblasti jsou buď sníh nebo kamenné pláně, na základě počtu srážek;

6 Oblasti vedle vody jsou pláž;

7 Oblasti s průměrnou teplotou a počtem srážek jsou louky a lesy;

8 **return** Biom přiřazený k poli na souřadnicích x, y;

Druhá metoda generování mapy využívá zcela odlišný princip: celulární automaty. Tato metoda si zakládá na mřížce buněk, kde každá buňka může být buď průchodná (podlaha, 0) nebo nepropustná (zed, 1). Inicializace mapy začíná náhodným rozdělením těchto stavů s určitou pravděpodobností, což dává každé buňce šanci stát se stěnou.

Hlavní logika pro transformaci mapy je poté realizována prostřednictvím iterativního procesu, který aplikuje pravidla celulárního automatu po definovaný počet cyklů. Během každé iterace je stav každé buňky aktualizován na základě stavu jeho sousedů, což je určeno Mooreovým okolím o velikosti 3x3 buněk. To zahrnuje osm sousedů každé buňky.

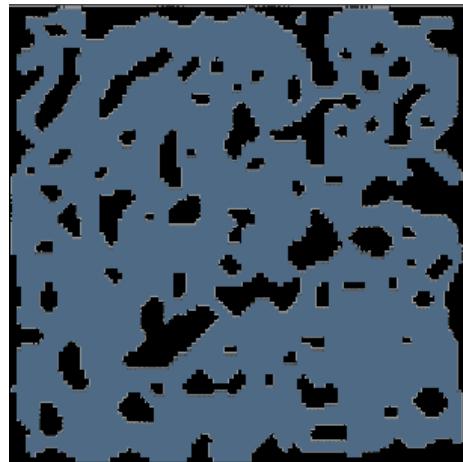
Pravidla pro změnu stávajících buněk jsou následující.

- Buňka se změní na stěnu (1), pokud počet jejích sousedních stěn překročí hodnotu „birthLim“¹.
- Buňka se stane průchodnou (0), pokud počet sousedních stěn je menší než hodnota „deathLim“².
- V případě, že počet sousedních stěn je mezi těmito dvěma prahy, stav buňky zůstává neměnný.

Tento dynamický proces (zobrazen na algoritmu 2) vytváří labyrint, připomínající strukturu opuštěných kobek (obrázek 4.5), které může hráč prozkoumávat.

¹V kódu je tato hodnota nastavena na 4

²V kódu je tato hodnota nastavena na 3



Obrázek 4.5: Vygenerovaná mapa kobky pomocí celulárních automatů.

Algoritmus 2: Algoritmus generování kobky celulárními automaty

Data: početIterací, birthLim, deathLim
Result: Vytvořená mapa kobky

```

1 Function GenerateDungeon(početIterací, birthLim, deathLim):
2   if existuje uložená mapa then
3     Načti danou mapu
4   else
5     Vygeneruj novou mapu kobky
6     Náhodně přiřaď stavu políčkům mapy
7     for i  $\leftarrow$  1 to počet iterací algoritmu do
8       foreach políčko na mapě do
9         Vypočítej počet sousedících stěn v dosahu daném Mooreovým okolím
10        if políčko je stěna then
11          if počet sousedících stěn  $<$  deathLim then
12            | Změn pole na podlahu
13          else
14            if počet sousedících stěn  $>$  birthLim then
15              | Změn pole na stěnu
16            end if
17          end foreach
18        end for
19        Polož políčka pro stěny a podlahy podle stavu finální mapy
20      end if
```

Proces generování objektů na mapě začíná vytvořením mapy objektů s využitím Perlinova šumu, který určuje potenciální lokality pro umístění objektů. Každý objekt má přiřazenou pravděpodobnost generování, která ovlivňuje, jak často se může objevit na mapě. Je zohledňován typ terénu a specifika lokality, aby bylo rozhodnuto, které objekty jsou pro danou oblast vhodné. Například v oblastech s písečným terénem mohou být generovány kaktusy, zatímco v lesních a travnatých oblastech mohou být umístěny různé typy stromů, keřů a balvanů. V zasněžených oblastech pak hra umísťuje objekty, které jsou vizuálně kon-

zistentní se zimním prostředím, jako jsou sněhové kry, protáhlé kmeny stromů nebo sněhové skály.



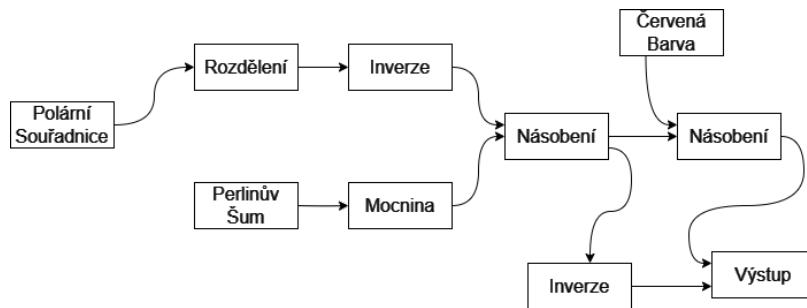
Obrázek 4.6: Příklady objektů nacházejících se na mapě.

4.3 Efekty

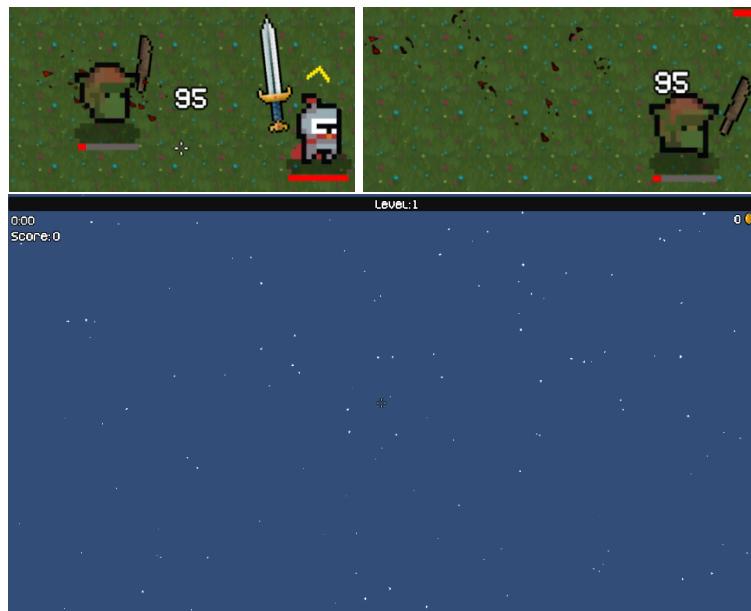
Efekty jsou ve hře realizovány pomocí dvou hlavních nástrojů, integrovaných v herním engine Unity. Jedná se o Unity částicový systém a Unity shader graph, kde každé přináší své unikátní možnosti pro vytváření vizuálních efektů.

Unity částicový systém je využíván pro scény, kde je potřeba efektivně generovat a spravovat velké množství častic, což je ideální pro vizuální efekty jako jsou exploze, oheň nebo simulace přírodních jevů. Díky své flexibilitě umožňuje částicový systém detailní nastavení atributů častic, jako je tvar, velikost, životnost a pohybové trajektorie. Například, efekt padání listů je realizován tak, že každý list je reprezentován jako částice s individuálně nastavenou trajektorií a rotací, což vede k realistické simulaci pohybu listů ve vzduchu. Stejným způsobem jsou dále implementovány efekty padajícího sněhu v zimním biomu, nebo smrt hráčovy postavy, při které se částice krve rozstříknou do různých směru v případě dosažení nulových životů.

Na druhou stranu, Unity shader graph je používán pro složitější vizuální efekty, kde je požadována větší míra kontroly nad vizuálními aspekty a texturami. Shader graph umožňuje vývojářům vizuálně sestavovat složité shadery bez nutnosti psát jejich kód, což je klíčové pro rychlé experimentování a implementaci efektů jako jsou vodní efekty, komplexní osvětlení nebo realistické texturování povrchů. Například, efekt úderu nepřítele využívá shader graph k simulaci náhodně generovaného šumu pro každou kapku krve, což zajišťuje, že každý detail vypadá unikátně a realisticky.



Obrázek 4.7: Animace krve po udeření nepřítele, realizované v Unity shader graph.



Obrázek 4.8: Efekt krve při udeření nepřítele a efekt sněžení, který se aktivuje v zimním biomu.

4.4 Souboj

Souboj ve hře je rozdělen do tří základních typů: úder od hráče, úder od nepřítele a kouzla na dálku. Mechanika blízkého boje pro hráče i nepřítele je založena na animaci, kde je zbraň pootočena o několik stupňů a seřazena za sebou v několika setinovém úseku, což vytváří dojem plynulého pohybu zbraně. V polovině této animace je aktivována metoda útoku, která vyhodnocuje, zda se v „hitboxu“ úderu nachází cíl. Pokud ano, úder zasáhne cíl, což vede k okamžitému ubrání životů závislého na množství způsobeného poškození, odhození nepřítele dozadu a vygenerování vizuálního efektu úderu. Tento způsob řešení boje přidává hře úroveň obtížnosti a dodává boji realističtější pocit, kdy hráč musí zvažovat časování svých úderů.

Stejným způsobem jsou zpracovány útoky nepřátel. Ty jsou iniciovány automaticky, nikoliv manuálním stiskem tlačítka jako u hráče. Neprítel zaútočí, pokud je hráč v dosahu a systém detekuje, že je možné útočit.

Kouzla na dálku, která jsou třetím typem útoku, fungují odlišně. Jsou aktivována, když se hráč dostane do specifické vzdálenosti, kterou nepřítel může pokrýt svým útokem. Po aktivaci animace kouzla, jako je například vyvolání ohnivé koule nebo koule slizu, je projektil vystřelen směrem k hráči (viz algoritmus 3). Pokud hráče projektil zasáhne, je mu uděleno odpovídající poškození.



Obrázek 4.9: Celková animace útoku, složená z jednotlivých částí úderu.

Algoritmus 3: Metoda pro detekci hráče a zahájení útoku

Result: Kontroluje, zda se hráč nachází v oblasti útoku a zahajuje animaci útoku

```
1 Function Update():
2     foreach collider v colliderech nacházejících se v oblasti útoku nepřítele do
3         if collider má tag Player then
4             | Attack();
5         end if
6     end foreach
7 Function Attack():
8     if nepřítel neútočí na dálku then
9         | v animatoru nastav proměnou attack na true;
10    else
11        | vytvoř projektil a vystřel jej na místo nalezeného collideru;
12    end if
```

Systém, který rozhoduje, komu útoky mohou způsobit poškození, je založen na využití tagů v Unity. Každý nepřítel má přiřazený tag³ „Enemy“, zatímco hráč a vystřelené projektily mají vlastní unikátní tagy. Toto řešení zabraňuje nechtěným kolizím, jako je poškození samotného vyvolavatele útoku nebo vzájemné poškození mezi nepřáteli.

Po zásahu postavy je vizualizována hodnota způsobeného poškození, která je prezentována prostřednictvím numerické indikace. V případě, že úder je proveden hráčem, indikované poškození je reprezentováno bílou barvou, zatímco poškození způsobené nepřítelem je vyjádřeno červenou. Vzhledem k potenciálně vysokému počtu nepřátel a možnému poškození několika postav najednou jedním útokem je tato vizuální odezva implementována s využitím poolingu číselných hodnot. Tento přístup je podrobněji analyzován a popsán v sekci věnované optimalizaci herního výkonu (viz 4.10).



Obrázek 4.10: Čísla indikující udělené poškození. Šipka nad hráčem je jedno z pasivních vylepšení 4.12.

³Neboli štítek, v Unity se můžou rozdělovat typy objektů podle tagů.

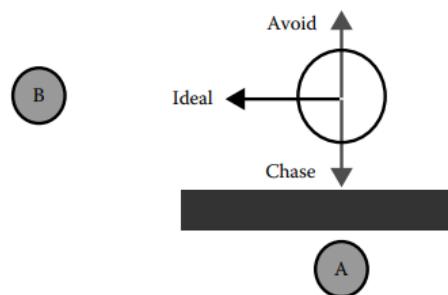
4.5 Pohyb nepřátele

Pohyb nepřátele ve hře je řízen metodou zvanou „context steering“. Tento přístup umožňuje nepřátelům analyzovat a interpretovat objekty v jejich bezprostředním okolí a na základě toho určovat nejefektivnější cestu k jejich cíli, což ve většině případů představuje hráčovu postavu.

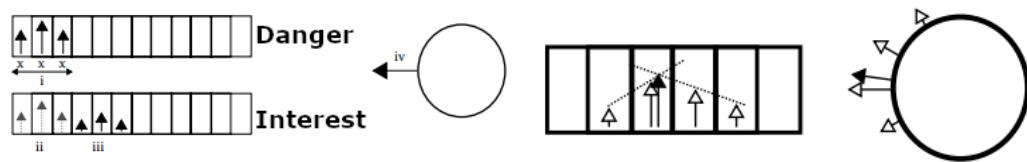
Při implementaci tohoto systému jsou všechny objekty v okolí rozdeleny do dvou hlavních kategorií: překážky a cíle. Překážky jsou definovány jako všechny elementy prostředí, které mohou fyzicky zablokovat cestu nepřítele a tím mu zabránit v dosažení jeho cíle. Na druhou stranu, cíle zahrnují všechny objekty, ke kterým má nepřítel zájem se přiblížit co nejkratší cestou — v našem případě je to pouze postava hráče.

Systém pro efektivní navigaci a rozhodování nepřátele vytváří pro každou skupinu objektů samostatné pole vektorů. Toto pole obsahuje osm vektorů, které pokrývají možné směry pohybu nepřítele. Intenzita každého vektoru je určena na základě relativní vzdálosti objektu od nepřítele – čím je objekt blíže, tím má příslušný vektor vyšší hodnotu. Výsledné směry pohybu se pak určují odečtením vektorových hodnot překážek od hodnot vektorů cílů, což vede k vytvoření finálního pole směrů pohybu.

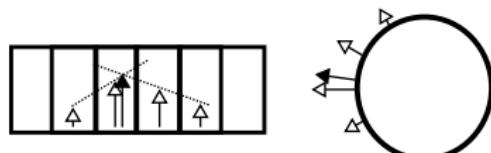
Následně algoritmus interpoluje nejvhodnější směr pohybu z tohoto pole, aby nepřítel mohl efektivně navigovat kolem překážek a úspěšně dosáhnout svého cíle (viz algoritmus 4).



(a) Vizualizace vektorů překážek a možných cílů.



(b) Pole překážek a možných cílů.



(c) Finální pole vektorů ideální cesty.

Obrázek 4.11: Context steering - proces vybíráni ideálního směru pohybu (Obrázky převzaty z [2]).

K dalšímu zlepšení schopností AI⁴ byla do systému implementována funkce sledování poslední viděné pozice hráče. Pokud hráč zmizí z dohledu nepřítele například tím, že se skryje za překážku, AI nezastaví své pohyby, ale místo toho se vydá k poslední známé

⁴Artificial Intelligence – česky umělá inteligence.

pozici hráče. Při dosažení této lokace pak nepřítel pokračuje ve „vyhledávacím režimu“, kde se pokouší znovu lokalizovat hráče.

V případě, že nepřítel ztratí hráče z dohledu a není schopen ho po určitou dobu najít, přechází do pasivního stavu. V tomto režimu může buď pokračovat v hledání, pokud hráče opět zaznamená, nebo po delší době nečinnosti a vzdálenosti od hráče dojde k nastavení neaktivnosti objektu. Tento mechanismus pomáhá optimalizovat výkon hry tím, že redukuje počet aktivních entit.

Algoritmus 4: Context Steering vyhodnocování cesty

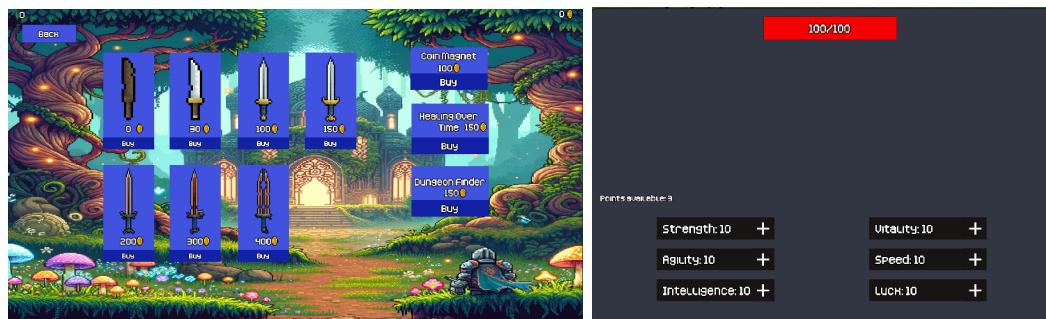
Result: Vypočte ideální směr pohybu nepřítele na základě okolních překážek a cílů

```
1 foreach Nepřítel nacházející se ve scéně do
2   while Nepřítel je aktivní do
3     Detekuj možné cíle a překážky v jeho okolí;
4     Vytvoř pole možných cílů a pole překážek;
5     Vypočítej vzdálenost všech možných cílů a překážek v 8 zadaných směrech a
       ulož je do polí;
6     Od pole cílů odečti pole překážek;
7     Vyber směr pohybu pomocí nejvyššího vektoru v poli možných cílů;
8     Aplikuj vybraný směr na nepřátelský pohyb;
9   end while
10 end foreach
```

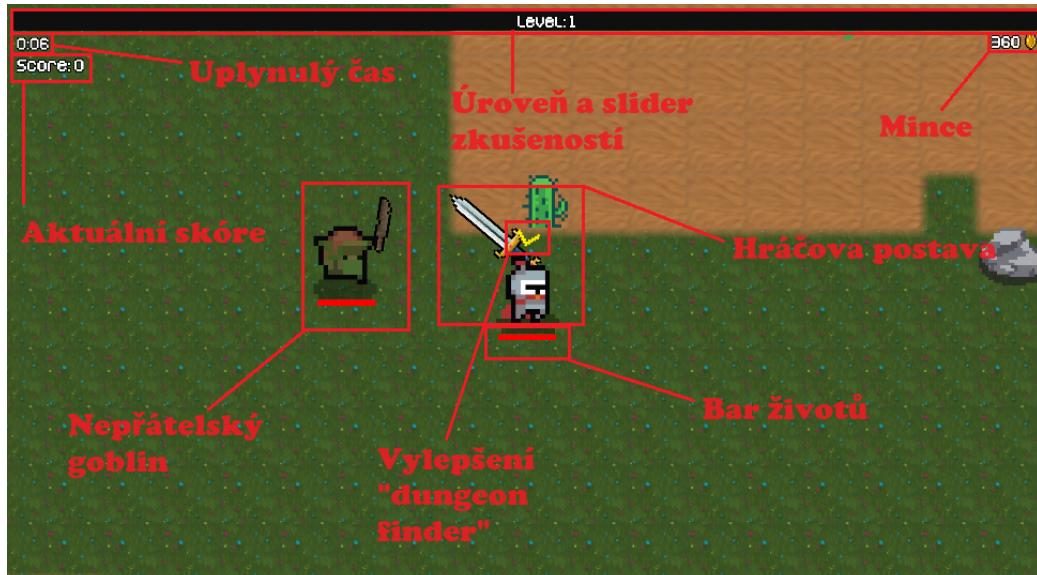
4.6 Uživatelské rozhraní

Uživatelské rozhraní (angl. User Interface, UI) ve hře je vytvořeno s použitím jednoduchých grafických prvků, jako jsou tlačítka a posuvníky (slidery), které jsou implementovány v herním engine Unity. Aktualizace klíčových informací na UI, jako jsou skóre hráče, jeho aktuální úroveň nebo množství nasbíraných minců, je zajištěna pomocí specifických částí kódu, které jsou přímo odpovědné za správu těchto dat.

Jedním z významných a zajímavých prvků uživatelského rozhraní je obchod umístěný v hlavním menu hry. Tento obchod využívá funkcionality Unity nazývanou Player Prefs (podrobněji popsanou v sekci systému ukládání 4.7), která umožňuje dlouhodobé ukládání vybraných herních statistik a preferencí hráče, což zahrnuje informace o již zakoupených předmětech. Díky tomu má hráč možnost v obchodě snadno identifikovat, které předměty již vlastní a může je podle toho nasazovat přímo v uživatelském rozhraní.



(a) Obchod s předměty v hlavním menu, které si hráč může nakupovat za nasbírané mince. (b) UI vylepšování postavy za body atri- hráč může nakupovat za nasbírané mince. butů, získané zabíjením nepřátel.



(c) Detailněji popsané UI, které hráče doprovází většinu hry.

Obrázek 4.12: Příklady implementovaného herního UI.

4.7 Systém ukládání

Ve hře je pro přechod mezi procedurálně generovanými mapami zásadní ukládání a načítání stavu map a hráčových dat. K tomuto účelu byly implementovány dvě hlavní metody: ukládání do JSON souborů pro data map a využití Unity Player Prefs pro osobní data hráče.

Ukládání map do JSON souborů

Každá mapa ve hře, po svém procedurálním vygenerování, je kompletně uložena do JSON souboru. Tento soubor obsahuje strukturované údaje, které reprezentují všechny aspekty mapy – od políček s jejich typy a pozicemi až po nepřátele s jejich životy, poškozením a specifickými pozicemi (viz algoritmus 5). Tyto soubory jsou umístěny v lokálním úložišti zařízení, obvykle v adresáři určeném pro aplikace, kde je každý soubor pojmenován podle unikátní identifikace mapy. Při načítání mapy systém vyhledá odpovídající JSON soubor a rekonstruuje mapu do jejího původního stavu bez potřeby opětovného generování (viz algoritmus 6) za využití šumových funkcí, které jsou náročně na paměť a využití procesoru.

Algoritmus 5: Ukládání map a objektů do JSON souboru

Data: Aktuální stav hry, zahrnující mapu a herní objekty
Result: JSON soubor representující stav hry

```
1 foreach políčko na mapě do
2   | ulož pozici políčka a jeho typ;
3   | vlož jej do listu uložených polí;
4 end foreach
5 foreach aktivní nepřítel na mapě do
6   | ulož jeho pozici a statistiky (životy, poškození);
7   | vlož jej do listu uložených objektů;
8 end foreach
9 foreach aktivní prop na mapě do
10  | ulož jeho pozici a typ objektu;
11  | vlož jej do listu uložených propů;
12 end foreach
13 změn listy na formát JSON;
14 ulož JSON na disk pod názvem dané mapy;
```

Algoritmus 6: Načítání map a objektů z JSON souboru

Data: JSON soubor pojmenovaný po načítané mapě
Result: Načtená mapa ve stavu, v jakém jí hráč opustil

```
1 if existuje soubor pro mapu then
2   přečti JSON soubor;
3   změn JSON zpět na list;
4   vyčisti aktuální mapu;
5   foreach uložené políčko v listu do
6     | polož políčko na uloženou pozici s jeho uložených typem;
7   end foreach
8   foreach uložený herní objekt v listu do
9     | vytvoř herní objekt na uložené pozici s danými atributy;
10  end foreach
11 end if
```

Využití Unity Player Prefs pro ukládání dat hráče

Unity Player Prefs je využíván pro ukládání menšího množství dat, která jsou specifická pro hráče, jako jsou životy, úroveň, nasbírané mince a body, a další herní statistiky. Tento systém ukládá data ve formátu klíč-hodnota přímo do registru operačního systému nebo do souborů specifických pro platformu, což zajišťuje jejich trvalost mezi herními sezeními. Při každém spuštění hry systém načítá tato data, což hráči umožňuje pokračovat přesně tam, kde hru naposledy ukončil. Player Prefs poskytuje jednoduché rozhraní pro ukládání a načítání těchto hodnot, což značně zjednoduší správu herních nastavení a postupu.

4.8 Objevování nepřátele

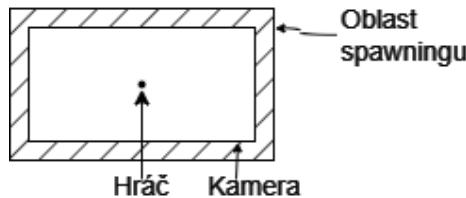
Ve hře je objevování (angl. spawning) nepřátele řízeno tak, aby zlepšilo interaktivitu a udrželo dynamické a náročné herní prostředí. Proces začíná určením velikosti kamery hráče, což je klíčové pro zajištění, že se nepřátelé objeví právě mimo okamžitý výhled hráče, čímž se přidává prvek překvapení a strategie. Systém vypočítá hrany zorného pole kamery a vybere náhodné místo pro objevení nepřítele na základě několika kritérií.

Za prvé, vybrané místo je kontrolováno, zda je platné v rámci herní mapy – nesmí být mimo hratelnou oblast nebo obsazeno objekty s vyšší prioritou, jako jsou zdi nebo významné prvky prostředí, jako jsou stromy. Tím se zajišťuje, že spawning nenarušuje logiku hry nebo navigaci hráče.

Za druhé, na základě biomu hráče – specifické environmentální zóny ve hře – jsou generováni konkrétní nepřátelé. Například v pouštních oblastech se objevují píseční goblini, zatímco v podzemí se objevují duchové, což sladí typ nepřítele s tématickými prvky biomu pro ponoření do hry.

Skript toto zvládá prostřednictvím systému, kde různé herní objekty nepřátel jsou instancovány na základě typu terénního políčka pod pozicí hráče. Komponenta Tilemap⁵ se používá k určení konkrétního políčka na pozici hráče, což poté spouští spawning nepřátele specifických pro daný biom. Tato metoda nejen kontextualizuje setkání s nepřáteli, ale také

⁵Komponenta Unity obsahující kompletní mapu s jednotlivými políčky.



Obrázek 4.13: Oblast, ve které se můžou objevit nepřátelé.

dodává vrstvu složitosti do strategie hráče, protože různí nepřátelé mohou vyžadovat různé přístupy.

Spawning nepřátel je regulován časovačem, který zajišťuje, že se nepřátelé objevují ve zvládnutelných intervalech, čímž udržuje hru vyváženou (viz algoritmus 7). Skript také zahrnuje funkci deaktivace a poolingu nepřátel, kteří se dostanou mimo dosah hráče. Tato metoda optimalizuje výkon tím, že znova používá herní objekty, místo vytváření nových, címž napomáhá v udržení výkonu hry.

Algoritmus 7: Spawning nepřátel

Result: Spawn nepřátel na herní mapě

```

1 inicializace seznamu nepřátel;
2 while hra běží do
3   čekání do nastaveného času pro spawn;
4   výběr náhodné pozice na mapě;
5   if pozice je vhodná then
6     vytvoření instance nepřítele na vybrané pozici;
7     přidání nepřítele do seznamu aktivních nepřátel;
8   else
9     hledání nové pozice;
10  end if
11 end while

```

4.9 Audio

Herní engine Unity umožňuje snadné přidávání a konfiguraci zvukových souborů prostřednictvím komponenty AudioSource. Tato komponenta slouží jako médium pro přehrávání zvuků ve hře a může být přiřazena k jakémukoli objektu v herní scéně. AudioSource nabízí řadu parametrů pro ovládání vlastností zvuku, jako je hlasitost, smyčka (loop), vzdálenostní dosah a směr, což umožňuje detailní nastavení zvukové stopy přizpůsobené konkrétním potřebám scény nebo akce. Kromě toho, zvukové efekty jsou aplikovány také při švihnutí zbraněmi a při zásahu nepřátel, což přispívá k realističnosti a dynamice herního prostředí.

Hudba ve hře je dynamicky řízena na základě kontextu a situace, ve které se hráč nachází. Například, každá lokace, jako je hlavní menu nebo klasický ostrov, má svůj vlastní hudební motiv. Speciální události, jako je objevení bosse nebo ukončení hry smrtí hráče, též aktivují hudební skladby. Tato hudba se automaticky spouští a zastavuje pomocí skriptů, které detekují vstup hráče do příslušné oblasti nebo změnu herního stavu.

4.10 Optimalizace

Tato sekce se zaměřuje na využité techniky pro optimalizaci běhu hry a paměti, zároveň jsou zmíněny možnosti další optimalizace, které nebyly realizovány.

Enemy Pooling

Tato metoda zahrnuje vytvoření předem definovaného počtu instancí nepřátel, které jsou uchovávány ve skladu, známém jako pool. Na začátku hry, nebo při načítání herní scény, systém inicializuje tyto instance a udržuje je v neaktivním stavu, připravené k použití.

Když hra vyžaduje vložení nového nepřítele do herního světa, místo aby vytvářela novou instanci, skript jednoduše reaktivuje instanci z poolu. Tato instance je poté umístěna na požadovanou pozici ve hře a jsou nastaveny její atributy, jako jsou životy, což odpovídá aktuálním požadavkům herního scénáře. Po ukončení existence nepřítele ve hře, například po jeho smrti nebo když se hráč vzdálí dostatečně daleko, instance nepřítele není zničena. Namísto toho je deaktivována a vrácena zpět do poolu pro opětovné použití (viz algoritmus 8).

Algoritmus 8: Pooling nepřátele

Result: Recyklace nepřátele pomocí pooling systému

```
1 inicializace poolu pro každý typ nepřítele;
2 while hra běží do
3   if nepřítel je poražen nebo mimo dosah then
4     deaktivace nepřítele;
5     vrácení nepřítele do příslušného poolu;
6   else
7     if je potřeba spawnout nového nepřítele then
8       if pool obsahuje dostupné instance then
9         aktivace instance nepřítele z poolu;
10        umístění na herní mapě;
11      else
12        vytvoření nové instance nepřítele;
13        přidání do poolu pro budoucí použití;
14      end if
15    end if
16  end if
17 end while
```

Tato technika nabízí několik významných výhod. Nejprve značně snižuje počet operací vytváření a ničení objektů, což jsou procesy, jež jsou náročné na procesor a paměť. Tím se minimalizuje riziko vzniku zpoždění, které by mohlo negativně ovlivnit uživatelskou zkušenosť během rychlých a intenzivních momentů hry. Dále, pooling umožňuje hře efektivněji spravovat paměť, protože množství objektů v paměti je konstantní a recyklovatelné. Tato konzistence pomáhá předcházet problémům s fragmentací paměti, která může nastat, když objekty neustále vznikají a zanikají.

Praktická implementace pooling systému často vyžaduje, aby byly spravovány různé pooly pro různé typy nepřátele nebo herní objekty, což umožňuje optimalizovat chování

a správu každé entity nezávisle. Například může existovat samostatný pool pro lesní gobliny a jiný pro slimáky.

Despawn nepřátele

Jak bylo zmíněno v sekci 4.5, systém sleduje pozici nepřátele relativně k pozici hráče, aby se minimalizovala výpočetní náročnost a paměťové požadavky hry. Nepřátelé, kteří se nacházejí mimo určitý definovaný dosah od hráče a nejsou viditelní ani přístupní, jsou automaticky deaktivováni a uloženi zpět do poolu. Tento proces zabírá zbytečnému zpracování dat a renderování objektů, které nejsou pro hráče aktuálně relevantní. Deaktivace a pooling nepřátele tedy zajišťuje, že zdroje počítače jsou využívány efektivněji, a to umožňuje hladší průběh hry a lepší reakční časy.

Výhled na další optimalizaci

Jedním z navrhovaných směrů pro další optimalizaci je implementace dynamického načítání předem generovaných segmentů mapy. Vzhledem k rozsahu map ve hře by tento přístup mohl významně přispět k redukci paměťové náročnosti a zefektivnění celkového výkonu aplikace. Metoda dynamického načítání by fungovala tak, že by se načítaly pouze ty části mapy, které jsou v bezprostřední blízkosti hráče, zatímco ostatní segmenty by byly dočasně deaktivovány nebo vyloženy z operační paměti. Jakmile by hráč pohyboval svou postavou do nových oblastí mapy, příslušné segmenty by se automaticky načítaly zpět do paměti. Tento model by nejenom snížil okamžitou náročnost na paměť, ale také by umožnil větší rozsah a detailnost herních map bez negativního vlivu na rychlosť a plynulost hry.

Kapitola 5

Závěr

Tato bakalářská práce podrobně prezentuje proces návrhu a implementace 2D rogue-like hry (příklad obrázků ze hry viz 5.1), která využívá procedurální generování a řízení pohybu, známé jako context steering, pro dynamické ovládání neprátelských jednotek. Hlavním cílem bylo vyvinout hru, která využívá náhodně generované mapy a další prvky z rogue-like her, aby poskytovala stále se měnící herní prostředí.

Během vývoje byla kladená značná důraz na integraci rozšířených herních mechanik, včetně soubojového systému a animací, které přispívají k celkové poutavosti a hratelnosti hry. Pro implementaci byl vybrán herní engine Unity, díky jeho schopnosti efektivně zpracovávat grafické a fyzikální aspekty herních objektů, což umožnilo rychlý vývoj.

Projekt nezahrnoval testování s reálnými uživateli, ale byl interně ověřen s cílem zajištění funkčnosti a plynulosti běhu hry. Tento interní testovací proces umožnil identifikaci a opravu technických chyb, což vedlo k vylepšení výkonu a stabilitě hry.

Výsledná hra nabízí mnoho herních prvků, včetně různorodých nepřátel, zbraní, biomů a bossů, což položilo základy pro další rozvoj a potenciální rozšíření hry. Další vývoj by měl zahrnovat přidání nových map, zbraní a herních prvků na již vytvořených základech v rámci této práce. Prioritou by mělo být dále rozšíření typů nepřátel a bossů, což by poskytlo hráčům větší variabilitu a nové výzvy. Taktéž by bylo vhodné zvážit implementaci dalších vylepšení a pasivních schopností, které by hráči mohli využít k optimalizaci svého herního stylu a strategie.



(a) Hlavní menu.

(b) Souboj s nepřáteli.

Obrázek 5.1: Obrázky z finální verze hry.

Literatura

- [1] COOK, R. Wavelet Noise. *ACM Transactions on Graphics*. Tony DeRose. Červenec 2005, sv. 24, č. 3, s. 803–811.
- [2] FRAY, A. Context steering: Behavior-driven steering at the macro scale. In: *Game AI Pro 360: Guide to Movement and Pathfinding*. CRC Press, 2019, s. 147–158.
- [3] LAGAE, A. Procedural noise using sparse Gabor convolution. *SIGGRAPH '09*. Sylvain Lefebvre, George Drettakis, Philip Dutré. Červenec 2009, s. 1–10.
- [4] PERLIN, K. An Image Synthesizer. *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*. Červenec 1985, sv. 19, s. 287–296.
- [5] PERLIN, K. Hypertexture. *Computer Graphics*. Hoffert, E. M. Červenec 1989, sv. 23, s. 253–262.
- [6] TOGELIUS, J. *Procedural content generation in games*. 1. vyd. Noor Shaker, Mark J. Nelson. Springer Cham, 2016. ISBN 978-3-319-42716-4.
- [7] VON NEUMANN, J., BURKS, A. W. et al. Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*. 1966, sv. 5, č. 1, s. 3–14.
- [8] WOLFRAM, S. *A new kind of science*. 1. vyd. Wolfram Media, 2002. ISBN 9781579550080.