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

ROGUELIKE HRA V UNITY

ROGUELIKE GAME IN UNITY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ADRIAN MANČUŠKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL VLNAS
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

146168

Ústav: Ústav počítačové grafiky a multimédií (UPGM)
Student: **Mančuška Adrian**
Program: Informační technologie
Specializace: Informační technologie

Název: **Roguelike hra v Unity**
Kategorie: Počítačová grafika
Akademický rok: 2022/23

Zadání:

1. Nastudujte techniky herního vývoje a prostředí Unity.
2. Navrhněte hru žánru roguelike s vhodnými herními mechanikami.
3. Implementujte navrženou hru.
4. Zhodnoťte dosažené výsledky a též možnosti publikace hry.
5. Vytvořte demonstrační video.

Literatura:

- Gregory, Jason. *Game engine architecture*. crc Press, 2018. ISBN 1351974289, 9781351974288
- Bishop, Lars, et al. "Designing a PC game engine." IEEE Computer Graphics and Applications 18.1 (1998): 46-53.
- Adams, Ernest, and Joris Dormans. Game mechanics: advanced game design. New Riders, 2012. ISBN 0321820274, 9780321820273
- Koster, Raph. Theory of fun for game design. O'Reilly Media, Inc., 2013.
- Schell, Jesse. The Art of Game Design: A book of lenses. CRC press, 2008.
- Unity Learn. Unity, https://learn.unity.com/.

Při obhajobě semestrální části projektu je požadováno:
Body 1 a 2 a funkční prototyp vedoucí k bodu 3.

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

Vedoucí práce: **Vlnas Michal, Ing.**
Vedoucí ústavu: Černocký Jan, prof. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 10.5.2023
Datum schválení: 31.10.2022

Abstrakt

Táto bakalárska práca sa zaoberá vývojom roguelike hry s viacerými generovanými úrovňami v hernom engine Unity. Práca preskúmava rôzne možnosti použitia procedurálneho generovania, medzi ktoré patria napríklad rôzne šumy, gramatiky a celulárne automaty. Zároveň sú skúmané aj možnosti kombinovania týchto techník s manuálnym vytváraním obsahu hier. Zaoberá sa aj inými oblasťami potrebnými pre vývoj hry, ako sú pseudonáhodné generátory a umelá inteligencia. V oblasti umelej inteligencie sa práca venuje niektorým možnostiam jej implementácie. Týmito možnosťami sú konečné stavové automaty a tzv. behaviour trees. Potrebné sú aj algoritmy na hľadanie cesty, napríklad algoritmus A*. Z návrhového hľadiska je dôraz kladený na splnenie rôznych vlastností a požiadaviek roguelike žánru hier.

Abstract

This bachelor's thesis deals with the development of a roguelike game with multiple generated levels using the game engine Unity. The thesis explores various uses of procedural generation, which include various noises, grammars, and cellular automata. At the same time, possibilities of combining these techniques with manual game content creation are also explored. This thesis also deals with other areas necessary for game development, such as pseudorandom generators and artificial intelligence. The thesis discusses some of the possibilities of implementation of artificial intelligence. These possibilities are finite state machines and behaviour trees. Pathfinding algorithms are needed too, for example the A* algorithm. From design standpoint, the focus is on fulfilling various characteristics and requirements of the roguelike genre of games.

Kľúčové slová

vývoj hier, roguelike žáner, Unity engine, procedurálne generovanie, umelá inteligencia, A* algoritmus, pseudonáhodné generátory, raycasting

Keywords

game development, roguelike genre, Unity engine, procedural generation, artificial intelligence, A* algorithm, pseudorandom generators, raycasting

Citácia

MANČUŠKA, Adrian. *Roguelike hra v Unity.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Vlnas

Roguelike hra v Unity

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Michala Vlnasa. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

. . . . . .
Adrian Mančuška
3. mája 2023

Poďakovanie

Ďakujem vedúcemu bakalárskej práce Ing. Michalovi Vlnasovi za kvalitné odborné vedenie, pripomienky a užitočné rady, ktoré mi poskytol pri vypracovávaní tejto práce.

Obsah

Kapitola 1

Úvod

Pre celé generácie ľudí hra bola a aj naďalej je súčasťou ich života. Je to záujmová voľnočasová aktivita, ktorá ľuďom v každom veku prináša radosť. Prostredníctvom hry si človek, dieťa ale aj dospelý, môže oddýchnuť, zabaviť sa, relaxovať, ale zároveň získať aj nové vedomosti a zručnosti.

V dnešnej dobe sa ľudia každodenne stretávajú s informačnými technológiami, ktoré sú neodmysliteľnou súčasťou ich života. Počítače ľudia nevyužívajú už len na štúdium a prácu, ale aj na trávenie voľného času. Významným prostriedkom na trávenie voľného času pri počítači sú predovšetkým počítačové hry, ktoré sa svojim spôsobom stali fenoménom dnešnej doby. V porovnaní s minulosťou si už v dnešnej dobe ľudia môžu vybrať medzi klasickými hrami alebo digitálnymi hrami. Hrávajú ich nielen mladí, ale aj starší, ktorí takto relaxujú. Počítačové hry sa stali významnou súčasťou zábavného trhu. Je možné ich považovať za súčasť našej kultúry, presne tak ako knihy, filmy alebo hudbu.

Hry svojim unikátnym spôsobom ľudí vedia zabaviť tak, ako to iné média nedokážu. Tvorcovia počítačových hier neustále vytvárajú nové koncepty, svety a príbehy. Pri vývoji hier sa nejedná len o samotné programovanie, ale je dôležitá aj spolupráca s dizajnérmi, animátormi, zvukármi a inými profesiami. Vytvorenie úspešnej hry nie je jednoduché, vyžaduje to mnoho úsilia, talentu a šikovnosti.

Cieľom bakalárskej práce bolo navrhnúť a vytvoriť roguelike hru v hernom engine Unity. Návrh hry spĺňa rôzne existujúce vlastnosti a požiadavky roguelike žánru. Súčasne bolo pre vytvorenie tejto hry nutné preskúmať rôzne možnosti procedurálneho generovania a vhodne ich využiť.

Obsahovú štruktúru bakalárskej práce tvorí 5 kapitol, ktoré sú pre prehľadnosť ďalej členené. V druhej kapitole je obsiahnutý stručný prehľad o vývoji hier, história roguelike žánru hier a vysvetlenie základných teoretických pojmov potrebných pri vývoji hry. Tretia kapitola je obsahovo zameraná na vytvorenie návrhu a konceptu hry. Táto kapitola obsahuje aj návrhy užívateľského rozhrania. Štvrtá kapitola sa venuje konkrétnej implementácii navrhnutej hry a užívateľskému testovaniu. Poslednou kapitolou je záver, v ktorom je zhrnutie výsledkov práce a možné ďalšie rozšírenia v oblasti vývoja hry.

Kapitola 2

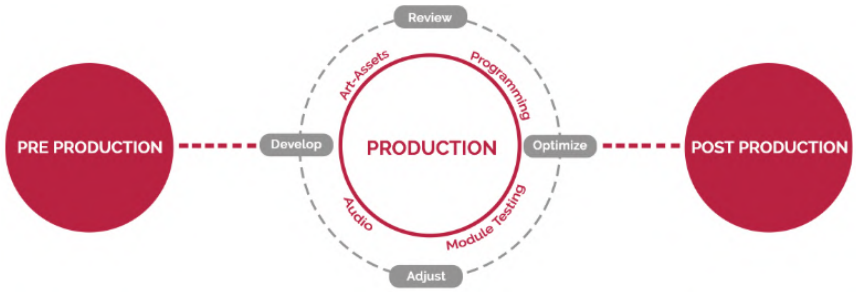
Teoretická časť

Pod pojmom počítačová hra (alebo všeobecne videohra) je možné si predstaviť interaktívny softvér komunikujúci s používateľom pomocou 2D alebo 3D grafiky a zvukov, ktorý je určený na účely zábavy, vzdelávania alebo simulácie. Existujú rôzne herné platformy, na ktorých sa hrajú videohry. Medzi tieto platformy sa zaraďujú napr. osobné počítače, konzoly, prenosné herné zariadenia (tzv. handhelds) a mobilné telefóny [27]. Postupným vývojom sa hry stali významnou kultúrnou a ekonomickou silou v modernom svete. Približne v poslednom desaťročí zisky videohier výrazne vzrástli. Podľa spoločnosti *NPD Group*, ktorá sa zaoberá marketingovým výskumom, dosiahli v USA v roku 2009 takmer 20 miliárd dolárov [31], pričom za rok 2022 táto suma dosiahla hodnotu 56,6 miliárd dolárov [3]. Počítačové hry ovplyvňujú aj iné formy médií, napr. film, hudbu, literatúru alebo umenie.

Videohry sa postupne rozčlenili na rôzne žánre a subžánre. Žánre videohier sa rozdeľujú podľa iných kritérií ako napr. žánre kníh alebo filmov. Žánre vo videohrách sú založené primárne na hrateľnosti a hernej interakcii na rozdiel od žánrov kníh alebo filmov, ktoré sa členia hlavne podľa deja alebo typu fiktívneho sveta. Videohry je problematické jednoznačne zaradiť do konkrétneho žánru nakoľko vo väčšine prípadov sú využité prvky viacerých žánrov. Delenie hier na jednotlivé žánre a subžánre je viac-menej subjektívne. Klasickým delením je možné hry rozdeliť napr. na akčné, FPS (First-Person Shooter, tzv. „strielačky“), dobrodružné, RPG (role-playing game – tzv. „hranie rolí“), arkádové, simulácie, stratégie, športové, puzzle, atď. [1].

2.1 Vývoj hier

Vývoj videohry je definovaný ako proces tvorby od konceptu až po dokončenie. Celý tento proces vývoja zahŕňa rôzne fázy. K týmto fázam patria predprodukcia, produkcia a postprodukcia [31] (viď obrázok 2.1). Jednotlivé fázy vývoja majú svoje vlastné úlohy a ciele. Celkový vývojový proces sa môže líšiť na základe žánru, rozsahu a rozpočtu videohry. Proces je ovplyvnený aj veľkosťou a štruktúrou vývojového tímu.



Obr. 2.1: Vizualizácia fáz vývoja videohier[1].

Predprodukčná fáza vývoja

V prvej fáze vývoja je nutné vytvoriť koncept hry – definuje sa hlavná myšlienka hry. Po schválení je tento koncept dizajnérom ďalej rozvíjaný. Pri tvorbe tohto rozšíreného návrhu je použitá spätná väzba viacerých členov tímu alebo aj údaje z prieskumov trhu. V rámci tejto fázy sa vyberá z viacerých návrhov, pričom je vybraný ten, čo najlepšie napĺňa vopred určené požiadavky.

V niektorých prípadoch spoločnosti nevyužívajú vyššie popísaný proces tvorby návrhu, ale preferujú tzv. iteračný proces návrhu [31]. Pri tomto procese sa môže vytvoriť niekoľko prototypov na experimentovanie s rôznymi fiktívnymi žánrami. Tieto prototypy prejdú testovaním cieľovými skupinami a na základe ich spätnej väzby je vytvorený finálny návrh. Po schválení dokumentácie alebo prototypu hry sa začnú zostavovať tímy, ktoré budú na vývoji hry pokračovať v ďalších fázach.

Produkčná fáza vývoja

V produkčnej fáze vývoja začína samotný vývoj hry – napr. tvorba grafických assetov alebo práca na hernom engine. Návrhári naďalej pokračujú v upresňovaní špecifikácie hry, pričom spolupracujú s tvorcami herného prostredia a úrovní. Postupne sa začína hra testovať a na základe získanej spätnej väzby majú tvorcovia možnosť vykonať v návrhu výrazné zmeny.

Verzia, v ktorej sa už hra dá hrať od začiatku do konca (aj keď vo veľmi hrubej podobe), sa nazýva tzv. „alpha build“ [31]. V tomto bode by už sa nemali robiť veľké zmeny v základnom návrhu hry. Ďalej pokračuje práca na tvorbe assetov, a zároveň sa testovaním hľadajú prípadné problémy v hre. Hra dosiahne tzv. „beta build“ [31] keď sú do hry pridané všetky assety a tím sa zameriava na záverečné dolaďovanie. V tejto fáze vývoja by sa už nemali robiť žiadne zmeny v návrhu a povolené by mali byť len opravy chýb. Produkčná fáza je najdlhšou fázou vývoja hry, môže trvať až niekoľko rokov. Jej posledným krokom je tzv. zlatá verzia – finálna verzia hry, ktorá je odoslaná do výroby.

Postprodukčná fáza vývoja

Poslednou fázou vývoja je fáza postprodukcie. Táto fáza sa prekrýva s neskoršími fázami produkcie a zahŕňa aj marketing a distribúciu finálnej hry. Z hľadiska vývojového tímu, hlavne programátorov, je táto fáza podporou vydanej hry. Jedná sa predovšetkým o aktualizácie opravujúce prípadné chyby, ktoré sa môžu vyskytnúť u užívateľov.

[1] Obrázok prevzatý z https://www.mobileapps.com/blog/game-design-process

Obr. 2.2: Logá herných enginov Unity[2], Unreal[3] a Godot[4].

Herný engine

Herný engine je softvér slúžiaci k vývoju hier, prostredníctvom ktorého sa v praxi implementujú pravidlá hry. Herné mechaniky na jednej strane detailne popisujú pravidlá hry tak, aby určili čo herný engine spraví, na druhej strane však tieto mechaniky neurčujú, ako to herný engine spraví [1]. Toto rozhodnutie je na programátoroch – ak existuje viac ako jeden spôsob na dosiahnutie toho istého efektu v hre, programátor rozhodne, ktorý z nich použije.

Termín „herný engine“ sa začal prvýkrát objavovať v polovici 90. rokov 20. storočia v spojitosti s hrami typu FPS (First-Person Shooter) [16], ako príklad môžeme uviesť hru Doom od *ID Sofware*. Vďaka využívaniu herného enginu sa vývojári nemusia starať o vytváranie jadra hry, ale môžu sa zamerať len na prostredie a hrateľnosť. Pre herný priemysel je toto rozdelenie veľmi výhodné.

Súčasné herné enginy nie sú zamerané na jeden typ hry alebo platformu, ale sú univerzálne. Princíp zostáva rovnaký – herný engine predstavuje znovu použiteľné jadro hry. Poskytuje základnú aj pokročilejšiu funkcionalitu, ktorá je viac-menej použiteľná pre všetky hry. Samozrejme platí, že vysokošpecializovaný herný engine urýchli vývoj hry viac ako univerzálny herný engine. Sťažuje sa tým však jeho opätovné použitie na iné typy hier.

Herný engine si vývojár môže vytvoriť sám alebo si môže zakúpiť licenciu na použitie niektorého už existujúceho enginu od rôznych firiem. Existujú aj voľne dostupné, bezplatné herné enginy (napríklad Unity, Unreal alebo Godot (viď obrázok 2.2)).

Herný engine Unity

- Multiplatformový herný engine, podporuje viac než 20 platforiem [43].
- Prvýkrát bol predstavený a vydaný v júni 2005 [11] spoločnosťou Unity Technologies na konferencii *Worldwide Developers Conference* spoločnosti *Apple* ako Mac OS X herný engine.
- Je flexibilný, poskytuje podporu na vývoj 2D aj 3D hier, mimo herného priemyslu je používaný aj vo filmovom a automobilovom priemysle, využívajú ho architekti, inžinieri a aj armáda Spojených štátov amerických [41].
- Ako hlavný programovací jazyk sa používa C# s využitím Mono (open-source vývojová platforma založená na *Microsoft* .NET Framework) [30], ale zároveň podporuje aj drag-and-drop vizuálne programovanie.
- K výhodám patria prístup k tzv. Asset store – internetovému obchodu prevádzkovanému spoločnosťou *Unity Technologies*, v ktorom sa dajú nájsť a zakúpiť užíva-

[2]Obrázok prevzatý z https://unity.com/legal/branding-trademarks
[3]Obrázok prevzatý z https://www.unrealengine.com/en-US/branding
[4]Obrázok prevzatý z https://godotengine.org/press/

teľmi vytvorené assety pomáhajúce s vývojom videohier; k rozsiahlej dokumentácii a ku komunitnému fóru.

- Ponúka viaceré možnosti licencií, od bezplatného osobného plánu až po priemyselnú licenciu s prístupom k všetkým funkciám a podpore.

Herný engine Unreal

- Podobne ako Unity má široké uplatnenie nielen v hernom priemysle, ale aj v iných oblastiach.
- Na trh bol uvedený už v roku 1998 [39] spoločnosťou *Epic Games* a pôvodne bol určený pre FPS hry (konkrétne ich vlastná hra Unreal).
- Hlavným rozdielom v porovnaní s herným enginom Unity je použitie programovacieho jazyka C++.

Herný engine Godot

- Multi-platformový herný engine slúžiaci na vytváranie 2D a 3D videohier.
- Prvá verejne dostupná verzia bola vydaná v januári 2014 [24], vydaný je ako open-source program pod MIT licenciou – je možné ho používať, upravovať a aj šíriť bezplatne [26].
- Umožňuje využitie programovacích jazykov C#, C++ a GDScript [25] – vysoko-úrovňový, objektovo-orientovaný, imperatívny a dynamicky typovaný jazyk vytvorený špecificky pre Godot.

Vývoj indie hier

Termín indie (odvodené od anglického slova *independent* – nezávislý) hra sa používa na popis videohier vytvorených nezávislými hernými štúdiami alebo dokonca aj jednotlivcami s obmedzenými zdrojmi. Rozvoj indie hier bol umožnený vďaka prechodu z predaja herných kópií v kamenných obchodoch na šírenie digitálnych verzií pomocou internetu [19]. K rozvoju hier prispela aj možnosť využívania herných enginov dostupných bezplatne na internete (viď *Herný engine* v sekcii 2.1).

Najvýraznejší nárast počtu indie hier nastáva približne po roku 2014 – na platforme *Steam* predávajúcej počítačové hry bolo podľa spoločnosti *Statista* v roku 2014 vydaných 1 348 hier, v roku 2015 ich bolo 2 500, v roku 2016 už 4 315 a v roku 2017 počet hier dosiahol číslo 6 306 [12]. Narastajúci trend vidíme aj v súčasnosti, v roku 2022 počet vydaných hier dosiahol maximum 10 963 [12]. Uvedené počty predstavujú celkový počet hier, ale výraznou väčšinou z nich sú indie hry.

Každým rokom narastajúci počet nových indie hier viedol k obavám z presýtenia trhu. Medzi nezávislými vývojármi sa dokonca začal šíriť pojem „indiepocalypse“ [48], ktorý sa týkal nadmernej ponuky hier, čím by sa celý trh stal nerentabilným. Tieto obavy sa síce úplne nenaplnili, ale pravdou je, že mnohé indie hry majú problém sa na trhu presadiť a tým pádom nie sú finančne ziskové.

Aj napriek týmto problémom indie hry sú ako celok vnímané pozitívne a sú populárne medzi užívateľmi nakoľko nie sú limitované očakávaniami, požiadavkami a obmedzeniami veľkých vydavateľov – väčšinou sú inovatívnejšie a experimentálnejšie ako mainstreamové

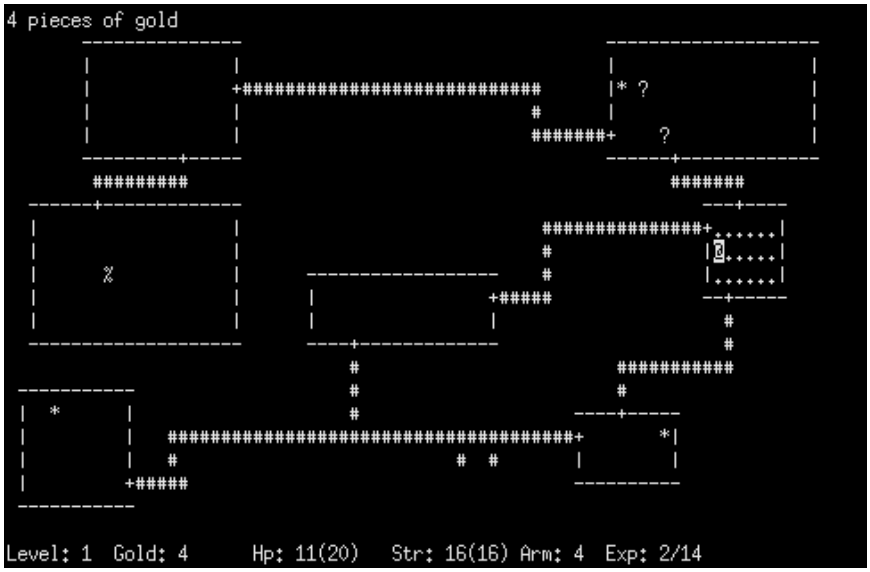
hry. Ponúkajú väčšiu tvorivú slobodu a umelecký prejav vývojárom. Mnohé indie hry dokázali, že aj nezávislí vývojári sú schopní vytvoriť hry na rovnakej alebo dokonca vyššej úrovni ako veľké vývojárske štúdia. Ako príklad sa môžu uviesť hry *Minecraft*, *Terraria*, *The Binding of Isaac* alebo *Hollow Knight*.

2.2 Roguelike žáner (podobné existujúce hry)

História roguelike žánru siaha do roku 1980, v ktorom bola vývojármi *Michaelom Toyom* a *Glennom Wichmanom* vydaná hra *Rogue* [18]. Táto hra inšpirovala mnohých ďalších vývojárov a postupne sa stala štandardom pre roguelike hry. Napriek tomu, že sa hra Rogue stala synonymom tohto žánru, nikdy sa nestala komerčne úspešnou. Pôvodná verzia z roku 1980 bola distribuovaná zadarmo. Komerčná verzia bola vydaná v roku 1984. Dôvodom komerčného neúspechu tejto verzie bol fakt, že stále bola k dispozícii bezplatná pôvodná verzia a taktiež medzitým vznikli ďalšie populárne hry inšpirované pôvodnou hrou – *Hack (1982, autor Jay Fenalson)* a *Moira (1983, autor Robert Koeneke)* [8].

Roguelike hry z 80.-tych rokov sú charakteristické spoločnými prvkami založenými na RPG systémoch (rôzne triedy postáv s rozdielnymi atribútmi, nachádzanie nového vybavenia v priebehu hrania) a procedurálnom generovaní obsahu (viď obrázok 2.3). Hlavne procedurálne generovanie obsahu je veľmi dôležitou vlastnosťou – pred vznikom roguelike žánru boli hry vnímané lineárne.

Prvé roguelike hry nepoužívali grafické assety, namiesto nich využívali na zobrazovanie hry znaky kódovacieho systému ASCII. Tieto roguelike hry obsahovali koncept známy ako „permadeath“ [8] – keď postava hráča zomrie, dáta o tejto postave a progrese sú vymazané a vyžaduje sa reštartovanie hry. Na tento koncept priamo nadväzuje aj fakt, že tieto hry neumožňovali hráčovi manuálne ukladať progres v hre. Niektoré z týchto hier obsahovali mechaniku ovplyvnenia nasledujúcich behov hry, napr. po úmrtí postavy mohol hráč nájsť jej pozostatky a predmety pri ďalšom hraní.



Obr. 2.3: Procedurálne generovaný dungeon v hre Rogue (1980)[5].

[5]Obrázok prevzatý z https://en.wikipedia.org/wiki/Rogue_(video_game)

Berlínska interpretácia

Jednoznačne definovať roguelike žáner nie je jednoduché. Jednotlivé definície sú odvodené od vlastností hier priamo inšpirovaných hrou Rogue. Jednou z populárnych interpretácií tohto žánra je tzv. Berlínska interpretácia vytvorená v roku 2008 na konferencii *International Roguelike Development Conference* [22]. Berlínska interpretácia využíva pri zaradení hier do roguelike žánru tzv. vysoko-hodnotové a nízko-hodnotové faktory [22]. Absencia niektorých z týchto hodnotiacich faktorov však nutne nemusí znamenať, že hra nie je zaradená do roguelike žánru.

Faktory s vysokou hodnotou

- **Náhodné generovanie prostredia** – prostredie je náhodne generované tak, aby bola zvýšená znovuhrateľnosť hry.
- **„Permadeath“.**
- **Ťahový systém** – hra nie je založená na reálnom čase, hráč má každé kolo k dispozícií jeden ťah.
- **Založené na mriežke** – svet je reprezentovaný ako mriežka, hráč aj ostatné postavy zaberajú jedno políčko.
- **Nemodálne** – každá akcia (napr. pohyb, útok) by mala byť prístupná v každej časti hry.
- **Komplexnosť** – hra by mala byť dostatočne komplexná na to, aby umožnila viaceré riešenia na jeden problém.
- **Spravovanie zdrojov** – hráč musí spravovať nájdené zdroje (napr. liečiace elixíry) a nájsť pre ne využitie.
- **„Hack'n'slash“** – hráč vs. svet – zabíjanie nepriateľov je veľmi dôležitou časťou hry, hra neobsahuje vzťahy medzi nepriateľmi (napr. diplomaciu).
- **Preskúmavanie a objavovanie** – hra vyžaduje preskúmavanie herného prostredia a objavovanie nových predmetov a vybavenia.

Faktory s nízkou hodnotou

- **Jedna postava hráča** – hráč ovláda len jednu postavu, ktorou je vnímaný celý svet a jej smrť znamená koniec hry.
- **Nepriatelia sú podobní hráčom** – nepriatelia a hráč podliehajú rovnakým pravidlám.
- **Taktická výzva** – na progres v hre je nutné postupne získavať znalosti o hre.
- **ASCII zobrazenie** – tradičné zobrazenie pomocou ASCII znakov.
- **Dungeons** – herné prostredie tvorené z miestností a koridorov.
- **Čísla** – čísla na popísanie postavy (napr. počet životov, hodnoty atribútov) sú priamo zobrazené.

V súčasnosti je však Berlínska interpretácia niektorými vývojármi aj užívateľmi považovaná za zastaralú a býva kritizovaná za nepresnú reprezentáciu moderného a otvoreného žánru hier.

„The eight rules of roguelike design“

V knihe *Exploring Roguelike Games* autor *John Harris* popisuje tzv. „eight rules of roguelike design“ [18] – 8 pravidiel roguelike dizajnu. Tieto pravidlá nie sú priamo záväzné, ale slúžia skôr ako užitočná nápoveda alebo pomôcka pre vývojárov.

- Postava hráča by nemala byť zabitá alebo príliš zranená jedným útokom.
- Použitie neidentifikovaného predmetu by nemalo byť smrteľné (v opačnom prípade sa hráč nebude pokúšať testovať nové predmety).
- Predmety by nemali byť jednoducho identifikovateľné ani pre skúsených hráčov.
- Účinky predmetov by sa mali prelínať takým spôsobom, aby v niektorých situáciách ich objavenie bolo prospešné, v iných situáciach by naopak uškodilo.
- Nájdenie nových predmetov by malo poskytovať zaujímavé možnosti pre hráča.
- Hráč s perfektnými znalosťami hry by mal byť schopný nájsť využitie pre každý predmet.
- Samotné plynutie času by malo nejakým spôsobom obmedzovať hráča a limitovať týmto možný “grind” (proces, kedy je na dosiahnutie výsledkov v hre nutné stráviť veľa času hraním).
- Obtiažnosť nepriateľov by mala rásť o trochu rýchlejšie než vylepšenia hráča.

Roguelike hry v súčasnosti

Rozmach moderných roguelike hier priamo súvisí s nárastom popularity indie hier v desiatych rokoch 21. storočia. Moderné roguelike hry striktne nedodržiavajú pôvodné definície žánru, ale často prichádzajú s novými inovatívnymi prvkami. Pre niektoré z týchto moderných hier sa začal používať aj pojem *roguelite*. Môže sa jednať napr. o hru s menším celkovým rozsahom a menšou variáciou procedurálneho generovania [8].

Jednou z prvých moderných roguelike hier je napr. hra *Faster Than Light* od štúdia *Subset Games* vydaná v roku 2012 [38]. Táto hra sa zameriava na súboje vesmírnych lodí – jej hracie mechaniky sú úplne odlišné od pôvodných roguelike hier. Ako ďalšie príklady môžeme uviesť hry *Rogue Legacy* (štúdio *Cellar Door Games*, rok 2013) [9], ktorá je akčnou platformovou hrou s prvkami *metroidvania* hier; *Nuclear Throne* (štúdio *Vlambeer*, rok 2015) [46], z hľadiska hrateľnosti sa jedná o tzv. *top-down shooter* (strielačka z pohľadu zhora) (viď obrázok 2.4) a *The Binding of Isaac* (vývojári *Edmund McMillen a Florian Himsl*, rok 2011 [45].

The Binding of Isaac je v podstate 2D top-down shooter, v ktorom hráč prechádza procedurálne generované úrovne. Táto hra (alebo presnejšie jej remake z roku 2014 *The Binding of Isaac: Rebirth* (viď obrázok 2.4) spolu s expanziami *Afterbirth*, *Afterbirth+* a *Repentance*) je považovaná za jednu z najlepších roguelike hier histórie. Jednou zo silných stránok hry je veľká variácia rôznych predmetov, vylepšení postáv a ich kombinácii.

Obr. 2.4: Snímky obrazovky z hier *Nuclear Throne*[6] a *The Binding of Isaac*[7].

2.3 Pseudonáhodné generátory

Pseudonáhodné generátory čísel (tzv. PNRGs) sú algoritmy využívajúce matematické funkcie na generovanie čísel, ktoré pôsobia náhodne. Produkované hodnoty nie sú skutočne náhodné, sú to deterministické sekvencie čísel, ktorých postupnosť záleží od inicializačnej hodnoty, ktorá sa nazýva *seed* [2]. Ako inicializačná hodnota sa môže použiť napr. čas od zapnutia počítača v milisekundách. Možnosť opakovaného využitia rovnakej inicializačnej hodnoty je užitočná pri simuláciách alebo debuggingu.

PRNG sa dá definovať ako konečný stavový automat s konečným počtom stavov, z čoho vyplýva, že po určitej dobe začne znovu generovať rovnaké sekvencie čísel – tento cyklus sa nazýva *perióda* [4]. Veľkosť periódy je dôležitá z hľadiska predvídateľnosti. V prípade nízkej periódy sa sekvencia generovaných hodnôt stáva predvídateľnou. PRNG s najvyššou možnou periódou sa označuje ako *maximum-period generator* [4].

Existujú viaceré druhy PRNGs určené pre rôzne potreby, napr. pri kryptografii sú potrebné z dôvodu bezpečnosti menej predvídateľné PRNGs. Pre účely herného vývoja sú však postačujúce jednoduché generátory.

Jedným z populárnych generátorov je *linear congruential generator (LGC)* – lineárny kongruentný generátor. LGC je definovaný nasledovným vzťahom [4]:

$$x_{n+1} = (ax_n + c) \mod m, \; n \geq 0, \tag{2.1}$$

kde a, c a m značia kladné celočíselné konštanty a inicializačná hodnota je x_0. Konštanta $a > 1$ sa nazýva *multiplier*, konštanta c je nazývaná *increment* a konštanta m je označovaná ako *modulus* generátora. Generátor produkuje hodnoty v intervale $0 \leq x_n < m$ [14]. Konštanty použité vo vzťahu 2.1 však musia byť vhodne zvolené. Existujú prípady, v ktorých generovaná sekvencia nie je náhodná. Aby mohol byť LGC označovaný ako maximum-period generátor musia platiť nasledujúce podmienky [4]:

1. c a m sú nesúdeliteľné – ich najväčší spoločný deliteľ je 1,
2. $a - 1$ je násobkom 4, ak m je násobkom 4,
3. $a - 1$ je násobkom každého prvku prvočíselného rozkladu m.

Ďalšou kategóriou generátorov sú *linear feedback shift register (LFSR)* – posuvný register s lineárnou spätnou väzbou. Posuvný register má svoje vstupy pripojené na výstupy takým

[6] Obrázok prevzatý z https://store.steampowered.com/app/242680/
[7] Obrázok prevzatý z https://store.steampowered.com/app/250900/

spôsobom, aby pri aktivácii obvodu boli dáta posunuté. LFSR je posuvný register, ktorého vstupný bit je výstupom lineárnej funkcie dvoch alebo viacerých predchádzajúcich stavov [13]. Do tejto kategórie generátorov patrí napríklad *Xorshift*, ktorý pomocou opakovaného použitia inštrukcie *exclusive-or (XOR)* generuje sekvenciu $2^{32} - 1$ kladných celočíselných hodnôt x, sekvenciu $2^{64} - 1$ dvojíc x, y, alebo sekvenciu $2^{96} - 1$ trojíc x, y, z, atď. [28]. Pri generovaní je inštrukcia XOR použitá na danú hodnotu a jej bitový posun. Xorshift je používaný triedou `Random` na generovanie náhodných hodnôt v hernom engine Unity [44].

Poslednou veľkou skupinou PRNG sú generátory založené na použití celulárnych automatov (viď sekcia 2.4). Využitie celulárnych automatov na generovanie hodnôt ako prvý navrhol *Stephen Wolfram* v roku 1985 v práci *Cryptography with cellular automata* [47]. Wolfram navrhol využitie jednodimenzionálnych, dvojstavových celulárnych automatov na generovanie náhodných sekvencií pomocou vzťahu [47]:

$$a'_i = a_{i-1} \text{ XOR}(a_i \text{ OR } a_{i+1}), \tag{2.2}$$

alebo jeho ekvivalentu:

$$a'_i = (a_{i-1} + a_i + a_{i+1} + a_i a_{i+1}) \mod 2. \tag{2.3}$$

V súčasnosti už ale využitie 1D celulárnych automatov nie je z bezpečnostného hľadiska dostačujúce (napr. v kryptografii), používané sú teda 2D alebo dokonca aj 3D [21] celulárne automaty.

2.4 Procedurálne generovanie

Procedurálne generovanie je proces vytvárania obsahu pomocou algoritmov, nie manuálne. Existuje veľa rôznych možností procedurálneho generovania obsahu, niektoré z nich sú popísané v nasledujúcej sekcii.

Perlinov šum

Jednou z možností procedurálneho generovania je využitie šumov. Šum sa dá všeobecne popísať ako séria náhodných hodnôt usporiadaných v línii alebo v mriežke [15]. V niektorých oblastiach (napr. spracovávanie obrázkov) je šum nežiadúci, ale pri procedurálnom generovaní pomáha prirodzenému vzhľadu generovaného obsahu. Medzi základné druhy šumu patrí tzv. *white noise (biely šum)* (viď obrázok 2.5). Tento šum je veľmi jednoduchý, jedná sa o rovnomerne vybrané náhodné čísla. Biely šum ale nie je vhodný na využitie pri procedurálnom generovaní. Existujú mnohé ďalšie druhy šumov, ktoré sú na toto využitie vhodnejšie, medzi najznámejšie patrí *Perlinov šum*, ktorý zaraďujeme do kategórie gradientových šumov.

Perlinov šum (viď obrázok 2.5) je používaný na generovanie prirodzene vyzerajúcich vzorov napríklad pri tvorbe mrakov, dymu, ohňa, textúr mramoru a podobne. Od klasického šumu sa líši tým, že v sekvencii čísel sú vždy za sebou nasledujúce čísla blízko k sebe. Toto zaručuje jeho prirodzený vzhľad [2]. Perlinov šum bol predstavený *Kenom Perlinom* v práci *An image synthesizer* [33] v roku 1985. Perlinov šum je v tejto práci popísaný ako funkcia `Noise()` s trojdimenzionálnym vektorom ako vstupnou hodnotou. Jeho princíp je možné popísať nasledovne:

1. Vytvorenie množiny bodov s celočíselnými x, y a z koordinátmi, takáto množina sa nazýva *celočíselná mriežka*. Každému bodu mriežky sú priradené pseudonáhodné hodnoty $[a, b, c, d]$, pričom:
$$[a, b, c, d] = H([x, y, z]), \tag{2.4}$$
kde $[a, b, c, d]$ definuje lineárnu rovnicu s gradientom $[a, b, c]$ a hodnotu d v bode $[x, y, z]$. $H()$ je vhodné implementovať ako hashovaciu funkciu.

2. Ak sa vstupný vektor $[x, y, z]$ nachádza na celočíselnej mriežke, tak platí:
$$Noise([x, y, z]) = d_{[x,y,z]}. \tag{2.5}$$
V opačnom prípade je použitá kubická interpolácia medzi koeficientami rovnice mriežky pomocou vzťahu:
$$s(t) = 3t^2 - 2t^3. \tag{2.6}$$

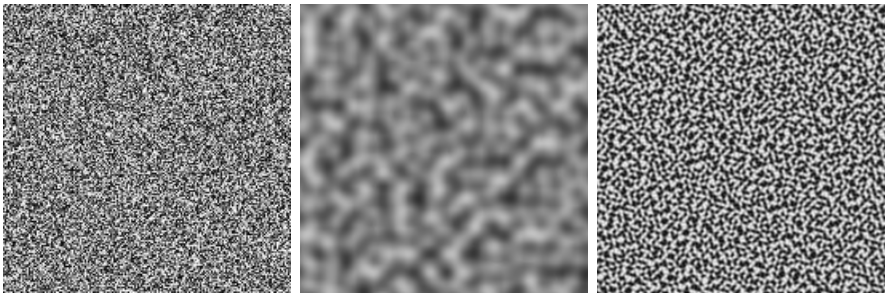
V roku 2002 Ken Perlin opísal v práci *Improving Noise* [34] vylepšenú verziu pôvodného Perlinovho šumu. Táto vylepšená verzia opravuje dva nedostatky originálu – nespojitosť interpolácie druhého rádu a neoptimálny výpočet gradientu. Nespojitosť je spôsobená tým, že druhá derivácia vzťahu 2.6, t.j. $6 - 12t$ sa nerovná nule ak $t = 0$ alebo $t = 1$. Tento problém je riešený nahradením tohto vzťahu nasledujúcim vzťahom:
$$s(t) = 6t^5 - 15t^4 + 10t^3, \tag{2.7}$$
ktorý má nulovú prvú aj druhú deriváciu pri $t = 0$ aj $t = 1$. Toto vylepšenie teda odstraňuje možné artefakty.

Druhý nedostatok súvisí s predpočítaným polom pseudonáhodných gradientov označovaných ako G, ktoré sú používané pri výpočtoch. Nastáva smerová asymetria (skrátenie pozdĺž osí a predĺženie po diagonálach), ktorá môže spôsobiť nežiadúci zhlukovací efekt. V tomto prípade sa blízke gradienty, ktoré sú takmer zarovnané s osou, navzájom zarovnajú a spôsobia týmto vysoké hodnoty v daných oblastiach. Riešením tohto nedostatku je odchýlenie smeru gradientov preč od osí a dlhých diagonál. Množina gradientov G je vo vylepšenej verzii Perlinovho šumu nahradená vopred určenými dvanástimi vektormi určenými smermi zo stredu kocky na jej hrany:
$$\begin{aligned}&(1,1,0),(-1,1,0),(1,-1,0),(-1,-1,0),\\&(1,0,1),(-1,0,1),(1,0,-1),(-1,0,-1),\\&(0,1,1),(0,-1,1),(0,1,-1),(0,-1,-1).\end{aligned} \tag{2.8}$$

Na zlepšenie výkonu je množina G rozšírená o ďalšie štyri vektory $(1, 1, 0), (-1, 1, 0), (0, -1, 1)$ a $(0, -1, -1)$, ktoré vizuálne neovplyňujú výsledok. Celkovo je podľa Perlina táto vylepšená verzia algoritmu rýchlejšia o približne 10%.

Simplex

V roku 2001 Ken Perlin predstavil šum *Simplex* (viď obrázok 2.5), ktorý mal prekonať rôzne limitácie originálneho Perlinovho šumu. Medzi jeho výhody patria napr. nižšia výpočtová zložitosť, lepšie škálovanie na vyššie dimenzie (4D, 5D a vyššie), absencia viditeľných smerových artefaktov a jednoduchá hardvérová implementácia [32]. Dôležitým prvkom je využitie *simplex mriežky* [17] – skladá sa z najjednoduchšieho a najkompaktnejšieho útvaru, ktorého opakovaním sa dá zaplniť priestor. Napríklad v prípade dvojdimenzionálneho priestoru sa



Obr. 2.5: Porovnanie rôznych šumov (white noise[8], Perlin[9], Simplex[10]).

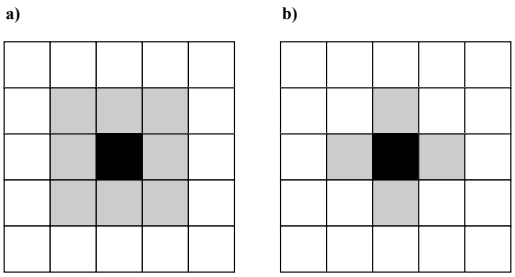
jedná o rovnostranný trojuholník. V trojdimenzionálnom priestore je to štvorsten. Výhodou takýchto útvarov je, že majú najmenší možný počet rohov, čo uľahčuje interpoláciu hodnôt.

Na rozdiel od originálneho Perlinovho šumu boli trojdimenzionálne a vyššie verzie Simplexu patentované. Z tohto dôvodu bol vytvorený voľne dostupný šum pod názvom *OpenSimplex* [37].

Celulárne automaty

Celulárne automaty sa používajú v mnohých oblastiach, ako napríklad v informatike, fyzike alebo aj v biológii, ako modely výpočtu, rastu, fyzikálnych javov, atď. Ako konkrétny príklad je možné uviesť napr. simuláciu šírenia ohňa alebo rastu rastlín a lesov [36]. Celulárny automat je diskrétny výpočtový model, ktorý sa skladá z n-dimenzionálnej mriežky (napr. 1D vektor alebo 2D matica), množiny stavov a množiny pravidiel prechodov medzi stavmi [35]. Každá bunka môže byť súčasne len v jednom z možných stavov.

Úvodné rozloženie stavov buniek určuje *počiatočný stav* celulárneho automatu. Následne sa stav bunky v čase t určuje na základe stavu samotnej bunky a stavu jej susedných buniek v čase $t - 1$. To, akým spôsobom susedné bunky ovplyvňujú stav bunky, sa líši od celulárneho automatu. Najčastejšie typy týchto tzv. *neighbourhoods* sú pri dvojdimenzionálnych celulárnych automatoch *Moore neighbourhoods* a *von Neumann neighbourhoods* [35]. Oba môžu mať veľkosť ľubovoľného celého čísla väčšieho alebo rovného 1. Pri veľkosti 1 Moore neighbourhood pozostáva zo všetkých 8 buniek okolo vybranej bunky, t.j. vrátane diagonálnych buniek. Naopak von Neumann neighbourhood tieto diagonálne bunky ignoruje – berie ohľad len na bunky hore/dole/vpravo/vľavo od vybranej bunky (viď obrázok 2.6).



Obr. 2.6: Porovnanie *Moore* a) a *von Neumann* b) neighbourhoods.

[8]Obrázok prevzatý z https://en.wikipedia.org/wiki/White_noise
[9]Obrázok prevzatý z https://en.wikipedia.org/wiki/Perlin_noise
[10]Obrázok prevzatý z https://en.wikipedia.org/wiki/Simplex_noise

Gramatiky

Gramatika v informatike označuje súbor pravidiel na prepisovanie reťazcov. Gramatiky predstavil v roku 1956 lingvista *Noam Chomsky* [10] ako spôsob na modelovanie prirodzeného jazyka. Každé pravidlo gramatiky je zapísané vo forme *znak* $\rightarrow$ *iný znak*. Samotné použitie gramatiky je pomerne jednoduché – prechádza sa vstupný reťazec a v prípade nájdenia znaku alebo sekvencie znakov, ktoré sa vyskytujú na ľavej strane nejakého pravidla, sa tieto znaky prepíšu podľa pravej strany daného pravidla [35]. Jednoduchý príklad pravidiel gramatiky môže byť v nasledovnej podobe:

1. $A \rightarrow AB$
2. $B \rightarrow b$

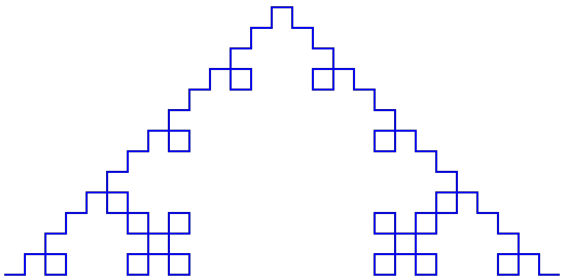
kde v prípade, ak je vstupným reťazcom A, je prvým krokom prepísanie podľa pravidla 1 – vznikne reťazec AB. V ďalšom kroku by bol znak A z výsledného reťazca znovu prepísaný na reťazec AB a znak B by bol podľa pravidla 2 prepísaný na znak b. Proces by takýmto spôsobom pokračoval ďalej. Vo väčšine prípadov veľké písmená predstavujú *neterminálne znaky* a malé písmená znaky *terminálne* (nie sú ďalej prepisované).

Existujú dve dôležité kritériá na delenie gramatík – to, či sú *deterministické*, a *poradie, v ktorom sú rozvíjané* [35]. Pri deterministických gramatikách existuje vždy len jedno pravidlo pre každý znak alebo sekvenciu znakov. Nemôže teda nastať situácia, v ktorej by na určitý znak alebo reťazec mohli byť aplikované dve rôzne pravidlá. Pri nedeterministických gramatikách je toto možné a tým pádom môže byť výsledný reťazec rozdielny na základe vybratého pravidla (tento výber môže byť náhodný alebo určený nejakými ďalšími parametrami).

Z hľadiska poradia rozvíjania vstupného reťazca sa môže jednať o *sekvenčné prepisovanie* alebo *paralelné prepisovanie* [35]. Pri sekvenčnom prepisovaní je vstupný reťazec čítaný v poradí zľava doprava a zároveň pritom prepisovaný. Výsledok spracovania pravidla je teda zapísaný priamo pred ďalší znak, ktorý bude spracovaný. V prípade paralelného prepisovania všetky prepisovania prebiehajú naraz.

L-systémy

L-systémy sú gramatiky využívajúce paralelné prepisovanie. Prvýkrát ich predstavil biológ *Aristid Lindenmayer* [23] v roku 1968 ako model rastu organických systémov, napr. rastlín alebo rias. Vstupný reťazec v L-systémoch sa označuje ako *axióm*.



Obr. 2.7: Príklad Kochovej krivky vytvorenej pomocou L-systému[11].

[11] Obrázok prevzatý z https://en.wikipedia.org/wiki/L-system

Ako príklad je možné uviesť jednoduchý L-systém definovaný Lindenmayerom na modelovanie rastu kvasníc:

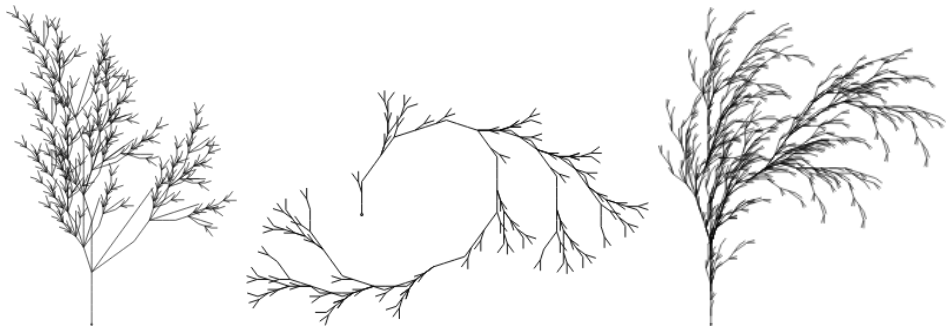
1. $A \rightarrow AB$
2. $B \rightarrow A$

Pri použití týchto pravidiel s axiómom A vyzerá prvých pár rozložení nasledovne:

1. A
2. AB
3. ABA
4. $ABAAB$
5. $ABAABABA$
6. $ABAABABAABAAB$

Z tohto výstupu je zrejmé opakovanie vzorov a každou iteráciou väčší nárast dĺžky reťazca.

Na grafické zobrazenie L-systémov je možné využiť napríklad tzv. *turtle graphics* [35]. "Korytnačka" predstavuje kurzor, ktorý sa hýbe určitým smerom a kreslí na plátno. Na interpretovanie L-systému takýmto spôsobom sa môžu použiť znaky F, $+$ a $-$, pričom F značí pohyb dopredu, $+$ je otočenie doľava o 90 stupňov a $-$ je otočenie doprava o 90 stupňov. Takýmto spôsobom sa dá vytvoriť útvar zvaný *Kochova krivka* (viď obrázok 2.7). Nevýhodou tejto interpretácie je ale fakt, že vykresľovanie je limitované len na jednu čiaru – všetko musí byť kreslené "jedným ťahom". Na odstránenie tejto limitácie je možné pridať ďalšie symboly [a]. Pomocou [sa uloží aktuálna pozícia a orientácia kurzoru,] zase slúži na načítanie predchádzajúcej pozície. Vďaka tomuto rozšíreniu je už možné vytvárať zložitejšie štruktúry ako možno vidieť na obrázku 2.8.



Obr. 2.8: Príklad rastlinných štruktúr vytvorených pomocou L-systémov[12].

[12]Obrázky prevzaté z https://www.vexlio.com/blog/drawing-simple-organics-with-l-systems/

2.5 Využitie procedurálneho generovania v oblasti herného vývoja

V oblasti herného vývoja sa používajú techniky manuálneho vytvárania obsahu, ale zároveň aj procedurálne generovanie. Procedurálne generovanie môže byť využité na vytváranie rôznych typov obsahu – herné prostredie, predmety, postavy, ale napríklad aj animácie alebo dokonca príbeh. Veľmi často je používaný aj kombinovaný prístup. V tomto prípade môže byť napríklad vytvorený základ herného sveta pomocou procedurálneho generovania, ktorý je potom dotvorený manuálne vývojármi (napr. hra *The Elder Scrolls V: Skyrim*) [36]. Ďalším vhodným príkladom je hra *The Binding of Isaac* (viď sekcia 2.2), v ktorej je herný dungeon generovaný procedurálne z manuálne vytvorených miestností.

Výber vhodného prístupu závisí od mnohých faktorov a rozhodnutia vývojárov. Faktory na použitie procedurálneho generovania môžu byť rozdelené na *technické* a *dizajnové* [2]. Medzi technické faktory patrí napríklad obmedzenie veľkosťou disku – v tomto prípade nie je možné vytvoriť obsah hry dopredu počas vývoja. Ako príklad je možné uviesť hru *Elite* z roku 1985, ktorá použila procedurálne generovanie na uloženie ôsmich galaxií obsahujúcich 256 hviezdnych systémov len v 32 KB. V súčasnosti už je vzhľadom na pokrok v technológiách tento objem dát zanedbateľný, ale rovnaký princíp môže byť využitý na modernejšie projekty. Jedným z príkladov je pokračovanie vyššie uvedenej hry s názvom *Elite: Dangerous* z roku 2014. Táto hra používa rovnaký prístup k procedurálnemu generovaniu na reprezentáciu Mliečnej dráhy obsahujúcej 400 miliárd preskúmateľných hviezdnych systémov. Bez využitia procedurálneho generovania by toto nebolo možné.

Dizajnový faktor je pri rozhodovaní o použití procedurálneho generovania použitý častejšie. Z pohľadu dizajnu využitie procedurálneho generovania umožňuje vývojárom pridať do hry variabilitu a znovuhrateľnosť. Veľmi populárne je využitie v roguelike hrách (viď sekcia 2.2). Pravdepodobne najznámejšie využitie procedurálneho generovania na vytváranie nových svetov pri hraní je v hre *Minecraft* (viď obrázok 2.9), ktorá používa na generovanie prostredia Perlinov šum (viď sekcia 2.4).

Obr. 2.9: Pohľad na procedurálne generovaný svet v hre *Minecraft*[13].

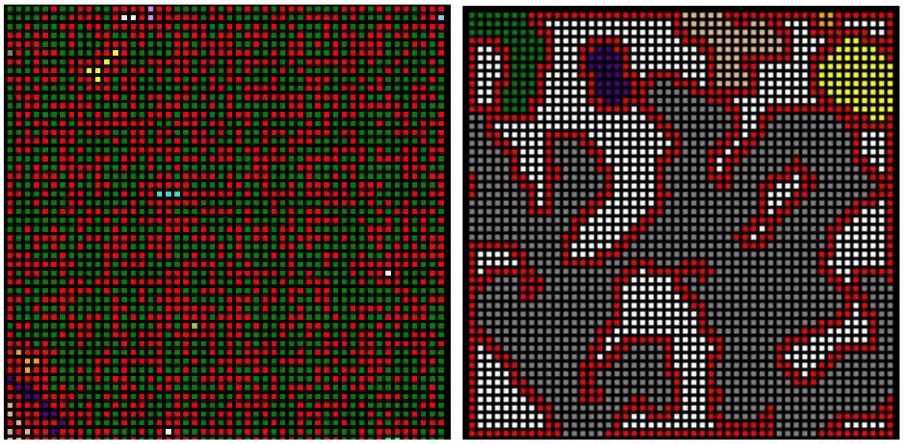
[13]Obrázok prevzatý z https://minecraft.fandom.com/wiki/History_of_world_generation

Zaujímavým príkladom využitia procedurálneho generovania obsahu je použitie celulárneho automatu na generovanie nekonečného jaskynného dungeonu popísané v práci *Cellular automata for real-time generation of infinite cave levels* [20] od autorov Johnson et al. Cieľom ich práce je vytvoriť herné prostredie, ktoré sa nekonečne a plynule rozprestiera všetkými smermi. Takéto prostredie samozrejme nie je možné uložiť na disk, tým pádom generovanie musí prebiehať za behu hry. Časové okno na generovanie nového prostredia sa vytvorí pri presune hráča medzi miestnosťami.

Každá miestnosť je reprezentovaná mriežkou s rozmermi 50x50. Bunky celulárneho automatu môžu nadobúdať dva stavy – prázdny alebo obsadený skalou. Na ovládanie generovania prostredia sú v práci využité štyri parametre:

1. Pomer obsadených buniek (t.j. nedosiahnuteľné oblasti).
2. Počet generácií celulárneho automatu (v priemere väčší počet generácií vedie k menšiemu počtu obsadených buniek).
3. Hraničná hodnota pre neighbourhood (viď sekcia 2.4), ktorá definuje skalu ($T = 5$).
4. Počet neighbourhood buniek.

V počiatočnom stave je miestnosť zaplnená prázdnymi bunkami. Následne sa každá bunka môže s určitou pravdepodobnosťou (napr. v práci je použitá hodnota 0.5) zmeniť na skalu. Ďalej je na mriežku aplikovaný celulárny automat. Prázdna bunka je konvertovaná na skalu, ak aspoň T jej susedných buniek je obsadených. Obsadené bunky, ktoré susedia s prázdnou bunkou môžu byť zmenené na tretí stav – stenu. Tento stav je z hľadiska funkcionality rovnaký ako skala, používa sa len z vizuálnych dôvodov. Týmto je vygenerovaná jedna miestnosť, ale hra vyžaduje viacero prepojených miestností. Po vygenerovaní miestnosti sú hneď ďalej generovaní jej susedia. V prípade, ak neexistuje žiadne prepojenie medzi najväčšími prázdnymi oblasťami v dvoch miestnostiach, na mieste, kde sú najmenej od seba oddelené, je vytvorený tunel. Výsledok generovania je graficky znázornený na obrázku 2.10.



(a) Random map (b) CA map

Obr. 2.10: Porovnanie náhodne generovanej mapy a) a mapy generovanej pomocou využitia celulárneho automatu b)[14].

[14] Obrázok prevzatý z *Cellular automata for real-time generation of infinite cave levels* [20]

2.6 Umelá inteligencia v hrách

Nevyhnutným a prirodzeným aspektom života živých organizmov je inteligencia. Počítačom však chýba akýkoľvek druh inteligencie, nakoľko sú len elektronické zariadenia, ktoré dokážu prijímať údaje a premieňať ich na výsledky pomocou logických a matematických operácií. Z tohto hľadiska sa dá povedať, že umelá inteligencia je oblasťou skúmajúcou ako umožniť strojom simulovať túto prirodzenú inteligenciu.

Umelá inteligencia sa môže deliť na *slabú* alebo *silnú* [7]. Slabá umelá inteligencia je navrhnutá na vykonávanie špecifických úloh. Silná umelá inteligencia je naopak tvorená tak, aby bola schopná nájsť riešenie aj na neznáme úlohy. Snaží sa kopírovať myšlienkové procesy ľudí. Tento typ inteligencie by teoreticky mal byť schopný prejsť *Turingovým testom* [40] – cieľom tohto testu je zistiť či testovaná umelá inteligencia je na rovnakej úrovni (t.j. nerozlíšiteľná) ako ľudská.

Z hľadiska potrieb herného vývoja je postačujúca slabá umelá inteligencia. V hrách nie je potrebná umelá inteligencia porovnateľná s ľudskou, postačujúce je aby bola dostatočne inteligentná na to, aby bola hra pre hráča zaujímavá a zábavná. Príliš jednoduchá umelá inteligencia môže kaziť dojem aj z dobre navrhnutej hry. Po čase sa stáva predvídateľnou a teda pre hráča nezaujímavou a nudnou. Na druhej strane však príliš dobrá umelá inteligencia môže byť pre hráča príliš náročná. V prípade, ak je umelá inteligencia taká perfektná, že v hre vždy zvolí tú najlepšiu možnosť, stáva sa pre hráča takmer neporaziteľnou a tým pádom hráča môže demotivovať od pokračovania hrania hry.

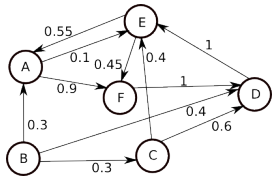
V hrách sa môže vyskytnúť *úmyselná* alebo *neúmyselná predvídateľnosť* [2]. Istá miera úmyselnej predvídateľnosti je v niektorých prípadoch dobrá, napr. predvídanie trasy nepriateľov v stealth hrách. Neúmyselná predvídateľnosť môže spôsobiť pre hráča pocit, že hra nie je dostatočne náročná alebo naopak, že hra nie je z hľadiska hráča fér. Možným riešením problematiky perfektnej/príliš jednoduchej umelej inteligencie v hrách je použitie úmyselných chýb, ktoré sa vyskytujú náhodne – do hier sa implementuje pravdepodobnosť a náhodnosť (viď sekcia 2.3).

Pathfinding

Pohyb *agentov* (autonómna postava vykonávajúca rozhodnutia na základe získaných informácií z hry) umelej inteligencie v hrách je daný trasou. Táto trasa môže byť fixne určená vývojármi alebo môže byť dynamická – prepočítavaná počas behu hry. Fixne určená trasa je pre vývojárov ľahko implementovateľná. Pre hráča fixne daná trasa nie je príliš zaujímavá a zároveň je veľmi jednoduché tento systém „pokaziť“, napr. prekážka vložená do trasy agenta môže spôsobiť, že sa zasekne. Z tohto dôvodu vývojári preferujú dynamické hľadanie cesty – *pathfinding* (alebo občas nazývané aj *path planning*) [29].

A* algoritmus

Na pathfinding herní vývojári najčastejšie používajú A* algoritmus. Tento algoritmus bol navrhnutý na tzv. *point-to-point pathfinding* [29] – hľadanie trasy medzi dvomi bodmi. Algoritmu A* je veľmi podobný algoritmus *Dijkstra*, ktorý slúži na hľadanie najkratších ciest z jedného bodu do všetkých ostatných. Toto je však z hľadiska herného pathfindingu zbytočné. Algoritmus A* však nevie priamo pracovať s herným prostredím v neupravenej podobe. Na základe existujúceho prostredia je najprv nutné vytvoriť dátovú štruktúru, ktorá bude reprezentovať herné prostredie vo formáte vyhovujúcom tomuto algoritmu.



Obr. 2.11: Príklad váženého orientovaného grafu[15].

Vhodnou reprezentáciou herného prostredia sú grafy, formálne definované ako $G = \{N, E\}$, kde N značí jednotlivé *nodes* (vrcholy) a E značí *edges* (hrany) [7]. Herné prostredie je rozdelené na malé časti v grafe reprezentované jednotlivými nodes, ktoré sú navzájom prepojené. Prepojenia medzi nodes majú určitú váhu, hovoríme teda o *weighted – vážených grafoch*. Tieto grafy je možné označiť ako *non-negative – nezáporné* [29], nakoľko táto váha udáva vzdialenosť alebo časovú náročnosť medzi dvomi bodmi v hernom prostredí a ani jedna z týchto hodnôt nemôže byť záporná. Táto reprezentácia sa môže rozšíriť aj o smer prepojenia, čo umožňuje presun postavy z bodu do bodu len jedným smerom (napr. použité pri skákaní z vyššej polohy na nižšiu). Tento typ grafu sa nazýva *directed – orientovaný graf* [29] (viď obrázok 2.11).

Algoritmus 1: A* ALGORITMUS

Input: graph with source node *start* and goal node *end*
Output: *path* from *start* to *end* with the lowest cost

```
open_list = { start }
closed_list = { }
while open_list is not empty do
    current = node of open_list with the lowest f
    if current == end then
        return
    end
    remove current from open_list
    for each child in current.children do
        cost = g(current) + distance(current, child)
        if child in open_list and g(child) ≤ cost then
            continue
        end
        if child in closed_list and g(child) ≤ cost then
            continue
        end
        if child not in open_list and child not in closed_list then
            add child to open_list
            g(child) = cost
            h(child) = heuristic(child, end)
            f(child) = g(child) + h(child)
        end
    end
    add current to closed_list
end
```

[15]Obrázok prevzatý z https://pi.math.cornell.edu/~numb3rs/blanco/Provenance.html

Algoritmus A* prebieha v iteráciách. Používa *open* a *closed list*. V open liste sú uložené nodes, ktoré boli navštívené ale nie sú ešte spracované. Do closed listu sú ukladané spracované nodes. Na začiatku každej iterácie je z open listu vybratý node s najnižšou odhadovanou celkovou cenou. Z tohto aktuálneho node následne posudzuje všetky vychádzajúce prepojenia. Na posúdenie vychádzajúcich prepojení a výber vhodného nasledujúceho node je použitá hodnota odhadu celkovej trasy. Táto hodnota je počítaná ako súčet hodnoty trasy od začiatku po aktuálny node a vzdialenosti z aktuálneho node do cieľového. Konkrétna podoba výslednej hodnoty záleží od použitej *heuristickej funkcie*. Algoritmus je vo väčšine prípadov ukončený v momente keď je prvýkrát navštívený cieľový node (viď pseudokód 1).

Zvolenie vhodnej heuristiky je veľmi dôležité, nakoľko priamo ovplyvňuje rýchlosť nájdenia trasy – čím presnejšia heuristika, tým rýchlejšie nájde A* trasu. Teoreticky by použitie perfektnej heuristiky, ktorá by vždy vracala minimálnu vzdialenosť medzi dvomi nodes, znamenalo, že by A* algoritmus išiel priamo po najlepšej možnej trase [29].

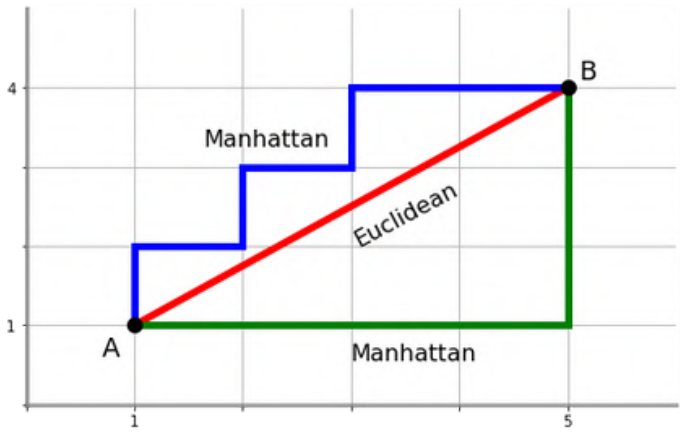
V prípade, že je heuristika príliš nízka (podceňuje dĺžku trasy), vykonávanie A* algoritmu trvá dlhšie. Preferuje nodes bližšie k počiatočnému node a celkové trvanie hľadania je dlhšie. V opačnom prípade, ak je heuristika príliš vysoká (preceňuje dĺžku trasy), A* algoritmus nemusí nutne vrátiť tú najlepšiu trasu. V tomto prípade preferuje vytváranie trasy s menším počtom nodes, aj vtedy ak majú tieto nodes vysokú váhu. Správne preceňovanie môže v ideálnom prípade A* algoritmus urýchliť, keďže v tomto prípade algoritmus preferuje pohyb k cieľovému node.

Medzi často používané heuristiky patria napr. heuristiky využívajúce *Manhattan* alebo *Euclidean* vzdialenosti (viď obrázok 2.12). Manhattan vzdialenosť je vhodné používať v prípade, ak je herné prostredie rozdelené na štvorce, medzi ktorými je možné prechádzať len v 4 smeroch – hore/dole/vpravo/vľavo. Počítaná je podľa nasledovného vzťahu [6]:

$$h(n) = |n.x - end.x| + |n.y - end.y|. \tag{2.9}$$

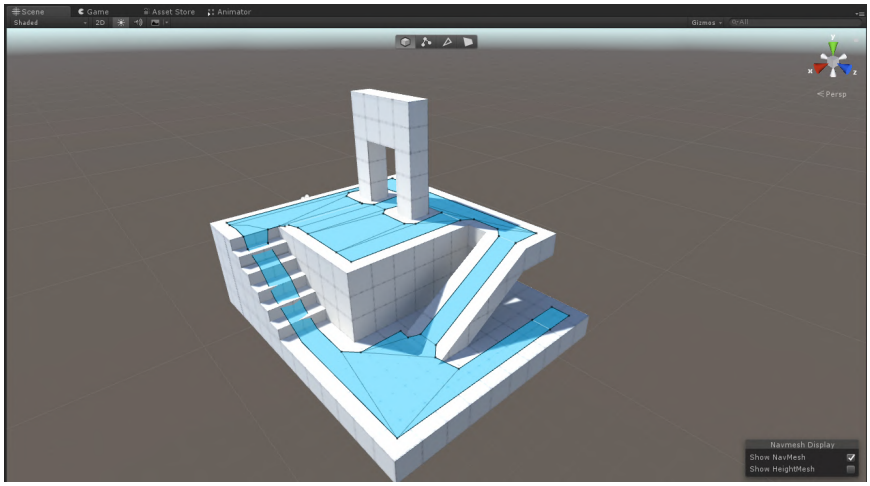
Euclidean vzdialenosť je používaná častejšie, nakoľko umožňuje pohyb všetkými smermi pod rôznymi uhlami. Týmto bližšie reprezentuje pohyb v reálnom prostredí. Počítaná je podľa vzťahu [5]:

$$h(n) = \sqrt{(n.x - end.x)^2 + (n.y - end.y)^2}. \tag{2.10}$$



Obr. 2.12: Ukážka Manhattan a Euclidean vzdialeností medzi dvomi bodmi v grafe[16].

[16]Obrázok prevzatý z https://www.omnicalculator.com/math/manhattan-distance



Obr. 2.13: Príklad vygenerovaného NavMesh povrchu v hernom engine Unity[17].

Unity NavMesh

Unity NavMesh je pathfinding systém používaný v hernom engine Unity (viď sekcia 2.1). Tento systém slúži na určenie povrchu, po ktorom sa môžu pohybovať agenti umelej inteligencie.

Agent umelej inteligencie je reprezentovaný ako valec. Dosiahnuteľné oblasti sú vygenerované na základe toho, kde agent vie stáť – miesta kde je možné umiestniť tento valec. Tieto miesta sa následne pripoja k povrchu ležiacom na vrchu herného prostredia. Tento povrch sa nazýva *navigation mesh* (NavMesh) [42]. Povrch NavMesh (viď obrázok 2.13) je reprezentovaný konvexnými polygónmi. Týmto je zabezpečené, že medzi dvomi bodmi vnútri polygónu nebude žiadna prekážka.

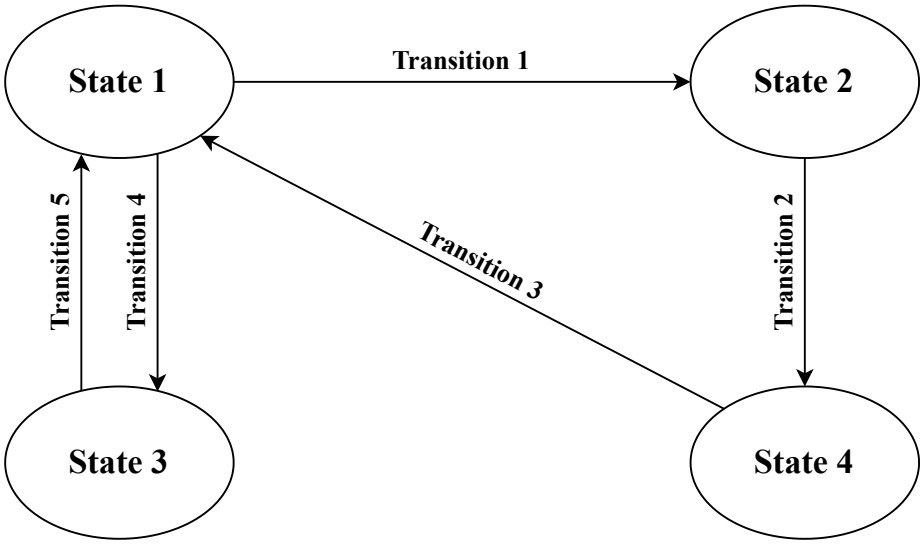
Na samotné nájdenie trasy využíva algoritmus A*. NavMesh nájde najbližšie polygóny k počiatočnej a cieľovej pozícií a vypočíta trasu medzi nimi. NavMesh poskytuje taktiež aj možnosti samotného pohybu agentov a vyhýbanie sa ostatným agentom a iným prekážkam.

Finite State Machine

Finite State Machine (FSM) – konečný stavový automat, je model automatu založený na konečnom počte možných *stavov* a *pravidiel* popisujúcich *prechody* medzi týmito stavmi. V prípade splnenia podmienky určitého pravidla sa vykoná prechod do ďalšieho stavu. Jeden stav z množiny možných stavov je označovaný ako *počiatočný stav*. FSM sa môže v danom okamihu nachádzať len v jednom zo stavov. Je možné ich zobrazovať viacerými spôsobmi, ako napríklad pomocou tabuliek alebo stavových diagramov (viď obrázok 2.14).

V oblasti herného vývoja predstavujú dobrú možnosť na reprezentáciu chovania sa agentov umelej inteligencie. Z ľudského hľadiska sú celkom intuitívne – nie je príliš zložité rozčleniť správanie sa agenta umelej inteligencie na jednotlivé stavy a vytvoriť pravidlá pre prechody medzi nimi [7]. Vďaka možnosti jednoduchého vizuálneho zobrazenia je taktiež možné diskutovať o nich aj s neprogramátorskými členmi tímu (napr. dizajnérmi). Vďaka týmto výhodám a ich pomerne jednoduchej možnosti implementácie sú veľmi populárne. V prípade potreby veľkého množstva rôznych stavov a prechodov však práca s nimi môže byť zložitá [2].

[17] Obrázok prevzatý z https://learn.unity.com/tutorial/navmesh-baking-1



Obr. 2.14: Príklad diagramu Finite State Machine.

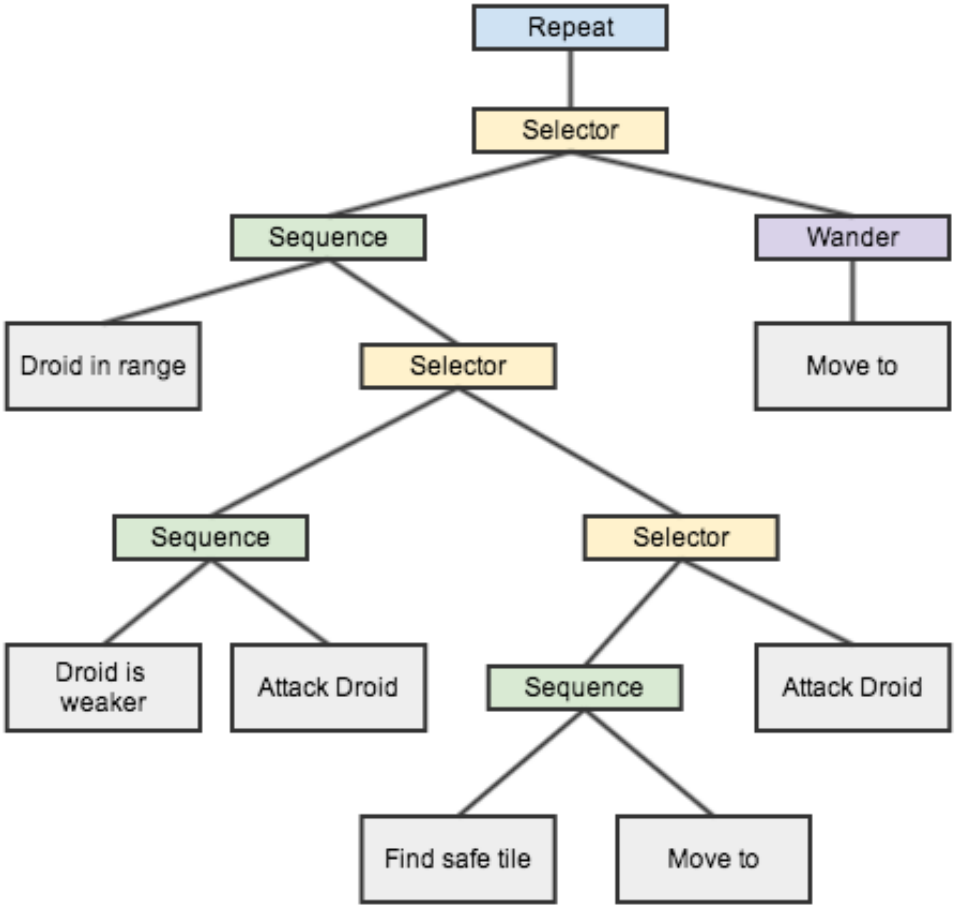
V jednoduchých konečných automatoch jednotlivé stavy reprezentujú konkrétnu akciu. V prípade potreby však stav môže reprezentovať ďalší, vnorený konečný automat. Po prechode do takéhoto stavu sa začne vykonávať vnorený FSM, pričom pôvodný FSM čaká na návrat. Takýto model je vhodný na implementáciu komplexnejšieho správania a nazýva sa *hierarchický konečný automat* [29].

Behaviour Tree

Behaviour Trees (BTs) sú v podstate podobné hierarchickým konečným automatom, ale na rozdiel od stavov sú ich stavebnými blokmi tzv. *nodes* [29]. BT je teda tvorený hierarchickým stromom nodes (viď obrázok 2.15), ktoré ovládajú správanie agenta umelej inteligencie. Každý node môže mať tri návratové stavy – *success*, *failure* alebo *running*. Ďalej sa nodes delia na tri typy [2]:

- **task** (nazývané aj **leaf (list)**) – node bez child objektov,
- **decorator (dekorátor)** – node s jedným child objektom,
- **composite (kompozit)** – node s viacerými child objektmi.

Leaf nodes reprezentujú konkrétnu akciu alebo podmienku (napr. kontrola či je hráč nablízku). Nodes tohto typu tvoria konce jednotlivých vetiev BT. Dekorátor je node, ktorý upravuje správanie alebo výsledok svojho child node. Ako príklad je možné uviesť dekorátory *Negate* a *Repeat* [2]. Dekorátor Negate invertuje návratovú hodnotu, dekorátor Repeat zase slúži na opakované vykonanie node. Dekorátory môžu ovplyvňovať listové alebo aj composite nodes. Composite node definuje začiatok vetvy stromu, spravuje teda skupinu child nodes a určuje ako sú vykonané. Medzi composite nodes patria napr. *Selector* alebo *Sequence* [2]. Selector vykonáva všetky svoje child nodes po poradí a vracia úspešnú návratovú hodnotu v okamihu, keď je jeden z týchto nodes úspešne vykonaný. V prípade vyčerpania vykonávateľných child nodes je návratovou hodnotou failure. Úspešné vykonanie Sequence nastane len v prípade, ak sú úspešne vykonané všetky príslušné child nodes.



Obr. 2.15: Príklad diagramu Behaviour Tree[18].

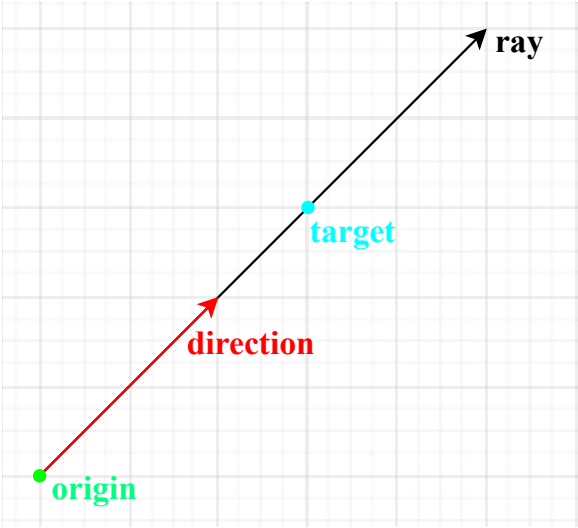
Jednotlivé nodes samozrejme musia komunikovať medzi sebou a aj s prostredím. Na toto slúži dátová štruktúra nazývaná *Blackboard* [2], s ktorou môžu nodes pracovať a zapisovať do nej alebo čítať z nej potrebné dáta.

2.7 Raycasting a jeho využitie

Raycasting je technika, pri ktorej sa pomocou využitia nekonečne tenkej čiary (lúča) dá zistiť či v danom smere nastane kolízia s nejakým objektom. Kolízia nastane, ak existuje priesečník medzi vytvoreným lúčom a nejakým objektom [2]. Na základe pozície kolízie sa jednoducho zisťuje vzdialenosť medzi objektmi. Raycasting je ďalej využiteľný na nájdenie prekážky medzi objektmi, určenie typu objektu, zistenie či agent umelej inteligencie „vidí“ hráča, alebo napr. aj klikanie na prvky užívateľského rozhrania.

Pri raycastingu je pre vytvorenie samotného lúča potrebná počiatočná pozícia a orientácia lúča. Napríklad v hernom engine Unity je možné tento lúč definovať pomocou dvoch vektorových premenných typu `Vector3` – vektor počiatočnej pozície $\vec{origin}$ a vektor určujúci

[18]Obrázok prevzatý z https://www.javacodegeeks.com/2014/08/game-ai-an-introduction-to-behaviour-trees.html



Obr. 2.16: Príklad smerovania lúča pri raycastingu.

smer lúča definovaný ako normalizovaná vzdialenosť medzi počiatočnou a cieľovou pozíciou (viď obrázok 2.16):

$$\vec{direction} = normalize(\vec{target} - \vec{origin}). \tag{2.11}$$

Veľký počet vysielaných lúčov za snímok môže mať negatívne dopady na výkon hry. Preto je vhodné hľadať kolíziu len s potrebnými objektmi, t.j. ignorovať niektoré objekty (napr. pomocou využitia rôznych *layers – vrstiev*), a zároveň určiť maximálnu dĺžku lúča.

Kapitola 3

Návrh hry

3.1 Koncept hry

Vytváraná hra primárne patrí do žánru roguelike (viď sekcia 2.2). Základné prvky roguelike hier sú rozšírené o prvky FPS hier – jedná sa teda o single-player (pre jedného hráča) strielačku z pohľadu prvej osoby v kombinácii s prvkami roguelike hier. Na vývoj hry je použitý herný engine Unity (viď sekcia 2.1). Hra je vytvorená v 3D prostredí. Hráč ovláda postavu, ktorá sa pohybuje v procedurálne generovaných dungeonoch. Na vývoj hry sú použité aj už existujúce assety, ktoré sú voľne dostupné alebo zakúpené cez oficiálny asset store Unity. Tieto assety sú hlavne použité pri tvorbe prostredia a nepriateľov.

Herné mechaniky

Návrh herných mechaník je pri vývoji hier dôležitým krokom. Pri navrhovaní jednotlivých mechaník ako nápoveda môžu slúžiť pravidlá roguelike dizajnu popísané v sekcii 2.2.

Hráč sa pohybuje v dungeone, ktorý je z dizajnového hľadiska štylizovaný na štýl podzemných ruin. Tento štylistický výber je doplnený vhodnou hudbou a zvukovými efektmi. Tento dungeon je rozdelený na úrovne. Každá úroveň je rozdelená na viaceré miestnosti. Hráč musí prechádzať medzi jednotlivými miestnosťami až kým nedosiahne poslednú miestnosť danej úrovne. Po dosiahnutí poslednej miestnosti je hráč presunutý do špeciálnej miestnosti, kde sa odohráva tzv. bossfight. Po úspešnom prejdení tejto miestnosti môže byť hráč presunutý do ďalšej úrovne. V prípade, že už bol hráč v poslednej úrovni, je hra ukončená.

Za úspešné prejdenie každej miestnosti je hráč odmenený. Odmeny môžu byť rôzne. Medzi základné odmeny patria napr. mince, ktoré môže hráč použiť na nákup predmetov alebo kľúče, ktoré môže hráč použiť na odomykanie dverí alebo truhlíc. Hráč môže taktiež získať vzácnejšie odmeny, ako napr. nové zbrane, predmety alebo použiteľné schopnosti. Zároveň má hráč po prejdení niektorých miestností možnosť vylepšiť niektorý z piatich atribútov svojej postavy. Jedným z možných typov miestností je obchod, v ktorom hráč môže použiť získané mince na zakúpenie rôznych predmetov a vylepšení.

Všetky predmety v hre nie sú odomknuté od začiatku hry. Niektoré predmety sú do zoznamu dostupných vecí postupne pridávané na základe herného progresu hráča. V prípade, ak hráč odomkne nejaký predmet, tento predmet bude možné pre hráča nájsť až po vstupe do ďalšieho dungeonu (nie úrovne). Nájdené predmety sú pomenované a krátko popísané, ale ich funkcionalita nie je pre hráča okamžite známa. Účinky predmetov zistí hráč až po ich použití. Použitie predmetov v bežných situáciách by nemalo spôsobiť zabitie hráča. Nájdenie nových predmetov môže zmeniť štýl hry hráča a poskytnúť hráčovi nové

možnosti, napr. rôzne typy zbraní. Dôležité je aby hráč bol schopný nájsť využitie pre každý existujúci predmet.

V jednotlivých miestnostiach sa nachádzajú nepriatelia a prípadné environmentálne hrozby, ktoré musí hráč prekonať. Hráč má možnosti na vyhýbanie sa týmto hrozbám – môže uskakovať útokom alebo preskočiť prekážky. Pre hráča by nemala byť žiadna z týchto hrozieb okamžite smrteľná. Je vhodné sa vyhnúť situácii, v ktorej by hráč s plným počtom životov bol zabitý na jeden zásah. Nakoľko hra je určená pre jedného hráča (nie je online) – nepriatelia sú ovládaní pomocou umelej inteligencie. V hre sa môže hráč stretnúť s viacerými typmi nepriateľov. Nepriatelia sa líšia vzhľadom, veľkosťou, útokmi, schopnosťami a hodnotami atribútov ako napr. ich počet životov alebo útočná sila. Obtiažnosť nepriateľov postupne rastie na základe aktuálnej úrovne. Tento nárast je rýchlejší než priemerný progres hráča. Z hľadiska environmentálnych hrozieb si hráč musí dávať pozor napr. na pád do vody, pasce alebo zhoršenú viditeľnosť.

Za jednotlivé akcie hráča je udeľované hodnotenie vo forme skóre. Celkové skóre môže byť navýšené napr. zabíjaním nepriateľov alebo prechádzaním miestností. Zároveň môže hráč dostať aj negatívne hodnotenie – skóre je znížené napr. keď je hráč zasiahnutý nepriateľom alebo v prípade úmrtia postavy hráča. Skóre sa znižuje aj samotným plynutím času, čo núti hráča aktívne prechádzať úrovne dungeonu.

Miera implementácie vlastností roguelike žánru

Na posúdenie miery implementácie jednotlivých vlastností žánru roguelike je možné sa inšpirovať knihou *Exploring Roguelike Games*, v ktorej autor *John Harris* hodnotí rôzne hry na základe faktorov Berlínskej interpretácie roguelike žánru (viď sekcia 2.2). Každý faktor je hodnotený hodnotou od 1 do 5, pričom nízko-hodnotové faktory majú v konečnom súčte polovičnú hodnotu.

Faktory s vysokou hodnotou

- **Náhodné generovanie prostredia** – náhodné generovanie dungeonu rozdeleného na miestnosti, ktoré sú vytvárané podľa šablón. Hodnotenie: 4.
- **„Permadeath“** – hráč má po smrti možnosť prechádzať ten istý dungeon, ale úplne od začiatku bez uloženia progresu. Hodnotenie: 4 – znížené kvôli počítaniu skóre aj pri ďalšom pokuse.
- **Ťahový systém** – hra je založená na reálnom čase. Hodnotenie: 1.
- **Založené na mriežke** – generovanie prostredia je založené na mriežke, ale samotná hra sa odohráva v 3D priestore. Hodnotenie: 2.
- **Nemodálne** – hráč môže vykonávať každú akciu jemu prístupnú v ľubovoľnom čase a priestore. Hodnotenie: 5.
- **Komplexnosť** – prechod herným levelom je možný s použitím rôznych zbraní, predmetov a vylepšení. Predmety neposkytujú veľkú mieru vzájomnej interakcie. Hodnotenie: 2.
- **Spravovanie zdrojov** – zdroje (mince, kľúče, elixíry na liečenie) majú jedno využitie, ale pre optimálne využitie ich hráč musí správne používať. Hodnotenie: 3.
- **„Hack’n’slash“** – zabíjanie nepriateľov je dôležitou časťou hry. Prechod medzi miestnosťami je možný až po zabití všetkých nepriateľov v miestnosti. Hodnotenie: 5.

- **Preskúmavanie a objavovanie** – hráč preskúmava miestnosti kvôli novým vylepšeniam/zbraniam/predmetom alebo použiteľným zdrojom (mince, kľúče, elixíry); ako motivácia slúži aj bodová odmena za vyčistenú miestnosť. Hodnotenie: 4.

Faktory s nízkou hodnotou

- **Jedna postava hráča** – hráč ovláda jednu postavu. Hodnotenie: 5.
- **Nepriatelia sú podobní hráčom** – nepriatelia a hráč sú založení na rovnakom základe – napr. životy, aplikácia rôznych efektov. Hráč má ale rozšírenia navyše (napr. atribúty, štít, inventár). Hodnotenie: 2.
- **Taktická výzva** – získanie znalostí o hre môže urýchliť progres, ale nie je nevyhnutné. Hodnotenie: 1.
- **ASCII zobrazenie** – hra je 3D. Hodnotenie: 1.
- **Dungeons** – herné prostredie tvorené z miestností a koridorov. Hodnotenie: 5.
- **Čísla** – základné zobrazenie života je čisto vizuálne, v rozšírenom zobrazení je ale vidieť priamo čísla a aj hodnoty atribútov. Skóre je viditeľné v číselnej podobe stále. Hodnotenie: 3.

Faktor	**Názov hry**						
	Dungeon Hack	**ToeJam &Earl**	**Shiren**	**NetHack**	**Dungeon Crawl**	**Diablo**	**Navrh. hra**
Náhodné generovanie prostredia	5	4	4	4	5	3	4
Permadeath	2	4	4	5	5	2	4
Ťahový systém	2	1	5	5	5	1	1
Založené na mriežke	4	1	5	5	5	1	2
Nemodálne	5	4	5	5	4	4	5
Komplexnosť	2	4	5	5	4	1	2
Spravovanie zdrojov	3	4	5	4	5	1	3
"Hack'n'slash"	5	1	5	5	5	5	5
Preskúmavanie a objavovanie	5	4	5	5	5	2	4
Jedna postava hráča	5	5	5	5	5	5	5
Nepriatelia sú podobní hráčom	2	1	3	5	4	1	2
Taktická výzva	1	3	5	4	5	3	1
ASCII zobrazenie	1	1	1	5	5	1	1
Dungeons	5	1	3	5	5	5	5
Čísla	5	1	5	5	5	5	3
SÚČET	**42.5**	**33**	**54**	**57.5**	**57.5**	**30**	**38.5**

Tabuľka 3.1: Tabuľka hodnotenia miery implementácie roguelike žánru[1].

[1] Faktory s nízkou hodnotou sú v tabuľke označené kurzívou.

Maximálne možné skóre podľa tohto hodnotiaceho systému je 60, minimum je 12. V tabuľke 3.1 je porovnanie hodnotenia navrhnutej hry s hodnotením vybraných hier podľa Johna Harrisa. Podľa jeho názoru by väčšina hier, ktoré nemajú nič spoločné s roguelike žánrom, dostala prinajlepšom približne 20 bodov [18].

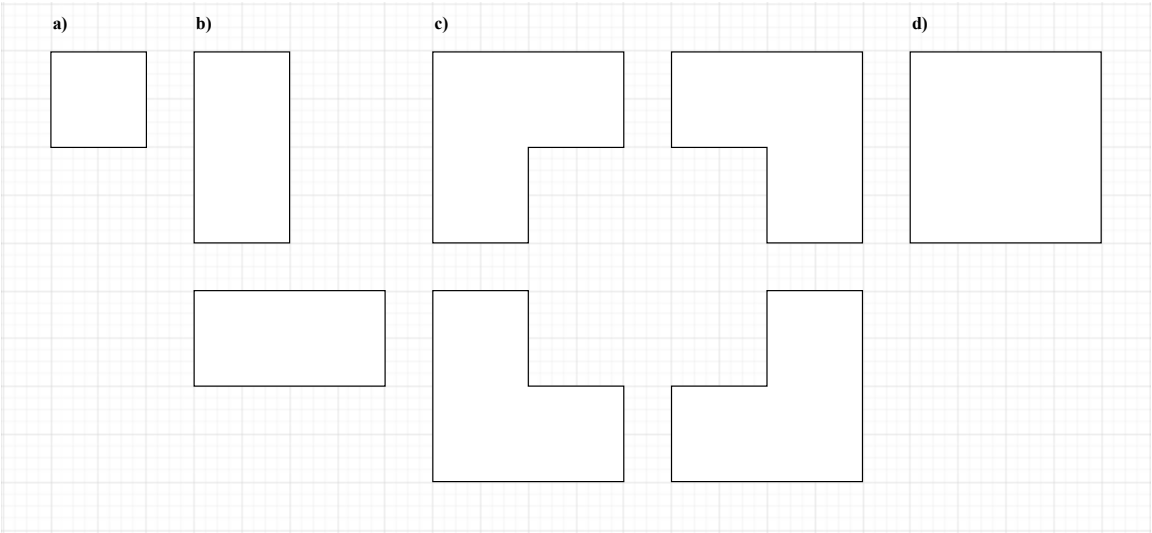
3.2 Návrh herného prostredia

Hra využíva procedurálne generovanie na vytváranie jednotlivých úrovní dungeonu, v ktorom sa hráč pohybuje. Úrovne dungeonu sú mriežkovito rozdelené, je možné ich reprezentovať maticou s rozmermi 10x10. Na základe tejto mriežky sú umiestňované rôzne typy miestností (viď obrázok 3.1). Podľa veľkosti sú miestnosti rozdelené na 4 základné typy:

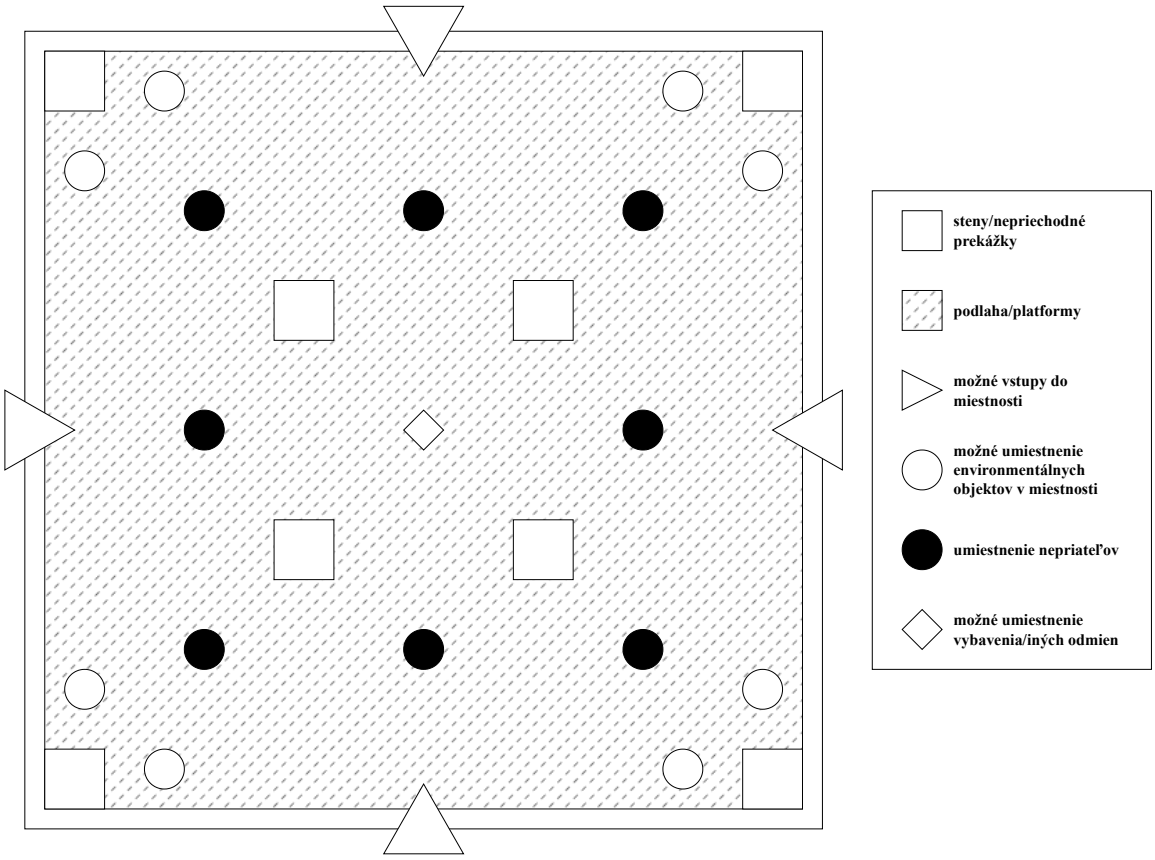
- Typ a) – najmenšia miestnosť v tvare 1x1 štvorca.
- Typ b) – dve miestnosti obdĺžnikového tvaru 2x1 alebo 1x2.
- Typ c) – štyri miestnosti v tvare L skladajúce sa z 3 štvorcov.
- Typ d) – najväčšia miestnosť v tvare 2x2 štvorca.

Pre každý z týchto typov miestností existujú rôzne variácie. Tieto variácie určujú špecifiká danej miestnosti.

Okrem týchto štyroch základných typov miestností existuje v hre aj špeciálna miestnosť. Táto miestnosť je koncovou miestnosťou jednotlivých úrovní. Jej úspešné prejdenie umožňuje hráčovi prechod do ďalšej úrovne alebo ukončenie hry. Na rozdiel od základných typov miestností nie je umiestňovaná náhodne, ale má fixnú pozíciu mimo základnej mriežky vytvoreného prostredia. Odlišuje sa aj svojou veľkosťou – je o niečo väčšia ako najväčšia miestnosť zo základných typov miestností (typ d).



Obr. 3.1: Základné typy miestností.



Obr. 3.2: Príklad návrhu rozloženia miestnosti.

Jednotlivé miestnosti sú generované na základe šablón. Tieto šablóny sú vytvorené manuálne. Popisujú rôzne možnosti vzhľadu a rozloženia miestnosti. Parametre miestnosti zahŕňajú:

- **Ohraničenie miestnosti** – určujú pozíciu stien, podlahy a stropu miestnosti.
- Pozície, na ktorých môžu existovať **vstupy** do miestnosti.
- **Vybavenie miestnosti** – možné umiestnenia rôznych predmetov bez reálneho využitia v hre (slúžia len ako dekorácia).
- **Predmety v miestnosti** – pozície, kde je možné umiestniť hráčom použiteľné predmety, vybavenie alebo iné odmeny.
- **Nepriatelia v miestnosti** – pozície určujúce možné umiestnenie nepriateľov v miestnosti. Každá šablóna môže mať priradených viacero týchto rozložení nepriateľov, z ktorých je vždy vybrané jedno rozloženie.

Príklad šablóny pre jednu variáciu miestnosti typu a) je znázornený na obrázku 3.2.

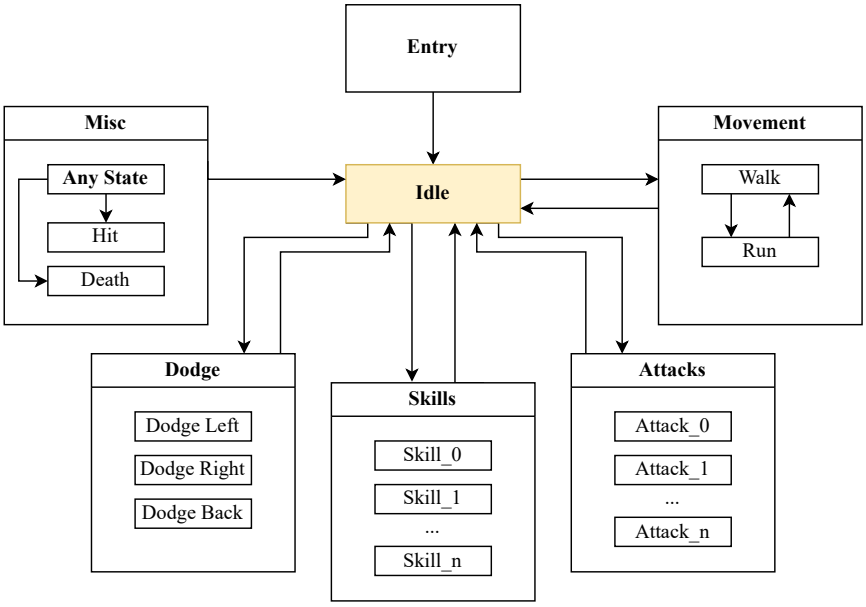
3.3 Koncepcia umelej inteligencie

Na realizáciu umelej inteligencie nepriateľov sa používa konečný stavový automat (FSM, viď *Finite State Machine* v sekcii 2.6). FSM má 6 základných stavov – 1 základný stav a 5 ďalších stavov riadiacich rôzne akcie nepriateľa. Každý z týchto piatich rozšírených stavov je vlastne sám ďalším FSM – presnejšie sa teda jedná o hierarchické FSM (viď obrázok 3.3).

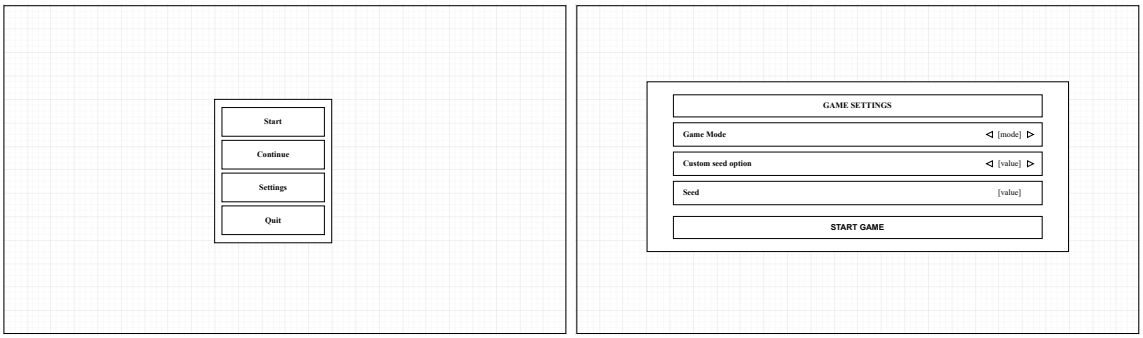
Každý stav predstavuje konkrétnu funkciu nepriateľa a jej možné akcie:

- **Idle** – základný stav, počas ktorého je nepriateľ v kľudovom stave. Tento stav slúži ako východzí stav v prípade, že žiadny z ostatných stavov nie je možné vykonať. Nie je rozdelený na ďalšie stavy.
- **Movement** – stav riadiaci pohyb nepriateľa. Obsahuje dva stavy – jeden určený na chôdzu a ďalší na beh.
- **Attacks** – stav riadiaci jednotlivé útoky. Počet stavov závisí od typu nepriateľa.
- **Skills** – stav riadiaci použiteľné schopnosti. Počet stavov závisí od typu nepriateľa.
- **Dodge** – stav riadiaci uhýbanie sa. Obsahuje tri stavy určujúce smer uhnutia – vľavo, vpravo alebo dozadu. Existencia tohto stavu je voliteľná, záleží od typu nepriateľa.
- **Misc** – stav riadiaci ostatné akcie, ako je napríklad reakcia na zasiahnutie alebo smrť nepriateľa. Tento stav je špeciálny tým, že nemá prepojenie so stavom idle ako ostatné stavy. Jeho vykonanie môže nastať kedykoľvek a dokáže prerušiť ostatné stavy.

Všetky stavy hlavného FSM a aj jednotlivých vnorených FSM majú vážené prechody. Tieto váhy prechodov sa môžu dynamicky meniť na základe aktuálneho stavu nepriateľa alebo aj pozície hráča. Výber stavov prebieha náhodným výberom s využitím pseudonáhodného generátora hodnôt (viď sekcia 2.3).



Obr. 3.3: Návrh konečného stavového automatu pre umelú inteligenciu nepriateľov v hre.



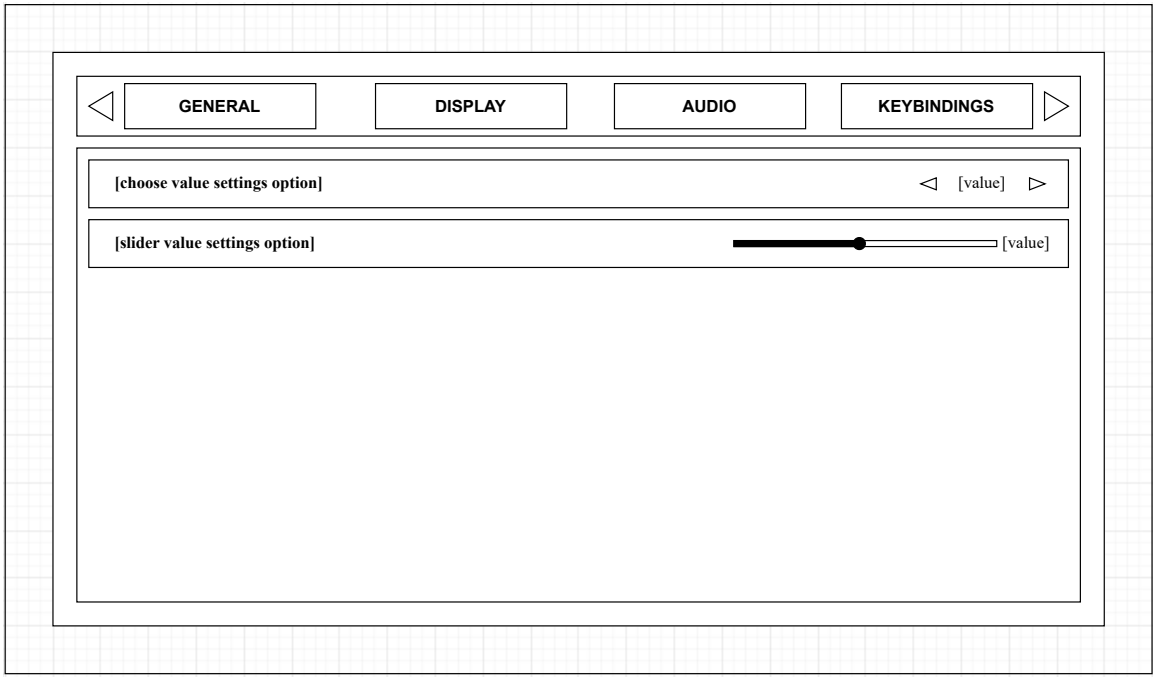
Obr. 3.4: Návrh hlavného menu hry.

3.4 Grafické užívateľské rozhranie

Užívateľské rozhranie (UI) je možné rozdeliť na tri hlavné časti – hlavné menu, nastavenia a herné UI.

Hlavné menu

Hlavné menu hry obsahuje základné možnosti ako začatie hry, otvorenie nastavení alebo vypnutie hry (viď obrázok 3.4). Jedná sa o jednoduché východzie menu, ktorého funkciou je prepínanie do ďalších zobrazení. Pod hlavné menu patrí aj menu na začatie hry, ktoré sa zobrazí vybratím možnosti začatia hry. Zobrazuje vybranú obtiažnosť a náhodný seed na generovanie herného prostredia.



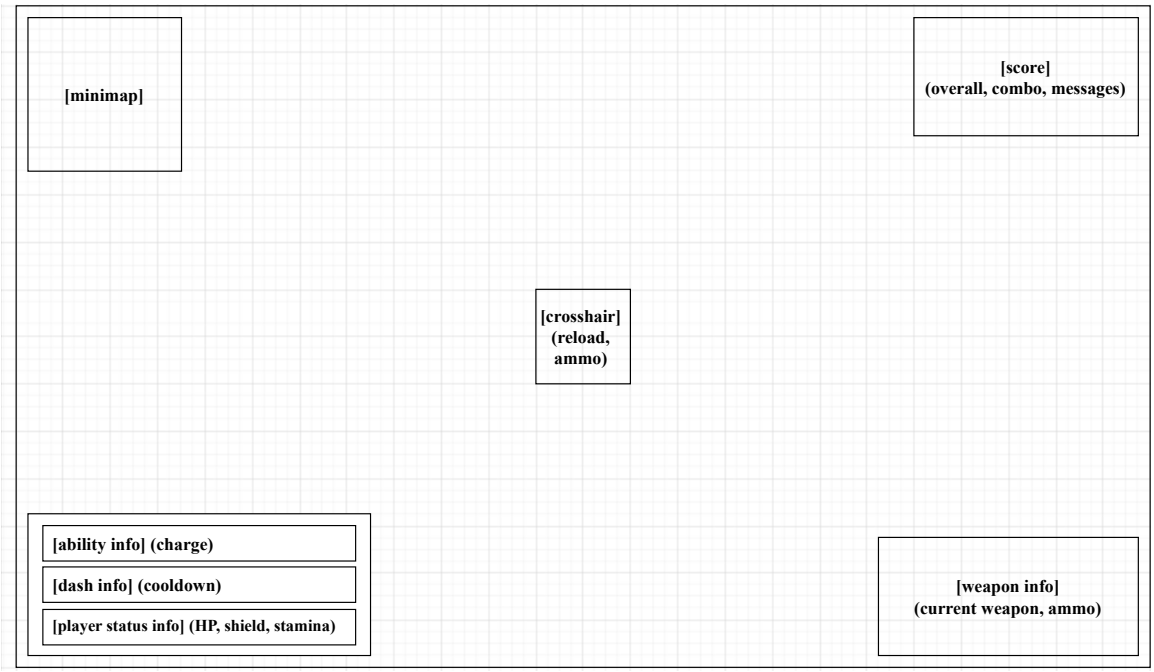
Obr. 3.5: Návrh menu pre všeobecné nastavenia hry.

UI pre nastavenia hry

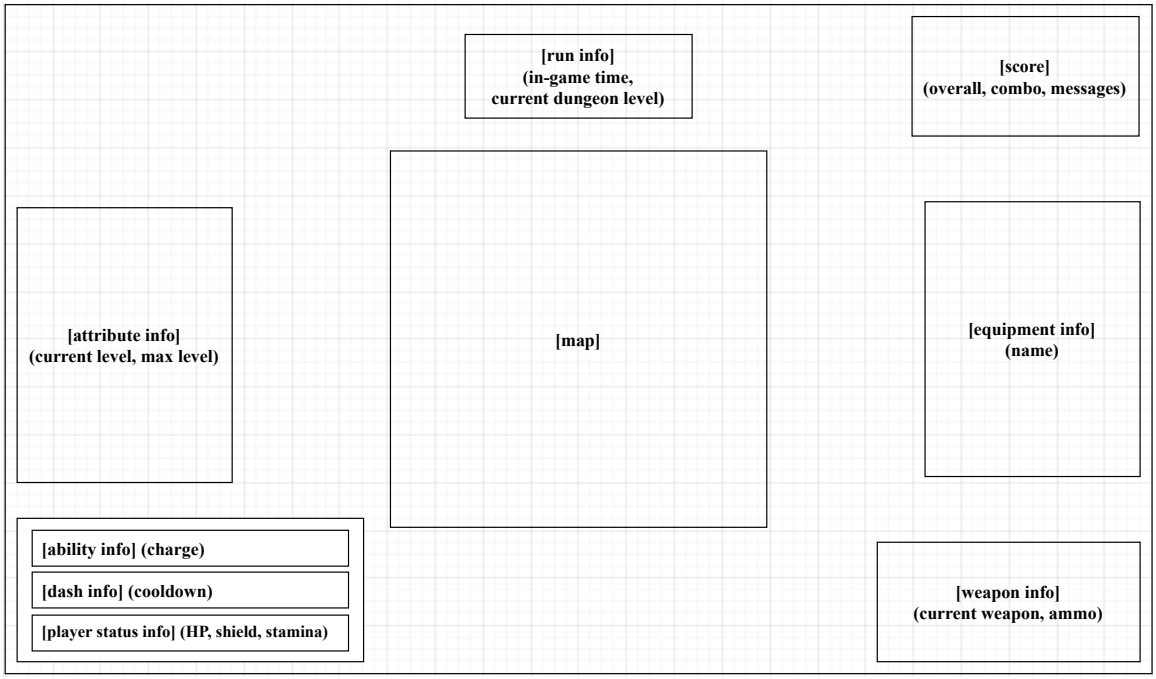
Užívateľské rozhranie pre nastavenia hry je rozdelené na štyri sekcie – nastavenia všeobecné, vizuálne, zvukové, nastavenia ovládania hry (viď obrázok 3.5). V týchto sekciách sa používajú dva typy ovládania nastavovaných hodnôt – prepínacie tlačidlo alebo posuvný slider. V sekcii všeobecných nastavení si hráč môže nastaviť napr. citlivosť pohybu myši alebo zobrazovanie počítadla snímkov za sekundu. Sekcia vizuálnych nastavení obsahuje možnosti určenia základných grafických nastavení ako napr. nastavenie rozlíšenia hry. V sekcii audio nastavení má hráč možnosť nastaviť hlasitosť zvukových efektov hry a hudby. Posledná sekcia nastavení umožňuje hráčovi prestaviť si východzie nastavenia ovládania hry.

Herné UI

Najdôležitejšou časťou užívateľského rozhrania je herné UI (viď obrázok 3.6). V ľavom dolnom rohu sa nachádzajú elementy zobrazujúce aktuálny stav postavy hráča. Zahrnuté sú tu informácie o počte životov, prípadnej hodnoty štítu, dostupnej energie a aj o možnej informácii o použiteľnej schopnosti. V ľavom hornom rohu je zobrazená minimapa – miniatúrny náhľad na mapu aktuálnej úrovne dungeonu. V pravom dolnom rohu sú viditeľné informácie o dostupných zbraniach. Zobrazený je tu aj počet nábojov pre dané zbrane. V pravom hornom rohu je ukazovateľ aktuálneho skóre. V tejto oblasti sa zobrazujú aj správy o pozitívnom alebo negatívnom hodnotení hráča. V strede obrazovky sa nachádza mieridlo a prípadne aj indikátor nabíjania alebo varovanie o nutnosti nabitia. V prípade nízkeho počtu životov je hráč upozornený zmenou farby krajov obrazovky.



Obr. 3.6: Návrh základného užívateľského rozloženia v hre.



Obr. 3.7: Návrh rozšíreného užívateľského rozloženia v hre.

Rozšírené herné UI

V hre má hráč možnosť zobraziť rozšírenie štandardného herného užívateľského rozhrania (viď obrázok 3.7). Toto zobrazenie poskytuje hráčovi viac informácií o priebehu hry. Minimapa je v tomto zobrazení nahradená celou mapou dungeonu. V ľavej časti obrazovky môže hráč vidieť atribúty svojej postavy. Rozšírené je aj zobrazenie aktuálneho stavu hráča – okrem vizuálneho zobrazenia počtu životov a štítu sú tieto hodnoty zobrazené aj číselne. Informácie o aktuálne používaných predmetoch môže hráč vidieť v pravej časti obrazovky. V strede hornej časti obrazovky je k dispozícii informácia o aktuálnej úrovni dungeonu a počítadlo uplynutého času.

Kapitola 4

Implementácia návrhu hry

Na implementáciu návrhu hry je využitý herný engine Unity verzie 2021.3.7f1. Vytvorené zdrojové kódy sú v jazyku C#. Pri vývoji hry sú použité aj rôzne už existujúce assety – voľne dostupné alebo zakúpené v asset obchode Unity Store. Bližšia špecifikácia týchto assetov je popísaná v relevantných sekciách tejto kapitoly. Na vytváranie niektorých grafických assetov je využitý softvér GIMP verzie 2.10.32.

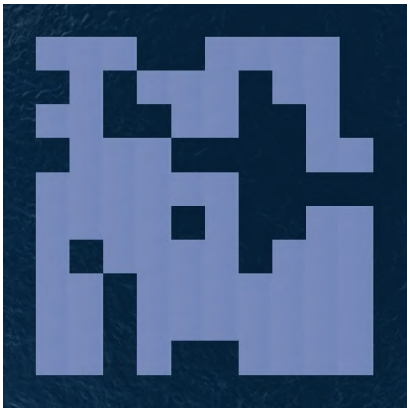
Táto kapitola práce je rozdelená do štyroch sekcií popisujúcich rôzne časti implementácie a piatej časti zameranej na užívateľské testovanie.

4.1 Generovanie herného prostredia

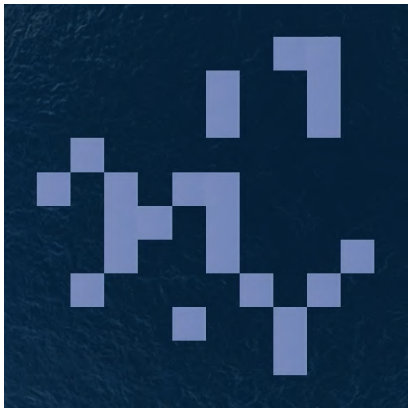
Vytváranie herného prostredia je realizované pomocou kombinovaného prístupu (viď sekcia 2.5). Rozloženie jednotlivých miestností v úrovni je generované náhodne. Každý typ miestnosti má manuálne vytvorené šablóny, podľa ktorých je neskôr generovaný obsah daných miestností. Pred začiatkom samotného procesu generovania úrovne je vytvorený seed využitý pri generovaní. Tento seed je reprezentovaný 8-znakovým reťazcom skladajúcim sa z náhodne vybraných alfanumerických znakov. Toto umožňuje jednoduché opakované použitie pri debuggingu a testovaní. Seed v podobe reťazca však samozrejme nie je možné priamo použiť na inicializáciu generátora `Random` v Unity. Jeho číselná hodnota je vypočítaná pomocou MD5 hashovacej funkcie.

Vytvorenie rozloženia generovanej úrovne

Po inicializácii generátora `Random` môže začať samotný proces generovania úrovne. Prvým krokom je určenie počtu jednotlivých typov miestností (viď sekcia 3.2). Rozsah možných hodnôt záleží od aktuálnej úrovne dungeonu – počet miestností sa mení podľa toho, ako hráč postupuje v hre. Každý level generovaného dungeonu je reprezentovaný mriežkou 10x10. Vytvorená je matica s touto veľkosťou obsahujúca objekty reprezentujúce jednotlivé polia mriežky. Polia mriežky reprezentujú prázdny priestor alebo priestor, ktorý môže byť súčasťou miestnosti. Hodnoty matice sú inicializované s využitím Perlinovho šumu (viď sekcia 2.4) – ak je hodnota pozície v matici pod určenou hranicou, pole v mriežke je označené ako prázdne (viď obrázok 4.1a). Hodnota tejto hranice je opakovane zvyšovaná a postupne sú odstraňované nadbytočné polia mriežky aktuálne generovanej úrovne, až kým nie je dosiahnutý požadovaný počet polí potrebný na vytvorenie predtým určeného počtu miestností (viď obrázok 4.1b).

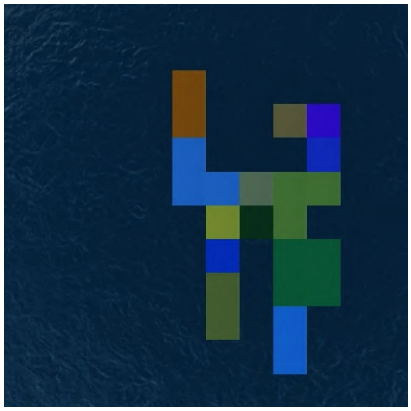


(a) Inicializácia hodnôt pomocou Perlinovho šumu



(b) Odstránenie nadbytočných polí

(c) Spojenie jednotlivých skupín do celku

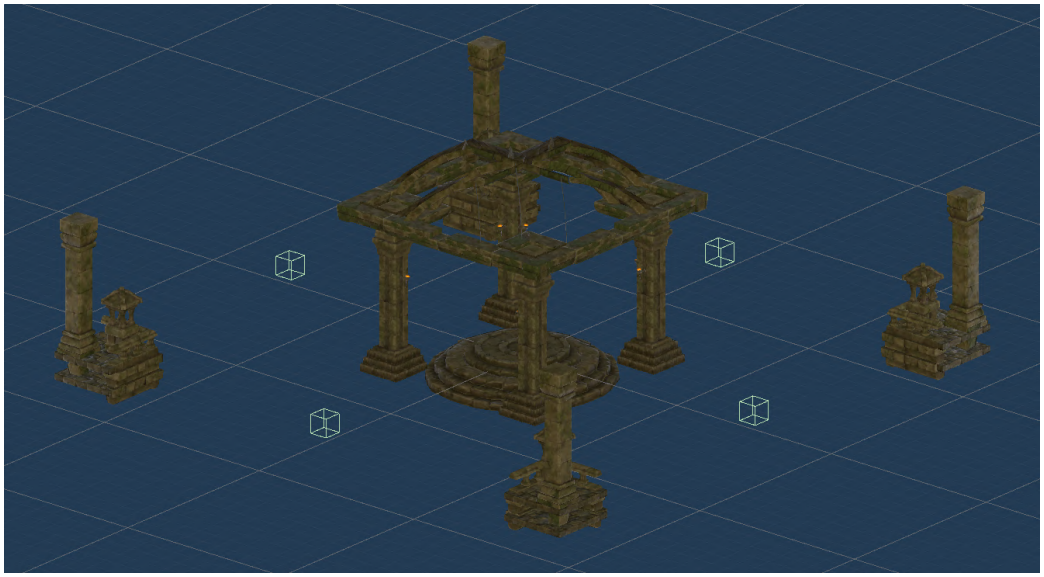


(d) Rozdelenie na miestnosti

Obr. 4.1: Príklad priebehu procesu generovania rozloženia úrovne herného dungeonu.

Týmto procesom vzniknú oddelené skupiny polí, ktoré je potrebné spojiť do jedného celku. Spojenie je možné dosiahnuť presunom jednotlivých skupín po mriežke. Presun začína najmenšou skupinou, ku ktorej sú postupne pripájané ďalšie skupiny. Tento proces sa ukončí v momente, keď všetky polia patria do jednej skupiny (viď obrázok 4.1c). Následne sú určené pozície na mriežke, na ktoré je možné umiestniť jednotlivé typy miestností. Výber pozície miestností začína od najväčšieho typu miestnosti. Príklad konečného stavu tohto procesu je znázornený na obrázku 4.1d.

Pred samotným začatím generovania jednotlivých miestností ako herných objektov je nutné určiť prechody medzi miestnosťami. Z miestností typu a) sa náhodne vyberie jedna miestnosť, ktorá je označená ako začiatočná. Následne sú pomocou algoritmu A* (viď sekcia 2.6) nájdené najkratšie trasy z tejto miestnosti do ostatných miestností. Podľa nájdených ciest medzi miestnosťami sú určené prechody medzi miestnosťami – týmto sú určené susedné miestnosti. Súčasne je pri tomto procese určená najvzdialenejšia miestnosť, ktorá potom slúži na prechod do finálnej miestnosti úrovne. V prípade, že existuje aspoň jedna miestnosť typu a), ktorá má len jedného suseda, a zároveň nie je začiatočnou ani najvzdialenejšou miestnosťou, táto miestnosť je označená ako obchod.



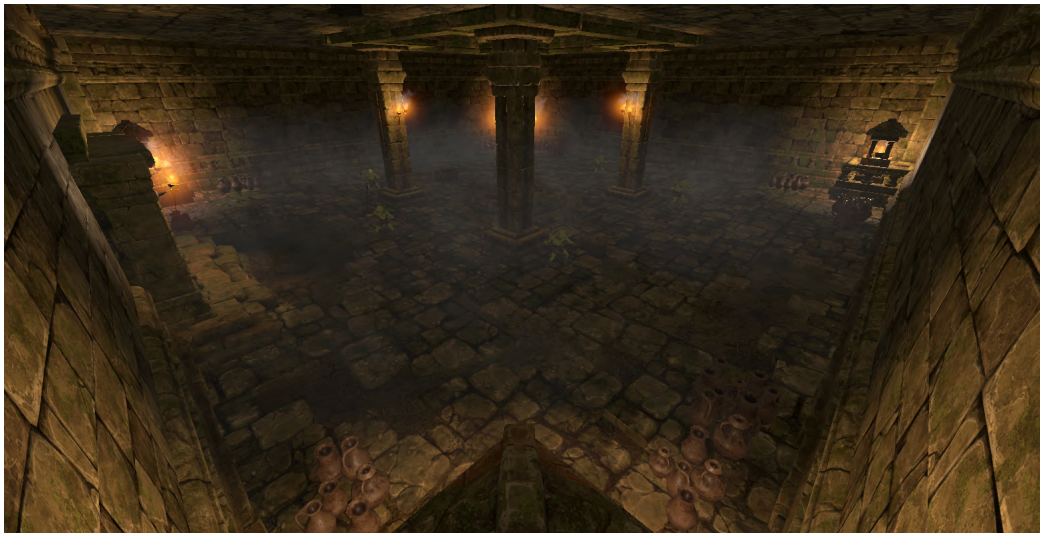
Obr. 4.2: Príklad šablóny pre miestnosť typu a) – znázornené sú určité fixne dané prvky miestnosti a možné pozície pre vstupy do miestnosti.

Spawnovanie úrovne

Po určení rozloženia miestností hernej úrovne nasleduje spawnovanie miestností – vytvorenie herných objektov miestností. Základ každej miestnosti je tvorený manuálne vytvorenou šablónou. Táto šablóna určuje to, ako môže miestnosť vyzerať. Obsahuje pozície, na ktoré môžu byť umiestnené rôzne prvky miestnosti. Okrem týchto variabilných prvkov môže šablóna obsahovať aj fixne dané prvky miestnosti (viď obrázok 4.2). Modely a textúry jednotlivých prvkov boli zakúpené z obchodu Unity Asset Store.

Na základe vytvoreného rozloženia miestností je pre každú miestnosť vybraná vhodná šablóna zo zoznamu pripadajúcemu danému typu miestnosti. Každá šablóna je inštancovaná ako herný objekt v scéne a jej ovládací skript je uložený do zoznamu aktuálnej úrovne. Okrem miestností na základnej mriežke sa na fixnej pozícii vytvorí aj miestnosť navyše, ktorá slúži na ukončenie úrovne. Počas spawnovania objektov herných miestností súčasne prebieha aj vytváranie vstupov do miestností. Každá miestnosť má určené pozície, na ktorých sa môžu nachádzať vstupy do miestnosti. Po vytvorení miestnosti je z každej takejto pozície vyslaný lúč pomocou raycastingu (viď sekcia 2.7). V prípade, ak tento lúč narazí na objekt určujúci možný vstup inej miestnosti, podľa rozloženia úrovne skontroluje či má byť medzi týmito dvomi miestnosťami prechod. Ak sú tieto miestnosti označené ako susedné, tak sa na spoločnej stene oboch miestností vytvoria vchody.

Po ukončení inštancovania šablón postupne ovládací skript každej miestnosti zavolá funkcie na generovanie rôznych variabilných prvkov, ako napr. stien, podlahy, stropu a možných prvkov interiéru. Zároveň sú pre miestnosti generovaní aj nepriatelia. Ku každej šablóne sú priradené viaceré možné zoznamy nepriateľov. Tieto obsahujú informácie o jednotlivých pozíciach, na ktorých sa môžu objaviť nepriatelia. Zároveň sú ku každej takejto pozícii priradené možné typy nepriateľov. Pred generovaním nepriateľov je ešte nutné vytvoriť NavMesh povrch (viď *Unity NavMesh* v sekcii 2.6), po ktorom sa môžu neskôr pohybovať. Verzia Unity používaná na vývoj hry vytváranie tohto povrchu za behu však



Obr. 4.3: Ukážka možnej finálnej podoby vygenerovanej miestnosti.

neumožňuje. Z tohto dôvodu je použité experimentálne rozšírenie *NavMeshComponents*[1], ktoré túto funkcionality podporuje.

Posledným krokom pri vytváraní úrovne dungeonu je rozmiestnenie použiteľných predmetov a iných odmien. Tieto odmeny sú rozdelené do dvoch základných skupín – odmeny za dokončenie miestnosti a odmeny získané zabitím nepriateľa. Odmeny sú vyberané z viacerých zoznamov podľa typu odmeny. K dispozícii sú tri hlavné zoznamy odmien (zbrane, predmety a schopnosti), ktorých prvky nemôžu byť použité v jednej úrovni opakovane. Opakovanie v ďalších úrovniach je možné, ak si hráč túto odmenu v danej úrovni nezoberie. Okrem týchto zoznamov existujú aj zoznamy opakujúcich sa odmien (mince, kľúče, elixíry). Odmenou za dokončenie miestnosti je vždy truhla obsahujúca nejaký náhodne vybraný predmet zo zoznamu možných predmetov. Existujú dva typy truhlíc – truhla čerpajúca zo zoznamu opakovateľných predmetov a vzácnejšia truhla využívajúca jeden z troch hlavných zoznamov odmien. V každej miestnosti existuje šanca, že bude vytvorený objekt slúžiaci na vylepšenie postavy hráča. Odmeny za zabitie nepriateľa sú vyberané zo zoznamu opakovateľných predmetov, prípadne nemusí byť odmena žiadna.

Špecifickým prípadom sú predmety poskytované v miestnosti typu obchodu – existuje viacero objektov slúžiacich na predaj predmetov. Každému z týchto objektov je v šablóne fixne priradený nejaký zoznam predmetov, z ktorého môže potom pri generovaní čerpať.

Výsledok procesu generovania konkrétnej miestnosti dungeonu je vidieť na obrázku 4.3. Na obrázku je možné vidieť vygenerovaný interiér miestnosti, rozloženie nepriateľov, odmenu za dokončenie miestnosti, ale aj jeden vchod do susednej miestnosti.

4.2 Vytvorenie nepriateľských postáv

Nevyhnutnou súčasťou hry sú objekty nepriateľských postáv. V hre je zakomponovaných viacero typov nepriateľských postáv. Základným delením týchto postáv je ich rozdelenie na regulárnych nepriateľov, „minibossov“ (väčší a silnejší nepriatelia vyskytujúci sa v niektorých miestnostiach) a „bossov“ (najsilnejší nepriatelia nachádzajúci sa vždy v poslednej

[1] Dostupné na https://github.com/Unity-Technologies/NavMeshComponents

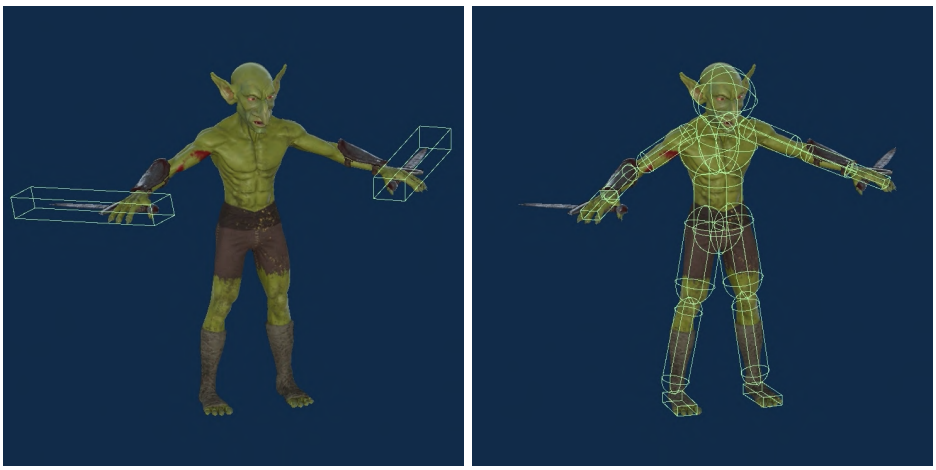
miestnosti úrovne). Vizuálne prvky postáv (modely, textúry a animácie) sú prevzaté z assetu zakúpeného z obchodu Unity Asset Store.

Objekt nepriateľa

Objekt nepriateľa sa skladá z viacerých častí. Základným prvkom je vybraný model nepriateľa s prislúchajúcimi animáciami. Na základe tohto modelu je možné vytvoriť *hitboxy* a *hurtboxy*. Hitboxy (viď obrázok 4.4a) sú využité pri útokoch nepriateľa na zasiahnutie hráča. Hurtboxy (viď obrázok 4.4b) naopak definujú oblasti, ktoré môže zasiahnuť hráč a tým nepriateľa zraniť. Hitboxy aj hurtboxy sú vytvárané pomocou `Collider` objektov, ktoré zabezpečujú kolízie s inými objektami v hre. Pre zabezpečenie správneho fungovania majú hitboxy aj hurtboxy určenú svoju vlastnú kolíznu vrstvu. Kolízia hurtboxov je možná len s útokmi hráča. Hitboxy nepriateľa môžu interagovať len s objektom hráča. Interakcia hitboxov s hráčom je však možná len počas určitej časti animácie útoku. Na vytvorenie tohto časového okna sú použité animačné eventy (udalosti). V každej animácii útoku nepriateľa je určený snímok, v ktorom sa pomocou skriptu zapne určitá skupina hitboxov. Zároveň je taktiež určený snímok, v ktorom je táto skupina potom opätovne deaktivovaná.

Okrem colliderov vytvorených pre hitboxy a hurtboxy, objekt nepriateľa obsahuje aj collider zabezpečujúci regulárnu kolíziu s hráčom. Tento collider má taktiež definovanú svoju vlastnú kolíznu vrstvu. Pri kolízii s hráčom je na hráča aplikované kontaktné poškodenie.

Ďalšími dôležitými prvkami objektu nepriateľa sú dva základné skripty. Prvý z nich zaznamenáva aktuálny stav nepriateľa. Uložené sú v ňom informácie o počiatočných, maximálnych aj aktuálnych hodnotách statusu nepriateľa (hodnoty ako napr. život, sila útoku alebo rýchlosť). Tento skript taktiež spravuje rôzne dočasné efekty (viď sekcia 4.3), ktoré môžu byť aplikované na nepriateľa (napr. dočasné navýšenie sily útoku). Zároveň tento skript obsahuje aj rozdelenie rôznych colliderov hurtboxu do častí tela nepriateľa. Tieto časti majú svoj vlastný počet života, ovplyvňujú poškodenie, ktoré hráč spôsobí zásahom, a pri ich zničení môže byť aplikovaný nejaký efekt na nepriateľa. Druhým dôležitým skriptom je skript, ktorý riadi napr. prepínanie animácií, prehrávanie zvukov, aktivovanie odmeny za zabitie nepriateľa alebo aj zničenie objektu nepriateľa po jeho smrti.



(a) Hitboxy daného nepriateľa (b) Hurtboxy daného nepriateľa

Obr. 4.4: Porovnanie hitboxov a hurtboxov nepriateľa.

Objekt nepriateľa má k sebe priradený aj svoj vlastný prvok UI. Tento prvok zobrazuje jeho aktuálny počet života, mení sa na základe hodnoty v skripte stavu nepriateľa. Viditeľný však nie je stále, jeho viditeľnosť závisí od toho, či hráč našiel potrebný predmet.

Umelá inteligencia a pohyb nepriateľa

Dôležitou časťou vytvárania objektu nepriateľa je zabezpečenie možnosti pohybu a vykonávania rôznych akcií. Na zabezpečenie pohybu v hernom prostredí je použitý NavMesh systém (viď *Unity NavMesh* v sekcii 2.6). Objektu nepriateľa je priradený *NavMesh agent*, ktorý je používaný NavMesh systémom na vyhýbanie sa prekážkam v prostredí. Každý agent má nastavenú určitú veľkosť (presnejšie sa určuje výška a polomer valca). Taktiež sú určené rýchlosť otáčania, zrýchlenie, vzdialenosť od cieľa, v ktorej má zastaviť, a maximálna rýchlosť pohybu (nastavená podľa hodnoty uloženej v skripte stavu nepriateľa). Cieľ pohybu je za bežných okolností určený podľa pozície hráča. Nejedná sa však o priamu pozíciu hráča, táto pozícia je upravená podľa aktuálneho pohybu hráča – nepriateľ „predvída“ smer pohybu hráča. Trasa k cieľu je prepočítavaná pomocou NavMesh implementácie A* algoritmu a samotný pohyb je vypínaný a zapínaný podľa aktuálne vykonávanej akcie.

Umelá inteligencia nepriateľa je implementovaná ako hierarchický konečný automat (ďalej FSM) (viď *Finite State Machine* v sekcii 2.6). Hlavný FSM teda obsahuje ďalšie vnorené FSM ako svoje stavy. Každý z týchto vnorených FSM obsahuje jednotlivé vykonateľné akcie. Skript hlavného FSM taktiež obsahuje frekvenciu, ako často sa má vykonať rozhodovacia funkcia. Zahrnuté sú v ňom aj pravdepodobnosti vykonania jednotlivých stavov.

Po uplynutí času určeného frekvenciou sa v prípade, ak už nie je vykonávaná nejaká akcia, vyhodnotí rozhodovacia funkcia. Úlohou tejto funkcie je vybrať ďalšiu akciu, ktorá bude vykonaná. Tento proces začína vyhodnotením toho, ktoré zo stavov je možné vykonať. Pre každý zo stavov prebehne kontrola jeho akcií – pokiaľ je možné vykonať aspoň jednu z nich, stav je označený ako vykonateľný. Každá z týchto akcií môže mať iný spôsob kontroly svojej vykonateľnosti – môže sa jednať napr. o kontrolu vzdialenosti k hráčovi alebo kontrolu toho, či sa hráč nachádza v triggeri priradenom k objektu danej akcie. Na základe počtu vykonateľných stavov sú prepočítané pravdepodobnosti prechodov k týmto stavom. Následne je náhodne vybraný jeden z týchto stavov a sú prepočítané pravdepodobnosti jeho akcií. Z týchto akcií je náhodne vybraná jedna akcia, ktorá je následne aj vykonaná. Pred jej vykonaním je uzamknutý rozhodovací proces. Akcia zavolá funkciu hlavného ovládacieho skriptu nepriateľa, ktorá spustí vybranú animáciu prislúchajúcu k tejto akcii. Zároveň začne odpočítavať cooldown – čas, ktorý musí uplynúť pred ďalším možným vykonaním tejto akcie. Rozhodovací proces je znovu odomknutý po skončení animácie akcie. Po každom vykonaní rozhodovacej funkcie sú pravdepodobnosti vykonania jednotlivých stavov resetované na pôvodné hodnoty.

Špeciálnym stavom je stav vyhýbania sa útokom. Tento stav je voliteľný, nepriateľ ho nemusí mať. Jeho kontrola prebieha nezávisle od hlavnej rozhodovacej funkcie v prípade, ak nepriateľ nie je v útočnom stave. Vykonanie tohto stavu záleží od toho, ako dlho hráč mieri na daného nepriateľa. Tento stav môže prerušiť ostatné animácie nepriateľa.

4.3 Gameplay – implementácia herných mechaník

Nasledujúca sekcia popisuje objekt postavy hráča a implementáciu rôznych herných mechaník, s ktorými sa hráč stretne pri hraní hry.

Status hráča

Aktuálny stav hráča je spravovaný skriptom založeným na rovnakom základe ako skript stavu nepriateľa. Z tohto vyplýva, že niektoré vlastnosti majú zdieľané. Skript stavu hráča má napríklad v sebe tiež obsiahnuté informácie o počte života hráča, jeho rýchlosti alebo aktívnych dočasných efektoch. Na rozdiel od nepriateľa však hráč nie je členený na viacero častí tela – informácie o jednotlivých častiach teda nie je nutné ukladať a poškodenie je aplikované priamo. Ďalším rozdielom je spôsob počítania sily útoku. Nepriateľ má vo svojom stave uloženú všeobecnú silu útoku, ktorá je pri vykonaní nejakého útoku vynásobená koeficientom daným pre tento útok. Hodnota sily útoku je u hráča použitá len ako prípadný bonus – hlavná sila útoku je viazaná na aktuálnu zbraň. Hráč sa ďalej odlišuje od objektu nepriateľa viacerými vlastnosťami, ktoré objekt nepriateľa nevyužíva. Medzi tieto vlastnosti sú zaradené napr. hodnota štítu alebo výdrže. Výdrž je používaná pri uhýbaní sa hráča nepriateľským útokom. Uložené sú aj hodnoty cooldownu (minimálny čas pred ďalším možným použitím akcie) a regenerácie tejto výdrže. V stave hráča sú uložené aj hodnoty o aktuálnom počte mincí a kľúčov.

Stav hráča môžu priamo ovplyvňovať rôzne atribúty. Tieto atribúty ale nie sú priamo súčasťou skriptu stavu hráča. Sú spravované svojim vlastným skriptom. Jednotlivé atribúty môžu mať hodnotu od 0 do 9. Každá úroveň atribútu má k sebe priradené dočasné efekty, ktoré sú pri jej dosiahnutí aplikované – pri zmene hodnoty atribútu prebehne kontrola a môže byť aplikovaný nový efekt alebo odstránený efekt, na ktorý už nie sú splnené požiadavky. V hre existujú nasledovné atribúty:

- **Sila útoku** – ovplyvňuje bonus útočnej sily zbraní.
- **Presnosť** – vylepšuje presnosť zbraní a silu kritických zásahov.
- **Rýchlosť** – vplýva na rýchlosť pohybu postavy a hodnotu výdrže.
- **Sila schopností** – vylepšuje modifikátor sily použiteľných schopností.
- **Šťastie** – má vplyv na frekvenciu kritických zásahov a zvyšuje šancu na nájdenie vzácnejších odmien. Po zmene hodnoty tohto atribútu sú pri najbližšom načítaní novej miestnosti prepočítané jednotlivé šance na nájdenie predmetov a všetky doposiaľ nepreskúmané miestnosti majú znovu vybrané odmeny.

Vybavenie hráča

V priebehu hry hráč môže nájsť rôzne typy vybavenia, ktoré je možné rozdeliť na *zbrane, predmety* a *schopnosti*. Každý typ vybavenia má svoj vlastný typ skriptu, ktorý ho spravuje. Každý typ vybavenia má svoju vlastnú štruktúru na uchovanie dát o jeho rôznych vlastnostiach. Tieto dáta sú uložené vo forme assetov založených na Unity triede `ScriptableObject`.

Na začiatku hry má hráč k dispozícii maximálne dve zbrane, neskôr sa môže pridať miesto pre tretiu zbraň. Ovládací skript zbrane má v sebe uložené dáta o vlastnostiach zbrane. Tieto dáta obsahujú aj informácie o projektiloch zbrane (viď *Útoky hráča* 4.3). Tento skript obsahuje rôzne funkcie na používanie zbrane, ako napr. streľba, nabíjanie

alebo prepínanie medzi zbraňami. Zvukové efekty zbrane sú tiež ovládané týmto skriptom. Navyše tento skript umožňuje aj aplikovanie dočasných efektov (viď *Dočasné efekty* 4.3) pri vyzbrojení sa zbraňou.

Hráč má dostupných päť miest na predmety. V priebehu hry môže získať jedno miesto navyše. Podobne ako pri zbraniach, aj pri predmetoch má každé miesto svoj vlastný ovládací skript. Tento skript obsahuje funkcie na aplikovanie a zrušenie efektov predmetu.

Počas hry môže byť hráč vybavený maximálne jednou schopnosťou. Ovládací skript schopnosti hráča zabezpečuje kontrolu a počítanie nabitia schopnosti a jej použitie. Použitie schopnosti je rozdelené do viacerých funkcií, ktorých funkcionalita záleží od typu schopnosti. Tieto funkcie sú volané podľa stavu stlačenia tlačidla použitia schopnosti – existujú funkcie volané pri prvom stlačení tlačidla, pri jeho podržaní a pri jeho uvoľnení. Schopnosti, ktoré umožňujú oddialiť aktiváciu držaním tlačidla, majú implementovanú aj funkciu na zrušenie použitia schopnosti. Samotný spôsob aplikácie efektu schopnosti závisí od typu schopnosti (napr. niektoré môžu využiť systém dočasných efektov).

Dočasné efekty

Hra obsahuje mechaniku dočasných efektov, ktoré nejakým spôsobom ovplyvňujú postavy (hráča aj nepriateľov). Implementácia dočasných efektov je založená na Unity triede `ScriptableObject` – tieto efekty sú uložené na disku ako assety. Pri každom dočasnom efekte je možné nastaviť viaceré vlastnosti. K hlavným vlastnostiam patria:

- **Kategória efektu** – čo vlastne daný efekt ovplyvňuje.
- **Cieľ efektu** – cieľový objekt aplikovaného efektu.
- **Hodnota efektu** – určuje nakoľko daný efekt ovplyvňuje vybranú kategóriu.
- **Typ hodnoty efektu** – fixná alebo škálovaná hodnota (v prípade, ak je hodnota škálovaná, je možné aj určiť na základe čoho je škálovaná).
- **Spôsob aplikovania efektu** – jednorázovo, jednorázovo s oneskorením, na určitú dobu, opakovane počas určitej doby alebo „permanentne“ pokiaľ nie je efekt odstránený (napr. pri predmetoch). Pri opakovanom aplikovaní efektu musí byť určená frekvencia tohto opakovania. Pri aplikovaní na určitú dobu musí byť definovaná táto doba trvania efektu.

Jednotlivé hodnoty týchto vlastností sú nastavené pomocou editora.

Dočasné efekty je možné priradiť k rôznym akciám (napr. k útokom) alebo k predmetom, ktoré ich potom aplikujú na postavu hráča alebo nepriateľa. Po aplikácii nejakého efektu sú informácie o ňom uložené do zoznamu v skripte stavu postavy. Je určená frekvencia kontroly odstraňovania už vypršaných efektov. V prípade „permanentných“ efektov aplikovaných na postavu hráča je ich odstraňovanie riadené skriptom, ktorý ich vytvoril. Napríklad pri použití nejakého predmetu si ovládač tohto predmetu uloží identifikátor (ID) aplikovaného efektu. Pri odstránení tohto predmetu je pomocou ID daný efekt označený ako vypršaný a je následne odstránený.

Pohyb hráča a kamery

Na implementáciu pohybu postavy hráča je použitý komponent `CharacterController`. Pohyb tohto komponentu na rozdiel od komponentu `Rigidbody` nie je priamo viazaný

na fyzikálny engine v Unity. Vďaka tomuto je vhodnejší na použitie pre hry s „arkádovým“ typom pohybu.

Na pohyb je použitá funkcia tohto komponentu s názvom `Move()`. Táto funkcia hýbe s objektom daným smerom, nepoužíva ale gravitáciu. Gravitáciu je preto nutné aplikovať explicitne pomocou gravitačnej konštanty $g \approx -9.81$. Smer pohybu je počítaný na základe vstupu z klávesnice – vstupný vektor je normalizovaný a rotovaný podľa rotácie kamery. Pred aplikáciou je ešte tento vektor nutné vynásobiť rýchlosťou hráča, ktorá je uložená v skripte stavu hráča, a časom od posledného zobrazeného snímku (keďže funkcia `Move()` je volaná každý snímok).

V prípade skoku hráča je k vertikálnej rýchlosti postavy v danom snímku pripočítaná hodnota $\sqrt{-3 * jumpDistance * g}$, kde $jumpDistance$ je vzdialenosť skoku hráča. Po skoku prebieha kontrola výšky hráča až pokiaľ jeho vertikálna rýchlosť opäť nenadobudne záporné hodnoty. Táto kontrola zabezpečuje ukončenie stúpania v prípade ak hráč narazí na nejakú prekážku – z objektu hráča je vyslaný lúč pomocou raycastingu (viď sekcia 2.7). Ak existuje nejaký priesečník tohto lúča s prekážkou, vertikálna rýchlosť hráča je pred aplikovaním gravitácie vynulovaná a tým pádom hráč začne padať.

Pri uhýbaní sa hráča je počas určitého časového okna v každom snímku aplikovaný pohyb v smere pohybu hráča, ktorý je vynásobený časom od posledného snímku a vzdialenosťou.

Kamera používa perspektívu pohľadu prvej osoby. Je priamo súčasťou objektu postavy hráča – pohybuje sa spolu s týmto objektom, ale je nutné riešiť jej nezávislú rotáciu. Vstupný vektor pohybu myši je vynásobený uloženou hodnotou citlivosti. Ďalej je vynásobený časom od posledného snímku a následne pripočítaný k aktuálnej rotácii kamery. Ak je táto hodnota mimo predom definovaného rozsahu možnej rotácie kamery, tak je upravená na minimálnu alebo maximálnu hodnotu tohto intervalu. Následne je rotácia objektu kamery nastavená na danú hodnotu.

Útoky hráča

Hráč môže spôsobiť poškodenie nepriateľom pomocou zbraní. Zbrane pri útočení vytvárajú projektily. Každý objekt projektilu má svoj vlastný ovládací skript. Projektily využívajú fyzikálny engine v Unity, každý projektil má k sebe priradený `Rigidbody` komponent. Vlastnosti tohto komponentu sú nastavené ovládacím skriptom na základe dát uložených v priradenom `ScriptableObject` dátovom assete.

Ovládací skript projektilu obsahuje viacero funkcií. Ako prvá je volaná inicializačná funkcia, ktorá nastaví vlastnosti `Rigidbody` komponentu a vypočíta základné poškodenie, ktoré bude aplikované v prípade zásahu nepriateľa. Následne je volaná funkcia, ktorá vystrelí projektil. Smer výstrelu je určený na základe smeru kamery s určitou odchýlkou. Táto odchýlka je určená ako náhodný smer z vypočítaného rozptylu – rozptyl je zistený na základe presnosti zbrane, vzdialenosti k cieľu (viď sekcia 2.7) a aktuálneho pohybu hráča. Po určení smeru je na objekt projektilu aplikovaná sila – projektil je vystrelený.

Kolízia projektilu je limitovaná na určité vrstvy objektov. Pri kolízii s prostredím sa môže projektil odraziť. V prípade, že projektil dosiahne maximálny počet odrazov alebo ak zasiahne nepriateľa, je zničený. Pri zasiahnutí nepriateľa je aplikované poškodenie na konkrétne zasiahnutú časť tela nepriateľa. Ak hráč opustí miestnosť, v ktorej zostali aktívne projektily, tieto projektily sú zničené.

Obr. 4.5: Ukážka mechaniky ukladania mincí do banky.

Ďalšie herné mechaniky

Hráč začína hru v miestnosti bez nepriateľov. Po vstupe do ďalšej miestnosti sú všetky vchody zavreté. Ovládací skript každej miestnosti má zoznam nepriateľov, ktorí boli v danej miestnosti vygenerovaní. Tento zoznam slúži ako podmienka vyčistenia miestnosti. Pri zabití nepriateľa je tento nepriateľ odstránený zo zoznamu. Po úplnom vyprázdnení zoznamu je miestnosť označená ako vyčistená a sú otvorené všetky jej neuzamknuté vchody.

Uzamknuté vchody vedú do obchodu. Tieto vchody je možné odomknúť pomocou kľúča. V obchode má hráč okrem možnosti zakúpenia rôznych predmetov aj možnosť využiť mechaniku banky a odomknutia rozšírenej časti obchodu. Banka slúži na ukladanie mincí pre ďalšie hranie s inými postavami (viď obrázok 4.5). Dáta o počte uložených mincí sú uložené na disk. Na odomknutie rozšírenej časti obchodu sú hráčovi ponúknuté viaceré možnosti. Hráč môže obetovať z určitého rozsahu náhodne určený počet mincí, kľúčov alebo života.

Hodnotenie hráča je zobrazované vo forme skóre. Táto hodnota sa mení na základe viacerých faktorov. Skóre je odpočítavané plynutím času – skript spravujúci skóre obsahuje časovač, ktorý každú sekundu zníži aktuálnu hodnotu skóre. Ďalším negatívnym faktorom ovplyvňujúcim skóre sú zasiahnutie hráča nepriateľom a úmrtie hráča. K pozitívnym faktorom ovplyvňujúcim skóre patria:

- **Zasiahnutie nepriateľa projektilom** – zvýšenie skóre je aplikované pomocou ovládacieho skriptu projektilu pri kolízii s objektom nepriateľa.
- **Úmrtie nepriateľa** – skóre je zvýšené prostredníctvom ovládacieho skriptu nepriateľa tesne pred jeho zničením.
- **Vyčistenie miestnosti** – po zabití všetkých nepriateľov v miestnosti ovládací skript tejto miestnosti zvýši aktuálne skóre hráča na základe typu miestnosti.

Po vyčistení najvzdialenejšej miestnosti je možný prechod do finálnej miestnosti obsahujúcej „boss“ nepriateľa. Zabitím tohto nepriateľa je umožnený prechod do ďalšej úrovne. Po aktivácii prechodu je inkrementovaná hodnota aktuálnej úrovne a znovu začína proces generovania prostredia (viď sekcia 4.1). V prípade, ak už bola dosiahnutá posledná úroveň herného dungeonu, namiesto prechodu do ďalšej úrovne je hra ukončená. Po ukončení hry

je porovnané dosiahnuté skóre s uloženým najlepším skóre. V prípade, že je nové dosiahnuté skóre lepšie, hodnota uložená na disku je prepísaná. Takéto porovnanie prebehne aj s uplynutým časom hrania.

4.4 Užívateľské rozhranie

Užívateľské rozhranie hry je vytvorené pomocou `Canvas` systému Unity. Jeho veľkosť je škálovaná na základe rozlíšenia hry. Rozdelené je na viacero vrstiev. Správu týchto vrstiev a ďalších všeobecných funkcií UI zabezpečuje hlavný spravovací skript. Niektoré časti UI majú svoj vlastný skript, ktorý riadi ich prvky. Niektoré ikony použité pri vytváraní UI sú prevzaté z voľne dostupných alebo zakúpených assetov (napr. ikony kláves alebo ikony predstavujúce symboly ako zdravie alebo mince).

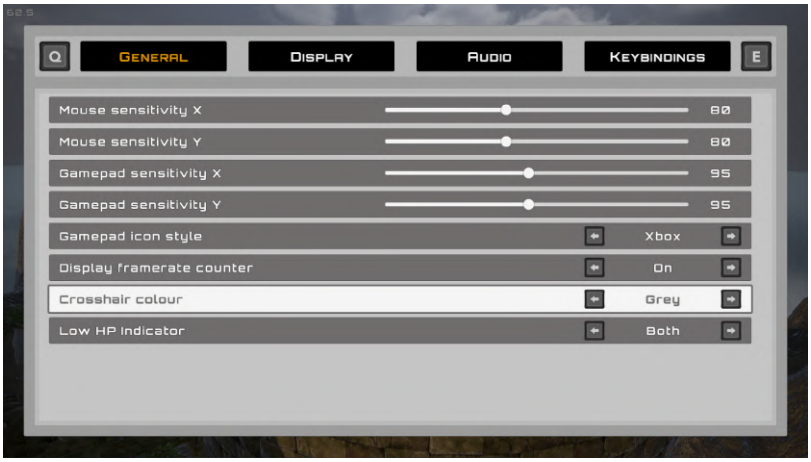
Hlavné menu

Hlavné menu existuje v separátnej scéne od hlavnej časti hry. Tvorené je scénou zobrazujúcou jednoduché vonkajšie prostredie. Dôležitou časťou hlavného menu je časť spravujúca začatie hry – nachádza sa tu možnosť na nastavenie obtiažnosti a tlačidlo, ktoré začína hru (a teda aj generovanie dungeonu).

Nastavenia hry

Užívateľské rozhranie nastavení hry je členené do viacerých kategórií. Prepínanie medzi týmito kategóriami je viazané na vstup z klávesnice (alebo gamepadu). Tento vstup je spracovaný hlavným spravovacím skriptom UI a následne je UI nastavení prepnuté na kategóriu vľavo alebo vpravo od aktuálnej.

Každá kategória obsahuje viacero možností nastavení (viď obrázok 4.6). Nastavenia je možné upraviť prostredníctvom tlačidiel. Existujú tri typy týchto tlačidiel. Prvým typom sú tlačidlá s posuvnou číselnou hodnotou, druhým typom sú tlačidlá s viacerými prepínateľnými možnosťami a posledným typom sú tlačidlá umožňujúce prestavenie keybindings (klávesových skratiek) akcií. Každé tlačidlo má k sebe priradený ovládací skript, ktorý jednak načíta inicializačnú hodnotu z uložených nastavení, a taktiež pri úprave hodnoty tlačidla prevedie dané zmeny priamo v hre a aj v uložených dátach.



Obr. 4.6: Ukážka užívateľského rozhrania nastavení hry.

Obr. 4.7: Ukážka herného užívateľského rozhrania.

Herné UI

Herné užívateľské rozhranie poskytuje hráčovi potrebné informácie o priebehu hry (viď obrázok 4.7). Rozčlenené je na viaceré menšie časti:

- **Status hráča** – poskytuje informácie na základe aktuálneho stavu hráča. Jeden z prvkov zobrazuje počet života, štítu a výdrže. Na toto zobrazenie sú použité obrázky, ktoré sú vyplnené podľa pomeru aktuálnej a maximálnej hodnoty. Ďalším prvkom je ukazovateľ cooldownu uhýbania sa. Tento prvok zobrazuje časovač, ktorý sa začne odpočítavať po tejto akcii. Posledným prvkom je prvok zobrazujúci informácie o aktuálnej schopnosti hráča. Zobrazuje jej názov a použiteľnosť na základe nabitia tejto schopnosti. Tento prvok je zobrazený len ak hráč počas hry našiel nejakú schopnosť.

- **Ukazovateľ mierenia** – zobrazuje rozptyl mierenia, indikátor nabíjania a varovanie o prázdnom zásobníku. Rozptyl mierenia je určený na základe presnosti aktuálnej zbrane vzhľadom na vzdialenosť k najbližšiemu zasiahnuteľnému objektu (prostredie alebo nepriateľ). Táto vzdialenosť je zistená pomocou raycastingu (viď sekcia 2.7). Indikátor nabíjania je implementovaný pomocou jednoduchej animácie kruhu.

- **Zobrazovanie skóre** – ukazuje aktuálne skóre hráča. Pri zmene skóre je vytvorený nový objekt správy popisujúcej túto zmenu. Tento objekt je po pár sekundách zničený.

- **Informácie o zbraniach** – v tejto časti sú zobrazené informácie o zbraniach hráča. Na začiatku hry sú zobrazené len dva prvky, v prípade nájdenia extra zbrane je vytvorený ďalší prvok UI.

- **Informácie o atribútoch hráča** – poskytuje informácie o aktuálnych atribútoch hráča. Jednotlivé prvky reprezentujúce hodnoty atribútov sú pridané alebo odobrané pri zmene hodnoty daného atribútu.

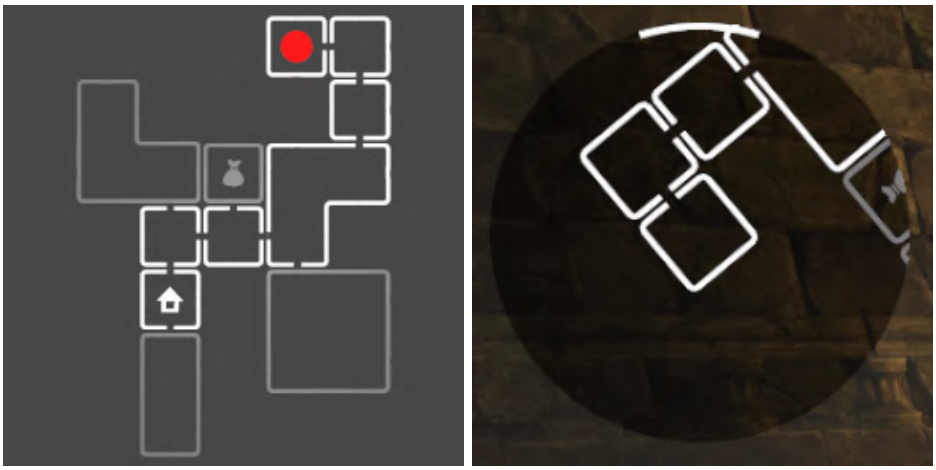
- **Zoznam predmetov** – táto časť herného UI ukazuje zoznam aktuálnych predmetov hráča. Zobrazené sú názvy a krátke popisy daných predmetov. Implementovaná je aj funkcia výmeny predmetov v prípade, ak sa hráč snaží získať predmet nad maximálny limit. Jednotlivé prvky predmetov majú v tomto prípade funkcionalitu tlačidiel – je možné medzi nimi prepínať a jeden z nich aktivovať.

- **Stav hry** – zobrazuje aktuálnu úroveň dungeonu a herný čas, ktorý je aktualizovaný na základe hodnoty času plynúceho počas hry (mimo menu).
- **Mapa a minimapa** – viď nasledujúca časť *Mapa a minimapa*.

Mapa a minimapa

Z hľadiska implementácie UI je zaujímavou časťou vytvorenie UI prvkov na zobrazenie mapy a minimapy, ktoré slúžia na navigáciu a uľahčenie pohybu v hernom dungeone. Počas inštancovania jednotlivých objektov miestností pri generovaní herného prostredia (viď sekcia 4.1) sú zároveň vytvárané v inej časti hernej scény objekty reprezentujúce mapu generovanej úrovne. Špeciálne typy miestností (začiatočná miestnosť a obchod) sú navyše označené unikátnou ikonou. Pre potreby zobrazovania mapy je vytvorený aj objekt ukazujúci aktuálnu pozíciu hráča v dungeone. Jednotlivé objekty miestností môžu byť označené ako navštívené alebo nenavštívené. Tieto objekty miestností sú počas generovania inicializované ako nenavštívené a sú deaktivované. Po presune objektu hráča do začiatočnej miestnosti je táto miestnosť označená za navštívenú. Zároveň je aj spolu s jej susednými miestnosťami aktivovaná a tým pádom zobrazená na mape. Ďalšie miestnosti a ich susedia sú aktivované presunom hráča z miestnosti do miestnosti (viď obrázok 4.8a).

Samotné zobrazovanie týchto objektov v UI je realizované použitím kamier a renderovacích textúr. Použité sú dve kamery – jedna pre celú mapu a druhá pre minimapu. Kamera použitá na zobrazenie celej mapy má fixnú pozíciu a je centrovaná. Kamera použitá na zobrazenie minimapy sa pohybuje spolu s indikátorom pozície hráča (ten však na nej nie je viditeľný). Obidve kamery renderujú len objekty vytvárajúce mapu úrovne. Výstup týchto kamier je ukladaný do renderovacích textúr, ktoré sú následne vložené do prvkov UI slúžiacich na zobrazenie mapy/minimapy. Takéto vytváranie textúr je mierne časovo náročné a z tohto dôvodu vždy prebieha počas načítavania ďalšej miestnosti pri prechode z miestnosti do miestnosti. UI prvok minimapy je zároveň rotovaný na základe orientácie objektu hráča (viď obrázok 4.8b).



(a) Príklad objektov mapy reprezentujúcich jednotlivé miestnosti úrovne

(b) Príklad minimapy ako prvku UI rotovaného podľa objektu hráča

Obr. 4.8: Grafické znázornenie mapy úrovne dungeonu.

4.5 Užívateľské testovanie a spätná väzba

Po ukončení hlavného vývoja hry je hra poskytnutá na testovanie viacerým užívateľom. Na spätnú väzbu pri užívateľskom testovaní slúži dotazník, ktorý je poskytnutý deviatim osloveným respondentom. Dotazník je rozdelený do štyroch kategórií. V každej kategórií je viacero otázok, na ktoré môže respondent odpovedať jednou z piatich možností s nasledujúcou škálou hodnotenia (pokiaľ nie je určené inak):

- **Možnosť 1** – veľmi nespokojný.
- **Možnosť 2** – nespokojný.
- **Možnosť 3** – neutrálny.
- **Možnosť 4** – spokojný.
- **Možnosť 5** – veľmi spokojný.

Prvé tri kategórie dotazníka sú zamerané na rôzne oblasti funkčnosti vytvorenej hry. Posledná kategória zisťuje spokojnosť s hrou ako celku. Výsledky užívateľského testovania sú spracované v jednotlivých kategóriách v percentuálnej podobe do prehľadných tabuliek. Percentuálne hodnoty sú zaokrúhlené na jedno desatinné miesto. Pri každej otázke je navyše v tabuľke vypočítaná priemerná hodnota hodnotenia.

Hrateľnosť

1. **Pocit zo súbojového systému** – táto otázka sa zameriava na rôzne faktory ovplyvňujúce hrateľnosť so zameraním na súbojový systém. Medzi tieto faktory sa zaraďujú napr. responzivita alebo presnosť kolízií medzi hráčom a nepriateľmi (hitboxy).
2. **Ovládanie hry** – respondent posudzuje celkový pohyb postavy. Okrem základného pohybu sem patria aj rozšírené možnosti pohybu ako uhýbanie sa útokom alebo skákanie.
3. **Obtiažnosť hry** – pri tejto otázke respondent na základe upravenej škály (rozsah od veľmi ľahká (1) po veľmi ťažká (5)) zdieľa svoje pocity o obtiažnosti hry.
4. **Kvalita a variácia obsahu generovaných herných úrovní** – cieľom tejto otázky je ohodnotiť generovanie jednotlivých herných úrovní na základe faktorov ako napr. variácie nepriateľov a predmetov.
5. **Vyváženosť a férovosť obsahu generovaných herných úrovní** – otázka je zameraná na hodnotenie faktorov ako napr. rozmiestnenie nepriateľov v miestnostiach alebo frekvencia a početnosť rôznych predmetov a vylepšení.
6. **Hodnotenie nepriateľov** – respondent posudzuje kvalitu nepriateľov na základe kvality ich umelej inteligencie, variácie ich útokov a ich atribútov.
7. **Celkové hodnotenie sekcie**.

Otázka	Možnosť 1	Možnosť 2	Možnosť 3	Možnosť 4	Možnosť 5	Priemer
č. 1	0 %	0 %	22,2 %	**55,6** %	22,2 %	***4,00***
č. 2	0 %	0 %	22,2 %	33,3 %	**44,4** %	***4,22***
č. 3	0 %	11,1 %	**44,4** %	22,2 %	22,2 %	***3,56***
č. 4	0 %	0 %	33,3 %	**55,6** %	11,1 %	***3,78***
č. 5	0 %	**33,3** %	22,2 %	**33,3** %	11,1 %	***3,22***
č. 6	0 %	0 %	**33,3** %	**33,3** %	**33,3** %	***4,00***
č. 7	0 %	0 %	11,1 %	**44,4** %	**44,4** %	***4,33***

Tabuľka 4.1: Výsledky hodnotenia hrateľnosti.

Oslovení respondenti celkovo hodnotili hrateľnosť hry pozitívne – najčastejším výberom boli možnosti „spokojný“ a „veľmi spokojný“ (viď tabuľka 4.1). Najspokojnejší boli s ovládaním hry. Kladne hodnotené boli aj otázky týkajúce sa súbojového systému a hodnotenia nepriateľov. Najnižšie hodnotené otázky boli otázky súvisiace s obtiažnosť a vyváženosťou hry. Takéto hodnotenie jednak môže súvisieť s preferenciami hráča, ale aj s náročnosťou správneho vyváženia hry. Vyváženie obsahu hry je časovo náročné, vyžaduje veľa testovania a z dôvodu limitovaného času vytvárania tejto práce nebolo na primeranej úrovni.

Audiovizuálna a technická stránka hry

1. **Vizuálna kvalita a dizajn hry** – respondent hodnotí objektívne faktory (napr. kvalitu modelov, textúr a osvetlenia) a subjektívne faktory (napr. výber vhodného vizuálneho štýlu a atmosféra hry).

2. **Zvukové efekty a hudba** – pri tejto otázke je hodnotená kvalita vybraných zvukových efektov ale aj vhodný výber hudby z hľadiska typu prostredia.

3. **Stabilita a výkon hry** – otázka sa zameriava na technickú stránku hry (napr. plynulý beh hry).

4. **Výskyt chýb v hre** – respondent pomocou upravenej škály hodnotenia (rozsah od výrazných problémov, prípadne až nehrateľnosti (1) po minimálny výskyt problémov, prípadne zanedbateľné problémy priamo neovplyvňujúce bezproblémové hranie (5)) hodnotí chybovosť hry.

5. **Celkové hodnotenie sekcie**.

Celkové hodnotenie sekcie bolo pozitívne – nadpolovičná väčšina respondentov vybrala možnosť „veľmi spokojný“ (viď tabuľka 4.2). Najlepšie bola hodnotená technická stránka hry. Respondenti nemali žiadne alebo mali len minimálne množstvo technických problémov. Problémy s výkonom boli hlásené len pri testovaní na staršom hardvéri. Audiovizuálna stránka hry bola hodnotená pomerne kladne. Najčastejšie vyskytujúcimi sa pripomienkami boli niektoré chýbajúce zvukové efekty a prípadne ich nevhodná hlasitosť. Pri slovnom hodnotení bola taktiež ocenená hudba, ktorá dotvárala atmosféru hry.

Otázka	Možnosť 1	Možnosť 2	Možnosť 3	Možnosť 4	Možnosť 5	Priemer
č. 1	0 %	11,1 %	11,1 %	33,3 %	**44,4** %	***4,11***
č. 2	0 %	11,1 %	11,1 %	**66,7** %	11,1 %	***3,78***
č. 3	0 %	11,1 %	0 %	11,1 %	**77,8** %	***4,56***
č. 4	0 %	0 %	11,1 %	11,1 %	**77,8** %	***4,67***
č. 5	0 %	0 %	11,1 %	33,3 %	**55,6** %	***4,44***

Tabuľka 4.2: Výsledky hodnotenia audiovizuálnej a technickej stránky hry.

Užívateľské rozhranie a nastavenia

1. **Jednoduchosť navigácie v užívateľskom rozhraní** – pri tejto otázke respondent hodnotí prípadné nedostatky alebo problémy pri používaní užívateľského rozhrania.

2. **Kvalita a prehľadnosť herného užívateľského rozhrania** – táto otázka sa zameriava na hodnotenie užívateľského rozhrania viditeľného pri hraní hry. Medzi hodnotené faktory patrí napr. rozloženie jednotlivých prvkov užívateľského rozhrania.

3. **Kvalita a ponuka možností v nastaveniach** – hodnotené sú rôzne poskytnuté možnosti nastavenia hry a ich upravovanie.

4. **Celkové hodnotenie sekcie**.

S užívateľským rozhraním a nastaveniami boli respondenti celkovo viac ako spokojní – dve tretiny respondentov vybrali možnosť „veľmi spokojný“ (viď tabuľka 4.3). Najpozitívnejšie bola hodnotená jednoduchosť navigácie v užívateľskom rozhraní. Z celkového počtu respondentov len jeden respondent vyjadril nespokojnosť s kvalitou a prehľadnosťou herného užívateľského rozhrania. V slovnom hodnotení jeden respondent pozitívne ocenil možnosť nastavenia citlivosti myši zvlášť pre x-ovú a y-ovú os.

Otázka	Možnosť 1	Možnosť 2	Možnosť 3	Možnosť 4	Možnosť 5	Priemer
č. 1	0 %	0 %	0 %	33,3 %	**66,7** %	***4,67***
č. 2	0 %	11,1 %	11,1 %	**44,4** %	33,3 %	***4,00***
č. 3	0 %	0 %	22,2 %	22,2 %	**55,6** %	***4,33***
č. 4	0 %	0 %	22,2 %	11,1 %	**66,7** %	***4,44***

Tabuľka 4.3: Výsledky hodnotenia užívateľského rozhrania a nastavení.

Otázka	Možnosť 1	Možnosť 2	Možnosť 3	Možnosť 4	Možnosť 5	Priemer
č. 1	0 %	0 %	33,3 %	**55,6** %	11,1 %	***3,78***
č. 2	0 %	0 %	0 %	22,2 %	**77,8** %	***4,78***
č. 3	0 %	0 %	11,1 %	**55,6** %	33,3 %	***4,22***

Tabuľka 4.4: Výsledky celkového hodnotenia.

Celkové hodnotenie

1. **Znovuhrateľnosť a „fun factor“ hry** – zameranie tejto otázky je na záujem respondenta o opakované hranie hry a zistenie, do akej miery bola hra z jeho pohľadu zábavná.

2. **Záujem o sledovanie ďalšieho vývoja** – táto otázka zisťuje možný záujem respondenta ohľadom ďalšieho vývoja hry (1 – nemá záujem, 5 – mal by veľký záujem sledovať ďalší vývoj hry).

3. **Celkové hodnotenie hry**.

Oslovení respondenti boli s hrou celkovo spokojní – len jeden respondent hodnotil hru ako celok neutrálne (viď tabuľka 4.4). Všetci respondenti vyjadrili záujem o ďalšie sledovanie vývoja hry.

Kapitola 5

Záver

Cieľom tejto práce bolo navrhnutie a vytvorenie roguelike hry v hernom engine Unity. Tento cieľ bol úspešne splnený.

Pre dosiahnutie cieľa bolo v prvom rade potrebné oboznámiť sa s problematikou vývoja hier a možnosťami rôznych herných vývojových prostredí. Po získaní všeobecných znalostí bolo nutné preskúmať históriu a vlastnosti roguelike žánru hier. Okrem toho bolo potrebné oboznámiť sa s rôznymi možnosťami procedurálneho generovania obsahu (napr. Perlinov šum, celulárne automaty) a kombináciou týchto techník s manuálnym tvorením obsahu. Ďalšou podstatnou časťou bolo naštudovanie tvorby umelej inteligencie v hrách. V tejto časti je zahrnuté aj skúmanie algoritmu A* používaného na hľadanie najkratšej trasy. Získané teoretické vedomosti sú popísané v kapitole 2. Po naštudovaní si potrebnej teórie bol vytvorený koncept hry a návrh jednotlivých herných mechaník. Tento koncept je bližšie popísaný v kapitole 3.

Na základe vytvoreného konceptu hry a návrhu jednotlivých herných mechaník prebehol samotný vývoj hry, ktorý je popísaný v kapitole 4. Hra spĺňa určené požiadavky roguelike žánru. Pri generovaní herných úrovní sú využité techniky procedurálneho generovania v kombinácii s manuálne vytvorenými šablónami miestností. Nepriateľské postavy majú implementovanú umelú inteligenciu na základe použitia hierarchických stavových automatov. Úspešne boli implementované aj rôzne herné mechaniky. Vývoj hry bol úspešný, čo potvrdzujú výsledky z užívateľského testovania.

Vďaka tejto práci som získal mnohé nové vedomosti, zručnosti a skúsenosti z rôznych oblastí súvisiacich s herným vývojom. Problematika vývoja hier ma oslovila, a veľmi rád by som sa jej v budúcnosti venoval aj naďalej.

Vytvorenú hru je samozrejme možné v budúcnosti vylepšiť a rozšíriť. Z hľadiska generovania úrovní sa dá zdokonaliť samotný spôsob generovania alebo rozšíriť variácia miestností, ktorá je obmedzená počtom manuálne vytvorených šablón. Vylepšenia sú možné aj v oblasti umelej inteligencie – je možné zlepšiť pohyb nepriateľov, prechody medzi rôznymi stavmi alebo aj implementovať zmenu správania na základe aktuálneho stavu nepriateľa. Ďalej je hru možné zdokonaliť aj doladením a prípadným rozšírením niektorých herných mechaník. Pri ďalšom vývoji hry by mohli byť použité aj rôzne návrhy a podnety poskytnuté respondentmi pri užívateľskom testovaní.

Literatúra

[1] Adams, E. *Fundamentals of Game Design*. 3. vyd. New Riders, 2013. ISBN 978-0-321-92967-9.

[2] Aversa, D. *Unity Artificial Intelligence Programming*. 5. vyd. Packt Publishing, 2022. ISBN 978-1-803-23853-1.

[3] Bauer, E. *The NPD Group: U.S. Video Game Industry Sales Increased 2% in the Fourth Quarter of 2022* [online]. 2. februára 2023 [cit. 2023-03-28]. Dostupné z: https://www.npd.com/news/press-releases/2023/the-npd-group-u-s-video-game-industry-sales-increased-2-in-the-fourth-quarter-of-2022.

[4] Bhattacharjee, K., Maity, K. a Dasa, S. A Search for Good Pseudo-random Number Generators: Survey and Empirical Studies. *CoRR*. 2018, abs/1811.04035, [cit. 2023-04-11]. Dostupné z: https://arxiv.org/abs/1811.04035.

[5] Black, P. E. *Euclidean distance* [online]. 17. decembra 2004 [cit. 2023-04-10]. V *Dictionary of Algorithms and Data Structures*. Dostupné z: https://www.nist.gov/dads/HTML/euclidndstnc.html.

[6] Black, P. E. *Manhattan distance* [online]. 11. februára 2019 [cit. 2023-04-10]. V *Dictionary of Algorithms and Data Structures*. Dostupné z: https://www.nist.gov/dads/HTML/manhattanDistance.html.

[7] Buckland, M. *Programming Game AI by Example*. 1. vyd. Jones & Bartlett Learning, 2004. ISBN 978-1-556-22078-4.

[8] Bycer, J. *Game Design Deep Dive: Roguelikes*. 1. vyd. CRC Press, 2021. ISBN 978-0-367-71371-3.

[9] Cellar Door Games. *Rogue Legacy* [online]. [cit. 2023-04-02]. Dostupné z: https://cellardoorgames.com/roguelegacy.

[10] Chomsky, N. Three models for the description of language. *IRE Transactions on Information Theory*. 1956, zv. 2, č. 3, s. 113–124. DOI: 10.1109/TIT.1956.1056813.

[11] Clarke, C. *Unity Technologies Celebrates Six Years of Continual Leadership and Innovation in Game Development* [online]. 6. júna 2011 [cit. 2023-03-30]. Dostupné z: https://www.globenewswire.com/en/news-release/2011/06/06/1251937/0/en/Unity-Technologies-Celebrates-Six-Years-of-Continual-Leadership-and-Innovation-in-Game-Development.html.

[12] Clement, J. *Number of games released on Steam worldwide from 2004 to 2022* [online]. Statista, 20. februára 2023 [cit. 2023-03-30]. Dostupné z: https://www.statista.com/statistics/552623/number-games-released-steam.

[13] Cusick, T. W. a Stanica, P. *Cryptographic Boolean Functions and Applications*. 2. vyd. Academic Press, 2017. ISBN 978-0-128-11129-1.

[14] Dunn, W. L. a Shultis, J. K. *Exploring Monte Carlo Methods*. 1. vyd. Elsevier Science, 2011. ISBN 978-0-444-51575-9.

[15] Games, R. B. *Noise Functions and Map Generation* [online]. 31. augusta 2013 [cit. 2023-04-21]. Dostupné z: https://www.redblobgames.com/articles/noise/introduction.html.

[16] Gregory, J. *Game Engine Architecture*. 1. vyd. A K Peters/CRC Press, 2009. ISBN 978-1-439-86526-2.

[17] Gustavson, S. Simplex noise demystified. Január 2005. Dostupné z: https://www.researchgate.net/publication/216813608_Simplex_noise_demystified.

[18] Harris, J. *Exploring Roguelike Games*. 1. vyd. CRC Press, 2020. ISBN 978-0-367-51372-6.

[19] Iuppa, N. a Borst, T. *End-to-End Game Development: Creating Independent Serious Games and Simulations from Start to Finish*. 1. vyd. Focal Press, 2009. ISBN 978-0-240-81179-6.

[20] Johnson, L., Yannakakis, G. N. a Togelius, J. Cellular automata for real-time generation of infinite cave levels. In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. September 2010, s. 1–4. DOI: 10.1145/1814256.1814266. Dostupné z: https://doi.org/10.1145/1814256.1814266.

[21] Koikara, R. SAC: G: 3-D Cellular Automata based PRNG. *ACM 978-1-4503-3738-0*. 2017, s. 1–5, [cit. 2023-04-15].

[22] Lait, J. *Berlin Interpretation* [online]. 26. septembra 2008 [cit. 2023-03-31]. Dostupné z: http://www.roguebasin.com/index.php?title=Berlin_Interpretation.

[23] Lindenmayer, A. Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs. *Journal of Theoretical Biology*. 1968, zv. 18, č. 3, s. 300–315. DOI: https://doi.org/10.1016/0022-5193(68)90080-5. ISSN 0022-5193. Dostupné z: https://www.sciencedirect.com/science/article/pii/0022519368900805.

[24] Linietsky, J. *First public release!* [online]. 14. januára 2014 [cit. 2023-03-30]. Dostupné z: https://godotengine.org/article/first-public-release.

[25] Linietsky, J., Manzur, A. a Godot community the. *GDScript reference* [online]. [cit. 2023-03-30]. Dostupné z: https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript_basics.html.

[26] Linietsky, J., Manzur, A. a Godot community the. *License* [online]. [cit. 2023-03-30]. Dostupné z: https://godotengine.org/license.

[27] Lowood, H. E. *Electronic game* [online]. Encyclopedia Britannica, 1. novembra 2021 [cit. 2023-03-28]. Dostupné z: https://www.britannica.com/topic/electronic-game.

[28] Marsaglia, G. Xorshift RNGs. *Journal of Statistical Software* [online]. 2003, zv. 8, č. 14, s. 1–6, [cit. 2023-04-11]. DOI: 10.18637/jss.v008.i14. Dostupné z: https://www.jstatsoft.org/index.php/jss/article/view/v008i14.

[29] Millington, I. a Funge, J. *Artificial Intelligence for Games*. 2. vyd. CRC Press, 2009. ISBN 978-0-123-74731-0.

[30] Mono Project. *About Mono* [online]. [cit. 2023-03-30]. Dostupné z: https://www.mono-project.com/docs/about-mono.

[31] Moore, M. *Basics of Game Design*. 1. vyd. A K Peters/CRC Press, 2016. ISBN 978-1-439-86776-1.

[32] Perlin, K. Chapter 2 Noise Hardware. In: *SIGGRAPH 2001 Course 24 Notes*. SIGGRAPH.

[33] Perlin, K. An Image Synthesizer. *SIGGRAPH Comput. Graph.* Association for Computing Machinery. Júl 1985, zv. 19, č. 3, s. 287–296. DOI: 10.1145/325165.325247. ISSN 0097-8930. Dostupné z: https://doi.org/10.1145/325165.325247.

[34] Perlin, K. Improving Noise. Association for Computing Machinery. Júl 2002, zv. 21, č. 3, s. 681–682. DOI: 10.1145/566654.566636. ISSN 0730-0301. Dostupné z: https://doi.org/10.1145/566654.566636.

[35] Shaker, N., Togelius, J. a Nelson, M. J. *Procedural Content Generation in Games*. 1. vyd. Springer, 2016. ISBN 978-3-319-42716-4.

[36] Short, T. X. a Adams, T. *Procedural Generation in Game Design*. 1. vyd. A K Peters/CRC Press, 2017. ISBN 978-1-138-74331-1.

[37] Spencer, K. *Noise!* [online]. 19. septembra 2014 [cit. 2023-04-21]. Dostupné z: https://uniblock.tumblr.com/post/97868843242/noise.

[38] Subset Games. *Subset Games | FTL: Faster Than Light* [online]. [cit. 2023-04-02]. Dostupné z: https://subsetgames.com/ftl.html.

[39] Thomsen, M. *History of the Unreal Engine* [online]. 24. februára 2010 [cit. 2023-03-30]. Dostupné z: https://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine.

[40] Turing, A. M. Computing Machinery and Intelligence. *Mind*. 1950, LIX, č. 236, s. 433–460, [cit. 2023-04-06]. DOI: 10.1093/mind/LIX.236.433. ISSN 0026-4423. Dostupné z: https://doi.org/10.1093/mind/LIX.236.433.

[41] Unity Technologies. *Government & Aerospace Simulation Training Software* [online]. [cit. 2023-03-30]. Dostupné z: https://unity.com/solutions/government-aerospace.

[42] Unity Technologies. *Inner Workings of the Navigation System* [online]. [cit. 2023-04-12]. Dostupné z: https://docs.unity3d.com/Manual/nav-InnerWorkings.html.

[43] Unity Technologies. *Multiplatform Game Development for the Future* [online]. [cit. 2023-03-30]. Dostupné z: https://unity.com/solutions/multiplatform.

[44] Unity Technologies. *Unity - Scripting API:* [online]. [cit. 2023-04-11]. Dostupné z: https://docs.unity3d.com/ScriptReference.

[45] Valve Corporation. *The Binding of Isaac on Steam* [online]. [cit. 2023-04-02]. Dostupné z: https://store.steampowered.com/app/113200/The_Binding_of_Isaac.

[46] Vlambeer. *Nuclear Throne* [online]. [cit. 2023-04-02]. Dostupné z: http://nuclearthrone.com.

[47] Wolfram, S. *Cellular Automata And Complexity: Collected Papers*. 1. vyd. CRC Press, 2002. ISBN 978-0-201-62664-3.

[48] Wright, S. *There are too many video games. What now?* [online]. 28. septembra 2018 [cit. 2023-03-30]. Dostupné z: https://www.polygon.com/2018/9/28/17911372/there-are-too-many-video-games-what-now-indiepocalypse.

Príloha A

Obsah priloženého pamäťového média

Na priloženom pamäťovom médiu sú uložené súbory v nasledujúcej štruktúre:

- `game/` – priečinok obsahujúci spustiteľnú hru
- `src/` – priečinok obsahujúci zdrojové kódy
- `latex_src/` – priečinok obsahujúci LaTeX kódy
- `vid/` – priečinok obsahujúci ukážkové video
- `README` – súbor readme