FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Magitech - Generátor dungeonů
Student:	Jiří Macháček
Vedoucí:	Ing. Jan Matoušek
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Magitech je počítačová hra pro jednoho hráče žánru dungeon crawler, v níž se hráč zapojuje do střetu magie s technologií v procedurálně generovaných prostředích ve světě zabydleném nehráčskými postavami.

Cílem bakalářské práce je vytvořit pro potřeby hry generátor těchto herních prostředí.

Pokyny k vypracování:
1) Prozkoumejte způsoby generování dungeonů v existujících počítačových hrách.
2) Analyzujte potřeby počítačové hry Magitech v souvislosti s generováním dungeonů.
3) Metodami softwarového inženýrství navrhněte příslušný modul generátoru.
4) Implementujte prototyp modulu generátoru.
5) Podrobte funkčnost modulu důkladnému testování.
6) Shrňte zjištěné výsledky a nastiňte další možnosti vývoje.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 3. února 2022 v Praze.

Bakalářská práce

MAGITECH - GENERÁTOR DUNGEONŮ

Jiří Macháček

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jan Matoušek
12. května 2022

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Rád bych poděkoval vedoucímu své práce Ing. Janu Matouškovi za jeho vedení a podporu v průběhu psaní této bakalářské práce. Poděkování také patří mé rodině za neustávající podporu po celou dobu studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 12. května 2022 .....

Abstrakt

Tato bakalářská práce se zabývá tvorbou procedurálního generátoru herních prostředí – dungeonů – pro hru Magitech. V práci jsou analyzovány metody procedurálního generování obsahu, které mohou být použity pro generování různých typů dungeonů. Z analyzovaných metod je vybrána metoda nejvhodnější pro použití v případě hry Magitech – využití agentů. Generátor dungeonů využívající agenty je následně navržen a implementován do hry pomocí prostředí Unity. Generátor je nakonec důkladně otestován. Způsob generování implementovaný v této práci splňuje požadavky stanovené zadavatelem, a bude tak do hry v budoucnu zakomponován.

Klíčová slova procedurální generování obsahu, počítačová hra, generování dungeonů, Dungeon Crawler, Unity

Abstract

The goal of this bachelor thesis is to create a procedural generator of game environments – dungeons – for the game Magitech. The paper analyzes the methods of procedural content generation, which can be used to generate different types of dungeons. From the analyzed methods, the most suitable method for use in the case of Magitech – the use of agents – is selected. The agent-based dungeon generator is then designed and implemented in the game using the Unity environment. The generator is finally thoroughly tested. The generator module implemented in this work meets the requirements set by the client, and will be incorporated into the game in the future.

Keywords procedural content generation, computer game, dungeon generation, Dungeon Crawler, Unity

Seznam zkratek

ASP	Answer Set Programming
BSP	Binary Space Partitioning
HMdMC	Hierarchical Multidimensional Markov chains
MdMC	Multidimensional Markov chains
MRF	Markov Random Fields
ORE	Occupancy-Regulated Extension
PCG	Procedural Content Generation
RPG	Role-Playing Game
SS	Side-Scrolling
TDCL	Top-Down Cavern-Like
TDML	Top-Down Mansion-Like

Úvod

Dungeon Crawler je jedním z populárních žánrů počítačových her, ve kterém musí hráči plnit úkoly v prostředí zvaném dungeon. Tato herní prostředí bývají typicky náhodně generovaná, aby měl hráč pocit, že je vždy součástí nového dobrodružství, a herním designérům odpadla nutnost taková prostředí ručně vytvářet. Počítačová hra *Magitech* spadá právě do tohoto žánru a mým úkolem je pro ni vytvořit modul pro generování dungeonů.

Hra *Magitech* vznikla jako semestrální projekt v předmětu *Virtuální herní světy* a tato bakalářská práce navazuje na práci, kterou jsem společně s mými kolegy na projektu v rámci předmětu odvedl.

Po absolvování předmětu nebyla hra *Magitech* v takovém stavu, jaký bych si přál, a tak jsem se rozhodl na ní dále pracovat. Zejména problematika procedurálního generování herních prostředí nebyla dosud ve hře uspokojivě řešena, a z toho důvodu se tomuto tématu věnuji ve své bakalářské práci. Tehdejší systém neumožňoval generování více propojených dungeonů, generování určitého počtu nepřátel a materiálů, které se v dungeonu musí nacházet, a nebyl jednoduše rozšiřitelný. Právě tímto směrem se ubíraly požadavky na systém generování, který jsem musel vyvinout v rámci bakalářské práce. Práce se zaměřuje na analýzu již existujících způsobů generování dungeonů a následný návrh, implementaci a otestování generátoru, který bude splňovat požadavky stanovené zadavatelem.

V teoretické části se v 1. kapitole zabývám popisem procedurálního generování obsahu, jeho historie a důvodů jeho použití v počítačových hrách. Součástí 1. kapitoly je i popis dělení metod procedurálního generování obsahu. Ve 2. kapitole popisuji, jak dungeony definují různí autoři. Ve 3. kapitole popisuji počítačovou hru *Magitech*. Ve 4. kapitole analyzuji již existující způsoby generování dungeonů. V 5. kapitole se věnuji analýze a popisu funkčních a nefunkčních požadavků. Na základě provedené rešerše volím pro požadovaný generátor takový způsob generování, který je nejvhodnější pro použití v případě hry *Magitech*.

V praktické části bakalářské práce se nejdříve v 1. části věnuji návrhu požadovaného modulu generátoru a následně se v 2. části zabývám implementační detaily výsledného produktu. V poslední části popisuji testování generátoru.

Cíle práce

Cílem bakalářské práce je vytvoření modulu pro generování herních prostředí do počítačové hry *Magitech*. Modul musí odpovídat požadavkům, které na něj klade zadavatel práce.

Cílem teoretické části je popis procedurálního generování obsahu v počítačových hrách. Teoretická část se dále věnuje analýze již popsaných způsobů generování dungeonů. V této části jsou také popsány funkční a nefunkční požadavky získané od zadavatele. Na jejich základě je následně navržen požadovaný modul.

Cílem praktické části je návrh a implementace prototypu generačního modulu v prostředí *Unity*. Návrh je založen na poznatcích získaných v teoretické části a implementován je tedy takový způsob generování, který nejlépe vyhovuje požadavkům kladeným na systém procedurálního generování. Funkčnost modulu je poté důkladně otestována.

Kapitola 1
Analýza

Tato kapitola se v sekci 1.1 věnuje popisu procedurálního generování obsahu, jeho historii a rozdělení. Sekce 1.2 se zabývá specifických herním prostorem zvaným dungeon a tomu, jak na něj nahlížejí různí autoři. Sekce 1.3 seznamuje čtenáře se hrou Magitech. V sekci 1.4 jsou rozebrány funkční a nefunkční požadavky kladené na modul generátoru. Konečně v sekci 1.5 jsou analyzovány různé metody používané pro generování dungeonů.

1.1 Procedurální generování obsahu

Cílem této sekce je seznámit čtenáře s pojmem procedurální generování obsahu. Krátce je zde shrnuta jeho historie, aby bylo zdůrazněno jeho úzké propojení s počítačovými hrami. Dále je vysvětleno, jaké jsou výhody použití procedurálního generování obsahu oproti jiným formám tvorby obsahu. Poslední část této sekce rozebírá různé druhy procedurálního generování obsahu a rozdíly mezi nimi.

1.1.1 Definice

Procedurální generování obsahu (dále jen PCG)[1] je běžně definováno jako automatické vytváření obsahu za použití algoritmických prostředků [2, 3, 4, 5, 6, 7]. Podle [7] se některé metody PCG stávají v herním průmyslu běžnou praxí, a zatímco někteří autoři [4, 6, 8] uznávají využití PCG i v jiných oblastech, než jsou hry, další autoři [2, 3, 5, 9] vztahují PCG k vytváření specificky herního obsahu. [8] kritizuje hlavně obecnost pojmu obsah, v praxi jsou totiž některé metody tvořící obsah brány jako PCG a jiné ne.

[3] dává procedurální generování obsahu do kontrastu s jinými formami generování obsahu, se kterými by mohlo být jednoduše zaměněno, a následně diskutuje vlastnosti, které bývají pro procedurální generování obsahu občas brány jako nutné, ale ve skutečnosti nejsou.

Diskutována je například míra lidského zásahu při samotném procesu generování. [3] zmiňuje editory postav a herních úrovní v počítačových hrách, které jsou volně dostupné hráčům pro tvorbu herního obsahu. Podle definice se nejedná o procedurální generátory obsahu, protože je obsah v tomto případě tvořen lidmi. Proti těmto editorům staví systémy, které s člověkem kooperují na vytváření obsahu. Jako příklad zmiňuje [3] systém *Tanagra*, který je používán na generování 2D herních úrovní pro plošinovky[2]. *Tanagra* umožní designérovi načrtnout rozložení herního levelu a následně je level systémem vygenerován tak, aby bylo pro hráče možné level

[1] Zkratka je hojně používaná v anglických textech. Vznikla z anglického Procedural Content Generation.

[2] Plošinovka neboli skákačka je žánr počítačových her, ve kterém je hráčovým úkolem projít herním prostorem od začátku do konce. Hráč musí pro dokončení herní úrovně využívat po prostoru rozmístěné plošiny.

úspěšně dokončit. Takové systémy jsou ovládány člověkem, ale na rozdíl od herních editorů za procedurální generátory obsahu považovány jsou. [3]

Rozdíl mezi těmito dvěma přístupy ke generování obsahu nalézá [3] v přímočarosti úprav prováděných v těchto typech nástrojů. V editorech lze pozorovat přímou a okamžitou spojitost mezi tím, co člověk dělá a jak se obsah mění. Oproti tomu u systémů PCG s určitou formou lidského vstupu odděluje lidský vstup od výsledné změny objektu nezanedbatelná míra výpočetní náročnosti. [3]

Na konci své práce tak [3] přichází s novou definicí PCG a jedná se tak podle něj o: „Algoritmické vytváření herního obsahu s limitovaným nebo nepřímým uživatelským vstupem."

Další vlastnost, kterou [3] ve spojitosti s procedurálními generátory obsahu zmiňuje, je jejich náhodnost. Jako příklad náhodných generátorů herního obsahu uvádí [3] generátory herních prostředí ve hrách *Civilization*, *Infinite Mario Bros* nebo *Rogue*. Tyto generátory nejsou podle [3] plně náhodné, ale spíše se řídí přísnými pravidly toho, jak má výsledný obsah vypadat, a zároveň v nich existují prvky náhody.

[3] oproti takovým náhodným generátorům staví deterministické generátory. Jako jejich příklad uvádí PCG ze hry *Elite*, která si kvůli limitacím své doby nebyla schopna ukládat celé herní prostředí do paměti, a proto bylo toto prostředí vždy deterministicky generováno na základě hodnoty seedu[3]. PCG tak ve hře sloužilo spíše jako jistá forma komprese dat. Tímto příkladem [3] podporuje tvrzení, že není důvod, aby musel být prvek náhody v PCG přítomen.

[10] naopak bere nějakou formu náhodnosti jako povinnou součást PCG. Definuje totiž PCG jako: „Programovou generaci herního obsahu za použití náhodného nebo pseudonáhodného procesu, jehož výsledkem je nepředvídatelný rozsah možných herních prostorů." K tomuto tvrzení se přidává i [11], která tvrdí že: „*Náhodnost je nutnou, ale ne postačující podmínkou pro to, aby byla hra považována za procedurální.*"

[10] dále odděluje termíny procedurální generování obsahu a procedurální generování a to na základě toho, jakou mírou obsah generovaný těmito metodami ovlivňuje hratelnost hry. [10] odkazuje na definici pojmu procedurální generování na Wikipedii, kde se píše, že procedurálně generovány mohou být například textury a ty podle [10] významně hratelnost neovlivňují. Naopak obsah generovaný metodami PCG má na způsob, jakým se hra hraje, velký vliv.

[8] považuje samotný pojem PCG za problematický, protože je příliš spjat s generováním herního a grafického obsahu. Existuje podle ní ale více oblastí výzkumu, které by svůj obsah mohly generovat za pomocí algoritmů. Jako příklad udává [8] generování muziky, umění nebo designu. [8] přichází s názvem Generativní metody, které zastřešují všechny funkce, které generují nějaké artefakty[4], tedy i samotné PCG.

1.1.2 Historie

[13] tvrdí, že procedurálně generovaný obsah může být generován člověkem, pokud člověk při jeho generování dodržuje přesně stanovený postup. [13] proto vidí předchůdce toho, co v dnešní době nazýváme PCG, ve stolních a karetních hrách. V roce 1976 byla vydána pomocná sada ke hraní hry *Dungeons & Dragons* jménem *Dungeon Geomorphs*. Tato sada obsahovala 10 velkých a 5 malých dlaždic, které mohly být použity pro sestavování různých dungeonů[5]. [13] zmiňuje, že na tomto principu staví mnoho moderních PCG a jako příklad udává hru *Spelunky*, která svá herní prostředí sestavuje z předpřipravených dlaždic. Dalším příkladem náhodného generování obsahu může být podle [13] generování náhodných střetnutí ve hře *Dungeons & Dragons*, které bylo založeno na pravidlech ze stolní hry *Outdoor Survival*, kde se na základě několika hodů kostkami určovala finální podoba střetnutí. Jako další hry, ve kterých dochází ke generování herního prostředí, uvádí [13] například *Carcassonne* nebo *Betrayal at House on the Hill*.

[3] Seed je číselná hodnota používaná pro inicializaci generátorů pseudonáhodných čísel.

[4] Pojem artefakt podle [12] označuje instanci obsahu.

[5] Dungeon je v kontextu her označení specifického herního prostoru. Definici dungeonu se tato práce věnuje v sekci 1.2.

Přechod ze stolního PCG do digitálního byl podle [13] veden dvěma hlavními směry. Prvním směrem je zájmové PCG, které šlo ruku v ruce s rozmachem počítačů a programování jakožto volnočasové aktivity. Zájmové časopisy *BYTE* a *Creative Computing* ukazovaly čtenářům programy, které zahrnovaly určitou formu PCG, a čtenáři si je mohli sami implementovat. Příkladem budiž haiku[6] generátor v roce 1976 nebo náhodný generátor muziky v roce 1979. [13]

Druhým směrem bylo zakomponování počítačů do stolních role-playing her[7]. Například monstra v doplňcích pro hru *Runequest*, měla své statistiky vygenerované počítačem. V roce 1977 vznikl počítačem vygenerovaný modul do hry *Tunnels & Trolls* jménem *A Computer Generated Dungeon*, který představoval dobrodružství v dungeonu pro jednoho hráče. [13]

Nakonec podle [13] docházelo k vytváření kompletních role-playing her, které se hrály na počítačích. [15] zmiňuje jako nejrannější dochovanou počítačovou role-playing hru *pedit5* z roku 1975, pojmenovanou také jako *The Dungeon*. Hra obsahovala pevně stanovené herní lokace, ale střetnutí hráče s nepřáteli byla náhodně generována. [2] zmiňuje hru *Akalabeth: World of Doom*, která již v roce 1979 obsahovala svá vlastní pravidla na tvorbu dungeonů. [13] jako příklad udává legendární hru *Rogue* z roku 1980.

[6] považuje hru *Rogue* za průkopníka procedurálního generování a to díky jeho procedurálně generovaným dungeonům. Podle hry *Rogue* je pojmenován jeden z oblíbených žánrů počítačových her – Roguelike – mezi jehož důležité atributy patří podle [16] náhodně generovaná prostředí. Popularita hry *Rogue* vedla ke vzniku mnoha dalších roguelike her. Jako jejich příklad udává [17] hru *Hack* z roku 1984 a hru *NetHack*, která na ni v roce 1987 navázala.

Mezi lety 1990 a 2000 se podle [4] také objevilo několik her inspirovaných hrou *Rogue*. Podle [4] mezi nejúspěšnější takové hry patří série *Diablo*, která má své herní levely náhodně sestavené z předem vytvořených dlaždic.

Procedurální generování je podle [18] v oblasti počítačové grafiky pojem s dlouhou historií. [18] zmiňuje několik zásadních technik, které byly v průběhu času objeveny a zkoumány. Jedná se například o šumy, fraktály, L-systémy nebo tvarové gramatiky. V roce 2000 zveřejnila společnost *Intel* technickou zprávu jménem „Procedural 3D content generation“, která je podle [8] zdrojem současného pojmenování této disciplíny. Zpráva podle [8] informovala vědce i laiky o generativních technikách, které byly prozkoumány v posledních desetiletích.

V dnešní době je procedurální generování obsahu hojně využíváno ve hrách všeho druhu a roguelike hry, jejichž součástí jsou procedurálně generované herní úrovně, jsou stále oblíbeným herním žánrem. Podle [19] bylo na distribuční službě *Steam* v roce 2021 vydáno více než 600 her se štítkem *Procedurální generování*.

1.1.3 Důvody použití

Zprvu nemusí být hned jasné, proč by někdo metod procedurálního generování obsahu chtěl využívat, když může být mnohdy kvalitnější obsah vytvářen člověkem. Z tohoto důvodu je do práce začleněn následující seznam důvodů použití procedurálního generování obsahu ve hrách. Věřím, že jednotlivé body je možné vztáhnout i na jiné oblasti, ve kterých se metod procedurálního generování obsahu využívá.

- **Automatické generování obsahu** – je možné generovat obsah automaticky, na tvorbu obsahu není potřeba najímat specializované zaměstnance, z čehož vyplývá nižší časová a finanční náročnost. [5, 7, 9, 10]
- **Znovuhratelnost** – generování nového originálního obsahu při každém startu hry, vytváření nového obsahu za běhu programu. [5, 9, 10, 13, 20]

[6] Haiku je třířádková báseň skládající se ze 17 slabik, pochází z Japonska.[14]

[7] Role-playing hra, označovaná také hra na hrdiny – anglicky role-playing game (RPG) – je žánr her, ve kterém je hráč postaven do pozice postav ve fiktivních prostředích. Hráč má možnost se svobodně za postavu rozhodovat a ovlivňovat tak její příběh.

- **Adaptivní obsah** – při použití dalších specializovaných metod, například neuronových sítí, je možné vytvářet obsah šitý na míru konkrétním hráčům. [9, 20]
- **Kreativita** – při použití PCG může vzniknout kreativnější obsah, který by designéři nemuseli vůbec brát v potaz. [9]
- **Podpora představivosti** – Obsah vzniklý použitím PCG může rozšiřovat lidskou představivost, podporovat lidskou kreativitu. [5, 13, 20]
- **Snížená spotřeba paměti** – generovaný obsah může zůstat „komprimován“, dokud ho není potřeba použít. [5, 10, 20]
- **Expresivní prostředek** – samotný proces vytváření systémů procedurálního generování obsahu může jedincům přinášet potěšení. [13]
- **Nezaujaté generování** – procedurální generátor obsahu může být oproti lidskému designérovi nezávislý a nezaujatý. [13]

1.1.4 Taxonomie

[5] rozděluje metody PCG na základě jejich vlastností. Poukazuje při tom, že se ve většině případů nejedná o binární rozdělení, ale spíše o škálu.

Online versus Offline

Procedurální generaci obsahu nazveme online, jestliže se děje za běhu programu. [5] udává jako příklad vygenerování nového herního prostoru, když hráč odejde z toho současného. Naproti tomu offline generování probíhá během vývoje hry a jeho výsledky mohou být následně upraveny designérem, než se opravdu stanou součástí hry. [5] poukazuje na možnost existence i přechodného stavu PCG, kdy jsou hráčům na základě jejich herního stylu každý den navrženy nové herní mapy.

Povinný versus Volitelný obsah

Další rozdělení se týká obsahu, který je pomocí PCG generován. Obsah povinný musí být vždy korektní. [5] udává jako příklad povinného obsahu herní prostory, kterými hráč musí projít, aby ve hře dosáhl nějakého pokroku – hráč musí být schopen projít prostorem od začátku do konce, prostor musí obsahovat monstra, která hráč přišel zneškodnit atd. Jako příklad volitelného obsahu jsou zmíněny například zbraně nebo herní prostory, které se hráč může rozhodnout ignorovat. Ve hře může být vygenerována nepoužitelná zbraň, jestliže ji hráč není nucen použít, nebo nesmyslný herní prostor, jestliže z něj může hráč odejít.

Náhodné seedy versus Vektory parametrů

[5] rozděluje PCG na základě toho, jak moc jsou parametrizované. Na jedné straně spektra leží generátory, které na svém vstupu přijímají seed, který parametrizuje jejich generátor pseudonáhodných čísel. Na straně druhé jsou generátory, které přijímají několikadimenzionální vektor parametrů, který specifikuje obsah vytvářený těmito generátory.

Stochastické versus Deterministické generování

Deterministické generátory, na rozdíl od stochastických, vždy vyprodukují stejný obsah, jsou-li jim zadány stejné parametry. [5] zmiňuje, že jako parametr generátoru zde nepovažuje seed pro generátor pseudonáhodných čísel, protože by to ze všech generátorů dělalo deterministické.

Konstruktivní versus Generuj-a-testuj

Konstruktivní algoritmy PCG jsou takové, které jednou svůj obsah vygenerují a jejich práce je tím dokončena. Musí v nich být ale zaručeno, že vygenerovaný obsah je vždy korektní. Algoritmy typu Generuj-a-testuj obsah vygenerují a pak testují, zda splňují specifikované požadavky[8]. Jestliže obsah testem neprojde, je vyřazen a znovu generován, dokud nejsou testované požadavky splněny. [5]

V [9] dochází k revizi přechozího rozdělení a jsou přidány další 2 kategorie, do kterých je možné metody PCG rozdělovat:

Generické versus Adaptivní

Adaptivní PCG je specifické tím, že bere v potaz hráčovo chování ve hře. Hráčův herní styl je analyzován a další obsah je generován na základě jeho předchozích akcích. Jako příklad adaptivního PCG je uvedena práce [6] nebo hra *Left 4 Dead*. Při generickém PCG není obsah hráčovými akcemi ovlivněn. [9]

Automatické generování versus Smíšené autorství

Při automatickém generování obsahu mají designéři jenom omezené možnosti, jak ovlivňovat generovaný obsahu, a to hlavně skrze upravování parametrů použitého generátoru. Proti automatickému generování je postaveno paradigma označované jako smíšené autorství, které staví designéra doprostřed samotného procesu generování. Designér nebo hráč v něm kooperuje s generujícím systémem tak, aby byl vytvořen požadovaný obsah. Příkladem využití tohoto paradigmatu je systém *Tanagra*, který zde byl již dříve popsán. [9]

Procedurální generátory obsahu můžeme dále dělit na základě toho, jaké metody pro generování obsahu používají. V práci se analýze některých metod generování věnuji v sekci 1.5.

1.2 Dungeon

Dungeon (česky žalář) je v reálném světě označení pro temné podzemního vězení, zejména uvnitř hradu či zámku [21]. Dungeon, který je pro nás v této práci zajímavý, je prostor, ve kterém se hráč může ve hře (stolní či počítačové) nacházet. Různí autoři však na tento prostor nahlížejí jinak.

1.2.1 Žánr

Pravidla stolní hry *Dungeons & Dragons* [22] (citováno v 2) popisují dungeon obecně jako „skupinu místností a chodeb, ve kterých je možné nalézt monstra a poklady.“ [2] označuje dungeon jako běžný herní prostor ve fantasy RPG hrách a předchozí definici považuje za spíše otevřenou. Podle něj by mohla zahrnovat herní prostory z ostatních žánrů her, a přestože sám tuto definici přijímá, zakládá analýzu dungeonů ve své práci na hrách, které spadají do žánru RPG. [7] se také přidává k vymezení dungeonu jakožto oblasti, která se vyskytuje v konkrétních žánrech her. [7] označuje dungeon jako typ herního levelu[9], který je specifický pro adventury[10] a hry žánru RPG. [23] označuje dungeon jako herní oblast, které se nachází v mnoha akčních adventurách.

[8] například zda existuje v dungeonu cesta od začátku do konce

[9] Herní levely či herní úrovně jsou části hry, které se od sebe odlišují například obtížností či prostředím, ve kterém se odehrávají.

[10] Adventura je žánr hry ve které hráč musí řešit různé hádanky a prozkoumávat herní lokace.

1.2.2 Topologie

[2] přichází s komplexním průzkumem her žánru RPG a rozděluje dungeony na základě jejich topologie na:

- **Spojené místnosti** – místnosti, mezi kterými se hráč přesunuje bez použití chodeb. Jako příklad [2] uvádí dungeony ve hře *Zork*.
- **Místnosti a chodby** – dungeon s malým počtem místností, které jsou spojeny nerozvětvenými chodbami. Příkladem hry, která obsahuje tento typ dungeonu, je například hra *Rogue*. [2]
- **Labyrint** – unikurzální (jednocestná) struktura – skrz dungeon vede jediná cesta, [2] udává jako příklad takového typu dungeonu dungeony ze hry *Ultima II*.
- **Bludiště** – skrz dungeon vede několik cest – tento typ dungeonu je přítomný například ve hře *Ultima I*. [2]
- **Otevřený prostor** – V dungeonu se nachází spousta prostoru pro volný pohyb hráče. [2] uvádí *Diablo* jako příklad hry, ve které se nachází dungeony tohoto typu.

Jednotlivé příklady jsou vyobrazeny na obrázku 1.1.

Ostatní autoři se neshodnou na přesné fyzické podobě dungeonu, převážně však pracují s jedním typem dungeonu z předchozího rozdělení.

[4] popisuje dungeon ve hrách žánru dungeon crawler[11] jako labyrint místností. Dále se zabývá generováním dungeonů, které se skládají z místností, které na sebe přímo navazují. Tento typ dungeonu se podle rozdělení výše nazývá Spojené místnosti. [24] se zabývá tvorbou generátoru pro roguelike dungeon crawler hru, v níž jsou dungeony reprezentovány jako na sebe navazující místnosti, stejně jako ve hrách *The Legend of Zelda* či *The Binding of Isaac*. Takovéto dungeony by byly podle [2] také klasifikovány jako Spojené místnosti.

[25] zkoumá metody generování dungeonů, které se skládají z obdélníkovitých místností propojených chodbami, což podle předchozího rozdělení spadá do kategorie Místnosti a chodby. Do této kategorie spadají i dungeony podle definice v [22] (citováno v 2). Podle [9] se dungeony ve většině adventur a her z žánru RPG skládají z několika místností propojených chodbami.

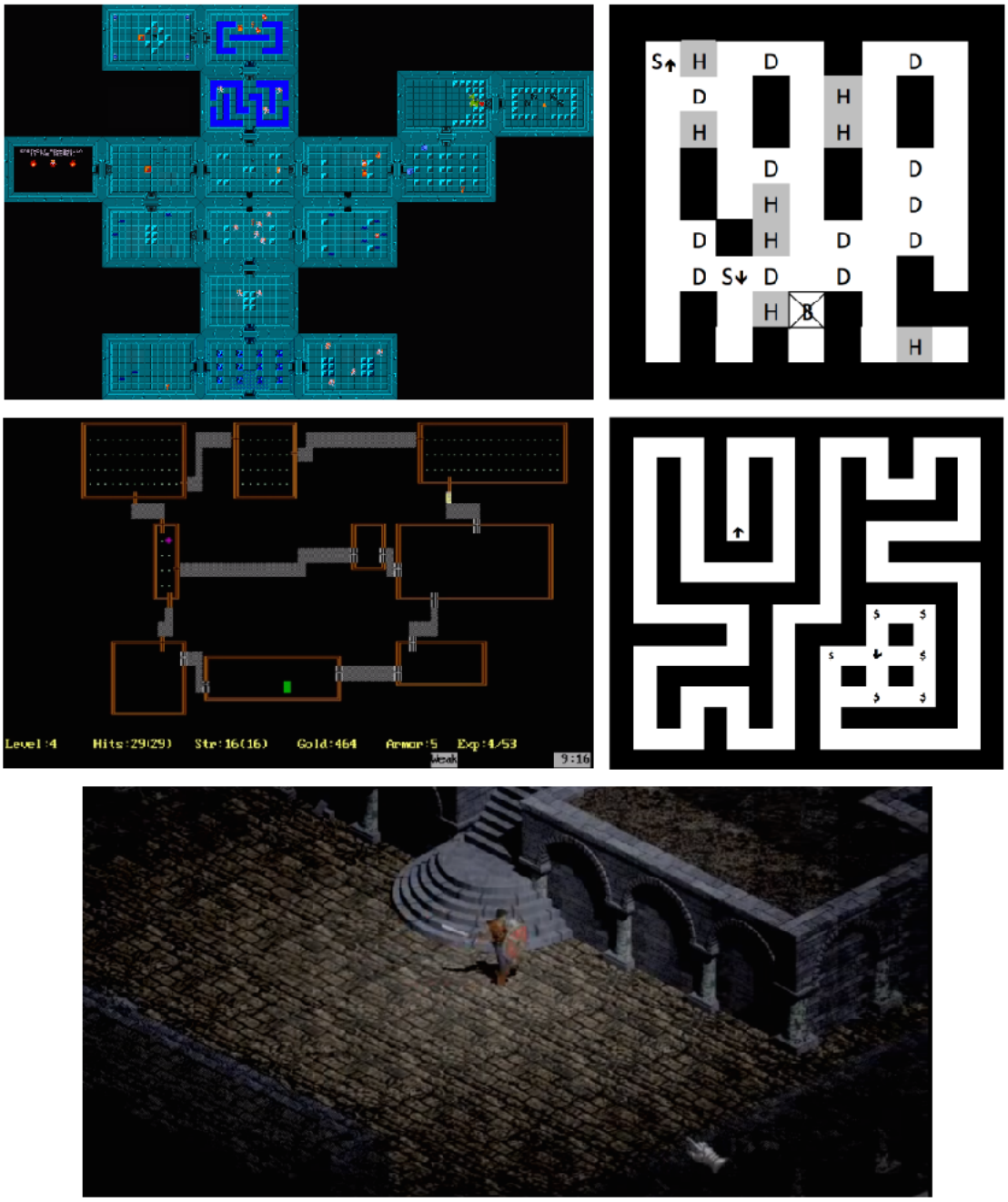
[7] popisuje dungeon jako labyrintické prostředí, které se skládá ze série výzev, odměn a hádanek. Jedná se o velice obecnou definici, podle které by se dungeon mohl nacházet hned v několika kategoriích zmíněných výše. Podle [7] se této definice striktně drží adventury. Moderní hry žánru RPG podle něj obsahují více otevřené dungeony, které jsou v mnoha ohledech komplexnější, například z hlediska jejich prozkoumávání. To by mohlo dungeony v moderních RPG hrách podle [7] zařadit do kategorie Otevřený prostor.

[26] přichází s novým pohledem na členění dungeonů na základě jejich struktury. Rozděluje zkoumané dungeony do 3 kategorií:

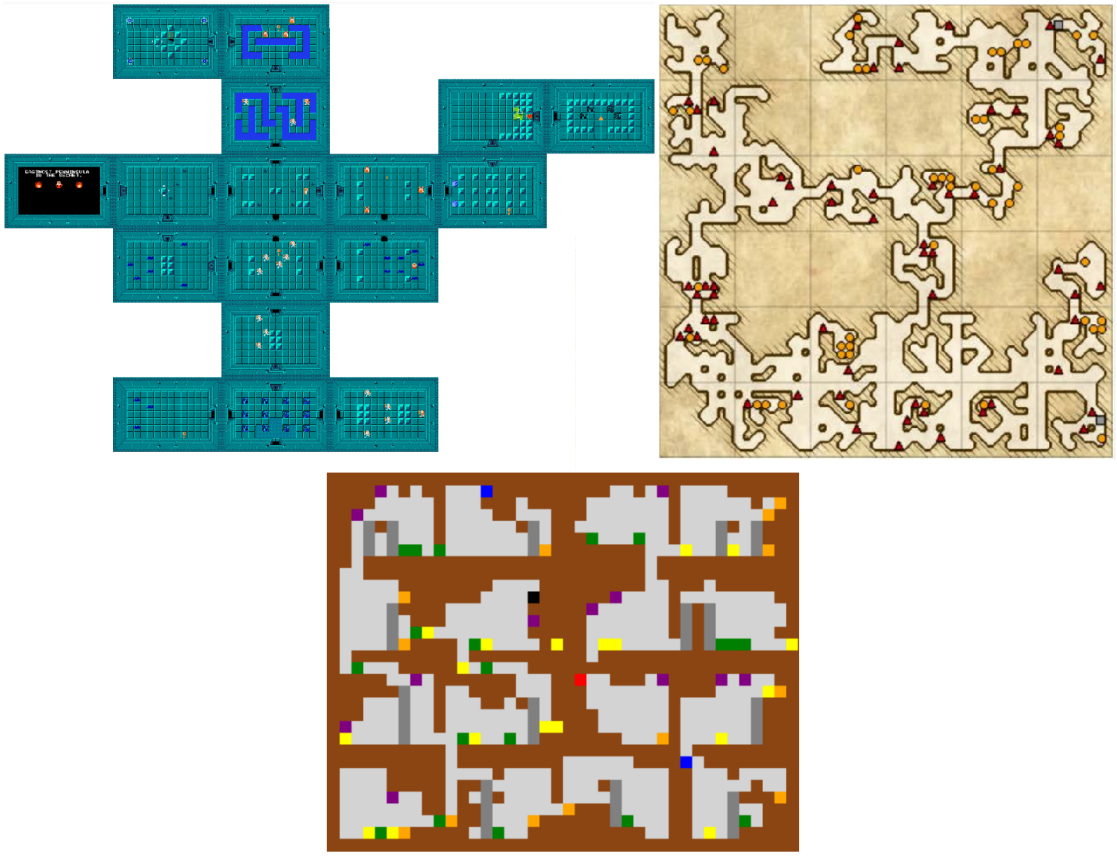
- **TDCL** – top-down cavern-like – jeskynní dungeony z ptačí perspektivy
- **TDML** – top-down mansion-like – dungeony připomínající velké sídlo z ptačí perspektivy
- **SS** – side-scrolling – dungeony, na které se hráč dívá ze strany

Příklady jednotlivých kategorií dungeonů můžeme vidět na obrázku 1.2. Dungeony z kategorie TDML by jednoznačně podle rozdělení [2] spadly do kategorie Spojené místnosti. Zbylé 2 kategorie už nejdou tak jednoznačně na rozdělení v [2] namapovat.

[11] Dungeon crawler je podžánr žánru RPG, ve kterém hráč prozkoumává dungeony a svádí v nich souboje s nepřáteli.



Obrázek 1.1 Druhy dungeonů podle [2]. Zleva po řádcích Spojené místnosti, Bludiště, Místnosti a chodby, Labyrint a Otevřený prostor



Obrázek 1.2 Příklady dungeonů podle rozdělení v [26]. Zleva po řádcích TDML, TDCL, SS.

1.2.3 Vlastnosti

[23] nezkoumá fyzickou strukturu dungeonu, ale spíše vlastnosti, které by měl dungeon splňovat. Mezi společné vlastnosti dungeonů napříč několika hrami podle nich mimo jiné patří:

- Koncept zámku a klíče – existence překážky a způsobu jejího překonání, který nutí hráče dungeon prozkoumávat.
- Přítomnost překážek, například ve formě soubojů nebo hádanek, za účelem vyvarování se repetetivnosti.
- Předání hráči možnosti prozkoumávat herní oblast spíše než ho vést lineární sekvencí událostí.
- Nutnost návratu hráče do již prozkoumaných oblastí, například návrat k zamčeným dveřím po nalezení klíče.
- „Boss“ – mimořádně těžký souboj, cíl uvnitř dungeonu. Výhra v souboji proti bossovi typicky znamená pokrok v herním příběhu.

[24] zakládá návrh svých dungeonů na hře *The Legend of Zelda*, která také obsahovala koncept zámku a klíče. Tento koncept je podle [24] často používán jako způsob otestování hráčova prostorového vnímaní a navigační paměti v dungeonech, které jsou otevřeny k prozkoumávání.

[27] vkládá koncept zámku a klíče do pravidel generativních gramatik a tím ho zakomponuje i do výsledného vygenerovaného dungeonu.[12]

[2] se ve svém průzkumu dále věnuje identifikaci návrhových vzorů v dungeonech, přičemž důraz dává na dungeony typu Místnosti a chodby, Labyrint a Bludiště. Jako jeden z mikro vzorů je zde uveden právě klíč jakožto předmět, který dokáže dostat dveře ze stavu *zamčené*.

[4] zmiňuje místnosti, ve kterých se nachází boss, jako jedny ze strategických místností dungeonu. Rozmístění strategických místností je podle něj důležitým aspektem návrhu dungeon. Jako příklad zajímavého rozmístění uvádí [4] situování možností získání lepšího vybavení před střetnutím s bossem. Hráč pak má pocit, že se v každé úrovni stává silnějším a je na souboje s bossem připravován.

Dungeony, které se nacházejí ve hře *Magitech* by se podle [2] daly klasifikovat jako Spojené místnosti a podle rozdělení ve [26] by spadaly do kategorie TDML. Neobsahují koncept zámku a klíče, který zde několik prací zmiňuje. Nachází se v něm ale překážky ve formě střetnutí s nepřáteli. Ve hře *Magitech* může dojít ke střetnutí s bossem, který se nachází na konci dungeonu a jeho poražení může být bráno jako splnění úkolu a tedy jako pokrok v herním příběhu, jak je popsáno v [23].

1.3 Magitech

Magitech je 3D izometrická počítačová hra pro jednoho hráče žánru dungeon crawler, ve které hráč využívá své magické schopnosti, aby odrazil invazi robotických vetřelců.

Hra vznikla jako semestrální projekt v rámci předmětu *Virtuální herní světy* a podílelo se na ní celkem 6 dalších kolegů. Před zahájením vývoje vznikl game design document[13], který měl popsat výsledný produkt. Byť se naše představy ohledně hry *Magitech* v průběhu projektu několikrát změnily, stále může dokument sloužit jako dobrý zdroj pro získání hrubé představy o tom, jak by hra *Magitech* měla vypadat. Game design document je k dispozici na přiloženém médiu.

Podle [28] se hra skládá ze 4 základních částí – mechanik, příběhu, vizuálu a technologie.

[12] více o generativních gramatikách v sekci 1.5.1

[13] Game design document je dokument popisující design hry. Obsahuje rozbor klíčových herních konceptů.

1.3.1 Mechaniky

Ve hře *Magitech* je hráčovým cílem plnění úkolů pro jednoho ze dvou vlivných obyvatel vesnice. Tyto úkoly hráče vždy zavedou do dungeonu, kde se setkává s nepřátelskými roboty. V dungeonech se také nacházejí různé materiály, které může hráč využít zpět ve vesnici k zakoupení různých vylepšení své postavy. Hráč se zpět do vesnice dostane, až splní úkol, který ho do dungeonu zavedl. Může se do vesnice ale také dobrovolně vrátit z místa, kde se v dungeonu poprvé objevil. V současnosti se ve hře nachází 3 typy úkolů – hráč musí zničit všechny nepřátele, zničit pouze jen nepřátelského bosse nebo posbírat určité množství materiálů.

Na souboj s nepřáteli je hráč vybaven 3 různými kouzly, které se liší svou účinností a spotřebou many. Nepřátelé ovšem také nejsou bezbranní a po hráči střílí rakety. Existují 2 druhy nepřátel – obyčejní roboti a boss. Boss může na hráče zaútočit několika různými způsoby. Jakmile množství hráčových životů klesne na nulu, hráčova postava umírá a hráč může dungeon opakovat nebo se vydat zpět do vesnice.

Méně důležité postavy ve vesnici mohou hráči nabídnout vedlejší úkoly, které se také plní v dungeonech a měly by pro hráče znamenat možnost nasbírat potřebné materiály pro zakoupení vylepšení. Ve hře se budou nacházet také úkoly, které hráče zavedou do nekonečných dungeonů.

Hra se skládá ze 3 hlavních scén:

- **scéna s vesnicí** – hráč se v ní bude setkávat s vesničany, získávat od nich úkoly a nakupovat vylepšení pro svou postavu,
- **scéna s hospodou** – v této scéně je umístěn hostinský, jedna ze dvou hlavních postav příběhu hry *Magitech*,
- **dungeon scéna** – scéna, kde bude hráč plnit úkoly. Pro každý úkol mu bude ve scéně vygenerován nový dungeon.

1.3.2 Příběh

Hrou hráče provází kratičký příběh o vesničce napadené roboty. Hráč na základě svých rozhodnutí může ovlivnit finální osud vesnice a dokončit tak hru se dvěma rozdílnými konci. Hra obsahuje příběhové úkoly, které hráč plní buď pro hospodského, který je zástupcem nižší vrstvy společnosti a stará se o obecné blaho vesnice, nebo pro starostu vesnice, který chce využít probíhající invaze pro své osobní obohacení. Pro méně důležité postavy hráč plní vedlejší úkoly, které nemají vliv na hlavní příběh a zaobírají se problémy obyčejných lidí.

1.3.3 Vizuál

Většina grafických modelů ve hře je laděna ve stylu low poly[14]. Modely robotů byly pro hru vlastnoručně vytvořeny stejně tak jako tilesety[15], které se používají pro generování dungeonů. Úkoly mohou hráče zavést do 3 různých druhů dungeonů, které se od sebe liší právě vizuálem – les, jeskyně a město. Hráče po celou dobu hry provází dynamická hudba, která byla pro hru speciálně vytvořena.

1.3.4 Technologie

Pro vytvoření hry byl zvolen herní engine *Unity*, který se ukázal být velice sofistikovaným nástrojem, který je zároveň velice jednoduchý na používání. Pro výrobu některých 3D modelů byl použit nástroj *Asset Forge*. Hudba pro hru byla vytvořena jedním ze spolupracovníků a byla do

[14] Low poly styl je charakteristický nízkým počtem polygonů ve 3D polygonové síti.
[15] Tileset je sada dlaždic, ze kterých je složen herní prostor. Další kapitoly se tilesetům věnují podrobněji.

hry integrována za pomocí nástrojů enginu *Unity* tak, aby se dynamicky měnila v závislosti na tom, zda se v současnosti hráč nachází v souboji či nikoliv.

1.4 Analýza požadavků

Cílem této kapitoly je představit požadavky kladené na modul procedurálního generování dungeonů hry *Magitech*. Tyto požadavky byly stěžejní pro výběr vhodné metody procedurálního generování, a proto je důležité s nimi čtenáře nejdříve seznámit.

Verze procedurálního generátoru, která pro hru *Magitech* vznikla v rámci předmětu *Virtuální herní světy*, trpěla nedostatky, které si tato bakalářská práce dala za cíl odstranit. Funkční a nefunkční požadavky, které jsou zmíněny níže, se soustředí právě na odstranění těchto nedostatků přidáním nových funkcionalit a úpravou těch starých.

1.4.1 Funkční požadavky

V této kapitole jsou vydefinovány požadavky na modul generátoru. Požadavky se nesoustředí pouze na rozšíření stávajících funkcionalit, ale také definují funkcionality nové.

F1: Využití tilesetů

Generátor bude využívat systém dlaždic, který je od počátku ve hře používán, včetně všech tilesetů, které byly pro hru zatím vytvořeny. Jeden dungeon se bude vždy generovat pouze za použití jednoho tilesetu. Generátor může počítat s tím, že všechny dlaždice ve všech tilesetech budou čtvercové a všechny dlaždice v jednom tilesetu budou mít stejnou velikost.

F2: Úprava tilesetů

Tilesety se budou skládat ze 6 typů dlaždic, které se navzájem liší účelem, který v dungeonu plní:

- **Border** – dlaždice používané jako hranice dungeonu,
- **Start** – dlaždice používané jako start dungeonu, hráč se na nich objeví při přechodu do dungeon scény. Obsahují objekt, který může přenést hráče zpět do vesnice,
- **Normal** – obyčejné dlaždice, které budou tvořit podstatnou část dungeonu,
- **Portal** – dlaždice, které obsahují portál do sousedního dungeonu,
- **Boss** – dlaždice, které jsou schopny vytvářet speciální nepřátele – bosse,
- **Special** – výjimečné dlaždice, například kill squad[16] dlaždice nebo dlaždice s pokladem.

Dlaždice se dále odlišují tím, jakými směry z nich vedou cesty neboli jejich tvarem. Všech 5 tvarů musí mít dlaždice v kategoriích Start a Normal. Ostatní typy dlaždic nemusí nutně obsahovat dlaždice všech 5 tvarů.

F3: Tvar dungeonu

Na tvar dungeonu nejsou kladena žádná omezení. Tvar dungeonu však nesmí být jakkoliv ovlivněn hranicemi mřížky, do které bude dungeon generován.

[16] Kill squad je skupina nepřátel, kteří byli vysláni nespokojenými vesničany do dungeonu, aby hráče zneškodnili. Bude-li mít hráč špatnou reputaci s některým vesničanem, generátor bude mít za úkol do dungeonu kill squad dlaždice umístit.

F4: Generování splnitelných dungeonů

Generátor musí vždy vygenerovat dungeon, který je splnitelný. Splnitelný dungeon je v tomto případě takový, ve kterém vede cesta od počátku ke všem jeho ostatním dlaždicím[17].

F5: Stejný způsob generování napříč tilesety

Generátor bude generovat dungeony vždy stejným způsobem, nezávisle na zvoleném tilesetu.

F6: Generování hranice dungeonu

Generátor musí být schopen libovolně velké okolí dungeonu vyplnit konkrétní dlaždicí.

F7: Spojování dungeonů

Systém musí umožňovat generování více dungeonů do jedné dungeon scény. Tyto dungeony musí být navzájem propojeny libovolným způsobem. Přechody do dalších dungeonů se musí nacházet na specifikovaných okrajích generovaného dungeonu. Na jednom okraji dungeonu se může nacházet maximálně jeden přechod do dalšího dungeonu. Každý dungeon, který je součástí dungeon scény, musí obsahovat možnost přechodu zpět do vesnice.

F8: Generování nekonečného dungeonu

Systém musí být schopen generovat dungeon pro nekonečný mód, ve kterém hráč musí přežít co nejdéle. Konkrétní způsob, jak by mělo být nekonečnosti dungeonu dosaženo, nebyl uveden. Postupem času není potřeba zvyšovat obtížnost nekonečného dungeonu, systém by si ale měl pamatovat, v kolikátém dungeonu se hráč již nachází. Parametry nekonečného dungeonu nebudou předem stanoveny a systém je tak musí náhodně vygenerovat.

F9: Vytváření nepřátelských skupin

Systém bude na určitých dlaždicích vytvářet skupiny nepřátel. Počet a velikost nepřátelských skupin jsou ovlivněny dalšími parametry dungeonu. Nepřátelské skupiny musí být v dungeonu rovnoměrně rozmístěny, neměly by spolu sousedit, pokud to není nutné. Samotné umístění skupin do scény není zodpovědností generátoru, tuto funkcionalitu budou obstarávat jednotlivé dlaždice. Úkolem generátoru je určit, které konkrétní dlaždice mají nepřátelské skupiny do scény umístit.

F10: Různé velikosti dungeonů

Generátor musí být schopen generovat dungeony o 3 velikostech – malý, střední a velký. Velikost dungeonu ovlivňuje požadovaný počet dlaždic, které se v něm musí objevit, nejsou-li na něj kladeny další požadavky. Je-li po systému požadováno vložení nutných, možných či úkolových dlaždic nebo sousedí-li generovaný dungeon s jiným dungeonem, je možné ignorovat požadovaný počet dlaždic a umístit jich do scény více. Počet nepřátelských skupin v dungeonu je také ovlivněn velikostí dungeonu. Počet dlaždic pro malý dungeon byl nastaven na 20, pro střední na 30 a pro velký na 40. Počet nepřátelských skupin pro malý dungeon byl nastaven na 8, pro střední na 14 a pro velký na 22. Jednotlivé hodnoty se budou v budoucnu měnit na základě zpětné vazby získané od hráčů.

[17] s výjimkou dlaždic typu Border

F11: Různé obtížnosti dungeonů

Generátor musí být schopen generovat dungeony o 3 obtížnostech – lehký, střední a těžký. Obtížnost dungeonu určuje, kolik nepřátelských jednotek bude v dungeonu tvořit jednu nepřátelskou skupinu. Počet nepřátelských jednotek byl pro lehký dungeon nastaven na 3, pro dungeon o střední obtížnosti na 5 a pro těžký na 7. Jednotlivé hodnoty se budou v budoucnu měnit na základě zpětné vazby získané od hráčů.

F12: Umísťování materiálů

Generátor bude muset v dungeonu umísťovat materiály. Počet jednotlivých materiálů bude jedním z parametrů dungeonu. Samotné umísťování materiálů do scény není zodpovědností dungeonu, tuto funkcionalitu budou obstarávat jednotlivé dlaždice. Úkolem generátoru je určit, jaké materiály a v jakém počtu je musí jednotlivé dlaždice do scény umístit. Materiály musí být umístěny uvnitř hranic dlaždice, která jejich umísťování do scény obstarává. Materiály budou umísťovat všechny dlaždice. Není potřeba kontrolovat, kam se materiály umístí. Jestliže budou umístěny uvnitř nějakého jiného objektu, bude tato situace vyřešena systémem kolizí.

F13: Nutné dlaždice

Generátor musí jako jeden ze svých parametrů brát seznam dlaždic, které se ve výsledném dungeonu musí objevit. Takové dlaždice jsou dále označovány jako nutné. Nezáleží na tom, kam generátor nutné dlaždice umístí. Pokud se v seznamu nutných dlaždic objeví dlaždice typu Start, musí se na ní hráč při vstupu do dungeonu objevit. Pokud bude do seznamu vloženo více startovacích dlaždic, jedna z nich se použije a zbytek bude ignorován.

F14: Možné dlaždice

Generátor musí jako jeden ze svých parametrů brát seznam dlaždic, které se ve výsledném dungeonu mohou objevit. U těchto dlaždic musí být uvedena pravděpodobnost, s jakou se v dungeonu dlaždice objeví, a jejich počet. Takové dlaždice jsou dále označovány jako možné. Počet u možné dlaždice označuje, kolikrát se systém bude na základě stanovené pravděpodobnosti rozhodovat, zda bude dlaždice součástí dungeonu nebo ne.

F15: Úkolové dlaždice

Generátor musí jako jeden ze svých parametrů brát seznam dlaždic, které jsou potřebné ke splnění úkolu, včetně údajů o tom, na jaké straně dungeonu se jednotlivé dlaždice musí objevit. Tyto dlaždice budou dále označovány jako úkolové. Na jednom okraji dungeonu se může objevit více úkolových dlaždic.

F16: Umísťování kill squad dlaždic

Systém musí do dungeonu umístit kill squad dlaždice v případě hráčovi špatné reputace. Na těchto dlaždicích se budou vytvářet nepřátelské skupiny, které nebudou ovlivněny parametry dungeonu jako je jeho velikost a obtížnost. Nepřátelé, kteří budou vytvořeni na kill squad dlaždicích se nezapočítávají do požadovaného počtu nepřátelských skupin, který je ovlivněn velikostí dungeonu. V nekonečném dungeonu se kill squad dlaždice umísťovat nebudou.

1.4.2 Nefunkční požadavky

Nefunkční požadavky na modul generátoru se soustředí hlavně na jednoduché rozšiřování generátoru o nová data.

N1: Jednoduché přidání nových tilesetů

Generátor musí být jednoduše rozšiřitelný z hlediska přidání nového tilesetu.

N2: Jednoduché přidání nových typů dlaždic

Generátor musí být jednoduše rozšiřitelný z hlediska přidání nového typu dlaždice.

N3: Jednoduché přidání nového materiálu

Systém musí být jednoduše rozšiřitelný z hlediska přidání nového materiálu.

N4: Jednoduché vkládání parametrů generátoru

Dungeon, jaktožto parametr pro systém generování, musí být jednoduché vytvořit a upravovat.

1.5 Analýza existujících řešení

Existuje hned několik metod procedurálního generování dungeonů. Všechny metody mají ovšem své klady a zápory a správný herní designér by se na jejich základě měl rozhodnout, kterou je vhodné použít v jeho konkrétním případě.

[5] identifikuje 3 přístupy k procedurálnímu generování obsahu – konstruktivní, založené na hledání (search-based) a generuj-a-testuj přístup. [5] dále označuje search-based přístup jako speciální typ generuj-a-testuj přístupu.

Obrázek 1.3 ilustruje rozdíly mezi různými přístupy k procedurálnímu generování obsahu. Můžeme vidět, že konstruktivní přístup oproti přístupu generuj-a-testuj svůj vygenerovaný obsah netestuje. Search-based metody zase místo testování obsahu využívají fitness funkci[18] a následnou úpravu vygenerovaného obsahu.

1.5.1 Konstruktivní přístup

[9] mezi konstruktivní metody řadí dělení prostoru, celulární automat, využití agentů, generativní gramatiky a pokročilé metody generování platforem. Podle [5] je algoritmus Markovova řetězce typická konstruktivní metoda.

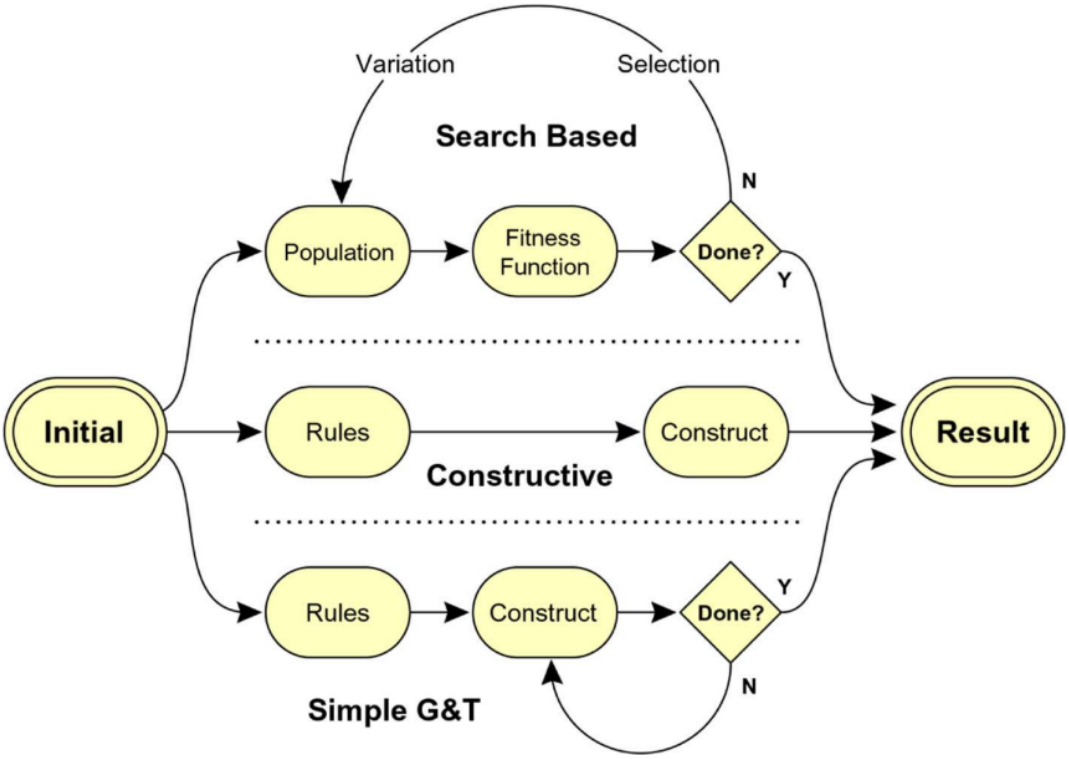
Dělení prostoru

Algoritmy rozdělování prostoru dělí 2D nebo 3D prostor na stejně velké disjunktní části, kterým se také říká buňky. Každé takto vytvořené buňky mohou být dále rekurzivně rozděleny na menší, dokud není splněna určitá podmínka[19]. Takovéto hierarchické rozdělování nám dovoluje reprezentovat celý rozdělovaný prostor jako stromovou datovou strukturu, která se nazývá *space-partitioning tree*. [9]

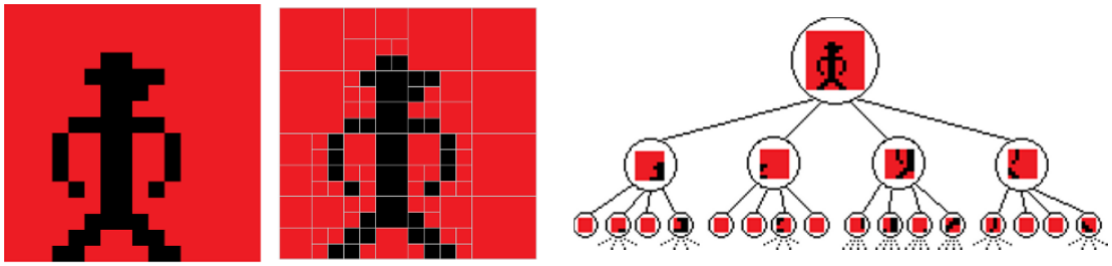
V kořeni stromu se nachází celý rozdělovaný prostor a v jeho synech jsou pak uloženy jeho buňky. Díky svým vlastnostem umožňuje takováto stromová struktura rychle provádět geometrické operace nad jakýmkoliv bodem v prostoru. Toho se s výhodou využívá v oboru počítačové grafiky například k efektivnímu detekování kolizí. [9]

Existuje několik variant algoritmů pro rozdělování prostoru, které se liší v počtu buněk, které vzniknou rozdělením daného prostoru. Nejznámější verzí je binární rozdělování prostoru[20], které

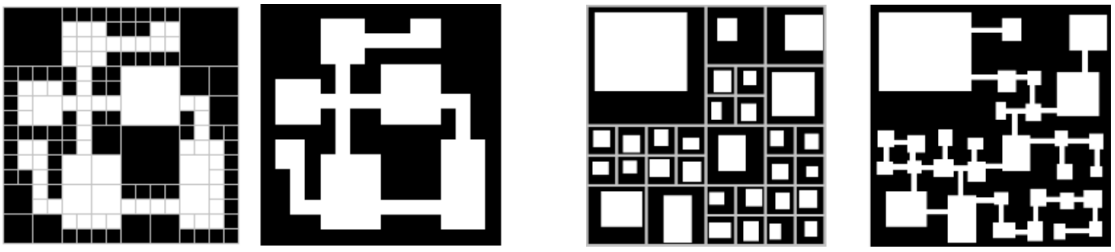
[18] Fitness funkce je funkce, která hodnotí kvalitu vygenerovaného obsahu.
[19] Buňka například dosáhla minimální možné velikosti.
[20] anglicky Binary Space Partitioning – BSP



Obrázek 1.3 Různé přístupy k procedurálnímu generování obsahu. V pořadí odshora dolů – přístup založený na hledání, konstruktivní, jednoduchý generuj-a-testuj. [5]



Obrázek 1.4 Příklad dělení obrázku pomocí stromové datové struktury *quadtree* [9]



Obrázek 1.5 Dungeony vzniklé dělením prostoru [9]

prostor dělí vždy na 2 části. Existují však algoritmy, které prostor vždy rozdělí na 4 nebo 8 částí. [9]

Dělení prostoru na 4 stejně velké části můžeme vidět na obrázku 1.4.

Díky disjunktnosti jednotlivých buněk vznikají prostory, které se nepřekrývají, čehož se dá využít při vytváření místností nebo jiných herních prostorů. Prostor může být dělen pomocí BSP algoritmu a vzniklé buňky mohou být náhodně vybírány na další dělení. Vznikne prostor, jehož buňky mohou mít různé velikosti, což přispívá k originalitě každého nově vytvořeného dungeonu. Takto vzniklé buňky mohou být náhodně označeny jako prázdné či plné nebo do nich mohou být vloženy místnosti o náhodných velikostech[21]. Jednotlivé místnosti musí být následně ještě propojeny cestami, aby mezi nimi mohl hráč volně přecházet. [9]

Příklady dungeonů vzniklých metodou dělení prostoru můžeme vidět na obrázku 1.5.

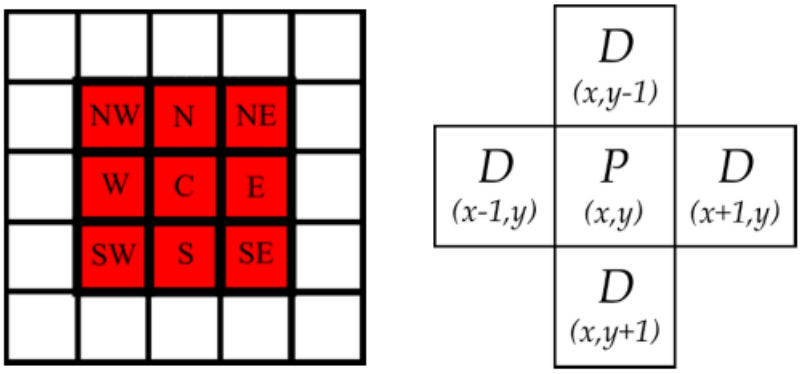
Výsledkem BSP algoritmu při generování dungeonů je velice strukturovaný prostor. Hierarchické struktury stromu se dá ještě využít například k nastavení motivů u skupin místností. Můžeme vybrat nějaký uzel stromu, který není listem, a říct, že všichni jeho synové budou mít stejný motiv[22]. Takto vybrané buňky budou umístěny ve výsledném prostoru blízko u sebe a při použití algoritmu pro vytváření cest zmiňovaného v [9] mezi nimi budou existovat cesty. [9]

[29] vidí nedostatek této metody v tom, že se místnosti musí nacházet celé uvnitř jedné buňky, což omezuje pozice, na které mohou být určité místnosti vloženy. Protože algoritmus BSP nemůže garantovat to, že spolu budou specifické tvary buněk sousedit, je komplexní uspořádání místností typicky vyloučeno. [29]

Celulární automat

Celulární automat je diskrétní výpočetní model, který je studován v oborech jako jsou výpočetní technologie, fyzika nebo biologie. Celulární automat sestává z n-dimenzionální[23] mřížky, která se skládá z buněk. Každá buňka se může nacházet v několika stavech, které se mění na základě pravidel stanovených pro konkrétní automat. Existuje několik druhů celulárních automatů, které

[21] Tak, aby velikost místnosti nepřekročila velikost buňky.
[22] Například budou obsahovat poklady, více nepřátel nebo budou mít odlišný vizuál
[23] nejčastěji jedno- nebo dvoudimenzionální



Obrázek 1.6 Dva typy sousedství, nalevo Moorovo, napravo Von Neumannovo. [9]

se navzájem liší strukturou svých mřížek, množinou stavů buněk nebo pravidly pro změnu stavů buněk. Některé z těchto druhů jsou dokonce Turingovsky kompletní[24]. [9]

Buňky automatu se na počátku, v čase t_0, nachází ve výchozím stavu. Následně dochází v diskrétních krocích ke změnám stavů buněk na základě předem určených pravidel. Stav buňky v čase t_n je ovlivněn stavem dané buňky a stavy jejích sousedů v čase t_{n-1}. Ve 2D mřížce existují dva druhy sousedství, které může celulární automat kontrolovat při zkoumání stavů sousedů určité buňky – Moorovo sousedství a Von Neumannovo sousedství. Na obrázku 1.6 můžeme vidět oba typy sousedství. Von Neumanovo sousedství je tvořeno buňkami, které leží na levém, pravém, horním a spodním okraji zkoumané buňky a společně se zkoumanou buňkou tvoří strukturu připomínající kříž. Moorovo sousedství přidává k sousedním buňkám z Von Neumannova sousedství i buňky, které se zkoumanou buňkou sousedí diagonálně. [9]

Cílem [30] bylo procedurální generování jeskynních levelů v reálném čase a pro tento úkol si zvolil celulární automat. [30] si vytvořil čtvercovou 2D mřížku, která se skládá z 50x50 buněk a nazval ji základovou mřížkou.

Každá z buněk se musí nacházet v jednom ze 3 stavů: *podlaha*, *zeď* nebo *skála*. Každá z buněk v mřížce začíná jako *podlaha* a následně je 50 % buněk náhodně změněno na skály. Celulární automat v každém kroku iteruje mřížkou a u každé buňky zkoumá její Moorovo sousedství. Je-li v sousedství 5 a více buněk ve stavu *skála*, pak bude následující stav zkoumané buňky také *skála*, jinak bude stav buňky *podlaha*. Takto celulární automat provede n kroků. [30] poznamenává, že čím vyšší je číslo n, tím širší jsou v jeskyni výsledné chodby[25]. [30]

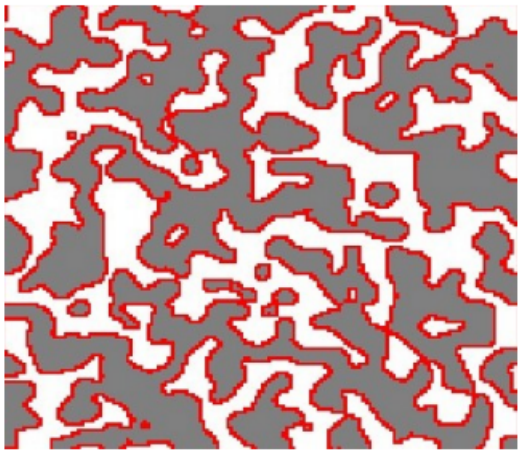
Po n iteracích algoritmu celulárního automatu jsou všechny buňky ve stavu *skála*, které sousedí s buňkou ve stavu *podlaha*, změněny do stavu *zeď*. [30] následně vytvořil další mřížky ve Von Neumannově sousedství základové mřížky a stejným způsobem v nich vygeneroval jeskynní chodby. Pro všech nynějších 5 mřížek je zkontrolováno, že existuje spojení s jejími sousedy. Jestliže ne, vybraly se 2 buňky ve stavu *podlaha*, které jsou nejblíže k hranici mezi dvěma mřížkami a mezi nimi se vygeneroval tunel určité šířky. Celá struktura, nyní tvořená 5 mřížkami, byla následně upravena dalšími 2 kroky celulárního automatu, aby se uhladily nově vygenerované tunely. [30]

Příklad takto vygenerované jeskyně můžeme vidět na obrázku 1.7.

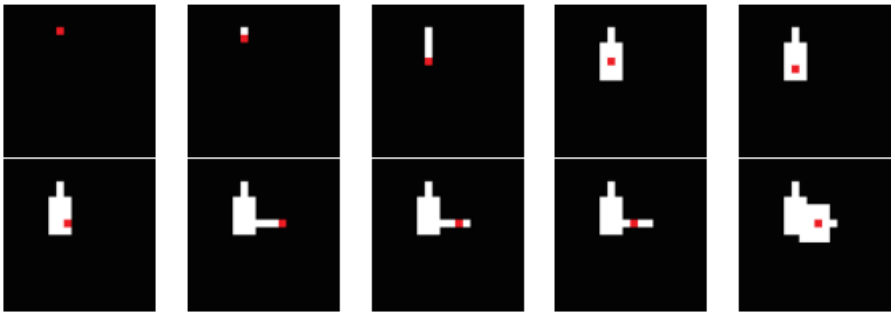
Využití agentů

▶ **Definice 1.1.** *Agent je počítačový systém, který je umístěn v nějakém prostředí a v tomto prostředí je schopen samostatné činnosti za účelem dosažení svých (anebo delegovaných) cílů. [31]*

[24] Mají stejnou výpočetní sílu jako Turingův stroj.
[25] Větší počet buněk se dostane ze stavu *skála* do stavu *podlaha*.



Obrázek 1.7 Příklad jeskyně vygenerované v [30]. Červenou barvou jsou znázorněny zdi, bílou barvou skály a šedou barvou podlahy.

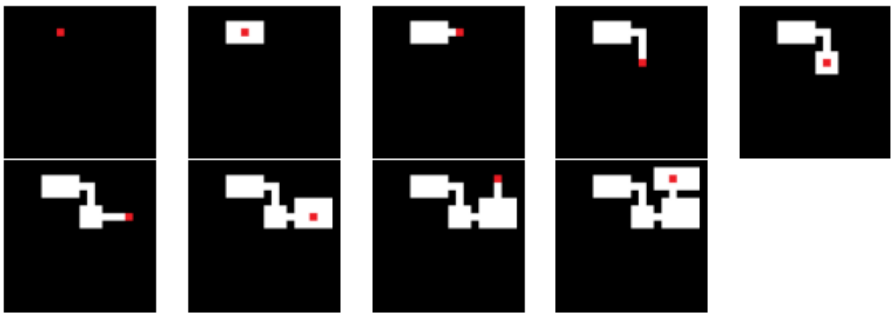


Obrázek 1.8 Znázornění krátkého běhu slepého agenta [9]

[9] uvádí využití agentů jako způsob schopný generování více organických a chaotických dungeonů ve srovnání s metodou rozdělování prostoru, která produkuje spíše úhledně uspořádané dungeony. Chaotičnost výsledného prostoru je ovlivněna stupněm náhodnosti agenta, byť je obecně obtížné určit, jak úprava parametrů chování agentů ovlivní výsledný vzhled dungeonu.[9] [9] navrhuje 2 různé přístupy generování dungeonů založené na agentech. Dungeony jsou v [9] reprezentovány 2D mřížkou, která je zpočátku zaplněna zdmi, tedy neprůchozími políčky. Při svém průchodu dělají agenti ze zdí chodby, které už průchozí jsou.

První agent, kterého [9] představuje je velice náhodný, [9] ho nazývá slepým. Agent se objeví na náhodné pozici v dungeonu, náhodně si vybere svůj směr a v něm začne kopat. Každé políčko, které je agentem vykopané, se počítá jako chodba. Po prvním kopání je 5% šance, že agent změní svůj směr, a 5% šance, že na své současné pozici umístí místnost náhodné velikosti. Pokaždé, když se agent pohne ve stejném směru, zvyšuje se šance, že změní svůj směr, o 5 %. Pokaždé, když agent na své současné pozici neumístí místnost, zvyšuje se šance, že místnost v dalším kroku umístí, o 5 %. V moment, kdy agent změní směr, šance na změnu směru se snižuje na 0 %. V moment, kdy agent umístí místnost, šance na umístění místnosti se také snižuje na 0 %. Obrázek 1.8 znázorňuje jeden krátký potenciální běh slepého agenta. Je možné z něj vypozorovat nevýhody tohoto přístupu. Agent vkládá místnosti náhodně, ty se tedy mohou překrývat. Další nevýhodou je možnost existence slepých uliček. [9]

[9] proto navrhuje agenta, který je méně náhodný a je si více vědom celkového vzhledu dungeonu. Agent opět začíná na náhodné pozici v dungeonu. Agent zkontroluje, jestli by se místnost umístěná na jeho současné pozici překrývala s jinou místností. Jestliže by umístění jakékoliv



Obrázek 1.9 Znázornění krátkého běhu informovaného agenta [9]

místnosti skončilo překrytím jiné místnosti, agent začne kopat. Agent zvolí takový směr a délku kopání, aby výsledná chodba vzniklá jeho kopáním neprotínala žádnou existující chodbu či místnost. Obrázek 1.9 ukazuje, jak by mohl probíhat běh takového agenta. Je z něj opět možné vypozorovat jednu z jeho nevýhod. Generování dungeonu skončilo, protože pro agenta nebylo možné pohnout se ze své pozice tak, aby neprotnul nějakou místnost či chodbu. Většina dungeonu však zůstala nezaplněná. [9]

Předchozí obrázky ukazují nejhorší možné scénáře generování těchto agentů. [9] dále zmiňuje, že se problémům s agentovým generováním dá vyhnout přidáním více či méně komplexních kusů kódu, je ale stále obtížné předvídat tyto problémy bez intenzivního testování agentova algoritmu.

[32] pro rozvržení dungeonu používá slepého agenta z [9]. Úkolem agenta je vyplnit 2D mřížku o velikosti 8x18 políček průchozími políčky. Agent je modifikován tak, aby na rozdíl od slepého agenta v [9] nemohl přidávat místnosti. Agent se tedy náhodně po mřížce pohybuje – kope dungeon – dokud v něm není určitý počet průchozích políček. Tento počet je náhodně stanoven na hodnotu mezi 75 až 95 políčky. [32] následně využívá agenty i na vložení různých objektů do dungeonu jako jsou například poklady, monstra, lektvary, portály či východy z dungeonu.

Objekty jsou ve [32] vloženy na náhodné pozice v dungeonu a po dungeonu se následně pohybují po tazích. Každý typ objektu má svá vlastní pravidla na pohyb v dungeonu. Například východy se snaží dostat co nejdál od vchodu, poklady se snaží dostat co nejblíže ke goblinům a lektvary se pohybují zcela náhodně. Po dobu 45 tahů se objekty na základě pro ně stanovených pravidel pohybují po dungeonu v závislosti na tom, co ze své pozice vidí a jaká je jejich vzdálenost od ostatních objektů. [32]

Gramatiky

Původním účelem generativních gramatik byl formální popis struktur v přirozeném jazyce. K tomuto popisu se využívá konečná množina rekurzivních pravidel, která ukazuje, jak se větší struktury staví z těch menších. [9]

Kromě rekurzivních pravidel je další důležitou součástí gramatiky její abeceda, tedy množina symbolů, se kterými gramatika pracuje. Gramatika tyto symboly přepisuje na základě svých pravidel na jiné symboly, které tvoří nový řetězec. Některé symboly, značené malými písmeny abecedy, už nelze dále přepsat, protože neexistují pravidla na jejich přepsání. Takové symboly nazýváme terminály. Symboly, pro jejichž přepis existují pravidla, nazýváme neterminály a značíme je obvykle velkými písmeny abecedy. Aby byla gramatika kompletní, potřebuje ještě počáteční symbol. Ten je obvykle označen písmenem S. [27]

Tedy například pravidlo, které říká, že se symbol S přepíše na symboly „ab“ by vypadalo následovně: S → ab. Pravidla umožňují rekurzi – lze pravidly produkovat neterminály, na které se dříve či později aplikuje stejné pravidlo. Gramatiky tak mohou vyprodukovat opravdu zajímavé výsledky. V případě pravidla S → abS lze vygenerovat nekonečný řetězec za sebou opakujících se symbolů „ab“. [27]

Jestliže jako abecedu použijeme množinu symbolů reprezentující specifické herní koncepty a jako pravidla použijeme různé způsoby, jak tyto koncepty kombinovat tak, aby vznikl správný herní level, můžeme gramatiku využívat na jejich generování. Příklad takových pravidel můžeme vidět níže.[26] [27]

1. Dungeon → Překážka + poklad
2. Překážka → klíč + Překážka + zámek + Překážka
3. Překážka → příšera + Překážka
4. Překážka → místnost

Generováním na základě těchto pravidel mohou vzniknout následující řetězce:

1. klíč + příšera + místnost + zámek + příšera + místnost + poklad
2. místnost + poklad
3. příšera + příšera + příšera + příšera + místnost + poklad

[27] na základě svého předchozího výzkumu rozděluje herní level na 2 struktury – misi a prostor. Mise je série úkolů, které hráč musí splnit, aby herního level dokončil. Prostor vyjadřuje geometrické rozložení herního levelu. Obě tyto struktury je možné reprezentovat grafem, jak můžeme vidět na obrázku 1.10. Klíčový je zde fakt, že byť jsou tyto dvě struktury na sobě nezávislé, je možné je na sebe namapovat. [27]

Na generování mise používá [27] grafové gramatiky. Grafové gramatiky jsou speciálním typem gramatik, které produkují místo řetězců grafy.

▶ **Definice 1.2.** *Graf je uspořádaná dvojice (V, E), kde V je neprázdná konečná množina vrcholů (nebo také uzlů), E je množina hran. Hrana je dvouprvková podmnožina V (čili neuspořádaná dvojice vrcholů). [33]*

Grafové gramatiky nahrazují jednu strukturu skládající se z uzlů spojených hranami jinou strukturou uzlů a hran. [27] Příklad takto vygenerovaného grafu, který může reprezentovat misi můžeme vidět na obrázku 1.11.

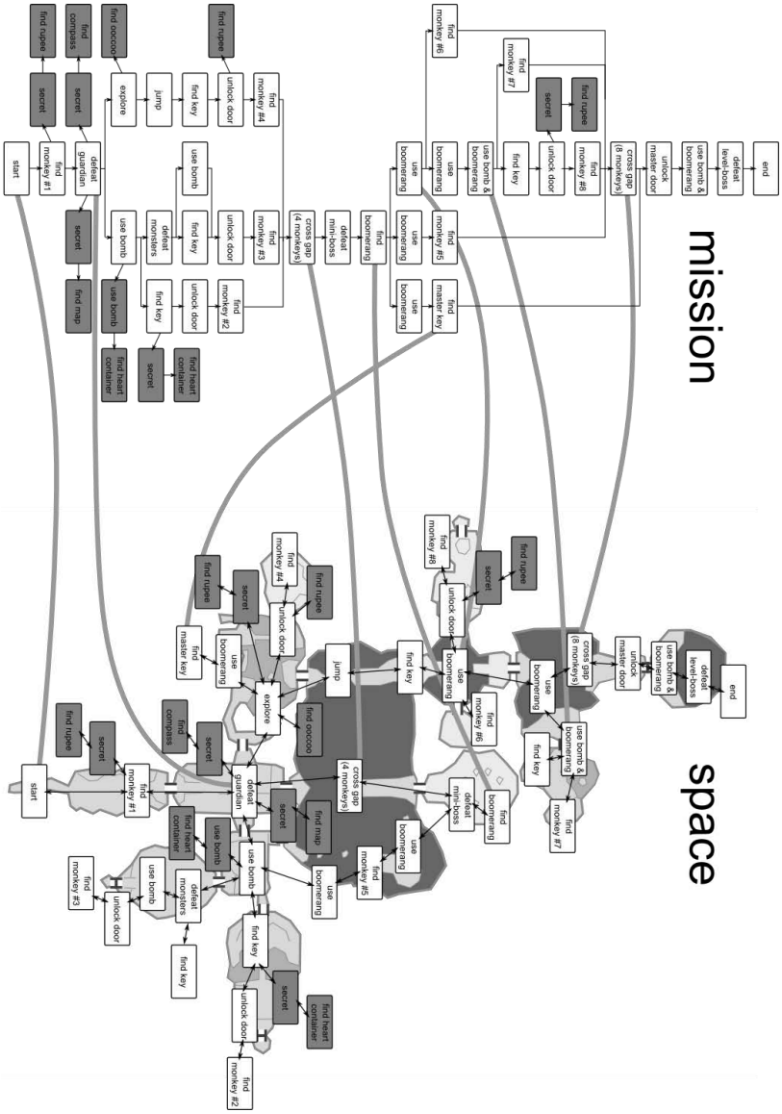
Na generování prostoru používá [27] tvarové gramatiky. Podobně jako v generativních a grafových gramatikách i v tvarových gramatikách existují přepisovací pravidla, jimiž se gramatika řídí. Tvarové gramatiky tato pravidla používají pro nahrazování tvarů. [27] jako příklad definuje abecedu se třemi prvky – zdí, volným prostorem a spojením – kde pouze spojení je neterminálním symbolem. Na obrázku 1.12 můžeme vidět abecedu, pravidla i příklad tvaru vygenerovaného tvarovou gramatikou.

[27] následně na základě mise generuje prostor. Každému pravidlu ve tvarové gramatice použité pro generování prostoru, je přiřazen příslušný terminální symbol z grafové gramatiky, která byla použita pro generování mise. Generování prostoru pak probíhá následovně:

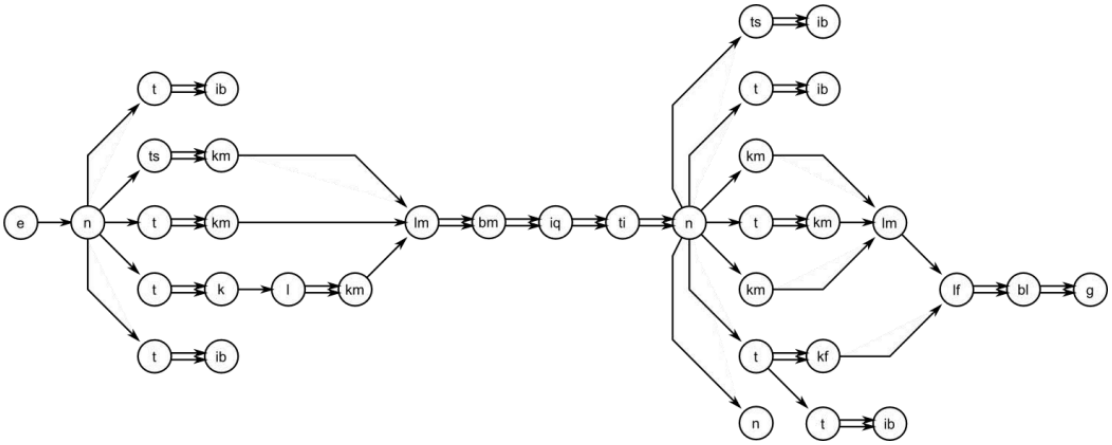
- V grafu reprezentujícím misi je nalezen další symbol, který ještě nebyl v prostoru reprezentován.
- Pro tento symbol je náhodně vybráno pravidlo z tvarové gramatiky.
- Na vhodné místo v již vygenerovaném prostoru je toto pravidlo aplikováno.

Nakonec jsou všechny neterminály v prostoru vygenerovaném tvarovou gramatikou nahrazeny terminálními symboly podle pravidel, která jsou určena pro finalizaci prostoru. Na obrázku 1.13 můžeme vidět, jak takové generování probíhá pro jeden konkrétní graf reprezentující misi.

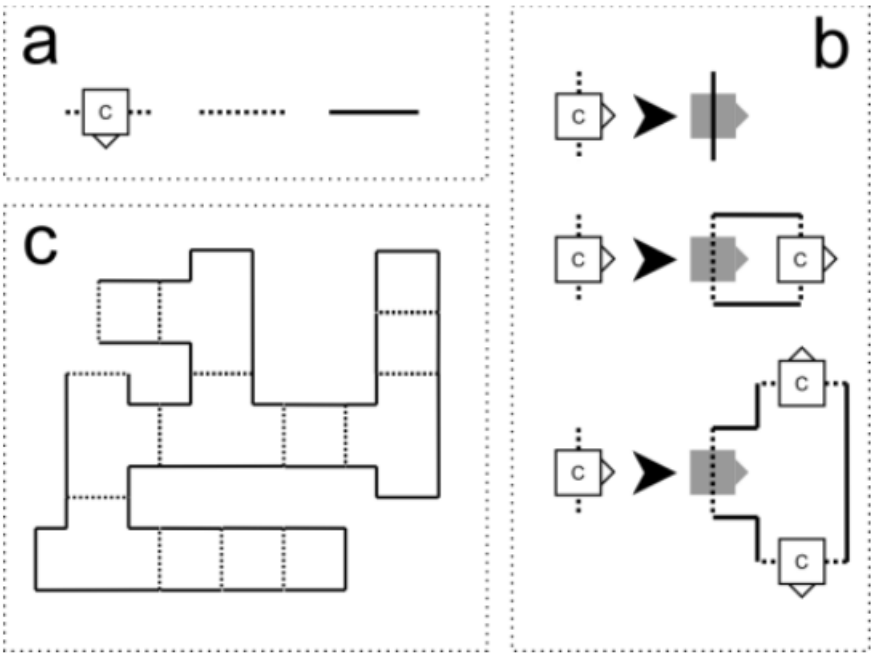
[26] Pravidla i jejich výsledky jsou převzaty z [27] a byly autorem přeloženy.



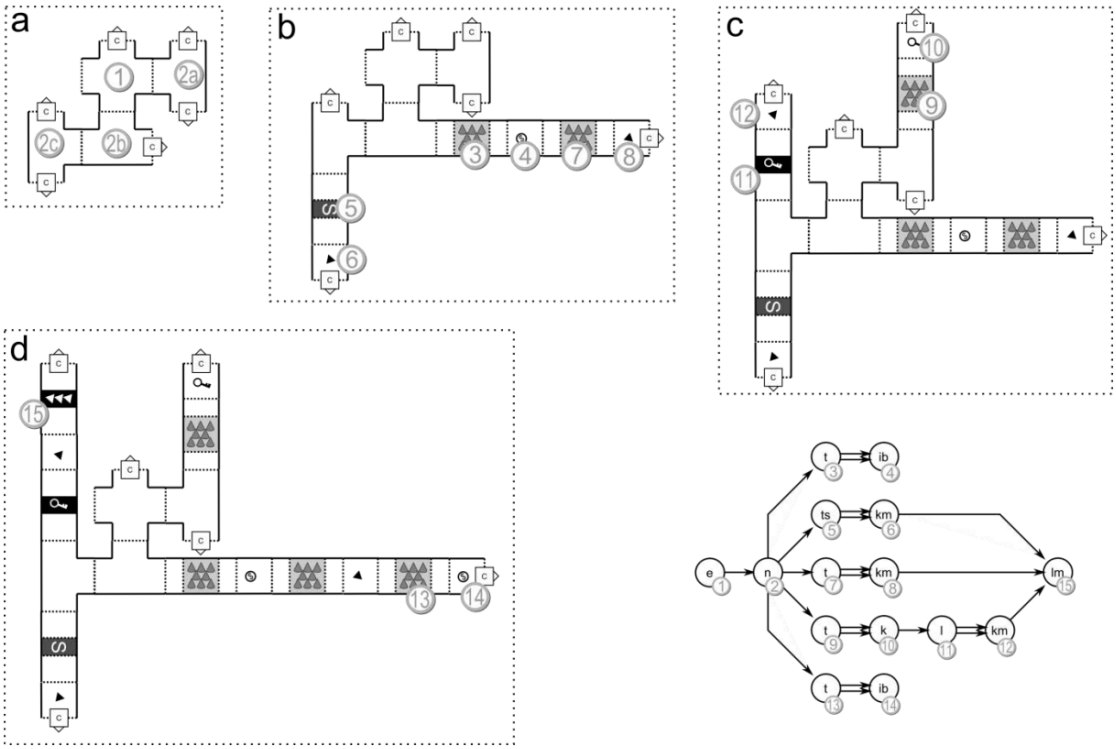
Obrázek 1.10 Příklad dvou struktur – mise a prostoru – a jejich namapování [27]



Obrázek 1.11 Příklad grafu struktury mise [27]



Obrázek 1.12 (a) Abeceda tvarové gramatiky (b) Pravidla tvarové gramatiky (c) Výsledek generování tvarové gramatiky [27]



Obrázek 1.13 Příklad grafu prostoru herního levelu vygenerovaného na základě mise v pravém dolním rohu obrázku. [27]

Pokročilé metody generování plošin

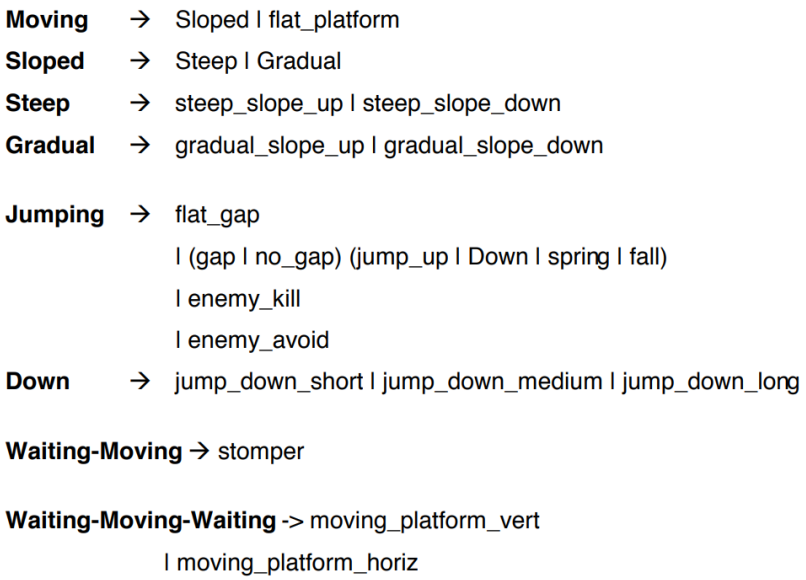
[9] představuje 2 metody, které byly původně navrženy pro generování plošinových levelů a nepodařilo se je zařadit do žádné z předchozích kategorií – Rythm-based generation[27] a Occupancy-regulated extension[28]. Byť tyto metody na první pohled nemusí vypadat jako využitelné pro generování dungeonů, [9] zmiňuje, že mimo jiné jejich centrální koncepty by samotné generování dungeonů mohly vylepšit. [9] jako jeden z příkladů počítačových her, kde se generování platforem využívá, uvádí hru *Spelunky*. *Spelunky* je 2D skákačka, jejíž herní úrovně by se daly podle některých definic klasifikovat jako dungeony, [26] by tento typ dungeonu v jejich rozdělení označila za side-scrolling dungeon.

Rythm-based generation

[1] přichází s přístupem generování levelů pro 2D plošinovky založeném na rytmu. Systém generování se skládá ze dvou přístupů založených na gramatikách. Nejdříve se vygeneruje množina akcí, které bude hráč v levelu muset vykonat, tak, aby tvořily rytmus. Následně se tato množina pomocí gramatik převede na geometrii daného levelu.

Vytvořený rytmus se skládá z názvu akce, kterou hráč musí vykonat, a dvojice časových údajů, které ohraničují vykonávání této akce. Příklad rytmu můžeme vidět na výpisu kódu 1.1.

[27] Do češtiny možno přeložit jako generování založené na rytmu.
[28] Do češtiny možno přeložit jako rozšiřování řízené obsazeností.



Obrázek 1.14 Pravidla geometrické gramatiky [1]

Výpis kódu 1.1 Příklad rytmu. Rytmus převzat z [1] a autorem přeložen.

```
pohyb 0 5
skok 2 2,25
skok 4 4,25
pohyb 6 10
skok 6 6,5
skok 8 8,5
```

Tento rytmus nám říká, že se hráč musí přesouvat prvních 5 sekund, ve 2. a 4. sekundě musí vyskočit, mezi 5. a 6. sekundou musí hráč čekat, následně se od 6. do 10. sekundy pohybuje a v 6. a 8. sekundě musí vyskočit. Součástí tohoto rytmu jsou dva druhy skoků, přičemž jeden trvá 0,25 sekundy a druhý 0,5 sekundy. Trvání skoku odpovídá množství času, po který musí hráč držet tlačítko vyhrazené pro skok herní postavy.

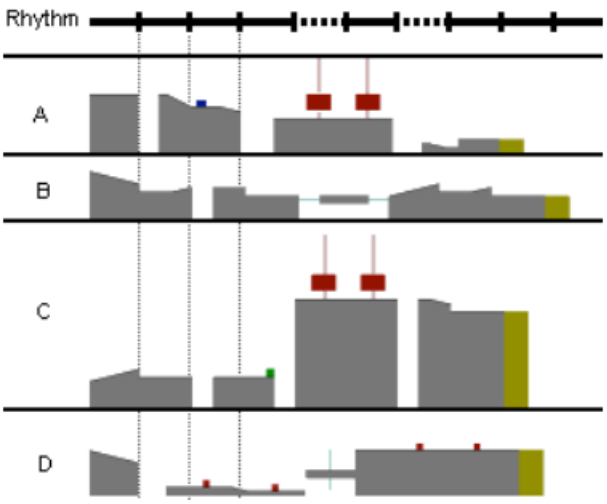
Na základě vygenerovaných rytmů je vytvořen seznam stavů hráče, které jsou použity jako neterminály gramatiky, která má za úkol vytvořit geometrii herní úrovně. Pravidla geometrické gramatiky využívané v [1] lze vidět na obrázku Na základě jednoho rytmu může vzniknout několik geometrií pro herní level. Na obrázku 1.15 můžeme vidět příklad jednoho rytmu a čtyř různých geometrií herní úrovně.

[9] poznamenává, že byť není koncept rytmu příliš použitelný pro generování dungeonů, tempo procházení místnostmi a chodbami by pro dungeony už hodnotné být mohlo. I využití dvouúrovňové gramatiky, kdy každá úroveň obstarává generování jiné části herního levelu, se velmi dobře pojí s generováním dungeonu, jak bylo ukázáno v sekci Gramatiky.

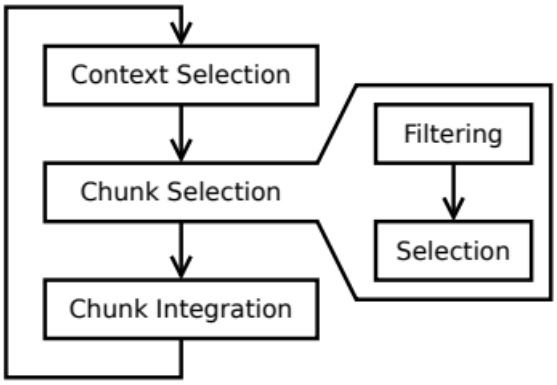
Occupancy-regulated extension

[34] generuje 2D plošinovky pomocí metody occupancy-regulated extension (ORE). Metoda skládá dohromady herní level z předpřipravených kusů levelu, které [34] nazývá chunky. Tyto kusy levelu vkládá na pozice – kotvy – které symbolizují místa, kam by se v levelu mohl hráč potenciálně dostat. Herní level začíná s kotvou na pozici, kde by měl v levelu hráč začínat a připravenou knihovnou chunků. [34]

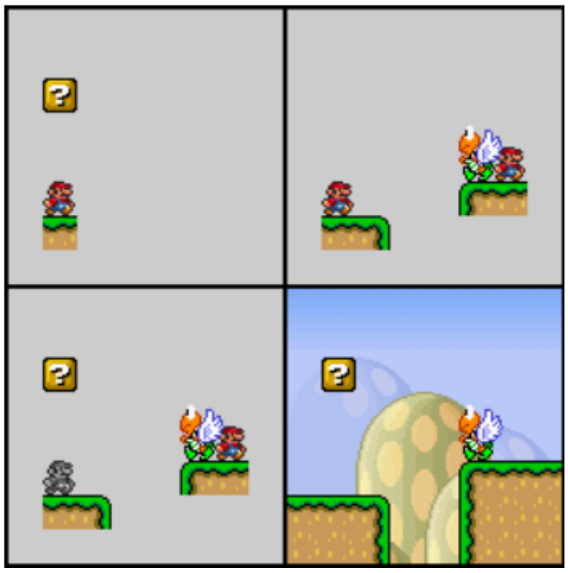
Fungování algoritmu je ilustrováno na obrázku 1.16.



Obrázek 1.15 Příklady herních levelů vygenerovaných rythm-based technikou [1]



Obrázek 1.16 Jednotlivé kroky algoritmu ORE [34]



Obrázek 1.17 Konkrétní případ generování pomocí metody ORE. V levém horním rohu vidíme kotvu (symbolizovanou ikonkou hráče), kterou jsme se v první fázi rozhodli rozšířit. V pravém horním rohu je zvolen chunk, který prošel druhou fází a bude vložen do herního levelu. Vlevo dole je současná kotva označena jako použitá. Vpravo dole vidíme zaintegrování vybraného chunku do existující geometrie. [34]

V první fázi – výběru kontextu – je vybrána kotva, která bude následně rozšířena o nový chunk. V druhé fázi algoritmus vyfiltruje všechny chunky na základě jejich kompatibility v současném kontextu. Z vyfiltrovaných chunků je následně vybrán jeden konkrétní na základě pro něj spočítané váhy. Váha chunku je ovlivněna například tím, kolikrát se tento chunk již v levelu objevil. Pravděpodobnost výběru konkrétního chunku koresponduje s jeho váhou. Poslední, třetí fáze sestává z integrace vybraného chunku do již existující geometrie levelu. Kotva, která byla rozšířena současným chunkem, je označena jako použitá. Potenciální lokace hráče na vybraném chunku jsou přidány do existujícího obsahu levelu, aby mohly být v budoucnu vybrány ve fázi výběru kontextu. [34]

Konkrétní případ generování můžeme vidět na obrázku 1.17.

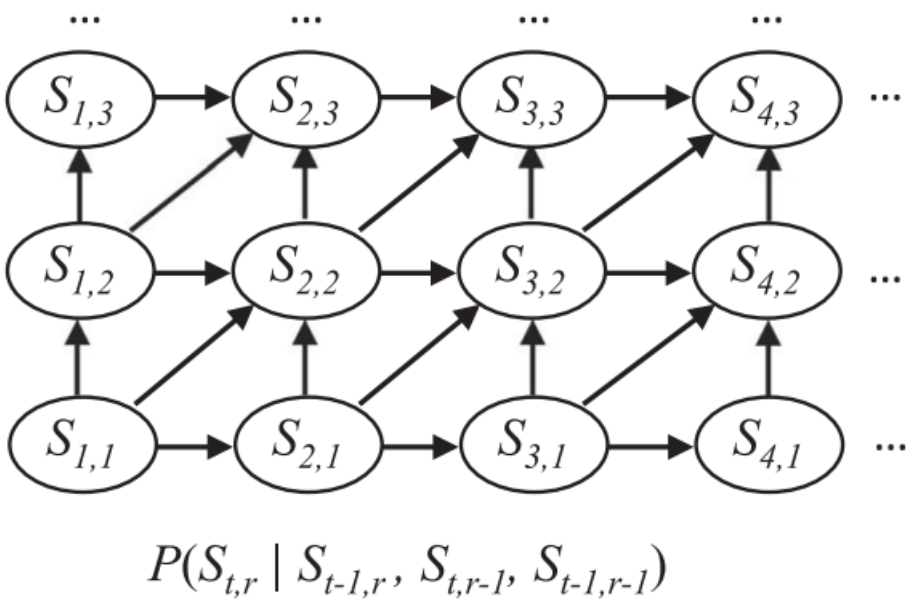
Tento způsob je určen pro vytváření obecného 2D levelu s platformami. Specifické herní elementy tak musí být do levelu ještě vloženy. [9]

[9] považuje myšlenku vytvoření knihovny předpřipravených částí levelu jako velmi dobře využitelnou i při generování dungeonů. Zároveň však dodává, že pro generování 3D herních levelů je potřeba knihovna větší a komplexnější než pro generování 2D levelů.

Markovovy modely

[35] zkoumá využití Markovových modelů pro generování videoherních map s cílem nalézt doménově nezávislý způsob generování. Konkrétně zkoumá 3 různé Markovovy modely – multidimenzionální Markovovy řetězce[29] (MdMC), hierarchické multidimenzionální Markovovy řetězce[30] (HMdMC) a Markovovy sítě[31] (MRF). Jedná se o metody strojového učení, které potřebují trénovací mapy pro cílovou doménu. Tyto 3 přístupy jsou následně testovány na 3 hrách – *Super Mario Bros.*, *Loderunner* a *Kid Icarus*.

[29] anglicky multidimensional Markov chains
[30] anglicky hierarchical multidimensional Markov chains
[31] anglicky Markov random fields



Obrázek 1.18 Markovův řetězec třetího řádu [35]

„Markovovy řetězce modelují stochastické přechody mezi stavy v průběhu času.“ [35] Markovův řetězec je tvořen množinou stavů S a distribucí podmíněných pravděpodobností $P(S_t|S_{t-1})$ představující pravděpodobnost přechodu do stavu $S_t \in S$ za předpokladu, že předchozí stav byl $S_{t-1} \in S$. Markovovy řetězce vyšších řádů se neomezují pouze na jeden předchozí stav, ale umožňují pravděpodobnost přechodu založit na d předchozích stavech, kde d je přirozené číslo. [35]

Rozšířením Markovových řetězců vyššího řádu jsou multidimenzionální Markovovy řetězce, které umožňují pravděpodobnost přechodu založit na okolních stavech v multidimenzionálním grafu. [35]

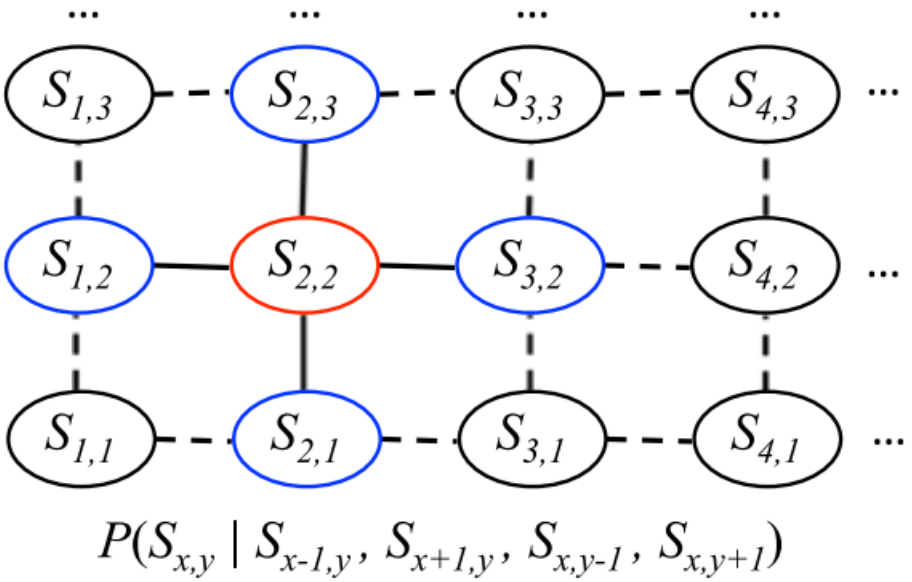
Na obrázku 1.18 můžeme vidět příklad multidimenzionálního Markovova řetězce třetího řádu, včetně obecného zápisu jeho distribuce podmíněných pravděpodobností.

„Markovovy sítě modelují pravděpodobnostní vztahy mezi okolními stavy v grafu.“ [35] Na obrázku 1.19 můžeme vidět příklad Markovovy sítě včetně obecného zápisu její distribuce podmíněných pravděpodobností.

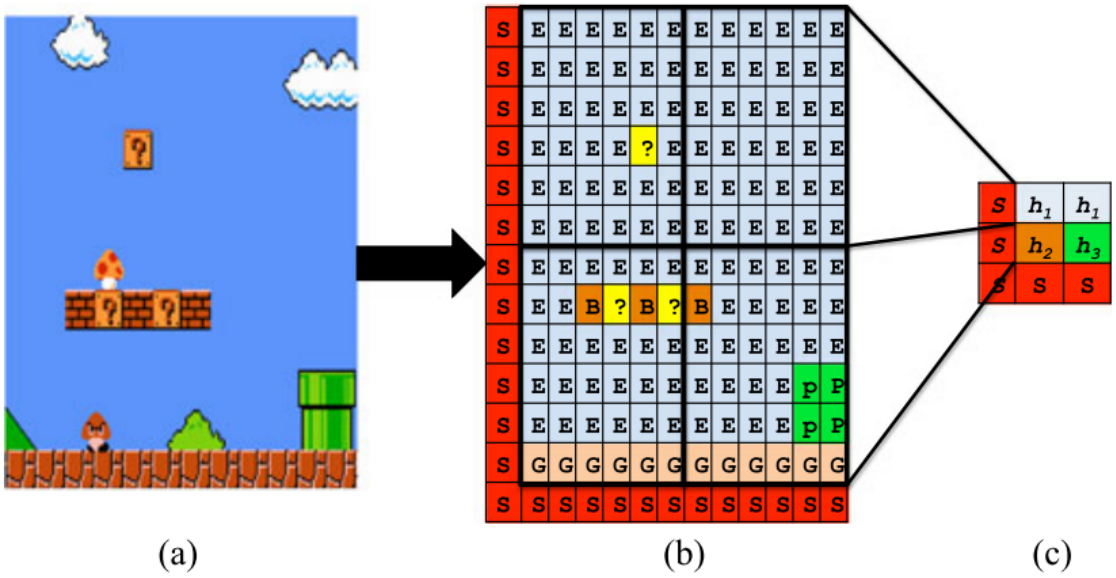
Herní mapa je v [35] reprezentována jako 2D mřížka, ve které má každá buňka hodnotu z konečné množiny dlaždic. Při trénování modelů je nejdřív spočteno, kolikrát se každý typ herní dlaždice nachází v každé konfiguraci nalezené na trénovacích mapách. Následně je určena pravděpodobnost každého typu dlaždice v každé konfiguraci, se kterou se algoritmus na testovacích mapách setkal. Na základě těchto pravděpodobností je vygenerován herní level. [35]

[35] vychází ze svých přechozích prací, ve kterých bylo zjištěno, že Markovovy modely nejsou schopny dostatečně dobře zachytit závislosti na větší vzdálenost. Rozhoduje se proto používat hierarchické MdMC (HMdMC).

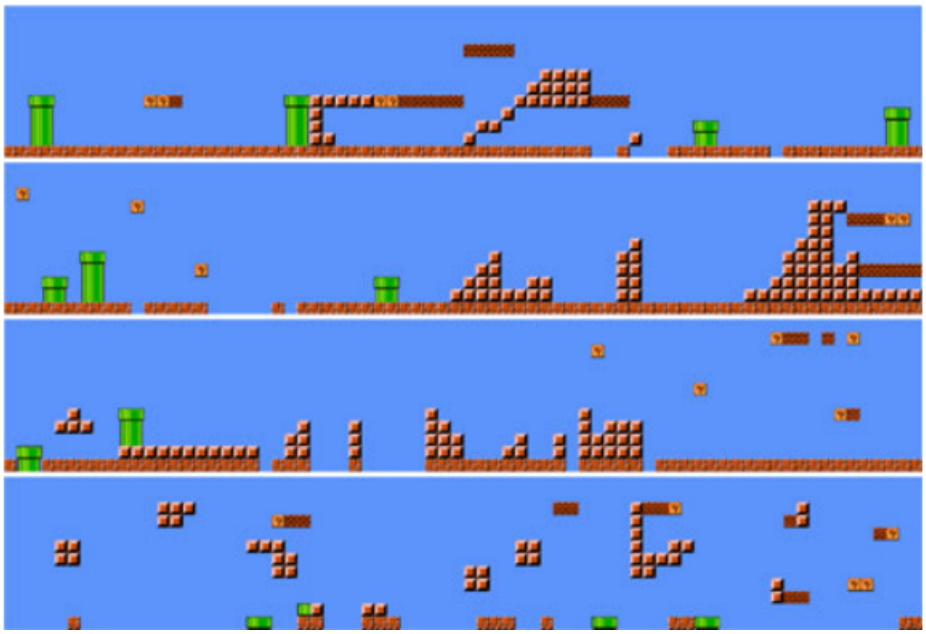
HMdMC pro trénování používá high-level reprezentaci herní mapy. High-level mapa o velikosti ZxZ vznikne rozdělením low-level mapy na ZxZ segmentů a jejich namapováním na buňky high-level mapy. Rozdíl mezi trénovací mapou, její low-level a high-level reprezentací je vyobrazen na obrázku 1.20. Algoritmus nejdříve trénuje klasický MdMC na high-level reprezentacích trénovacích herních map. Následně jsou trénovány další MdMC pro každou buňku high-level reprezentace mapy. Při generování herní mapy je nejdříve vygenerována její high-level reprezentace a následně její low-level reprezentace. Při vytváření buňky v low-level reprezentaci je v potaz brána pozice high-level buňky, pod kterou příslušná low-level buňka spadá. [35]



Obrázek 1.19 Markovova síť, kde každý stav závisí na čtyřech okolních stavech. Stav označený červenou barvou je závislý na stavech označených modrou barvou. [35]



Obrázek 1.20 Ukázka trénovací mapy (a), její low-level (b) a high-level (c) reprezentace [35]



Obrázek 1.21 Příklady herních map vygenerovanách metodou Markovových modelů [35]

MRF modeluje závislosti ve všech směrech v grafu. V první části trénování MRF je spočteno, kolikrát je dané políčko obklopeno každou konfigurací. Pro každou kombinaci konfigurace a políčka algoritmus nastaví pravděpodobnost, že bude dané políčko obklopeno danou konfigurací. Mapa je následně vytvořena na základě distribuce vypozorované z trénovacích map nezávislé na okolích vytvářených políček. Následně jsou náhodně vybrány 2 pozice na mapě a políčka na nich jsou s určitou pravděpodobností prohozeny. Toto se děje, dokud nebylo prohozeno určité množství políček. [35]

Tyto 3 metody vytvořily hratelné mapy pro hru *Super Mario Bros.* ve 21,9–90,4 % případů, u zbylých dvou her se pravděpodobnosti pohybovaly v rozmezí 0-–3,2 %. [35] Na obrázku 1.21 můžeme vidět příklady herních map vygenerovaných pro hru *Super Mario Bros.*.

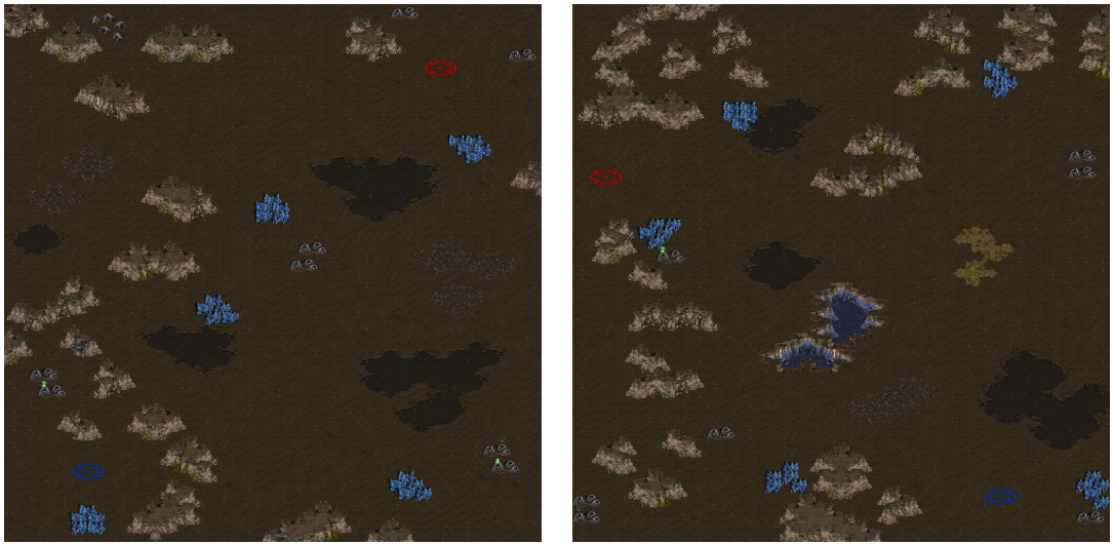
Nízká úspěšnost u her *Kid Icarus* a *The Loderunner* je způsobena neschopností modelů zachytit závislosti na větší vzdálenost. U hry *Super Mario Bros.* tato neschopnost často nevedla k vytvoření nehratelných levelů. [35]

1.5.2 Search-based metody

Podle [5] je procedurální generování obsahu založené na hledání (search-based) speciální typ generuj-a-testuj přístupu k PCG a má následující vlastnosti:

- Vygenerovaný obsah není ve fázi testování dělen na vyhovující a nevyhovující, ale je mu přiřazena hodnota nebo vektor hodnot na základě tzv. fitness funkce.
- Generování nového obsahu je závislé na hodnotách, které byly přiřazeny již vygenerovanému obsahu. Cílem je vytvořit nový obsah s lepší hodnotou.

Přístupů k search-based PCG je mnoho, ale podle [5] jich nejvíce využívá nějakou formu evolučních algoritmů. [9] i [26] řadí mezi search-based algoritmy takové, které ke generování obsahu využívají programování množinou dotazů.



Obrázek 1.22 Mapy vygenerované do hry *StarCraft* pomocí metody evolučních algoritmů [36]

Evoluční algoritmy

[9] popisuje evoluční algoritmus jako stochastický algoritmus hledání inspirovaný Darwinovskou evolucí.

Evoluční algoritmus si v paměti udržuje instance kandidátského obsahu. V každé generaci jsou tito kandidáti ohodnoceni fitness funkcí. Nejhorší kandidáti jsou nahrazeni kopiemi dobrých kandidátů, které byly náhodně pozměněny. Pro funkci evolučního algoritmu je potřeba správně navrhnout způsob reprezentace generovaného obsahu a fitness funkci, která bude obsah hodnotit. [5]

Podle [5] se dá v evoluční terminologii popsat hledání vhodného způsobu reprezentace generovaného obsahu jako hledání vhodného způsobu mapování genotypů na fenotypy. [9] připodobňuje rozdíl mezi genotypem a fenotypem k rozdílu mezi instrukcemi k vytvoření herního levelu a samotným levelem. [9] jako příklad udává [36], která se zaobírala generováním herních levelů ve hře *StarCraft*. [36] jako genotyp levelů brala pole reálných čísel a fenotypem byl pak samotný level se vším, co se v něm očekávalo. Příklad vygenerovaných map můžeme vidět na obrázku 1.22

[5] zmiňuje jako jeden z dobře prostudovaných způsobu reprezentaci prostřednictvím vektoru reálných čísel. Jako výhody tohoto přístupu je zmíněna dobrá analýza takových dat a fakt, že hodně standardních algoritmů je připraveno pracovat s takovou formou dat. Při návrhu takového vektoru je potřeba brát v potaz jeho dimenzionalitu. Vektor skládající se z malého množství čísel nebude schopen dostatečně dobře reprezentovat generovaný obsah, velké vektory se naopak pojí s jevem zvaným *Prokletí dimenzionality*[32]. Další vlastnost, kterou u reprezentace obsahu sledujeme, je jeho lokalita. Od reprezentace požadujeme vysokou lokalitu, což znamená, že malá změna v genotypu vyústí v malou změnu fenotypu a hodnoty fitness funkce. [5]

Jakmile je obsah vygenerován, musí pro něj být spočtena hodnota fitness funkce. Při návrhu fitness funkce je potřeba si určit, jaká vlastnost obsahu by měla být optimalizována a jak ji formálně vyjádřit. Designér se může snažit navrhnout algoritmus tvořící obsah, který bude pro hráče zábavný nebo imerzivní. Emociální stavy člověka se neformalizují zrovna lehce, a navíc by spousta hráčů mohlo jako zábavný obsah brát něco jiného. Lepší je proto měřit vlastnosti obsahu, které budou pro danou hru specifické. [5]

[5] dále dělí fitness funkce do 3 skupin:

[32] Prokletí dimenzionality je jev, který nastává při analýze vícedimenzionálních dat. Výpočetní nároky na zpracování dat rostou exponenciálně s počtem jejich dimenzí.

- **Přímé hodnotící funkce** – hodnota fitness funkce je odvozena od vlastností generovaného obsahu.
- **Hodnotící funkce založené na simulaci** – je vytvořen umělý agent, který má za úkol hrát část hry, která souvisí s generovaným obsahem. Hodnota fitness funkce je následně odvozena od vlastností herního stylu pozorovaného agenta.
- **Interaktivní hodnotící funkce** – reálný hráč hraje hru. Hodnota fitness funkce je určena během vlastního hraní hry. Potřebná data mohou být od hráče získána explicitně použitím dotazníků nebo implicitně prostřednictvím různých měření.

[5] dodává, že nelze přesně rozhodnout, zda jakýkoliv metaheuristický algoritmus[33] skončí v rozumném čase a vyprodukuje řešení v dostačující kvalitě. Čas, který takovému algoritmu trvá nalézt řešení, je nejvíce ovlivněn fitness funkcí. Podle provedeného průzkumu trvá generování kvalitního obsahu v řádu sekund až dní, což by podle [5] z search-based procedurálního generování obsahu dělalo metodu vhodnou spíše pro offline generování.

Vzory, které se běžně vyskytují v herních dungeonech, identifikované v [2], by mohly být podle [2] využity pro tzv. generování dungeonů založené na vzorech. [37] zakládá výpočet fitness funkce na kvalitě některých mikro-vzorů identifikovaných právě v [2]. Dalším zajímavým konceptem zmíněným ve [37] je ukončení algoritmu generování po pevně stanoveném počtu generací, spíše než po dosažení požadované hodnoty fitness funkce. Důvodem je dosažení konzistentní doby odezvy aplikace napříč různými uživatelskými vstupy.

[6] přichází s pojmem experience-driven PCG, ve kterém celému procesu generování předchází modelování hráčovy zkušenosti. Model hráčovy zkušenosti předpovídá, jaký typ zkušenosti by hráč mohl v určité herní situaci zažívat. Těmito modely je následně ovlivněno hodnocení kvality generovaného herního obsahu.

Answer set programming

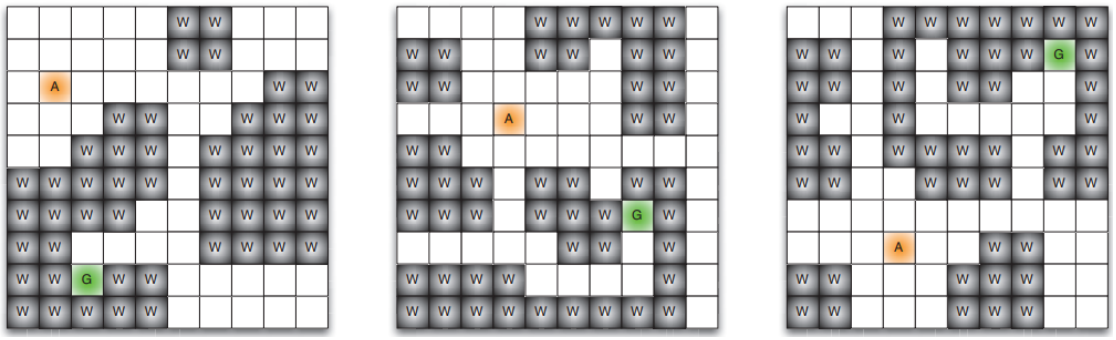
Answer set programming[34] (dále jen ASP) je využíváno v [9] jako metoda procedurálního generování obsahu založená na logickém programování. Generování obsahu probíhá tak, že nejdříve pomocí ASP definujeme, jaký obsah se má generovat, a následně předhodíme náš program nástroji jménem ASP solver, jehož výstupem je požadovaný obsah. Tak jako jsme museli v případě evolučních algoritmů navrhnout způsob reprezentace obsahu a fitness funkci, musíme v případě ASP definovat logiku domény generovaného obsahu a omezení, která na generovaný obsah klademe.

Logika domény obsahu zahrnuje herní mechaniky a strukturu obsahu. Není potřeba popisovat, jak funguje celá hra, stačí popsat herní mechaniky související s generovaným obsahem. Pokud chceme vygenerovat dungeon, který se skládá z 2D mřížky, musíme specifikovat, že se na ní mohou nacházet zdi, že se po ní pohybuje hráč, že hráč na nějaké pozici začíná, že se hráč může pohybovat pouze na prázdná políčka atd. [9]

Jakmile máme definovanou logiku obsahové domény, musíme popsat, jaké vlastnosti musí mít každý vygenerovaný obsah s ohledem na herní logiku. Jestliže se v každém levelu musí nacházet cesta od začátku do konce, musíme ji definovat. Musíme definovat například, že takováto cesta musí mít nějakou minimální délku nebo že začátek dungeonu musí být na opačné straně mřížky než jeho konec. [9] zmiňuje že úpravy těchto omezení se často dějí iterativně. Po několika vygenerovaných instancích obsahu si můžeme všimnout, že nám v nich něco schází a tak omezení ještě dále specifikujeme. [9]

Logika a omezení vlastností jsou následně předány nástroji ASP solver, který řeší logický problém. Logickým problémem je v tomto případě myšleno vygenerování takového obsahu, který je v souladu s logikou hry a jeho vlastnosti splňují na ně kladená omezení. [9]

[33] Metaheuristický algoritmus se snaží najít dostatečně dobré řešení optimalizačního problému.
[34] Některé zdroje překládají answer set programming jako programování množinou dotazů.



Obrázek 1.23 Příklad dungeonu generovaného technikou ASP. Šedou barvou jsou znázorněny zdi, bílou barvou volné pozice, zelenou drahokam a oranžovou oltář. [9]

Pro ASP využívá [9] jazyk AnsProlog. Nejjednodušším konstruktem v ASP je fakt. Fakt označuje nějaké tvrzení, které prohlašujeme za pravdivé. [9] Zápis faktu v jazyce AnsProlog můžeme vidět na výpisu kódu 1.2.

Výpis kódu 1.2 Zápis faktu v jazyce AnsProlog

```
game_over.
gravity_enabled.
```

Dalším konstruktem popisovaným v [9] je predikát, což je pravdivé tvrzení, které má parametry. Zápis predikátu můžeme vidět na výpisu kódu 1.3.

Výpis kódu 1.3 Zápis predikátu v jazyce AnsProlog

```
max_jump(3).
contains((2,2),wall).
```

Na definování herních mechanik nám AnsProlog umožňuje definovat pravidla. Zápis pravidla můžeme vidět na výpisu kódu 1.4.

Výpis kódu 1.4 Zápis pravidla v jazyce AnsProlog

```
impassable(Tile) :- contains(Tile,wall).
```

Pravidlo zapsané ve výpisu kódu 1.4 by se značením predikátové logiky dalo vyjádřit následovně:

$$(\forall Tile)(impassable(Tile) \Longleftarrow contains(Tile, wall)).$$

Dalším užitečným pravidlem v AnsPrologu jsou výběrová pravidla. Jestliže chceme například stanovit, že počet zdí v mřížce musí být mezi 5 a 10, můžeme to pomocí výběrových pravidel napsat jako ve výpisu kódu 1.5.

Výpis kódu 1.5 Zápis výběrového pravidla v jazyce AnsProlog

```
5 { contains(T,wall) : tile(T) } 10.
```

Cílem [9] bylo s využitím techniky ASP vytvořit dungeon tvořený 2D mřížkou, do kterého by hráč v levém horním rohu vstoupil, uvnitř dungeonu našel drahokam obklopený zdmi, donesl ho k oltáři a z dungeonu odešel východem v pravém dolním rohu. Po několika iteracích spočívajících v úpravách podmínek, které omezovali vlastnosti generovaného obsahu, došel k dungeonům, které si můžeme prohlednout na obrázku 1.23.

Kapitola 2
Návrh

V této kapitole je nejdříve v sekci 2.1 na základě provedené analýzy vybrána vhodná metoda generování dungeonů pro hru *Magitech*. Sekce 2.2 popisuje proces generování a třídy, které pro tento proces byly navrženy. Nakonec je v sekci 2.3 okomentován návrh uživatelského rozhraní.

2.1 Výběr metody generování

Na základě stanovených požadavků je nyní potřeba z analyzovaných metod vybrat takovou, která požadavky splní nejlépe. V následujících podkapitolách je každá metoda zmíněná v sekci 1.5 okomentována a je u ní rozhodnuto, zda je vhodná pro použití v případě hry *Magitech* či nikoliv.

2.1.1 Dělení prostoru

Rozdělování prostoru není metodou použitelnou v případě hry *Magitech*. Jak již bylo zmíněno, dlaždice používané ve hře *Magitech* mají fixní velikost. Rozdělování prostoru dělí prostor na části o různé velikosti. Místnosti v různě velkých buňkách jsou následně spojeny cestami, což v případě hry *Magitech* není možné, protože cesty jsou již součástí samotných dlaždic.

Zajímavá je ovšem myšlenka rozdělit prostor dungeonu na několik stejně velkých čtvercových podprostorů a ty zaplňovat dlaždicemi z tilesetů. Na toto ovšem algoritmus rozdělování prostoru není potřeba a stačí nám obyčejná 2D mřížka.

2.1.2 Celulární automat

Celulární automat by mohl být ve hře *Magitech* tvořen 2D mřížkou se čtvercovými buňkami. Nad touto mřížkou by automat iteroval a upravoval ji podle stanovených pravidel. Pravidla popsaná v rámci kapitoly 1.5 by byla v našem případě značně nevyhovující.

Popsaná pravidla spoléhají na mřížky o velkých rozměrech, do kterých jsou schopna vytvořit rozsáhlé struktury připomínající jeskyně. Při současných velikostech dlaždic je jejich vhodný počet v jednom dungeonu přibližně 30. Při větším počtu už hráč tráví průchodem dungeonu příliš mnoho času, což by nemuselo být bráno jako zábavné.

Již zmíněná pravidla navíc nezaručují, že bude vygenerována splnitelná herní úroveň – není zaručena existence cesty od jedné buňky ke všem ostatním. Přidávání dalších dlaždic s účelem vytvořit mezi všemi ostatními dlaždicemi cesty by navíc nebylo kompatibilní s požadavkem na přítomnost specifického počtu dlaždic v dungeonu. Korigování automatu s tímto požadavkem by znamenalo výrazné zvýšení složitosti procesu generování.

Máme však možnost si nadefinovat svá vlastní pravidla. Taková pravidla by musela obsáhnout požadavky na generování dungeonů o stanovené velikosti, dungeony vygenerované pomocí těchto pravidel by musely být splnitelné, muselo by v nich být také obsaženo generování do různých stran. Taková pravidla by byla mnohem složitější než pravidla, která byla popsána v kapitole 1.5.

Metodu celulárních automatů ve fázi návrhu označuji za použitelnou, ale kvůli množství složitých požadavků, které by pravidla musela uspokojit, ji označuji za spíše nevhodnou pro generování dungeonů ve hře *Magitech*.

2.1.3 Využití agentů

Agenti zmíněni v kapitole 1.5 s sebou nesli spoustu nevýhod. Byli používáni na kopání chodeb a umísťování místností, což je opět neaplikovatelné v našem případě, protože se dungeony skládají z dlaždic, na kterých se cesty už nacházejí. Jejich pravidla pohybu způsobovala vytváření nevyhovujících dungeonů.

Těmto nevýhodám by se ale dalo vyhnout vytvořením složitějších pravidel pohybu. Agent by v našem případě nekopal chodby, ale spíše zaplňoval pozice, na kterých by se následně umístily předem vytvořené dlaždice.

Implementací vhodné logiky agenta by se daly uspokojit požadavky kladené na výsledný dungeon. Agent by mohl vložit jenom určitý počet dlaždic, mohl by dungeon generovat určitým směrem, agentovým generováním by z dlaždic vznikaly cesty, v nichž by byly všechny dlaždice propojeny. Agent by mohl při průchodu mřížkou dungeonu pokládat například nutné či úkolové dlaždice.

Metoda využití agentů vypadá velice slibně, je schopná uspokojit požadavky na výsledný vygenerovaný dungeon a proto ji označuji za velice vhodnou pro využití ve hře *Magitech*.

2.1.4 Gramatiky

Použití generativních gramatik je opravdu výhodné v případě, kdy generujeme herní lokace, které obsahují koncept zámku a klíče. Zámky ani klíče se ve hře *Magitech* ale nevyskytují. Využití struktur mise a prostoru vypadá slibně, ale graf mise by byl v případě hry *Magitech* příliš jednoduchý. Úkoly jsou ve hře přímočaré a v současné chvíli by uzly obsahovaly pouze ničení nepřátel a sbírání materiálů. Využití tvarových gramatik ve hře nepřipadá v úvahu kvůli předpřipraveným dlaždicím.

Využívání gramatik je metoda skýtající spoustu výhod, v případě hry *Magitech* je ale nejsme schopni využít. Pro některé požadavky jako je například generování do určitých směrů by se pravidla generativních gramatik vymýšlela pouze obtížně. Gramatiky tedy hodnotím jako metodu použitelnou ve hře *Magitech*, ale spíše nevhodnou.

2.1.5 Markovovy modely

Nízká úspěšnost generování hratelných levelů, neschopnost zachytit závislosti na větší vzdálenost a nutnost vytvářet trénovací mapy dělá z Markovových modelů metodu nepříliš vhodnou pro využití ve hře *Magitech*.

2.1.6 Pokročilé metody generování plošin

Metody zmíněné v kapitole 1.5 nejsou použitelné pro generování dungeonů takového typu, jaký je použit ve hře *Magitech*.

2.1.7 Evoluční algoritmy

Herní prostory se ve hře *Magitech* musí generovat za běhu hry, jedná se tedy podle [5] o online generování. Jak už zaznělo v kapitole 1.5, evoluční algoritmy nejsou vhodné pro online generování z důvodu časové náročnosti procesu generování.

2.1.8 Answer set programming

Pomocí ASP byly generovány dungeony velice podobné těm, které se mají generovat ve hře *Magitech*. Prostředí *Unity*, ve kterém má být generátor implementován, bohužel ale nepodporuje možnost programování množinou dotazů.

Jednou z možností řešení tohoto problému by bylo volání programu v jazyce AnsProlog v moment, kdy by hráč vstoupil do dungeon scény. Výstup tohoto programu by musel být předán do prostředí *Unity*, musel by být interpretován a následně by na jeho základě byl výsledný dungeon vygenerován. Další možností je napsání rozšíření prostředí *Unity*, které by bylo schopné s jazykem AnsProlog pracovat. Pracnost obou těchto přístupů převažuje výhody této metody generování, a proto tento přístup označuji za nevhodný pro použití ve hře *Magitech*.

2.1.9 Zhodnocení

Předchozí kapitoly rozdělily metody generování dungeonů na nepoužitelné, nevhodné a vhodné. Rozdělování prostoru, evoluční algoritmy a pokročilé metody generování plošin se ukázaly jako nepoužitelné v případě hry *Magitech*. Metody využívající celulární automaty, gramatiky, ASP či Markovovy modely by mohly být použitelné, ale označuji je na základě jejich zhodnocení v předchozích kapitolách za nevhodné. Využití agentů vypadá ve fázi návrhu jako metoda vhodná pro generování dungeonů ve hře *Magitech*, schopná splnit všechny požadavky na ně kladené.

2.2 Popis architekury

Následující kapitola se věnuje popisu tříd a datových struktur, které budou ve hře *Magitech* použity. V rámci kapitoly bude také vysvětlen průběh generování od hráčova vstupu do dungeon scény až po jeho umístění na startovací dlaždici. Součástí kapitoly je i krátký komentář k návrhu uživatelského rozhraní.

2.2.1 Třídy a datové struktury

Tato kapitola se věnuje popisu tříd a datových struktur, které budou v procesu generování dungeonů používány. Některé zmíněné třídy[1] již byly součástí hry *Magitech*, ovšem aby mohly být splněny všechny požadavky, budou muset být pozměněny.

Tileset

Tato část se věnuje popisu tilesetů, které jsou používány pro generování dungeonů ve hře *Magitech*. Součástí požadavků na modul generátoru bylo totiž využití systému tilesetů, který byl do hry již zakomponován.

Tileset je množina dlaždic, které se skládají vedle sebe a společně tvoří dungeon. Ve hře se v současnou chvíli nachází 3 tilesety, které jsou graficky odlišeny tak, aby připomínaly les, jeskyni a město. Dlaždice ve všech tilesetech jsou tvaru čtverce o velikosti 16x16 jednotek. Generátor by měl podporovat i dlaždice jiných rozměrů, je však zaručeno, že dlaždice budou vždy čtvercové.

[1] Například třídy *DungeonController* a *DungeonSceneGenerator* – předchůdce třídy *AgentDungeonSceneGenerator* – se ve hře již nacházely.

Obrázek 2.1 5 tvarů dlaždic ve hře *Magitech* – zleva DeadEnd, Bend, Line, Branch a Crossroad

Existuje 6 různých druhů dlaždic v tilesetu, které se od sebe liší jejich účelem v dungeonu:

- **Start** – Start dlaždice slouží jako první dlaždice v dungeonu. Hráč se na nich poprvé v dungeonu objeví a je na nich přítomen objekt, který je schopen hráče přenést z dungeonu zpět do vesnice.
- **Boss** – Boss dlaždice jsou dlaždice uzpůsobené k vytváření bossů. Dává smysl, aby se nacházely co nejdál od startovací dlaždice, aby boss představoval v dungeonu finální výzvu.
- **Portal** – Portal dlaždice slouží k přesunu hráče do jiného dungeonu. Tyto dlaždice obsahují objekt, který je schopen přesunout hráče do sousedního dungeonu. Portal dlaždice musí být umístěny na okrajích dungeonu. Dává také smysl je umísťovat daleko od startovací dlaždice, aby hráč nejdříve splnil všechny výzvy, které na něho v dungeonu čekají, než se přesune do dalšího dungeonu.
- **Special** – Special dlaždice je množina dlaždic, které oproti ostatním nějak vynikají. Mohou to být dlaždice, které na sobě obsahují zdroje léčení pro hráče, nebo to mohou být dlaždice, na nichž se nachází kill squad.
- **Border** – Border dlaždice ohraničují celý dungeon. Jsou celé pokryty překážkami, které zabraňují hráči vyjít z dungeonu. Překážky na border dlaždicích slouží také k tomu, aby hráč neviděl ven z dungeonu.
- **Normal** – Posledním druhem dlaždic jsou Normal dlaždice, které nejsou nijak výjimečné. Generátor je bude pokládat na místech, která nejsou vyhrazena pro předchozí druhy dlaždic.

Všechny druhy dlaždic budou schopny na sobě umístit nepřátele i materiály.

Dlaždice v tilesetech na sobě obsahují graficky vyznačené cesty. Cesty je možné na dlaždicích uspořádat do 5 tvarů, které můžeme vidět na obrázku 2.1. Dlaždice musí být na místech, kde z nich nevedou cesty, ohraničeny Border dlaždicemi.

DungeonConfig

Pro nastavení všech parametrů požadovaného dungeonu bude sloužit třída *DungeonConfig*. Diagram tříd třídy *DungeonConfig* můžeme vidět na obrázku 2.2.

Designér bude moct ve třídě *DungeonConfig* nastavovat následující parametry:

- **BorderRadius** – do jaké vzdálenosti od vygenerovaného dungeonu mají být umístěny Border dlaždice
- **BorderTilePrefab** – model Border dlaždice
- **Difficulty** – obtížnost dungeonu – Easy, Medium nebo Hard
- **DungeonSize** – velikost dungeonu – Small, Medium nebo Large
- **MaterialCounts** – materiály, které se v dungeonu musí objevit, včetně jejich počtu.

- **NecessaryTiles** – dlaždice, které se musí v dungeonu objevit.
- **NeighbouringDungeons** – dungeony, se kterými bude současně tvořený dungeon sousedit. Bude možné u nich nastavit, na které straně dungeonu se mají objevit.
- **PossiblySpawnedTiles** – dlaždice, které se v dungeonu mohou objevit. Součástí těchto dlaždice bude i pravděpodobnost, že se v dungeonu objeví a počet dlaždic. Počet dlaždic značí kolikrát se bude rozhodovat o tom, zda se dlaždice v dungeonu objeví.
- **QuestTiles** – dlaždice, které slouží pro plnění úkolu v dungeonu. Součástí těchto dlaždic bude i strana dungeonu, na které se mají úkolové dlaždice objevit.
- **Tileset** – tileset, který má být použit pro generování.

Ve třídě *DungeonConfig* bude možné do seznamu *NeighbouringDungeons* vložit až 4 sousední instance třídy *DungeonConfig*, které budou také moct mít svoje sousedy. Součástí dungeon scény tak bude mřížka, ve které budou jednotlivé dungeony ve scéně umístěny. Pojem dungeon tak bude označovat právě jeden prvek této mřížky, dungeon scéna bude odkazovat na všechny dungeony ve scéně. V případě, že bude součástí dungeon scény více dungeonů, ten, který byl vygenerován na základě instance třídy *DungeonConfig* předané ve třídě *Quest*, bude označen jako počáteční a na jeho Start dlaždici bude hráč v dungeon scéně umístěn.

Quest

Třída *Quest* byla již součástí hry a reprezentuje úkol, který hráč ve hře musí plnit. Součástí třídy *Quest* bude proměnná typu *DungeonConfig*, do které se bude přiřazovat předpřipravená instance této třídy. Instance třídy *Quest* jsou ve hře uloženy ve třídě *QuestManager*. *QuestManager* nám dokáže vrátit objekt typu *Quest* reprezentující hráčův aktivní úkol[2]. Parametr *DungeonConfig* tohoto úkolu bude použit pro generování dungeonu.

Třída obsahuje proměnnou typu *QuestType*. *QuestType* je výčtový typ, který nám říká, co musí hráč v dungeonu udělat, aby úkol splnil. *QuestType* bude rozšířen o hodnotu Endless, která bude reprezentovat úkol s nekonečným dungeonem.

DungeonController

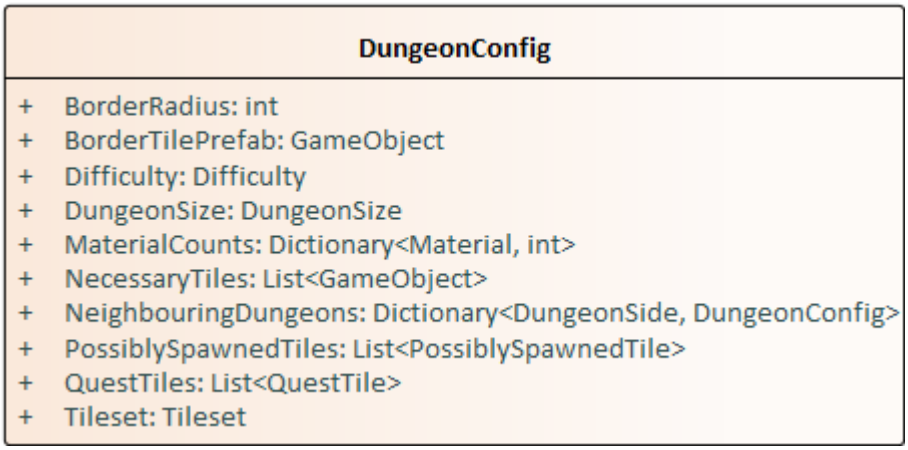
DungeonController je třída, která bude součástí dungeon scény a bude zodpovědná za její inicializaci a správu. Na obrázku 2.3 můžeme vidět třídu *DungeonController* společně s jejím rodičem – třídou *SceneController*.

Třída *SceneController* slouží pro inicializaci scén ve hře a dědí z ní třídy spravující ostatní scény, například *MainHubController* nebo *PubController*. Jak můžeme na diagramu vidět, obsahuje metodu *SpawnPlayer*, která umístí hráče do scény na specifikovanou lokaci. Po umístění je hráč rotován na základě proměnné typu *Quaternion*. *SceneController* také obsahuje metodu *ChangeScene* sloužící pro změnu scény na základě jména nové scény.

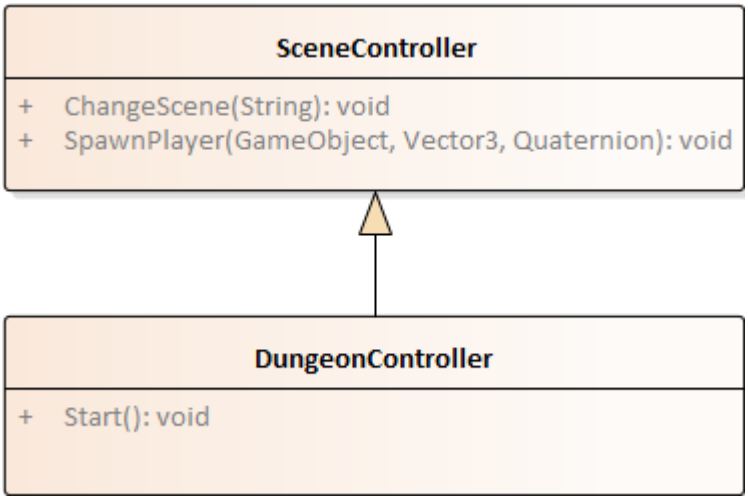
Třída *DungeonController* obsahuje metodu *Start*. Tato metoda je zavolána prostředím *Unity* vždy, když dojde k přechodu do nové scény. Metoda se bude starat o inicializaci dungeon scény a prostřednictvím třídy *AgentDungeonSceneGenerator* vygeneruje nový dungeon. Třídě *AgentDungeonSceneGenerator* bude předán objekt typu *DungeonConfig* uložený v instanci třídy *Quest*, která reprezentuje aktivní hráčův úkol.

Zodpovědností třídy *DungeonController* je i dokončení samotného procesu generování. Na konci procesu je potřeba na počáteční dlaždici počátečního dungeonu umístit hráče, spustit

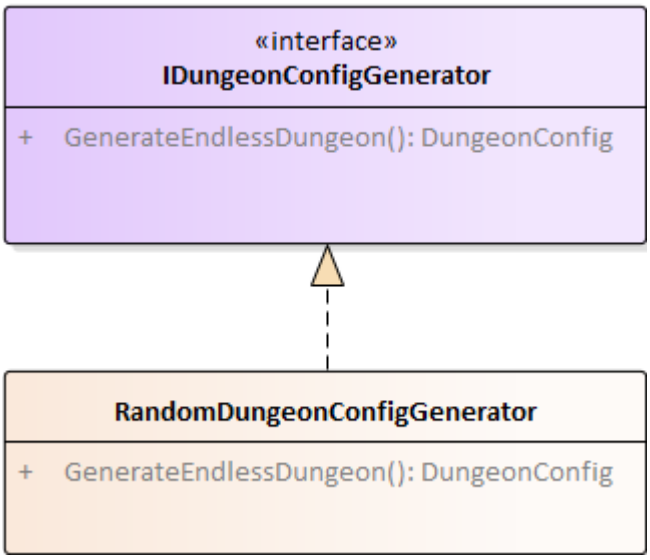
[2]Hráč má možnost jako aktivní zvolit jakýkoliv úkol, který se nachází v jeho seznamu úkolů. Při vstupu hráče do dungeon scény budou dungeony ve scéně vygenerovány právě na základě aktivního úkolu. Není-li žádný úkol zvolen jako aktivní, hráči se nepodaří přesunout do dungeon scény.



Obrázek 2.2 Diagram tříd – *DungeonConfig*



Obrázek 2.3 Diagram tříd – *DungeonController*



Obrázek 2.4 Diagram tříd – *RandomDungeonConfigGenerator*

správnou hudbu asociovanou s tilesetem dungeonu a nastavit správné světelné podmínky ve scéně.

Podle požadavku F8 bude nutné generovat novou náhodnou instanci třídy *DungeonConfig* v případě, že je potřeba generovat nekonečný dungeon. Generování těchto instancí bude obstarávat třída *RandomDungeonConfigGenerator*. Tato třída pro *DungeonController* vygeneruje nový náhodný objekt typu *DungeonConfig* také pokaždé, když hráč v nekonečném dungeonu projde portálem.

RandomDungeonConfigGenerator

Třída *RandomDungeonConfigGenerator* bude zodpovědná za generování náhodných instancí třídy *DungeonConfig*. Její diagram tříd můžeme vidět na obrázku 2.4.

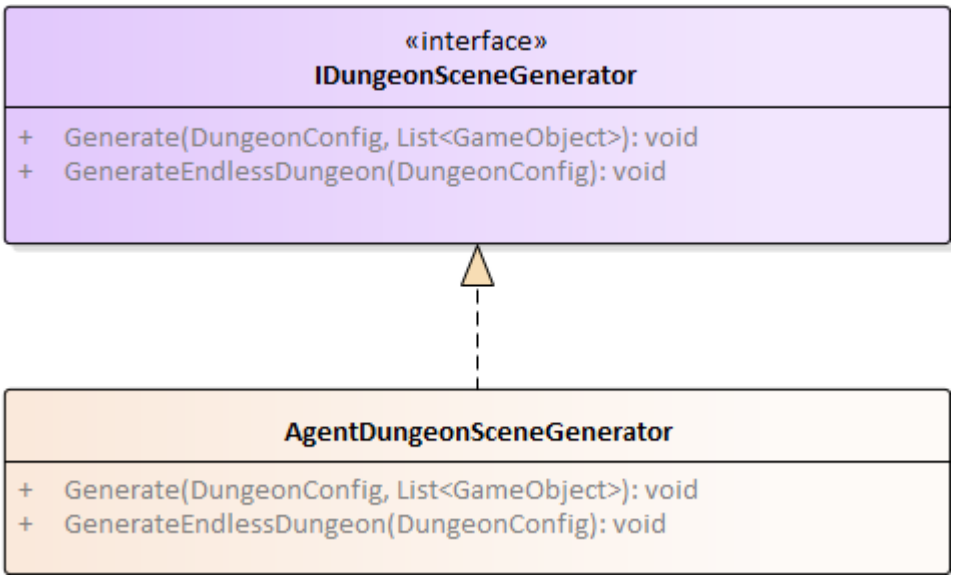
Třída *RandomDungeonConfigGenerator* implementuje rozhraní *IDungeonConfigGenerator*. Pokud bychom do budoucna chtěli generovat nové instance třídy *DungeonConfig*, jejíž parametry nebudou tak náhodné, stačí pouze implementovat stejné rozhraní.

AgentDungeonSceneGenerator

Třída *AgentDungeonSceneGenerator* je zodpovědná za vygenerování dungeonů do dungeon scény na základě instance třídy *DungeonConfig*, která jí byla předána třídou *DungeonController*. Třída vygeneruje dungeon definovaný předanou instancí třídy *DungeonConfig* včetně všech sousedících dungeonů do mřížky. Následně je umístí do dungeon scény a vytvoří v nich nepřátele a materiály. Její diagram tříd můžeme vidět na obrázku 2.5.

Na diagramu můžeme vidět, že třída implementuje rozhraní *IDungeonSceneGenerator*. Při návrhu bylo myšleno na to, že by se v budoucnu mohly dungeony generovat i jiným způsobem než pomocí agentů. Například každý tileset by mohl být generován jiným způsobem, což by mohlo podpořit dojem jedinečnosti jednotlivých tilesetů.

Kromě proměnné typu *DungeonConfig* je metodě *Generate* předán i seznam kill squad dlaždic, které se v dungeonu musí objevit. Metodě *GenerateEndlessDungeon* na základě požadavku F16 seznam kill squad dlaždic předán nebude.



Obrázek 2.5 Diagram tříd – *DungeonSceneGenerator*

RandomAgent

Třída *RandomAgent* bude sloužit k pokládání dlaždic do dungeonu. Agent se bude pohybovat po 2D mřížce reprezentující rozložení dlaždic v dungeonu. Na vhodné pozice v mřížce bude agent pokládat dlaždice. Agentův pohyb po mřížce bude ovlivněn náhodou. Diagram třídy *RandomAgent* můžeme vidět na obrázku 2.6.

RandomAgent bude implementovat rozhraní *IAgent*. Rozhraní bylo tvořeno s myšlenkou, že by v budoucnu mohla vzniknout jiná implementace agenta, která by byla méně náhodná ve svých pohybech.

Agent musí implementovat 2 metody. V metodě *FillLayout* agent mřížku vyplní určitým počtem dlaždic typu Normal. V metodě *PlaceTiles* agent do mřížky vloží specifické dlaždice[3].

UniformDungeonEnemySpawner

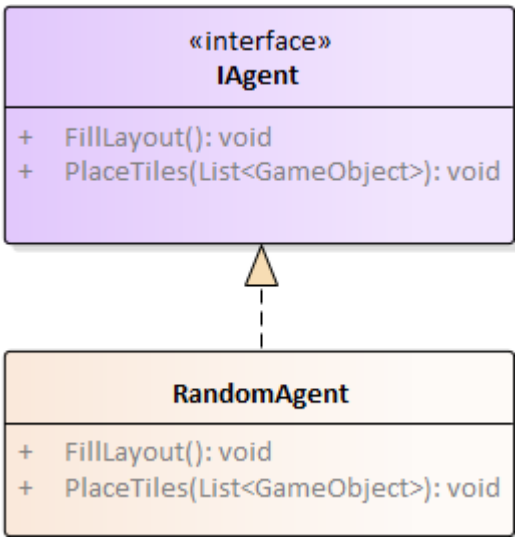
Třída *UniformDungeonEnemySpawner* bude zodpovědná za vkládání nepřátelských skupin do dungeonu. *UniformDungeonEnemySpawner* pro každý dungeon v dungeon scéně označí dlaždice, na kterých budou nepřátelské skupiny umístěny. Nepřátelské skupiny budou v dungeonu rozmístěny rovnoměrně. Na označených dlaždicích poté nechá *UniformDungeonEnemySpawner* nepřátele vytvořit. Diagram třídy můžeme vidět na obrázku 2.7.

Třída *UniformDungeonEnemySpawner* implementuje rozhraní *IDungeonEnemySpawner*. Je tak kromě umísťování nepřátel povinna vracet celkový počet umístěných nepřátel. Rozhraní v budoucnu může implementovat jiná třída, která může například počet nepřátelských skupin v dungeonu postupně zvyšovat.

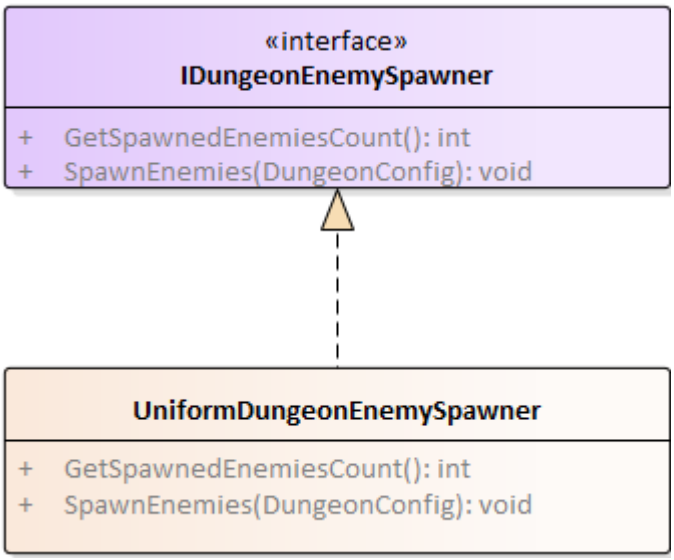
UniformDungeonMaterialSpawner

Třída *UniformDungeonMaterialSpawner* bude zodpovědná za umísťování materiálů na jednotlivé dlaždice. Ze seznamu *MaterialCounts* v třídě *DungeonConfig* reprezentující současně generovaný dungeon bude spočten celkový počet materiálů v generovaném dungeonu a tento počet bude

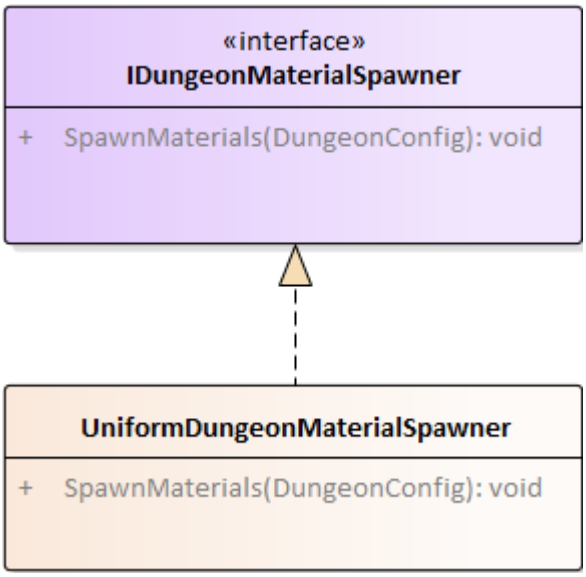
[3]Specifickými dlaždicemi jsou myšleny všechny dlaždice, které mají předem určený jejich model, tedy i jejich tvar. Jsou to převážně dlaždice, které se zadávají jako parametry třídy *DungeonConfig* – nutné, možné, úkolové, kill squad dlaždice.



Obrázek 2.6 Diagram tříd – *RandomAgent*



Obrázek 2.7 Diagram tříd – *UniformDungeonEnemySpawner*



Obrázek 2.8 Diagram tříd – *UniformDungeonMaterialSpawner*

rovnoměrně rozdělen mezi dlaždice. Následně budou ze seznamu *MaterialCounts* náhodně pro jednotlivé dlaždice vybrány konkrétní materiály, které se na nich objeví. Diagram tříd třídy *UniformDungeonEnemySpawner* je možné vidět na obrázku 2.8.

Třída *UniformDungeonMaterialSpawner* bude implementovat rozhraní *IDungeonMaterialSpawner*. Existence rozhraní dovoluje v budoucnu implementovat další třídu zodpovědnou za umísťování materiálů na dlaždice, která bude například postupně v dungeonu na dlaždicích zvyšovat počty materiálů.

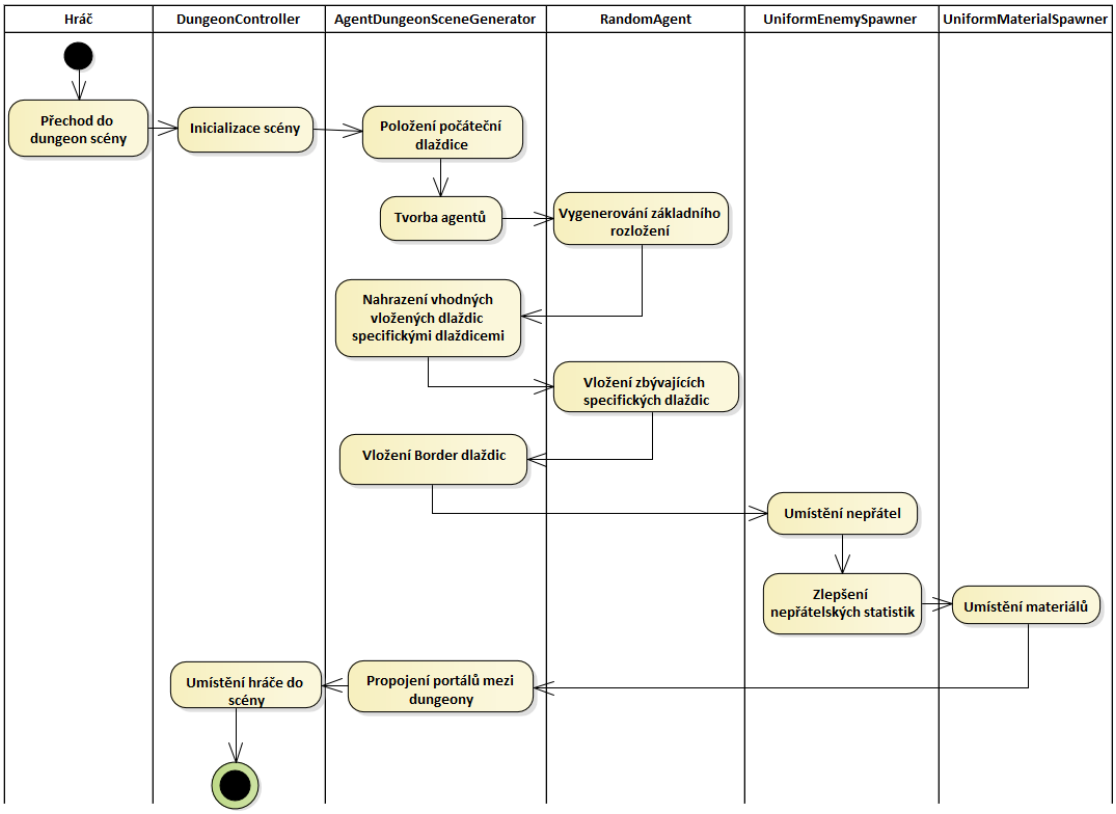
2.2.2 Postup při generování dungeonu

Posloupnost akcí při generování dungeonu je zobrazena na obrázku 2.9. Generování začne hráčovým příchodem do dungeon scény. Třída *DungeonController* se následně postará o inicializaci dungeon scény a předání parametru *DungeonConfig* hráčova aktivního úkolu třídě *AgentDungeonSceneGenerator*. Cesta instance třídy *DungeonConfig* z třídy *Quest* až do třídy *AgentDungeonSceneGenerator* je vyobrazena na obrázku 2.10.

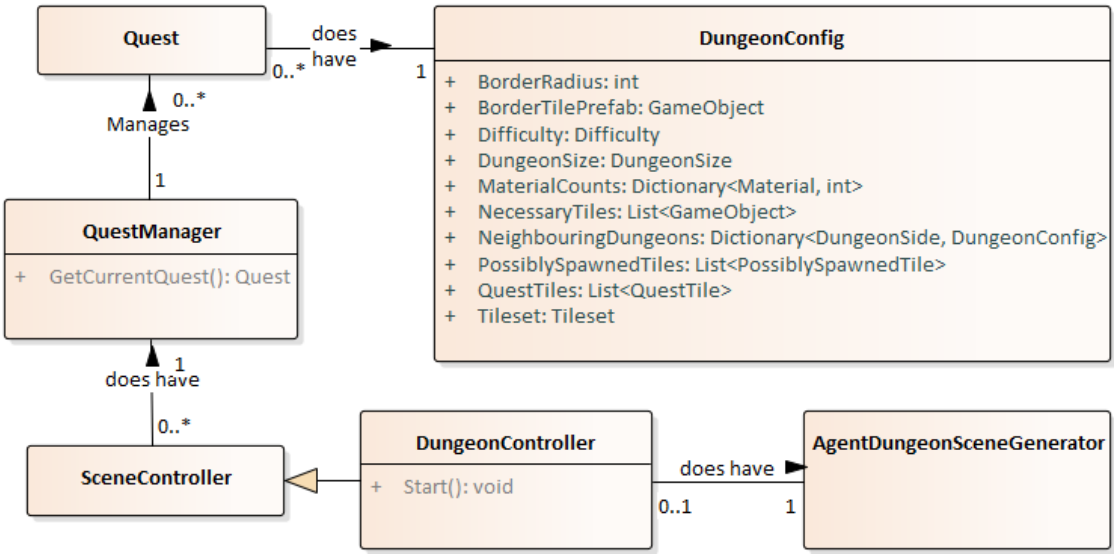
AgentDungeonSceneGenerator

V metodě *Generate* bude nejdříve sestavena 2D mřížka, do které budou vloženy všechny dungeony, které se ve scéně musí objevit. Dungeony budou v mřížce rozloženy na základě sousedností dungeonů definovaných v předané instanci třídy *DungeonConfig*. Každému dungeonu bude příslušet vlastní čtvercová 2D mřížka, do které budou vkládány jednotlivé dlaždice. Mřížky dlaždic jednotlivých dungeonů budou následně zaplněny.

Velikost mřížky dlaždic bude odvozena od parametrů instance třídy *DungeonConfig* pro jednotlivý dungeon tak, aby tvar výsledného dungeonu nebyl mřížkou omezován podle požadavku F3. Jednotlivé buňky mřížky symbolizují dlaždice, které se budou v dungeonu vyskytovat. U takových dlaždic si budeme pamatovat, jaký je jejich účel, případně jejich konkrétní model.



Obrázek 2.9 Activity diagram



Obrázek 2.10 Doménový model části systému

Mřížka začne jako prázdná a do jejího středu bude vložena dlaždice typu Start. V závislosti na tom, do kterých směrů bude potřeba dungeon generovat[4], se vytvoří 1–4 agenti. Mezi agenty bude rozdělen počet dlaždic dungeonu, který je stanoven parametrem *DungeonSize*.

Agenti

Agent je objekt, který bude procházet 2D mřížkou reprezentující rozložení dlaždic dungeonu. Když se agent přesune na prázdnou buňku, zaplní ji novou dlaždicí typu Normal. Přesune-li se agent na již obsazenou pozici, dlaždici nepokládá a pohybuje se dál. Jestliže se agent přesune na pozici, na které leží Border dlaždice, vrací se na svou předchozí pozici. Agentovi bude přiřazen směr, ve kterém se bude po mřížce pohybovat, a počet dlaždic, které svým procházením musí umístit. Směr přiřazený agentovi je dále nazýván také jako směr primární.

Agent se bude před každým svým krokem na základě pravděpodobností rozhodovat, kterým směrem se pohne. Každému směru pohybu[5] bude přiřazena nenulová pravděpodobnost. Stejně jako v [9] se tato pravděpodobnost bude upravovat po každém agentově kroku. Nejvyšší však vždy zůstane pravděpodobnost pohybu do agentova primárního směru.

Pro každého agenta bude vytvořena kontrolní oblast. Kontrolní oblast je obdélníkovitá část mřížky, uvnitř které musí agent ukončit svůj pohyb. Jestliže agent svůj pohyb uvnitř oblasti neukončí, bude restartován a začne generovat znovu. Kontrolní oblast bude umístěna v určité vzdálenosti od startovací dlaždice ve směru, který byl agentovi přiřazen. Kontrolní oblast bude tedy testovat, že se agent skutečně ve svém primárním směru pohyboval.[6]

Agent bude mít při generování 2 funkce – bude generovat základní rozložení dlaždic a vkládat specifické dlaždice.

Základní rozložení dlaždic bude agentem vytvářeno na počátku generování. Agent se po mřížce bude pohybovat tak dlouho, dokud nevyčerpá počet dlaždic, který mu byl přidělen. Na prázdné buňky mřížky agent vloží dlaždice typu Normal bez specifikovaného modelu.

AgentDungeonSceneGenerator se následně snaží nahradit co nejvíce dlaždic typu Normal specifickými dlaždicemi, aby případně agent vůbec nemusel specifické dlaždice vkládat.

Během vkládání specifických dlaždic se agent bude opět pohybovat po mřížce a na vhodná volná místa bude vkládat jemu předané specifické dlaždice.

Dokončení dungeonu

Po vložení všech specifických dlaždic budou následně okolo dungeonu ve specifikovaném počtu vloženy Border dlaždice. Dlaždice budou poté vloženy do dungeon scény na základě vyplněné 2D mřížky. Jestliže nebyl u některé dlaždice stanoven její konkrétní model, je pro ni náhodně vybrán na základě požadovaného tvaru[7] a účelu dlaždice.

Nepřátelské skupiny budou v dungeonu rovnoměrně rozmístěny na jeho dlaždicích. Počet nepřátelských skupin je ovlivněn parametrem *DungeonSize*, počet nepřátelských jednotek v jedné skupině se odvíjí od parametru *Difficulty*. Materiály budou taktéž rovnoměrně rozděleny mezi jednotlivé dlaždice a budou na nich umístěny. Portály v jednotlivých dungeonech budou náležitě propojeny mezi sebou na základě sousednosti jednotlivých dungeonů.

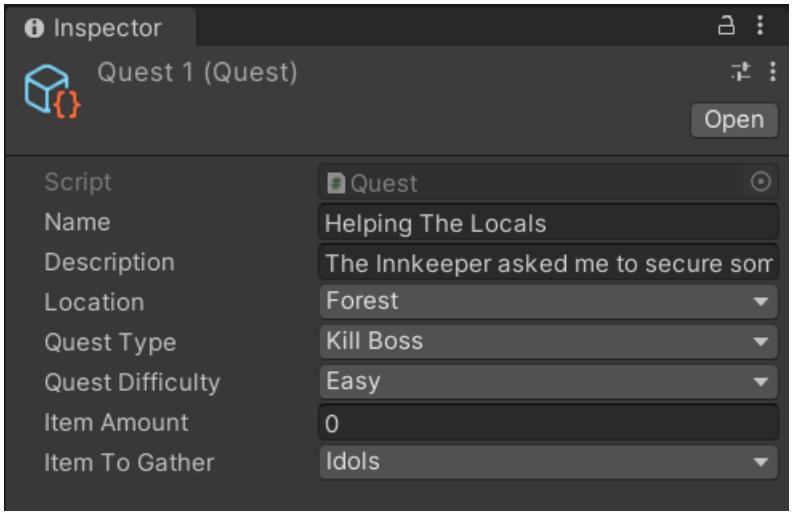
Ve třídě *DungeonController* bude následně generování dokončeno – bude spuštěna příslušná hudba a hráč bude umístěn na startovací pozici dungeonu.

[4] Dungeon bude generován do stran, které byly zadány v seznamech *QuestTiles* a *NeighbouringDungeons*

[5] Agent se bude moct po mřížce pohybovat směrem nahoru, dolů, doleva a doprava.

[6] Zařazení kontrolní oblasti dělá z metody využití agentů metodu PCG typu generuj-a-testuj. Toto je rozdíl oproti [9], kde součástí této metody žádné testování není a tato metoda je označena jako konstruktivní.

[7] Tvar je pro každou dlaždici určen na základě typů sousedních dlaždic. Jestliže dlaždice na určité straně sousedí s dlaždicí typu Border, nesmí z dlaždice tímto směrem vést cesta. Sousedí-li dlaždice s dlaždicí jakéhokoliv jiného typu, musí z ní do tohoto směru cesta vést.



Obrázek 2.11 Ukázka vyplnění třídy *Quest* v okně Inspector

2.3 Návrh uživatelského rozhraní

Pro modul generátoru není potřeba uživatelské rozhraní navrhovat. Designéři budou pro zadávání parametrů instancí třídy *DungeonConfig* používat uživatelské rozhraní prostředí *Unity*.

Aby byla tvorba nových instancí třídy *DungeonConfig* designérem co nejjednodušší a nejpřehlednější, bude pro třídu *DungeonConfig* zvolena reprezentace ve formě třídy odvozené od třídy *ScriptableObject*.

ScriptableObject je datový kontejner, který je v *Unity* používán k ukládání velkého množství neměnných dat. *Unity* umožňuje *ScriptableObject* vyplnit daty v okně Inspector, uložit ho mezi ostatní používané objekty a následně ho používat i na několika různých místech najednou.

Na obrázku 2.11 můžeme vidět, jak vypadá grafické rozhraní pro tvorbu nové instance třídy *Quest*, která také dědí od třídy *ScriptableObject*.

Designéři budou v každé instanci třídy *DungeonConfig* vyplňovat všechny parametry zmíněné dříve v sekci věnující se třídě *DungeonConfig*.

Kapitola 3
Implementace

Modul generátoru je implementován v herním enginu *Unity* stejně jako ostatní části hry *Magitech*. Specificky se jedná o verzi 2020.3.20f1. Programovacím jazykem používaným v herním enginu *Unity* je C#. Kromě mě přispíval do vývoje hry i můj kolega Štěpán Vejvoda, který pracoval na modulu pro interakci s nehráčskými postavami. Stejně jako já se rozhodl výsledky své práce prezentovat ve své bakalářské práci.

Tato kapitola obsahuje 12 sekcí, které se převážně věnují zajímavým problémům, na které jsem narazil ve fázi implementace. Některé sekce popisují funkcionality, které byly implementovány nad rámec specifikovaných požadavků.

3.1 Parametry dungeonů

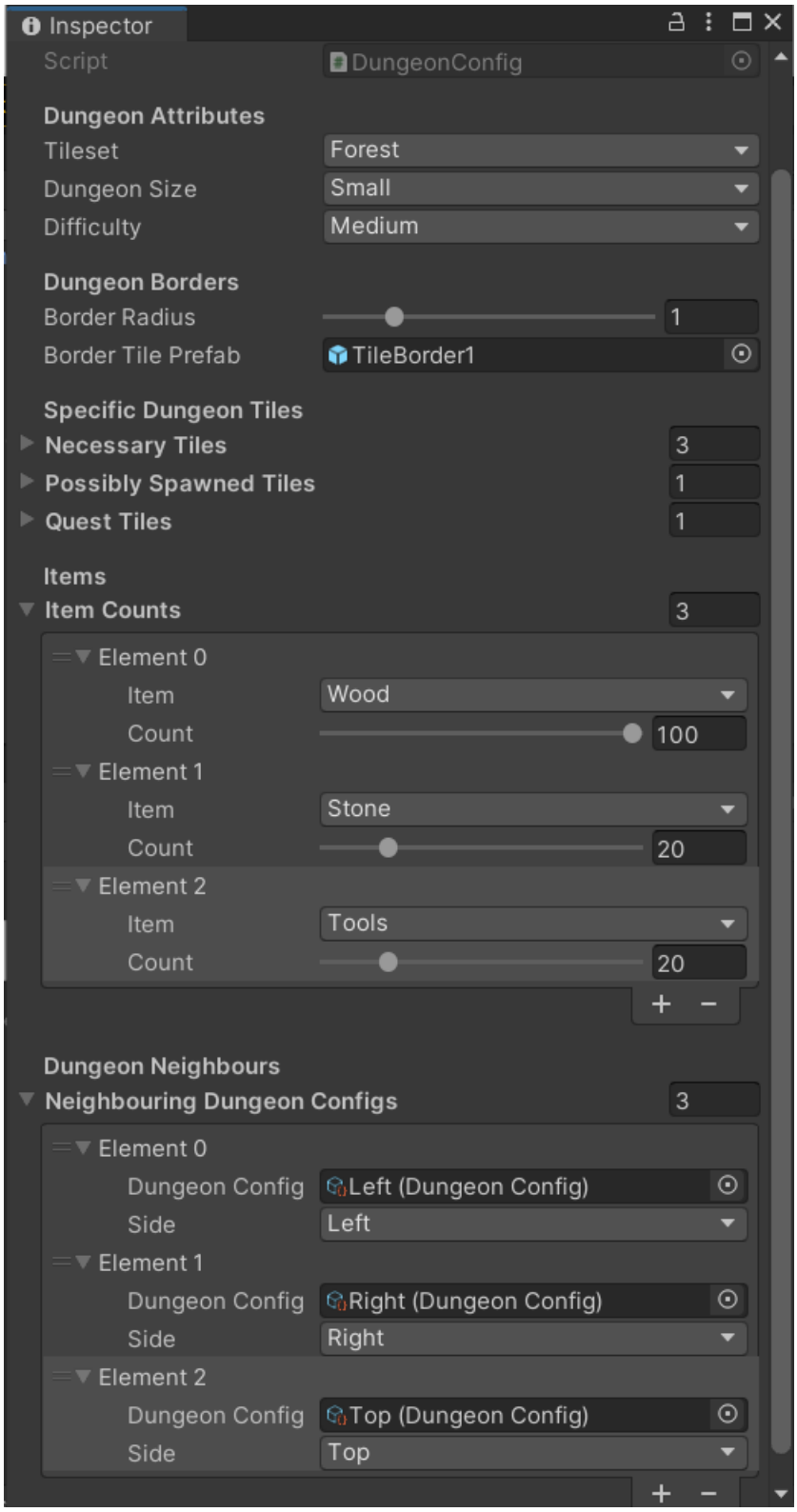
Při zadávání parametrů, které se skládají z více různých dat, nám *Unity* umožní tato data zadávat pospolu. Na obrázku 3.1 můžeme vidět například několik sousedních instancí třídy *DungeonConfig*, které byly přidány do seznamu *Neighbouring Dungeon Configs*. U sousedních dungeonů je možné v okně Inspector zadat i stranu dungeonu, na které se do těchto sousedních dungeonů mají objevit přechody. Podobným příkladem může být parametr *ItemCounts*, kde musí designér zadat jak druh předmětu, tak i počet instancí tohoto předmětu v příslušném dungeonu.

Taková data by bylo vhodné reprezentovat kolekcí klíčů a hodnot, v jazyce C# zvanou Dictionary. *Unity* však bohužel neumožňuje kolekci Dictionary serializovat, což mimo jiné znamená, že není možné její prvky zadávat v okně Inspector. Tento problém lze obejít vytvořením speciální třídy, která v sobě obsahuje oba druhy dat. Tuto třídu je potřeba označit anotací Serializable. Instance této třídy můžeme ukládat v kolekci zvané List a je možné je vytvářet a upravovat v okně Inspector. Příklad takové třídy ukazuje výpis kódu 3.1, její využití v třídě *DungeonConfig* je zobrazeno ve výpisu kódu 3.2.

Stejná technika je následně využita i v případě tříd, které by v kolekci Dictionary uložit nešly, například *PossibleTile*.

3.2 Validace dungeonů

Prostředí *Unity* umožňuje reagovat na změnu dat ve třídě *ScriptableObject* implementováním metody *OnValidate*. Toho je využito pro kontrolu zadaných parametrů. Je-li nesprávně zadaný parametr identifikován, je designér varován prostřednictvím textových výpisů v okně Console. Designér tak může být například varován, že zadal více než 4 sousední dungeony, zadal více sousedních dungeonů na stejnou stranu tvořeného dungeonu nebo nenastavil model Border dlaždice, přestože zvolil nenulovou velikost hranic dungeonu. Validační metoda je zobrazena na výpisu 3.3.



Obrázek 3.1 Okno Inspector při vytváření dungeonů

Výpis kódu 3.1 Definice párové třídy *ItemCountPair*

```
[Serializable]
public class ItemCountPair
{
    public ItemName Item;
    [Range(0, 100)]
    public int Count;

    public ItemCountPair(ItemName item, int count)
    {
        this.Item = item;
        this.Count = count;
    }
}
```

Výpis kódu 3.2 Členské proměnné třídy *DungeonConfig*

```
public class DungeonConfig : ScriptableObject
{
    [Header("Dungeon Attributes")]
    public Tileset Tileset;
    public DungeonSize DungeonSize;
    public Difficulty Difficulty;

    [Header("Dungeon Borders")]
    [Range(0, 5)]
    public int BorderRadius;
    public GameObject BorderTilePrefab;

    [Header("Specific Dungeon Tiles")]
    public List<GameObject> NecessaryTiles;
    public List<PossiblySpawnedTile> PossiblySpawnedTiles;
    public List<QuestTileRequiredSidePair> QuestTiles;

    [Header("Items")]
    public List<ItemCountPair> ItemCounts;

    [Header("Dungeon Neighbours")]
    public List<NeighbouringDungeonConfig> NeighbouringDungeonConfigs;

    (...)
}
```

Výpis kódu 3.3 Validace třídy *DungeonConfig*

```
public bool IsValid()
{
    return  !HasMultipleSameItems() &&
            !HasMultipleSameSideNeighbours() &&
            BorderPrefabSetCorrectly() &&
            AllPortalsLeadSomewhere() &&
            !DungeonCycleExists();
}
```

Výpis kódu 3.4 Kontrola nedokonalostí dat

```
private void WarnUser()
{
    CheckNecessaryTiles();
    CheckPossiblySpawnedTiles();
    CheckQuestTiles();
}
```

Designérovi však nic nebrání nevalidní dungeon úkolu přiřadit. Nevalidnost dungeonu je detekována před procesem generování a v takovém případě je pro generátor dungeonu vytvořen nový náhodný validní *DungeonConfig*, který bude pro generování použit.

Pro generování nových náhodných instancí třídy *DungeonConfig* byla rozšířena třída *RandomDungeonConfigGenerator*, která nově obsahuje metodu *GenerateDungeonConfig*. Některé parametry nově vytvořené instance třídy *DungeonConfig* jsou odvozeny od hráčova aktivního úkolu. Takto je například určena obtížnost dungeonu, použitý tileset nebo přítomnost potřebných úkolových dlaždic[1].

Designér je také varován, pokud je součástí zadaných dat nějaká nedokonalost, která nutně neznamená neschopnost generátoru podle dat vygenerovat dungeon[2]. Příkladem může být nulová zadaná pravděpodobnost výběru možné dlaždice nebo nezadání jejího konkrétního prefabu[3]. Designér v takových situacích nejspíš z nepozornosti nezadal správná data, a proto jej na tuto skutečnost *Unity* upozorní prostřednictvím varovné zprávy. Metodu varující designéra můžeme vidět na výpisu kódu 3.4.

3.3 Generování nekonečných dungeonů

Původní myšlenka generování nekonečných dungeonů spočívala v tom, že by v jedné scéně vždy byly umístěny 2 dungeony. V jednom z nich by se hráč zrovna nacházel a do druhého by se mohl okamžitě teleportovat z dungeonu prvního. V moment, kdy by se hráč přesunul mezi dungeony, by byl první dungeon zničen a místo něho by se začal generovat dungeon nový. Zatímco by hráč procházel druhým dungeonem, nový dungeon by se stačil vygenerovat a zamezilo by se tak čekání na jeho vygenerování.

[1] Pokud je cílem úkolu eliminovat bosse, je do seznamu úkolových dlaždic přidána dlaždice s bossem. Musí-li hráč v dungeonu posbírat určitý počet materiálů, jsou tyto materiály přidány do seznamu *ItemCounts* generované instance třídy *DungeonConfig*.

[2] Nedokonalý dungeon je brán jako validní a bude použit pro generování.

[3] Prefab je v prostředí *Unity* uložený herní objekt. Chová se jako šablona, ze které můžeme ve scéně vytvářet nové objekty. [38]

Podmínkou využití tohoto přístupu však bylo využití více vláken, které není v prostředí *Unity* příliš dobře podporováno. Pro implementaci byl použit Job systém prostředí *Unity*, Coroutine metody i samotná vlákna, od jejichž použití v *Unity* je silně odrazováno. Žádný z těchto přístupů však nevedl k funkčnímu výsledku.

Nekonečný dungeon se tedy začne generovat ve chvíli, kdy hráč vstoupí do portálu, což znamená, že musí hráč na vygenerování nového dungeonu počkat. Přenastavením systému pro generování NavMeshe[4] však došlo ke snížení času generování dungeonu, takže je generátor schopen nový dungeon vygenerovat v řádu nižších jednotek sekund.

3.4 Pohyb agenta

V algoritmu pohybu agenta se nachází několik pasáží, které by mohly být potenciálně problematické. Jedná se konkrétně o nekonečné cykly, musíme se ale také ujistit, aby agent nedošel na konec mřížky.[5]

3.4.1 Hranice mřížky

Mřížka, po které se agent pohybuje, je dostatečně velká, aby se agent nemohl dostat na její okraj. Při výpočtu její velikosti je počítáno s nejhoršími možnými výsledky generování základního rozložení i vkládání specifických dlaždic. Při výpočtu je počítáno i s tím, že jeden agent vygeneruje několik dlaždic v primárním směru jiného agenta a ten tak při pohybu svým primárním směrem několik dlaždic ušetří.

3.4.2 Nekonečné cykly

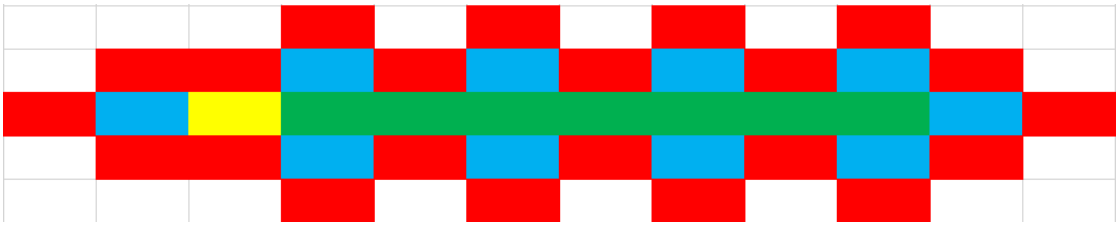
Agent se v nekonečném cyklu pohybuje po mřížce, dokud nesplní určitý úkol. V případě generování základního rozložení je tímto úkolem položení určitého množství dlaždic. Během generování základního rozložení je šance agentova zacyklení velice nízká. Agent se může pohybovat ve všech směrech, protože nejmenší pravděpodobnost pohybu do každého směru je nastavena na 5 %. Díky dostatečné velikosti mřížky se agent nemůže dostat na její okraj. Jediné překážky, které se v této fázi v mřížce nachází obklopují specifickou startovací dlaždici. Při výběru směru pohybu jsou překážky brány v potaz a agent se je snaží obejít.

Protože je agentův pohyb ovlivněn náhodou, nelze jednoznačně v jakýchkoliv případech zacyklení vyloučit. Pro pohyb do všech směrů je nastavena nenulová pravděpodobnost, a proto se agent může například donekonečna pohybovat mezi dvěma sousedními pozicemi v mřížce.

Při vkládání specifických dlaždic jsou po vložení specifické dlaždice okolo ní vloženy dlaždice typu Border ve směrech, kam ze specifické dlaždice nevedou cesty. Toto chování může vést k situacím, kdy se agent nachází v prostoru, který je zcela obklopen Border dlaždicemi. Agentovi se volnou pozici v takovém případě nikdy najít nepodaří. Tato situace je znázorněna na obrázku 3.2.

S touto situací se v kódu počítá. Po určitém počtu kroků dojde k restartování procesu vkládání specifických dlaždic agentem. Jestliže takové restartování proběhne několikrát, agent vkládání ukončí a je možné, že nebudou všechny specifické dlaždice vloženy. K zacyklení agenta by nemělo docházet při vkládání počtu specifických dlaždic v řádu jednotek. Ve fázi testování bude potřeba určit, jak často k zacyklení agenta dochází.

[4] NavMesh neboli Navigation Mesh je vrstva, která v *Unity* usnadňuje hledání cesty v prostoru.
[5] což by bylo v rozporu s požadavkem F3 ze sekce 1.4



Obrázek 3.2 Ukázka uzavření agenta v prostoru, žlutou barvou je zobrazena dlaždice typu Start, zelenou dlaždice typu Normal, červenou dlaždice typu Border a modrou specifická dlaždice tvaru DeadEnd.

Výpis kódu 3.5 Kontrola správnosti agentovy vygenerované cesty

```
private bool PathIsCorrect()
{
    return controlArea.IsPointInsideControlArea(agentGridPosition) ||
           IsStartingMovementRestricted();
}
```

3.5 Specifické startovací dlaždice

Do seznamu nutných dlaždic je možné umístit i dlaždice typu Start. Jestliže je takových dlaždic do seznamu vloženo více, jedna z nich je použita jako počáteční dlaždice dungeonu a ostatní jsou ze seznamu odstraněny. Specifikování tvaru počáteční dlaždice může však vést ke konfliktu s požadavkem generování dungeonu do specifických směrů.

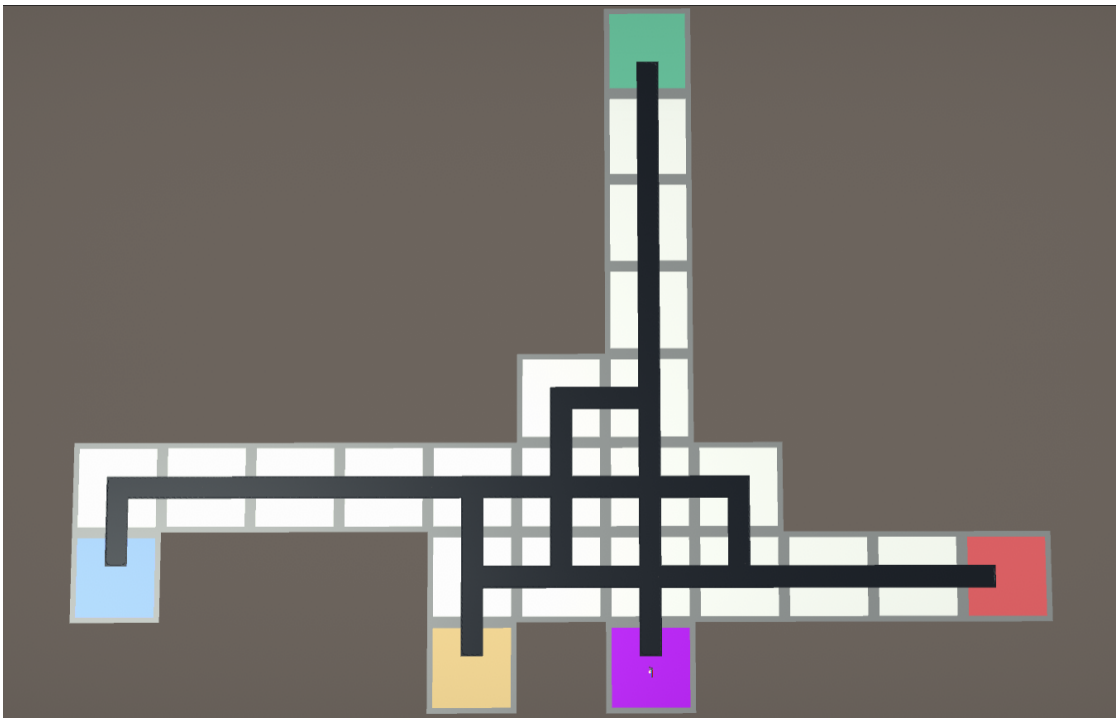
Na obrázku 3.3 můžeme vidět příklad dungeonu, který se nepodařilo vygenerovat korektně. Generátor měl v tomto případě za úkol umístit Boss dlaždice na všechny 4 strany dungeonu. Protože byla vybrána startovací dlaždice tvaru DeadEnd, vede z ní jen jeden východ. Přestože by agent chtěl generovat směrem dolů, musí učinit alespoň jeden pohyb směrem nahoru a do strany. Obcházením hranic startovací dlaždice se agent připravuje o dlaždice, které může položit, a není tak schopen ukončit svůj pohyb v kontrolní oblasti umístěné pod startovací dlaždicí.

Po domluvě se zadavatelem byl zrušen požadavek na nutnost umístění dlaždic na specifické strany dungeonu v případě, že je agentův pohyb na počátku omezen kvůli výběru konkrétního tvaru startovací dlaždice. Od agenta tedy v tomto případě nebude vyžadováno, aby ukončil svůj pohyb uvnitř své kontrolní oblasti. Kontrola správnosti agentovy vygenerované cesty probíhá jako na výpisu kódu 3.5.

3.6 Nedostatek křižovatek

Při vytváření základního rozložení agenty se objevil problém s nedostatkem křižovatek, respektive pozic pro specifické dlaždice tvaru Crossroad. Jestliže bylo do seznamu nutných dlaždic předáno několik křižovatek, agent je do mřížky musel vkládat až ve fázi vložení specifických dlaždic, protože je nebylo možné vložit do vygenerovaného základního rozložení. Stejný problém se týkal dlaždic tvaru DeadEnd. I v případech, kdy nebylo do dungeonu nutné vkládat Crossroad dlaždice, bylo toto chování považováno za problematické. Cesty bez křižovatek vygenerované jednotlivými agenty působily příliš jednolitě.

Jako jedno z možných řešení bylo otestováno pozměnění pravděpodobností pohybů agentů do určitých směrů. Na úkor pravděpodobnosti pohybu v agentově primárním směru byly zvýšeny pravděpodobnosti pohybů do směrů ostatních. Nepodařilo se však najít optimální nastavení



Obrázek 3.3 Nekorektní generování dungeonu při výběru specifické startovací dlaždice. Fialovou barvou je znázorněna startovací dlaždice. Na červené dlaždici skončil agent generující doprava, na zelené agent generující nahoru, na modré agent generující doleva. Na žluté dlaždici skončil agent generující dolů. Kvůli výběru specifické startovací dlaždice nebyl schopen ukončit svůj pohyb v kontrolní oblasti.

Výpis kódu 3.6 Rozhodování se o položení křižovatky

```
float randomResult = Random.Range(0f, 1f);

int filledNeighbours = tilesGrid.CountFilledNeighbours(agentGridPosition);
int neededTilesToLayCrossroad = 5 - filledNeighbours;
if (crossroadSpawnChance > randomResult &&
    tilesLeftToPlace >= neededTilesToLayCrossroad)
{
    PlaceCrossroad();
    crossroadSpawnChance -= 0.01f;
}
```

pravděpodobností. V případě že byla příliš snížena pravděpodobnost pohybu do primárního směru, agentovi dělalo potíže ukončit pohyb v pro něho stanovené kontrolní oblasti a agent musel být mnohokrát při generování restartován.

Do agentova algoritmu byla tak přidána nová pravděpodobnost – pravděpodobnost vložení křižovatky. Při vytváření základního rozložení dungeonu má nyní agent v základu 5% šanci, že místo jedné Normal dlaždice vloží okolo sebe do mřížky více Normal dlaždic tak, aby tvořily křižovatku. Tato pravděpodobnost se při každém položení křižovatky sníží o 1 %. Algoritmus rozhodování o položení křižovatky můžeme vidět na výpisu kódu 3.6.

Při vytvoření křižovatky tímto způsobem ve většině případů vznikne i několik dlaždic tvaru DeadEnd. Tato úprava algoritmu řeší tedy i jejich nedostatek.

3.7 Tvary dlaždic Portal

V nynější podobě hry jsou do scény umísťovány Portal dlaždice pouze tvaru DeadEnd. K tomuto rozhodnutí došlo po dohodě se zadavatelem. Dává totiž smysl, aby k východu z dungeonu hráč přišel z jednoho směru a za ním se už nenacházel žádný další herní obsah. Hráč tedy k portálu přijde ideálně až bude celý dungeon prozkoumán a nebude v něm už na hráče čekat žádná další výzva či odměna. Jestliže není v seznamu nutných dlaždic dlaždice typu Portal, je konkrétní model dlaždice generátorem vybrán náhodně.

3.8 Umísťování nepřátelských skupin

Podle požadavku F9 musí generátor rozmístit nepřátelské skupiny rovnoměrně a ideálně tak, aby spolu dlaždice, na kterých jsou nepřátelské skupiny umístěny, nesousedily. V průběhu implementace byl několikrát algoritmus vybíraní dlaždic pozměněn.

3.8.1 Dělení cest

První vyzkoušenou metodou bylo dělení cest agentů. Celkový počet nepřátel byl rovnoměrně rozdělen mezi jednotlivé agenty. Cesty, které agenti vygenerovali, byly následně rozděleny na části, jejichž počet odpovídal přiřazenému počtu nepřátelských skupin. Doprostřed každé části byla poté vložena nepřátelská skupina. Negativum této metody bylo rozmísťování nepřátel na křižovatkách. Nepřátelské skupiny na křižovatkách byly umístěny blízko u sebe a v některých případech spolu skupiny dokonce sousedily.

3.8.2 Víceúrovňový výběr

Další použitou metodou byl víceúrovňový výběr dlaždice vhodné pro umístění nepřátelské skupiny. Seznam kandidátů – dlaždic – je postupně filtrován na základě různých kritérií. Mým cílem bylo vybalancovat časovou složitost tohoto výběru a kvalitu samotného výběru. Zvažovaná kritéria:

- počet nepřátelských skupin v okolí o specifikované velikosti,
- počet nepřátelských skupin ve Von Neumannově sousedství,
- počet nepřátelských skupin v Moorově sousedství,
- vzdálenost od nejbližší nepřátelské dlaždice,
- počet sousedních dlaždic typu Normal,
- náhoda.

Různé kombinace kritérií vedly k různým výsledkům, které byly prezentovány zadavateli práce. Společně s ním bylo rozhodnuto o finální kombinaci kritérií.

Pro všechny kandidáty je spočtena vzdálenost od nejbližší dlaždice, která obsahuje nepřátelskou skupinu. V seznamu zbydou pouze dlaždice, které mají od nepřátelských skupin nejvyšší možnou vzdálenost. Je-li součástí seznamu více než jedna dlaždice, filtrování pokračuje. U zbylých kandidátů je zkoumáno, kolik nepřátelských skupin je umístěno v jejich okolí, jehož velikost je rovna vzdálenosti od nejbližší nepřátelské dlaždice. V seznamu zbydou pouze dlaždice, které mají počet takových skupin nejmenší možný. Jestliže je součástí seznamu stále více než jedna dlaždice, je z tohoto seznamu dlaždice vybrána náhodně.

Výsledná dlaždice je označena a bude s ní počítáno při dalším umísťování nepřátelských skupin jako s dlaždicí, která nepřátelskou skupinu obsahuje. Seznam všech takto označených dlaždic je po ukončení výběru dlaždic proiterován a na jednotlivých dlaždicích jsou skutečně v dungeon scéně vytvořeny nepřátelské skupiny.

3.9 Kill squad dlaždice

V průběhu implementace došlo ke změně požadavku na umísťování kill squad dlaždic do dungeonu. Štěpán Vejvoda vytvořil skript, který je předán třídě *DungeonController*, jehož součástí je seznam objektů typu *KillSquadMember*. Třída *KillSquadMember* v sobě obsahuje model nepřítele a počet nepřátel s tímto modelem, kteří tvoří jeden kill squad.

Tento seznam je předán až do třídy *AgentDungeonSceneGenerator* a ta pomocí třídy *UniformDungeonEnemySpawner* umístí kill squady stejným mechanismem, který je použit pro umísťování obyčejných nepřátelských skupin.

3.10 Zesílení nepřátel

Od Štěpána Vejvody přišel požadavek na upravování statistik nepřátel v dungeonu – konkrétně jde o upravování množství životů nepřátel a jejich poškození. Tento požadavek je vyjádřen skriptem *RobotAmplifier*, který je z třídy *DungeonController* propagován až k jednotlivým objektům *EnemySpawner*, které jsou umístěné na dlaždicích a jsou zodpovědné za umístění nepřátel na dlaždicích. Když *UniformDungeonEnemySpawner* vybere dlaždice na vytvoření nepřátel, je jim předán *RobotAmplifier* skript a po vložení nepřátel do scény jsou nepřátelům na základě tohoto skriptu zlepšeny jejich statistiky.

Bohužel se při návrhu nepřátel s vylepšováním jejich poškození nepočítalo a v současném stavu není možné poškození nepřátel rozumným způsobem upravit. Systém nepřátelských statistik tak

Výpis kódu 3.7 Původní algoritmus rozdělování počtů předmětů mezi dlaždice

```
int itemCountForTile = itemCountLeft / remainingTiles;
itemCountLeft -= itemCountForTile;
--remainingTiles;
```

potřebuje projít refaktoringem, který nebyl předmětem této práce. Nepřátelské životy se ale nepřátelům úspěšně upravit podařilo.

3.11 Umísťování předmětů

Práce Štěpána Vejvody rozšířila druhy předmětů, které se v dungeonu mohou objevit a mohou být sebrány hráčem. Kromě materiálů byly mezi tyto objekty přidány i dary a lektvary. Došlo proto k přejmenování MaterialSpawneru na ItemSpawner a byl tedy rozšířen seznam předmětů, které designér může v instanci třídy *DungeonConfig* nastavit v proměnné *MaterialCounts* (nyní *ItemCounts*).

3.12 Rozdělování předmětů

Po dokončení generování dungeonu je potřeba na vygenerovaných dlaždicích rovnoměrně umístit předměty, které hráč bude moct sebrat. Pro každou dlaždici je potřeba určit, kolik takových předmětů se na ní objeví a následně jaké konkrétní předměty to budou. V případě, že je mezi 20 dlaždic potřeba rozdělit 50 předmětů, se ideálně na 10 dlaždicích musí objevit 2 předměty a na 10 dlaždicích 3 předměty. Při zpracovávání jednotlivých dlaždic ve scéně nejdříve docházelo k tomu, že se pro konkrétní dlaždici počítal počet předmětů, které se v ní musí objevit, tak, jak je ukázáno na výpisu kódu 3.7.

Pokud při výpočtu proměnné *itemCountForTile* vyšel neceločíselný výsledek, byl výsledek oříznut o desetinou část a efektivně zaokrouhlen směrem dolů. Docházelo tedy k tomu, že prvních 10 dlaždic dostalo 2 předměty a posledních 10 dlaždic 3 předměty. Dlaždice byly do seznamu vloženy v pořadí, ve kterém byly agentem vloženy do mřížky. Jestliže generoval pouze jeden agent, znamenalo to, že prvních 10 vložených dlaždic obsahovalo méně předmětů než posledních 10.

Mohlo by se jednat o úmyslnou mechaniku, tedy čím více by hráč dungeonem procházel, tím více by byl odměňován ve formě sebratelných předmětů. Takováto funkcionalita nebyla v požadavcích vydefinována a bude jistě předmětem budoucí diskuze se zadavatelem.

Počet předmětů pro jednotlivé dlaždice je nyní vkládán do seznamu, jehož obsah je následně náhodně promíchán. Jednotlivé dlaždice počty předmětů ze seznamu sekvenčně vybírají a dochází tak k rovnoměrnějšímu rozmístění počtu předmětů po dungeonu.

Kapitola 4
Testování

Tato kapitola se v sekci 4.1 věnuje testovacím instancím třídy *DungeonConfig*, které byly vytvořeny v průběhu vývoje. Sekce 4.2 se věnuje uživatelskému testování, které proběhlo ve dvou různých formách. V sekci 4.3 je zmíněno testování v laboratoři *SAGElab*.

4.1 Testovací konfigurace dungeonů

Za účelem průběžného testování generátoru vznikly testovací instance třídy *DungeonConfig*, jejichž účelem bylo otestování specifických funkcionalit generátoru. Jednotlivé instance je možné přiřadit k libovolnému úkolu a následně přechodem do dungeon scény zkontrolovat korektnost vygenerovaného dungeonu. Tato testovací data se nachází v samostatných složkách ve složce Assets/ScriptableObjects/DungeonConfigs/Test.

4.1.1 Testovací tilesety

Pro testovací účely vznikly 2 tilesety – BlackAndWhite a BlackAndWhite2. Dlaždice v těchto tilesetech obsahují pouze nejnutnější objekty – portály, objekty pro umísťování nepřátel a sebratelných předmětů a rozcestníky sloužící k návratu do vesnice.

Dlaždice v tilesetu BlackAndWhite2 mají jinou velikost než dlaždice z ostatních tilesetů – 17 jednotek. Ve složce DifferentTileSize se nachází instance třídy *DungeonConfig*, která používá tileset BlackAndWhite2. Cesty ve vygenerovaném dungeonu na sebe nutně nenavazují, což nevadí, protože se zde hlavně testuje, aby se dlaždice o liché velikosti správně do scény vložily – nepřekrývaly se a nebyly mezi nimi mezery.

4.1.2 Testování propojení dungeonů

Ve složce DungeonGrid se nachází instance třídy *DungeonConfig*, které dohromady v dungeon scéně vytvoří mřížku o velikosti 3x3. Názvy jednotlivých instancí odpovídají jejich pozici v mřížce – například *DungeonConfig* TopRight se nachází v pravém horním rohu mřížky. Na svém spodním okraji by měl obsahovat portál do dungeonu definovaného konfigurací dungeonu jménem Right a na svém levém okraji by měl sousedit s dungeonem, který je definován konfigurací dungeonu jménem Top.

Tato testovací data slouží k otestování správného propojení portálů mezi dungeony. Krom toho jsou jednotlivým instancích třídy *DungeonConfig* přiřazeny různé kombinace parametrů *DungeonSize*, *Difficulty*, *Tileset* a *Border Radius*. V každém tilesetu je generátorem také umístěn jiný předmět.

4.1.3 Testování specifických startovacích dlaždic

Ve složce SpecificStart se nachází *DungeonConfig*, kterému byla přiřazena specifická startovací dlaždice. V seznamu Quest Tiles má tato instance třídy *DungeonConfig* také přiřazeny specifické dlaždice. Kromě toho, že se specifická startovací dlaždice opravdu použila, je testována i její rotace na základě toho, do jakých směrů se budou cesty v dungeonu ubírat. Směry cest byly dány parametry *Required Dungeon Side* u jednotlivých položek v seznamu *Quest Tiles*.

Rotace dlaždice by měla být nastavena tak, aby z dlaždice vycházely cesty alespoň do jednoho směru cest v dungeonu. V případě, že tvar startovací dlaždice odpovídá uspořádání směrů, do kterých se v dungeonu generuje, by měla počáteční dlaždice mít takovou rotaci, aby z ní agenti mohli rovnou do daných směrů vyrazit a nenarazili na překážku ve formě Border dlaždice.

Touto konfigurací dungeonu bylo také testováno, že je správně vyplněno okolí specifické startovací dlaždice dlaždicemi typu Normal a Border.

4.1.4 Testování zacyklení agentů

Ve složce NecessaryTiles se nachází testovací konfigurace dungeonů testující zacyklení agentů při vkládání různého počtu specifických dlaždic. Konfigurace, jejichž jména začínají na VariousTileShapes testují vkládání specifických dlaždic, které mají různé tvary. Konfigurace, jejichž jména začínají na DeadEndTiles testují vkládání specifických dlaždic tvaru DeadEnd, které jsou největším rizikem pro zacyklení agenta.[1]

V rámci testování zacyklení bylo vytvořeno celkem 6 konfigurací dungeonů. Parametr *DungeonSize* všech konfigurací byl nastaven na Small, agent tedy při vytváření základního rozložení dungeonu musel vložit 20 dlaždic. Nebyla zvolena specifická startovací dlaždice.

3 konfigurace dungeonu byly nastaveny tak, aby generátor musel do dungeonů s jistotou vložit 5, 10 a 15 dlaždic typu DeadEnd. Další 3 konfigurace byly nastaveny tak, aby generátor musel do dungeonů vložit 10, 15 a 20 dlaždic, přičemž zde byly tvary dlaždic rovnoměrně rozloženy. Tyto konfigurace tedy obsahují 2, 3 a 4 specifické dlaždice od každého možného tvaru dlaždice.

Jedna specifická dlaždice byla vždy vložena do seznamu Quest Tiles a agent byl poslán vygenerovat ji na pravou stranu dungeonu. Zbytek specifických dlaždic byl vložen do seznamu *Necessary Tiles*. Důvodem pro toto rozhodnutí je snaha o otestování jednoho agenta. Kdyby se dungeon generoval do více směrů, nutné dlaždice by byly mezi agenty rovnoměrně rozděleny a agenti by při jejich pokládání dosahovali lepších výsledků.

Každá z těchto 6 konfigurací dungeonu byla 100× použita pro generování dungeonu a u každé vygenerované instance bylo zaznamenáváno, kolikrát došlo k restartu procesu vkládání specifických dlaždic agentem a zda byl proces úspěšně dokončen či nikoliv. K restartu procesu došlo vždy poté, co agent udělal více než 100 000 kroků po mřížce. Proces vkládání byl ukončen poté, co byl takto agent restartován 10× a stále se mu do mřížky nepodařilo vložit všechny specifické dlaždice. Agent vkládal nejdříve specifické dlaždice ze seznamu *Necessary Tiles* a odděleně od nich specifickou dlaždici ze seznamu Quest Tiles. Mohlo tedy dojít k 0–20 restartům procesu vkládání. Výsledky testování jsou zaneseny v tabulce 4.1.

U případu vkládání 15 dlaždic s tvarem DeadEnd si můžeme všimnout, že byl průměrný počet restartů agenta 5,65. Každý restart agenta negativně ovlivňuje čas čekání na vygenerování dungeonu. I v případech, kdy k ukončení procesu vkládání nedošlo, byla doba čekání často výrazně vyšší než u ostatních testovaných instancí třídy *DungeonConfig*.

Ve většině případů nejspíš nebude počet specifických dlaždic tak vysoký, jedná se i přesto o nepříjemné zjištění. Lepších výsledků by mohlo být dosaženo úpravou algoritmu pro pohyb agenta po mřížce. Z kapitoly Návrh je zřejmé, že je na tuto úpravu – implementaci agenta s méně náhodnými pohyby – generátor připraven.

[1] Je to proto, že je s jejich vkládání spojeno vložení největšího množství dlaždic typu Border, které mohou agenta uzavřít v prostoru.

Tabulka 4.1 Výsledky testování zacyklení agentů. Každý *DungeonConfig* byl vygenerován 100×. Při každém generování byl spočten počet restartů agenta.

DungeonConfig	Počet ukončení procesu vkládání	Průměrný počet restartů
DeadEndTiles5	0	0
DeadEndTiles10	8	1,68
DeadEndTiles15	21	5,65
VariousTileShapes10	0	0
VariousTileShapes15	0	0,03
VariousTileShapes20	2	0,17

4.2 Uživatelské testování

Ve fázi testování proběhly 2 druhy uživatelského testování – designérské a hráčské. Pro oba druhy testování byly vytvořeny vlastní scénáře, pro designérské testování i návod. Testeři následně odpovídaly na otázky týkající se testování v dotaznících.

Pro uživatelské testování vznikla nová verze hry *Magitech*, ve které hráč při spuštění hry již začínal s několika úkoly a byl ve vesnici přesunut poblíž přechodu do dungeonu. Účelem této úpravy bylo minimalizování kontaktu testerů s ostatními systémy, které s generováním dungeonů nesouvisely, a testeři se soustředili hlavně na samotné dungeony.

4.2.1 Designérské testování

Cílem designérského testování bylo získání zpětné vazby k procesu designu dungeonů. Designérského testování se celkem účastnili 3 testeři, z nichž 2 byli členové týmu zodpovědného za vývoj hry *Magitech*, jedním z nich byl i zadavatel práce.

Během testování měli designéři k dispozici příručku pro designéry, která jim měla posloužit v případě, kdy by se při plnění scénáře na některém kroku zasekli. Příručku je možné nalézt v příloze.

Scénář

V prvním scénáři měli designéři za úkol vytvořit instanci třídy *DungeonConfig* o specifických parametrech a zkontrolovat, že dungeon, který se na jeho základě vygeneroval odpovídá parametrům, které zadali.

V druhém scénáři designéři vytvořili druhou instanci třídy *DungeonConfig*, upravili její parametry a obě dvě instance propojili portály. Následně zkontrolovali, že mohou mezi vygenerovanými dungeony pomocí portálů přecházet.

Ve třetím scénáři designéři změnili parametry úkolu tak, aby se pro něj vygeneroval nekonečný dungeon. Otestovali, že se pro ně v rámci dungeon scény stále generují nové dungeony, kterými mohou procházet.

Průběh testování

Jednomu z testerů se nepodařilo dokončit třetí scénář. Kvůli deaktivovanému skriptu nepřátelského bosse byl proces generování přerušen. Hráčova postava nebyla správně umístěna na startovací dlaždici a tester tak nemohl zkontrolovat, že je vygenerovaný dungeon skutečně nekonečný. Chybu se podařilo identifikovat a opravit.

V dungeonu, který byl vygenerován pro jednoho z testerů v rámci třetího scénáře, se portál na přechod do dalšího dungeonu nevygeneroval u konce dungeonu, ale zhruba v jeho středu.

Výsledky dotazníku

Jeden z testerů uvedl, že nemá žádné zkušenosti s jakoukoliv formou designování herních lokací. Další dva uvedli, že párkrát již herní lokace designovali. Dva z testerů vůbec příručku pro designéry nepotřebovali, jeden ji označil za spíše srozumitelnou, potřebovala by ale rozšířit o více obrázků.

Dva testeři označili proces designování za jednoduchý a intuitivní, jeden uvedl, že je prostor pro zlepšení. Proces návrhu v prostředí *Unity* označil jako těžkopádný. Tato těžkopádnost by se dala vyřešit vytvořením vlastního externího programu pro návrh dungeonů.

Možnosti, které nyní *Magitech* pro designování dungeonů nabízí, se všem testerům jevily jako dostačující pro vygenerování potřebných dungeonů. Jeden z testerů následně poznamenal, že pokud by se systém používal pro generování v jiných hrách, mohl by být rozšířen.

Ani jeden z testerů nezaznamenal ve vygenerovaných dungeonech odchylky od parametrů, které zadal v instancích třídy *DungeonConfig*.

Zpětná vazba

Testerům přišlo neintuitivní pojmenování dlaždic v tilesetech. Název dlaždice by kromě jejího účelu v dungeonu měl obsahovat i tileset, ve kterém se dlaždice nachází. Bylo pro ně obtížné vybrat dlaždici z konkrétního tilesetu pomocí grafického prostředí, které *Unity* k výběru dlaždice nabídlo. Informaci o tom, ze kterého tilesetu dlaždice pochází, šlo vypozorovat pouze z cesty k danému prefabu dlaždice, která byla sotva čitelná.

Při průchodu portálem poznamenali dva testeři, že není vhodné přesun hráče aktivovat při kolizi hráčovy postavy s portálem. Jeden z testerů navrhl, že by bylo vhodné přesun aktivovat zmáčknutím specifické klávesy.

4.2.2 Hráčské testování

Cílem hráčského testování bylo získání zpětné vazby k dungeonům, které jsou ve hře *Magitech* generovány. Hráčského testování se zúčastnilo 6 testerů. Testeři při spuštění hry měli v seznamu úkolů již zadány 4 úkoly, pro které byly vygenerovány různé dungeony – předpřipravený dungeon pro 1. příběhový úkol, náhodně vygenerovaný dungeon, dungeon scéna skládající se z více dungeonů propojených portály a nekonečný dungeon.

Dungeon pro 1. příběhovou misi vznikl z konfigurace dungeonu s parametrem *DungeonSize* nastaveným jako Small a parametrem *Difficulty* nastaveným jako Easy. Parametry konfigurace dungeonu pro náhodný a nekonečný dungeon byly vygenerovány náhodně. Dungeon scéna s portály se skládala z dungeonů, které už byly popsány v sekci 4.1.2.

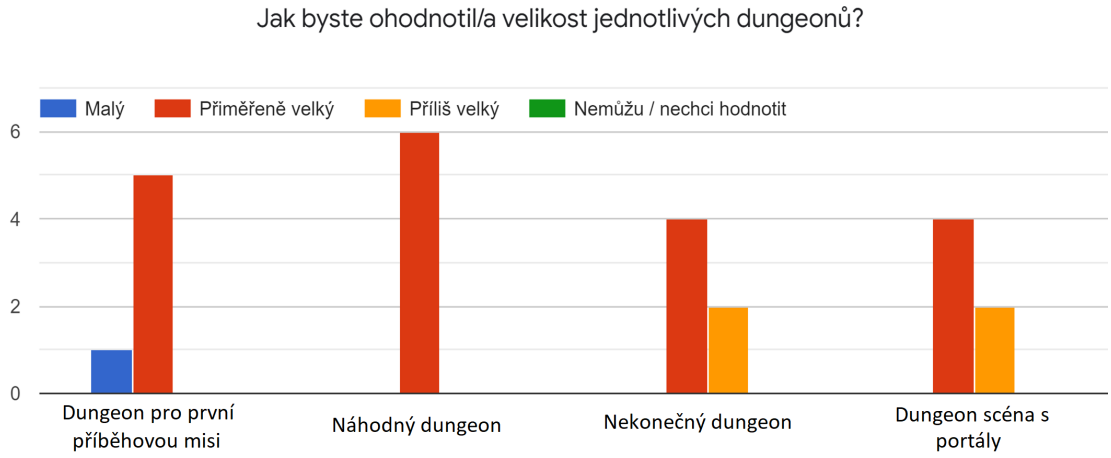
Scénář

Úkolem testerů ve scénáři bylo splnění co nejvyššího počtu úkolů a průběžné sdělování dojmů moderátorovi. Jestliže měli testeři potíže s plněním úkolu, byl jim úkol moderátorem připomenut a testeři byli nasměrování k cíli. Po skončení testování testeři vyplnili dotazník.

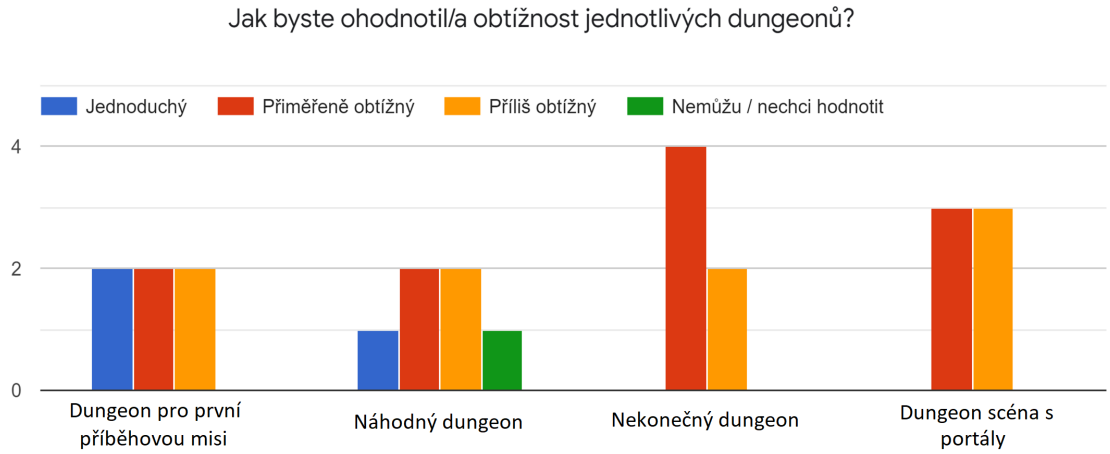
Průběh testování

Pouze některým hráčům se podařilo úspěšně dokončit všechny úkoly[2], všichni si ale nechali dungeony pro jednotlivé úkoly vygenerovat a alespoň chvíli v nich strávili.

[2] S výjimkou nekonečného úkolu, který nelze splnit z jeho podstaty.



Obrázek 4.1 Hodnocení velikosti jednotlivých dungeonů



Obrázek 4.2 Hodnocení obtížnosti jednotlivých dungeonů

Výsledky dotazníku

Všichni testeři byli muži aktivně hrající počítačové hry[3] a měli alespoň nějakou zkušenost s hrami obsahující element procedurálního generování herních lokací.

Jednomu z testerů se jako nejzajímavější zdál dungeon připravený pro 1. příběhový úkol, jednomu náhodně vygenerovaný dungeon a dvěma zase dungeon s portály. Ostatní ohodnotili všechny dungeony jako stejně zajímavé.

Obecná obtížnost dungeonů ve hře *Magitech* byla ohodnocena čtyřmi testery jako těžká a dvěma jako přiměřená. Následně testeři hodnotili jednotlivé vlastnosti 4 vygenerovaných dungeonů jako je jejich velikost, obtížnost či schopnost orientace v nich.

Hodnocení velikostí a obtížností jednotlivých dungeonů můžeme vidět na obrázcích 4.1 a 4.2.

Zajímavá, ale nikterak překvapivá data byla zjištěna při testování hráčské orientace v dungeonech. Hráči se v dungeonech, které neobsahovaly portály, většinou po celou dobu zcela orientovali, případně se orientovali obtížně. Naopak v nekonečném dungeonu se zcela orientoval pouze 1 tester, v dungeonu s portály žádný. 2 testeři se v těchto dungeonech dokonce ztratili.

[3] To znamená, že hrají počítačové hry několik hodin denně či týdně.

Žádný z testerů nenarazil na chybu, která by se týkala generování dungeonů. Všechny dungeony bylo možné dohrát. Jeden z testerů poznamenal, že popisky úkolů nesouvisely s testovanými dungeony.

Zpětná vazba

3 testeři měli problémy s interakcí své postavy s portálem a s ukazatelem, který hráče přesune zpět do vesnice. Interakce ve formě kolize s těmito objekty byla zhodnocena jako nevhodná. Jeden z testerů navrhl dotázat se při kolizi hráče, jestli se chce opravdu přesunout. Dva testeři navrhli interakci s těmito objekty ve formě zmáčknutí tlačítka akce při přiblížení se k danému objektu.

Dva testeři měli obtíže poznat, které předměty lze v dungeonu sebrat a které jsou jen součástí předpřipravené dlaždice a slouží k dekorativním účelům.

Podle jednoho z testerů se hráč dostává do akce příliš brzo. Když se hráč objeví v dungeonu, neměl by být hned pod útokem nepřátel.

V tilesetu vesnice by jeden z testerů zvýšil diverzitu jednotlivých dlaždic. Klikatící se cesty v tilesetu jeskyně byly jedním z testerů kladně ohodnoceny. Byť se jedná o designérský trik, proces generování tak vypadá sofistikovaněji. Jeden z testerů ohodnotil tileset lesa jako příliš generický, pravoúhlé cesty podle něj dělají z dungeonu spíš bludiště.

Další zpětná vazba se týkala systémů, které nejsou předmětem této bakalářské práce a byla předána ostatním členům vývojářského týmu.

4.3 Testování v laboratoři

Hra *Magitech* byla testována v síťové multimediální laboratoři *SAGElab*. Účelem testování bylo zjistit, zda se veškeré prvky uživatelského rozhraní správně škálují i na zařízení o netradičním rozlišení 9600×4320 px. Herní uživatelské rozhraní však s mou prací příliš nesouvisí, a proto byli o výsledcích testování zpraveni ostatní členové vývojářského týmu zodpovědní za práci s uživatelským rozhraním.

Závěr

Cílem práce bylo vytvoření procedurálního generátoru herních prostředí – dungeonů – do hry *Magitech*. Implementaci modulu generátoru předcházel popis samotného procedurálního generování obsahu.

V rámci teoretické části byla vypracována analýza již existujících řešení, na jejímž základě byl modul navržen. Bylo popsáno několik metod, pomocí kterých je možné ve hrách dungeony generovat, včetně konkrétních případů jejich použití. Analytická část ukázala jako jedno z vhodných řešení procedurálního generování dungeonů ve hře *Magitech* využití agentů. Modul generátoru byl následně naimplementován v prostředí *Unity*, které je použito i pro implementaci ostatních části hry *Magitech*. Výsledný procedurální generátor byl do hry úspěšně zakomponován a důkladně otestován.

Výsledný modul generátoru umožňuje designérům jednoduše definovat herní prostory, které by v průběhu hry měly být vygenerovány. Tyto předdefinované herní prostory může designér přiřadit k určitým úkolům, které hráč musí během hry plnit. Na jejich základě jsou v průběhu hry herní prostory vygenerovány. Jestliže není herní prostor k úkolu přiřazen, generátor je schopen ho pro úkol náhodně vytvořit. Generátor umožňuje generování nekonečných herních prostorů, v nichž je hráčovým úkolem přežít se svou postavou co nejdéle. Designér může u herního prostoru ovlivňovat jeho vzhled, velikost, obtížnost či seznam dlaždic, které musí být při generování použity. U dlaždic, které slouží k tomu, aby na nich hráč plnil úkol, je možné specifikovat, na jaké straně dungeonu se musí nacházet. Generátor umožňuje spojovat více dungeonů, mezi kterými může hráč v průběhu hry přecházet. Uvnitř dungeonů na základě parametrů herního prostoru generátor vytváří skupiny nepřátel, se kterými hráč musí bojovat. Do dungeonu mohou být generátorem umístěny i předměty, které může hráč sbírat.

V budoucnu by bylo možné modul rozšířit o možnost upravovat rozložení konkrétních dlaždic v tilesetech. Do jednotlivých dlaždic by mohly být generátorem umístěny specifické herní prvky, například portály, což by výrazně zjednodušilo vytváření nových tilesetů designérem. Dále by mohl být implementován sofistikovanější algoritmus pro pohyb agentů v mřížce, který by dokázal i složitější požadavky na generovaný dungeon splnit v rozumném čase.

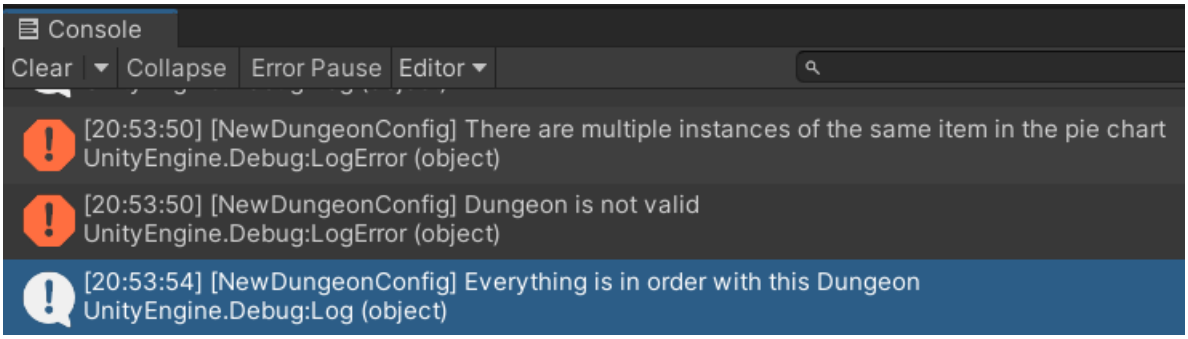
Příloha A

Příručka pro designéry

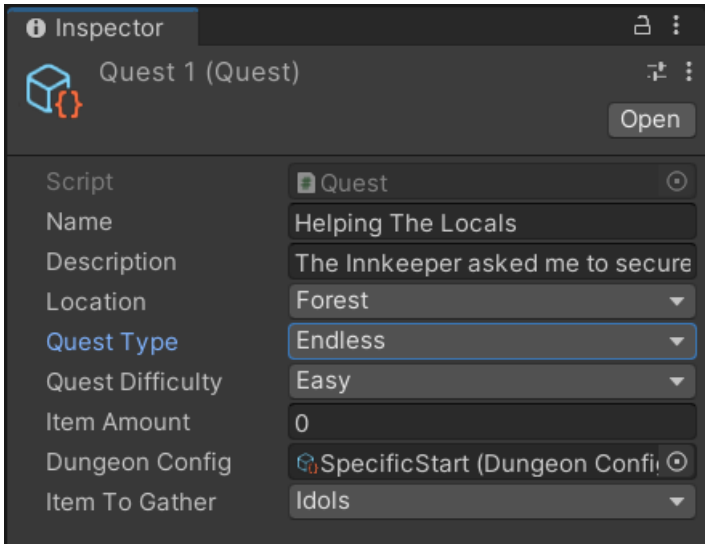
Vzhled generovaného dungeonu je založen na parametrech s ním související třídy DungeonConfig. Toto je návod, jak DungeonConfig vytvořit, nastavit a přiřadit ho ke konkrétnímu úkolu. V průběhu tvoření dungeon configu systém kontroluje jeho validitu. Nebude-li dungeon config po jeho vytvoření validní, nebude použit generátorem pro vytvoření reálného dungeonu v dungeon scéně. Současný stav validity dungeon configu je možné vypozorovat z výpisů v záložce Console. Ukázka výpisů vztahujících se k validitě dungeon configů v okně Console je na obrázku A.1.

Tento návod je určen pro vytváření dungeon scény skládající se z konečného počtu dungeonů. Jestliže chcete pro úkol nastavit nekonečný dungeon, musíte tak učinit ve scriptable objectu reprezentující daný úkol – parametr Quest Type tohoto úkolu nastavte na Endless. Příklad úkolu, který je nastaven jako nekonečný je na obrázku A.2.

1. Ve složce ScriptableObjects/DungeonConfigs vytvořte scriptable object DungeonConfig.
   a. Klikněte levým tlačítkem na symbol „+“ v levém horním rohu záložky Project.
   b. Klikněte levým tlačítkem na ScriptableObjects/DungeonConfig.
2. Pojmenujte nově vytvořený DungeonConfig.
3. Rozklikněte nově vytvořený DungeonConfig tak, aby se Vám objevil v záložce Inspector.
4. V záložce Inspector upravte parametry DungeonConfigu.
   a. Nastavte požadovaný tileset, který má být pro generování dungeonu použit.



Obrázek A.1 Výpisy v okně Console



Obrázek A.2 Ukázka nastavení nekonečného úkolu

b. Nastavte požadovanou velikost dungeonu.

c. Nastavte požadovanou obtížnost dungeonu.

d. Nastavte parametr border radius. Border radius značí velikost hranic dungeonu, tedy kolik hraničních dlaždic bude vygenerováno na okrajích dungeonu. ideální velikost hranic je 1 nebo 2.

e. Nastavte prefab hraniční dlaždice. Jestliže jste v přechozím kroku nastavili nenulovou velikost hranic dungeonu, musíte nějaký prefab hraniční dlaždice nastavit, jinak nebude Váš dungeon config validní. Při výběru dlaždice by Vám její název měl napovědět k jakému účelu dlaždice v dungeonu slouží. Název dlaždice, která se používá pro generování hranic, bude obsahovat řetězec „TileBorder“. Číslo v názvu dlaždice odkazuje na její tvar. V případě hraničních dlaždic nemá tvar na vzhled dlaždic vliv.

f. Do seznamu Necessary Tiles vložte prefaby dlaždic, které chcete, aby se v dungeonu s jistotou objevily.

g. Do seznamu Possibly Spawned Tiles vložte prefaby dlaždic, které chcete, aby se v dungeonu s určitou pravděpodobností objevily. U těchto dlaždic nastavte i pravděpodobnost, že se v dungeonu objeví, a jejich počet. Tento počet značí kolikrát se bude o dlaždici rozhodovat, zda se v dungeonu objeví, na základě její pravděpodobnosti.

h. Do seznamu Quest Tiles vložte prefaby dlaždic, které souvisí s úkolem, který se bude v dungeonu plnit. U těchto dlaždic nastavte, na jaké straně dungeonu se musí tyto dlaždice objevit. Jestliže nespecifikujete konkrétní úkolovou dlaždici, systém automaticky použije náhodnou dlaždici s bossem.

i. Do seznamu Item Counts nastavte, které předměty se musí v dungeonu objevit. U každého předmětu nastavte počet, ve kterém chcete, aby se v dungeonu daný předmět vyskytoval. V seznamu Item Counts se nesmí jeden druh předmětu vyskytnou více než jednou, jinak bude Váš dungeon config nevalidní.

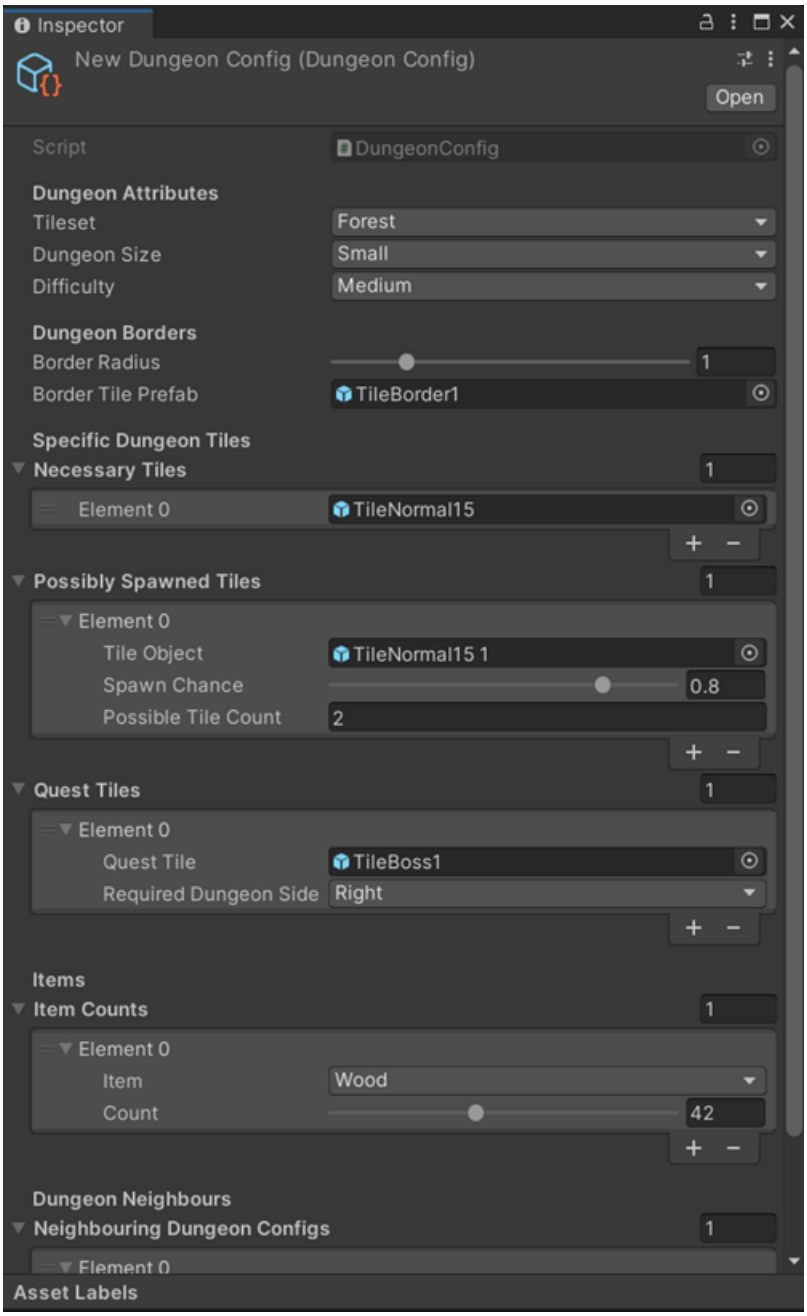
j. Do seznamu Neighbouring Dungeon Configs vložte DungeonConfigy dungeonů, se kterými má Váš dungeon sousedit. U každého dungeonu specifikujte, na jaké straně nyní tvořeného dungeonu se má nacházet přechod do tohoto sousedního dungeonu. Na jedné straně dungeonu se může nacházet přechod do maximálně jednoho sousedního dungeonu. Jestliže tomu tak není, Váš dungeon config nebude validní. Jestliže do seznamu Neighbouring Dungeon

Configs nevložíte konkrétní DungeonConfig, Vámi tvořený DungeonConfig také nebude validní. Jestliže do seznamu Neighbouring Dungeon Configs vložíte Vámi tvořený DungeonConfig, takový DungeonConfig taktéž nebude validní. Jestliže na jednu stranu dungeonu umístíte sousední dungeon, ujistěte se, že na opačnou stranu sousedního dungeonu configu umístíte Vámi právě tvořený dungeon config.

5. V záložce Console zkontrolujte, zda je dungeon config, který jste vytvořili, validní. Jestliže není, výpisy v záložce Console Vám pomohou zjistit proč. Nevalidní dungeon config prosím upravte podle výpisů v záložce Console nebo nebude pro generování použit. Příklad správně vyplněného DungeonConfigu můžete vidět na obrázku A.3.

6. Ve složce ScriptableObjects/Quests se nachází questy, kterým můžete Vámi vytvořený dungeonConfig přiřadit. Pro nejrychlejší otestování Vámi vytvořeného dungeonuConfigu doporučuji přiřadit Váš dungeonConfig k úkolu Quest 1 ve složce ScriptableObjects/Quests/MainQuests/Innkeeper. Rozkliknete-li si tento úkol tak, aby se Vám zobrazil v záložce Inspector, uvidíte jako jeden z jeho parametrů parametr DungeonConfig. Do tohoto parametru prosím přiřaďte Vámi vytvořený DungeonConfig.

7. Po spuštění hry přijměte první úkol od hospodského a běžte s tímto úkolem přes most do dungeonu. Zkontrolujte, že vygenerovaný dungeon odpovídá Vašim požadavkům.



Obrázek A.3 Příklad správně vyplněného DungeonConfigu

Bibliografie

1. SMITH, Gillian; TREANOR, Mike; WHITEHEAD, Jim; MATEAS, Michael. Rhythm-Based Level Generation for 2D Platformers. In: *Proceedings of the 4th International Conference on Foundations of Digital Games* [online]. Orlando, Florida: Association for Computing Machinery, 2009, s. 175–182 [cit. 2022-04-24]. FDG '09. ISBN 9781605584379. Dostupné z DOI: 10.1145/1536513.1536548.
2. DAHLSKOG, Steve; BJÖRK, Staffan; TOGELIUS, Julian. Patterns, Dungeons and Generators. In: *Foundations of Digital Games* [online]. 2015 [cit. 2022-04-24]. Č. 30. Dostupné z: http://urn.kb.se/resolve?urn=urn:nbn:se:mau:diva-16749.
3. TOGELIUS, Julian; KASTBJERG, Emil; SCHEDL, David; YANNAKAKIS, Georgios N. What is Procedural Content Generation? Mario on the Borderline. In: *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games* [online]. Bordeaux, France: Association for Computing Machinery, 2011 [cit. 2022-04-24]. PCGames '11. ISBN 9781450308724. Dostupné z DOI: 10.1145/2000919.2000922.
4. WERNECK, Mariana; CLUA, Esteban W. G. Generating Procedural Dungeons Using Machine Learning Methods. In: *2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)* [online]. 2020, s. 90–96 [cit. 2022-04-24]. Dostupné z DOI: 10.1109/SBGames51465.2020.00022.
5. TOGELIUS, Julian; YANNAKAKIS, Georgios N.; STANLEY, Kenneth O.; BROWNE, Cameron. Search-Based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games* [online]. 2011, roč. 3, č. 3, s. 172–186 [cit. 2022-04-24]. Dostupné z DOI: 10.1109/TCIAIG.2011.2148116.
6. YANNAKAKIS, Georgios N.; TOGELIUS, Julian. Experience-Driven Procedural Content Generation. *IEEE Transactions on Affective Computing* [online]. 2011, roč. 2, č. 3, s. 147–161 [cit. 2022-04-24]. Dostupné z DOI: 10.1109/T-AFFC.2011.6.
7. LINDEN, Roland van der; LOPES, Ricardo; BIDARRA, Rafael. Procedural Generation of Dungeons. *IEEE Transactions on Computational Intelligence and AI in Games* [online]. 2014, roč. 6, č. 1, s. 78–89 [cit. 2022-04-24]. Dostupné z DOI: 10.1109/TCIAIG.2013.2290371.
8. COMPTON, Kate; OSBORN, Joseph C; MATEAS, Michael. Generative methods. In: *The Fourth Procedural Content Generation in Games workshop, PCG* [online]. 2013, sv. 1 [cit. 2022-04-23]. Dostupné z: http://www.pcgworkshop.com/archive/compton2013generative.pdf.
9. SHAKER, Noor; TOGELIUS, Julian; NELSON, Mark J. *Procedural Content Generation in Games* [online]. Springer, 2016 [cit. 2022-04-18]. ISBN 978-3-319-42716-4. Dostupné z: https://link.springer.com/content/pdf/10.1007%2F978-3-319-42716-4.pdf.

10. DOULL, Andrew. The Death of the Level Designer. *Ascii Dreams* [online]. 2008 [cit. 2022-04-18]. Dostupné z: http://njema.weebly.com/uploads/6/3/4/5/6345478/the_death_of_the_level_designer.pdf.
11. *Procedural content generation wiki: Andrew Doull* [online]. 2008 [cit. 2022-04-18]. Dostupné z: http://pcg.wikidot.com/pcg-algorithm:randomness.
12. SMITH, Adam M.; MATEAS, Michael. Answer Set Programming for Procedural Content Generation: A Design Space Approach. *IEEE Transactions on Computational Intelligence and AI in Games* [online]. 2011, roč. 3, č. 3, s. 187–200 [cit. 2022-04-25]. Dostupné z DOI: 10.1109/TCIAIG.2011.2158545.
13. SMITH, Gillian. An Analog History of Procedural Content Generation. In: *Proceedings of the 2015 Conference on the Foundations of Digital Games* [online]. Monterey, CA, USA, 2015 [cit. 2022-04-23]. Dostupné z: http://www.fdg2015.org/papers/fdg2015_paper_19.pdf.
14. AMERICAN POETS, Academy of. *Haiku* [online]. 2022 [cit. 2022-04-25]. Dostupné z: https://poets.org/glossary/haiku.
15. *The Earliest CRPGs* [online]. 2011 [cit. 2022-04-18]. Dostupné z: http://crpgaddict.blogspot.com/2011/12/earliest-cprgs.html.
16. *Berlin interpretation* [online]. 2008 [cit. 2022-04-18]. Dostupné z: http://www.roguebasin.com/index.php/Berlin_Interpretation.
17. BREWER, Nathan. Computerized Dungeons and Randomly Generated Worlds: From Rogue to Minecraft [Scanning Our Past]. *Proceedings of the IEEE* [online]. 2017, roč. 105, č. 5, s. 970–977 [cit. 2022-04-24]. Dostupné z DOI: 10.1109/JPROC.2017.2684358.
18. GREUTER, Stefan; PARKER, Jeremy; STEWART, Nigel; LEACH, Geoff. Real-Time Procedural Generation of 'pseudo Infinite' Cities. In: *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia* [online]. Melbourne, Australia: Association for Computing Machinery, 2003, 87–ff [cit. 2022-04-24]. GRAPHITE '03. ISBN 1581135785. Dostupné z DOI: 10.1145/604471.604490.
19. DJUNDIK, Pavel. *SteamDB Instant Search* [online]. 2013 [cit. 2022-05-07]. Dostupné z: https://steamdb.info/instantsearch/?refinementList%5Btags%5D%5B0%5D=Procedural%20Generation&refinementList%5BreleaseYear%5D%5B0%5D=2021.
20. TOGELIUS, Julian; CHAMPANDARD, Alex J.; LANZI, Pier Luca; MATEAS, Michael; PAIVA, Ana; PREUSS, Mike; STANLEY, Kenneth O. Procedural Content Generation: Goals, Challenges and Actionable Steps. In: LUCAS, Simon M.; MATEAS, Michael; PREUSS, Mike; SPRONCK, Pieter; TOGELIUS, Julian (ed.). *Artificial and Computational Intelligence in Games* [online]. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, sv. 6, s. 61–75 [cit. 2022-04-24]. Dagstuhl Follow-Ups. ISBN 978-3-939897-62-0. ISSN 1868-8977. Dostupné z DOI: 10.4230/DFU.Vol6.12191.61.
21. *Dungeon* [online]. 2022 [cit. 2022-04-18]. Dostupné z: https://dictionary.cambridge.org/dictionary/english/dungeon.
22. GYGAX, Gary; ARNESON, Dave; MENTZER, Frank. *Dungeons & Dragons Set 1: Basic Rules* [Role-playing game]. TSR, 1983. ISBN 978-0880383387.
23. SMITH, Thomas; PADGET, Julian; VIDLER, Andrew. Graph-Based Generation of Action-Adventure Dungeon Levels Using Answer Set Programming. In: *Proceedings of the 13th International Conference on the Foundations of Digital Games* [online]. Malmö, Sweden: Association for Computing Machinery, 2018 [cit. 2022-04-24]. FDG '18. ISBN 9781450365710. Dostupné z DOI: 10.1145/3235765.3235817.
24. GELLEL, Alexander; SWEETSER, Penny. A Hybrid Approach to Procedural Generation of Roguelike Video Game Levels. In: *International Conference on the Foundations of Digital Games* [online]. Bugibba, Malta: Association for Computing Machinery, 2020 [cit. 2022-04-24]. FDG '20. ISBN 9781450388078. Dostupné z DOI: 10.1145/3402942.3402945.

25. BARON, Jessica R. Procedural Dungeon Generation Analysis and Adaptation. In: *Proceedings of the SouthEast Conference* [online]. Kennesaw, GA, USA: Association for Computing Machinery, 2017, s. 168–171 [cit. 2022-04-24]. ACM SE '17. ISBN 9781450350242. Dostupné z DOI: 10.1145/3077286.3077566.

26. VIANA, Breno M. F.; SANTOS, Selan R. dos. A Survey of Procedural Dungeon Generation. In: *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)* [online]. 2019, s. 29–38 [cit. 2022-04-24]. Dostupné z DOI: 10.1109/SBGames.2019.00015.

27. DORMANS, Joris. Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games. In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games* [online]. Monterey, California: Association for Computing Machinery, 2010 [cit. 2022-04-24]. PCGames '10. ISBN 9781450300230. Dostupné z DOI: 10.1145/1814256.1814257.

28. SCHELL, Jesse. *The Art of Game Design A Book of Lenses, Second Edition*. 2. vyd. A K Peters/CRC Press, 2014. ISBN 9780429169755.

29. WHITEHEAD, Jim. Spatial Layout of Procedural Dungeons Using Linear Constraints and SMT Solvers. In: *International Conference on the Foundations of Digital Games* [online]. Bugibba, Malta: Association for Computing Machinery, 2020 [cit. 2022-04-24]. FDG '20. ISBN 9781450388078. Dostupné z DOI: 10.1145/3402942.3409603.

30. JOHNSON, Lawrence; YANNAKAKIS, Georgios N.; TOGELIUS, Julian. Cellular Automata for Real-Time Generation of Infinite Cave Levels. In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games* [online]. Monterey, California: Association for Computing Machinery, 2010 [cit. 2022-04-24]. PCGames '10. ISBN 9781450300230. Dostupné z DOI: 10.1145/1814256.1814266.

31. ŘEHOŘEK, Ing. Tomáš. *Multiagentní systémy, Teorie her* [online]. 2017 [cit. 2022-05-09]. Dostupné z: https://courses.fit.cvut.cz/BI-ZUM/media/lectures/07-mas-games-v5.0.pdf.

32. GREEN, Michael Cerny; KHALIFA, Ahmed; ALSOUGHAYER, Athoug; SURANA, Divyesh; LIAPIS, Antonios; TOGELIUS, Julian. Two-Step Constructive Approaches for Dungeon Generation. In: *Proceedings of the 14th International Conference on the Foundations of Digital Games* [online]. San Luis Obispo, California, USA: Association for Computing Machinery, 2019 [cit. 2022-04-24]. FDG '19. ISBN 9781450372176. Dostupné z DOI: 10.1145/3337722.3341847.

33. MALÍK, Josef; SUCHÝ, Ondřej; TVRDÍK, Pavel; VALLA, Tomáš. *Základy grafů* [online]. 2021 [cit. 2022-05-09]. Dostupné z: https://courses.fit.cvut.cz/BI-AG1/media/lectures/bi-ag1-p1-handout.pdf.

34. MAWHORTER, Peter; MATEAS, Michael. Procedural level generation using occupancy-regulated extension. In: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games* [online]. 2010, s. 351–358 [cit. 2022-04-24]. Dostupné z DOI: 10.1109/ITW.2010.5593333.

35. SNODGRASS, Sam; ONTAÑÓN, Santiago. Learning to Generate Video Game Maps Using Markov Models. *IEEE Transactions on Computational Intelligence and AI in Games* [online]. 2017, roč. 9, č. 4, s. 410–422 [cit. 2022-04-24]. Dostupné z DOI: 10.1109/TCIAIG.2016.2623560.

36. TOGELIUS, Julian; PREUSS, Mike; BEUME, Nicola; WESSING, Simon; HAGELBÄCK, Johan; YANNAKAKIS, Georgios N. Multiobjective exploration of the StarCraft map space. In: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games* [online]. 2010, s. 265–272 [cit. 2022-04-24]. Dostupné z DOI: 10.1109/ITW.2010.5593346.

37. BALDWIN, Alexander; DAHLSKOG, Steve; FONT, Jose M.; HOLMBERG, Johan. Mixed-initiative procedural generation of dungeons using game design patterns. In: *2017 IEEE Conference on Computational Intelligence and Games (CIG)* [online]. 2017, s. 25–32 [cit. 2022-04-24]. Dostupné z DOI: 10.1109/CIG.2017.8080411.

38. UNITY TECHNOLOGIES. *Prefabs* [online]. 2018 [cit. 2022-05-08]. Dostupné z: https://docs.unity3d.com/Manual/Prefabs.html.

Obsah přiloženého média

- `readme.txt` — Stručný popis obsahu média
- `Magitech.pdf` — Text práce ve formátu PDF
- `MagitechLatex` — Adresář s textem práce ve formátu LaTeX
  - `Obrazky` — Adresář s obrázky použitými v práci
  - `text` — Adresář s textem práce
- `GameDesignDocument.docx` — Game design document, který vznikl před započetím vývoje
- `MagitechBuild` — Složka se spustitelnou verzí hry Magitech
  - `Project_Magitech.exe` — Spustitelný soubor
- `MagitechProject.7z` — Komprimovaný adresář s Unity projektem hry