VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

PROCEDURÁLNÍ GENEROVÁNÍ STRUKTUR TYPU DUNGEON

PROCEDURAL GENERATION OF DUNGEON TYPE STRUCTURES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MAREK ŠIPOŠ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Šipoš Marek**
Program: Informační technologie
Název: **Procedurální generování struktur typu dungeon**
Procedural Generation of Dungeon Type Structures
Kategorie: Web

Zadání:

1. Seznamte se s existujícími metodami procedurálního generování, zaměřte se na generování herních úrovní typu dungeon.
2. Analyzujte požadavky na struktury herních úrovní tohoto typu a po konzultaci s vedoucím navrhněte systém pro jejich generování na základě vstupních parametrů.
3. Navržený systém z předchozího bodu implementujte ve formě knihovny použitelné pro různé hry tohoto typu.
4. Vytvořte webovou aplikaci umožňující praktické využití vytvořené knihovny. Ověřte funkčnost knihovny a její využití demonstrujte na vhodném příkladě.
5. Zhodnoťte dosažené výsledky a diskutujte další možné pokračování tohoto projektu.

Literatura:

- Shaker, N., Togelius, J., Nelson, J. M.: Procedural Content Generation in Games, Springer International Publishing Switzerland, 2016. ISBN 978-3-319-42714-0.
- Korn, O., Newton, L.: Game Dynamics. Best Practices in Procedural and Dynamic Game Content Generation, Springer International Publishing AG, 2017. ISBN 978-3-319-53087-1.
- Baron, J. R.: Procedural Dungeon Generation Analysis and Adaptation, ACM Southeast Regional Conference, 2017, s. 168-171.
- Linden, v. d. R., Lopes, R., Bidarra, R.: Procedural generation of dungeons, IEEE Transactions on Computational Intelligence and AI in Games, 2014, sv. 9, č. 1, s. 78-89. URL:

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

Vedoucí práce: **Bartík Vladimír, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 31. července 2020
Datum schválení: 21. října 2019

Abstrakt

Hlavním cílem této bakalářské práce bylo navrhnout a vytvořit knihovnu pro procedurální generování dungeonů a webovou aplikaci pro její praktické využití. Knihovna nabízí konfigurační rozhraní, generování výstupu podle této konfigurace a souborovou podporu. Webová aplikace pak umožňuje získaný výstup vizualizovat. Pro implementaci byl zvolen jazyk Java.

Abstract

The main aim of this bachelor thesis was to design and develop a library for procedural generation of dungeons and web application for its practical use. The library offers a configuration interface, output generation according to this configuration and file support. The web application allows the visualization of the obtained input. The implementation was done in Java language.

Klíčová slova

procedurální generování, generátor, dungeon, místnost, chodba, knihovna, Java, Maven, roguelike, celulární automat, graf, delaunayova triangulace, minimální kostra

Keywords

procedural generation, generator, dungeon, room, corridor, library, Java, Maven, roguelike, cellular automaton, graph, delaunay triangulation, minimum spanning tree

Citace

ŠIPOŠ, Marek. *Procedurální generování struktur typu dungeon.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Procedurální generování struktur typu dungeon

Prohlášení

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

.....
Marek Šipoš
30. července 2020

Poděkování

Rád bych poděkoval panu Ing. Vladimíru Bartíkovi, Ph.D. za odborné vedení této práce a za rady, které mi v průběhu vypracovávání poskytl.

Obsah

Kapitola 1

Úvod

Cílem této bakalářské práce je vytvořit knihovnu pro procedurální generování struktur typu dungeon, které budou odpovídat požadavkům a parametrům vývojářů. Knihovna by tedy měla být navržena tak, aby podchytila co možná nejvíce možných scénářů, pro které si vývojář může přát vygenerovat herní úroveň. Funkčnost knihovny bude demonstrována na jednoduché webové aplikaci, která umožní uživatelům zadat vstupní konfiguraci a vygenerovanou strukturu poté vizualizovat.

Se stále se zvyšující oblíbeností sandboxových her a s rostoucími náklady na tvorbu herního obsahu roste také poptávka po technikách, jak tento obsah automaticky generovat. K tomuto účelu se využívají některé techniky procedurálního generování obsahu, díky kterým mohou vývojářská studia snížit objem práce výtvarníků a kreativních pracovníků a nabídnout hráčům variabilnější herní zážitek. Protože tyto techniky často využívají náhodnosti nebo neurčitosti, která je činí nepředvídatelnými, může být obtížné je zakomponovat do celkové koncepce hry tak, aby vytvořený obsah přesně odpovídal požadavkům vývojářů. Herní studia se proto snaží dosáhnout ideální rovnováhy mezi složitostí těchto algoritmů a přesností výsledků jimi navrácených.

Významným herním prvkem, na kterém se tyto techniky dají uplatnit, jsou herní úrovně, ve kterých se hra odehrává. Jejich vytváření je obvykle značně zdlouhavý a nákladný proces, který vyžaduje práci výtvarníků z mnoha oborů a musí být realizován tak, aby výsledné úrovně odpovídaly požadovanému tempu herního postupu hráče, měly správně rozmístěné vizuální a herní prvky, byly rozmanité a případně aby se dokázaly hráčovu postupu přizpůsobovat a měnit svůj charakter. Specifickým typem herních úrovní jsou tzv. dungeony (z angl. podzemní žalář nebo kobka). Dungeony se obvykle používají v dobrodružných hrách a hrách na hrdiny (RPG – Role-Playing Game) a jedná se o labyrinty, které se většinou skládají ze vzájemně propojených částí s vysoce strukturovaným herním obsahem. Jsou tedy vhodným kandidátem pro techniky procedurálního generování. Hráči se mohou sami rozhodnout, jakým způsobem budou dungeon prozkoumávat, jejich herní postup je však pozvolný a má gradující charakter, některé části jsou obtížnější, jiné se mohou například odemknout až po dokončení těch předchozích. Právě tímto strukturovaným postupem se dungeony odlišují například od otevřených sandboxových světů. Knihovna umožňující jejich automatické generování by tak mohla posloužit jako základ, na kterém by se tyto úrovně daly vytvářet efektivněji a s menším úsilím.

Text práce je rozčleněn na kapitoly. V první části kapitoly 2 je definováno procedurální generování obecně, druhá část obsahuje popis herní úrovně typu dungeon a v poslední části jsou představeny techniky procedurálního generování herního obsahu, zejména herních úrovní. Kapitola 3 výše představené techniky pak prezentuje v souvislostí s již existujícími

hrami. V další části kapitoly jsou prozkoumány existující knihovny pro procedurální generování dungeonů. Vlastní knihovna je navržena v kapitole 4. Důraz je kladen zejména na zadávání potřebných vstupů, které umožní výslednou strukturu přizpůsobit požadavkům vývojáře. V další části kapitoly je navržena výstupní struktura, která bude sloužit jako prostředek pro uložení vygenerovaných dat, na jejímž základě může vývojář herní úroveň vybudovat a dále ji modifikovat. Na závěr kapitoly je navržen způsob, jakým bude na základě těchto požadavků vytvořena výsledná struktura ve výstupním formátu s využitím procedurálního generování. Kapitola 5 popisuje pak samotnou implementaci knihovny. Knihovna je realizována v jazyce Java a nabízí uživateli objektově orientované rozhraní pro její intuitivní využití. Kapitola 6 se věnuje návrhu a tvorbě webové aplikace, která knihovnu využívá a vizualizuje její výstup. V kapitole 7 je prověřena základní funkčnost knihovny a vlastnosti, které vykazuje. Knihovna je zde také porovnána s již existujícími řešeními. V poslední kapitole 8 je práce shrnuta, jsou zhodnoceny dosažené výsledky a navrženo pokračování a rozšíření knihovny o další funkce.

Kapitola 2

Procedurální generování herního obsahu

Tato kapitola pojednává o technikách procedurálního generování. Začátek kapitoly se věnuje procedurálnímu generování obecně a v kontextu herních úrovní typu dungeon. Následně je definována herní úroveň typu dungeon. Největší část kapitoly zaujímají nejvyužívanější techniky procedurálního generování pro herní obsah, zejména pro herní úrovně, ale i jiné techniky, které mohou být v tomto kontextu užitečné nebo zajímavé.

2.1 Definice a historie procedurálního generování

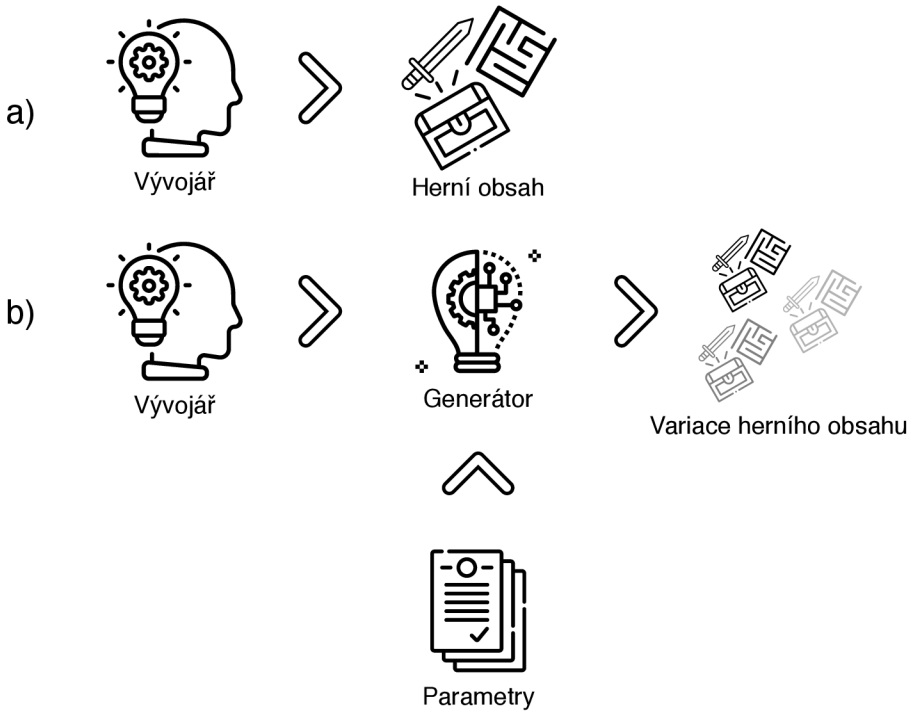
Tato část obsahuje úvod do procedurálního generování, je vysvětlen samotný pojem a ten je poté zasazen do kontextu generování herních úrovní typu dungeon.

2.1.1 Procedurální generování jako pojem

V kontextu výpočetní techniky představuje procedurální generování tvorbu dat pomocí algoritmů (a nikoliv ručně). Obecně se tedy jedná o procedurální generování obsahu (zkráceně PCG – Procedural Content Generation). Obsah v kontextu počítačových her pak může znamenat úrovně, textury, úkoly, nepřátelé apod. Slovo „procedurální“ v tomto případě znamená, že je obsah vytvářen pomocí počítačových procedur (nebo též funkcí, metod, rutin)[19]. Procedurální generátor může mít ve hrách mnoho podob a využití:

- Metoda, která na základě vstupních parametrů (např. číslo úrovně a požadovaná velikost) vytvoří náhodné bludiště určité obtížnosti
- Dohledový systém, který analyzuje oblíbené a často používané zbraně hráče a vygeneruje mu vylepšené verze těchto zbraní, které bude moci během hraní získat
- Funkce, která generuje náhodnou hádanku typu „puzzle“ na určitém místě ve statickém a předem navrženém aztéckém chrámu
- Mechanismus, který pro rozsáhlý kontinent automaticky vygeneruje vegetaci podle podnebí
- Grafický systém, který náhodně generuje 3D modely obydlí domorodých kmenů

Procedurální tvorba obsahu může být pro vývojáře časově i finančně výhodná, protože stačí *jednou* vytvořit generátor, který následně vyprodukuje *neomezené množství* variací na herní obsah podle zadaných parametrů (viz obrázek 2.1). Největší výzvou při tvorbě takového generátoru je, aby jím vytvářený obsah splňoval požadavky na kvalitu, věrnost a variabilitu. Protože je však možnost přeměnit konečné úsilí v potenciálně nekonečný obsah velmi lákavá, procedurální generování je předmětem neustálého zkoumání.



Obrázek 2.1: Porovnání způsobů tvorby herního obsahu, a) herní obsah musí tvořit vývojář ručně, b) vývojář vytvoří generátor, který generuje neomezené množství variací herního obsahu podle zadaných parametrů, pro obrázek byly použity materiály ze stránky https://www.flaticon.com/, autoři: Freepik, Kiranshastry, fps web agency, geotatah.

2.1.2 Procedurální generování dungeonů

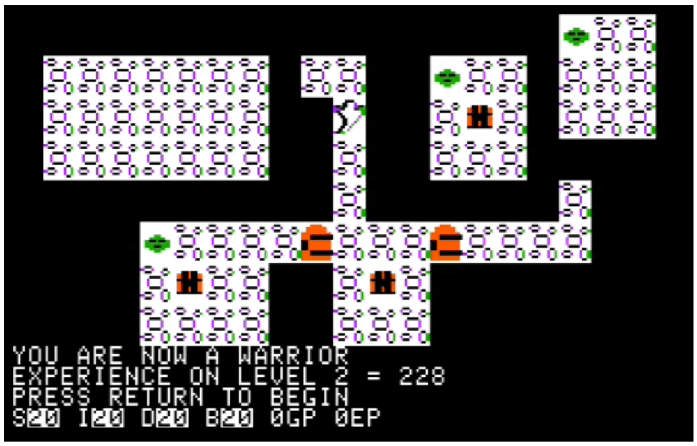
Velmi významnou částí procedurálního generování je generování herních úrovní typu dungeon. Dungeony jsou prostory skládající se z různě tvarovaných místností a chodeb, ve kterých hráč bojuje proti různým nepřátelům, řeší (nejčastěji logické) hádanky a nachází poklady. Svým tvarem a uspořádáním často připomínají labyrint. Herní úroveň typu dungeon je přesněji a podrobněji definována v části 2.2. Dungeony je možné generovat mnoha způsoby: pomocí celulárních automatů, generativních gramatik, na základě specifikovaných omezení a pravidel, pomocí genetických algoritmů atd. Mnoho z těchto metod vychází z výzkumů, které se však do praxe promítají velmi zřídka. Tyto výzkumy přesto nesou nezanedbatelný potenciál – umožňují generovat dungeony abstraktně jako topologickou síť bez geometrie, přizpůsobovat generátor dynamicky hráčovu postupu či chování, vytvářet úrovně „na míru“ pro definovanou sadu herních úkonů nebo dokonce generovat dungeony pouze zadáním požadovaných parametrů zábavy, obtížnosti a velikosti.

2.1.3 Historie dungeonů v počítačových hrách a žánr Roguelike

Před definováním herní úrovně typu Dungeon je vhodné podívat se alespoň stručně na historii žánru, který stál za jejich vznikem. Tradice her, ve kterých hráč prochází temné úrovně a bojuje v nich proti nepřátelům sahá až do roku 1974, kdy Gary Gygax a Dave Arneson vydali velmi úspěšnou stolní hru *Dungeons and Dragons*. Tato hra obsahující prvky náhody měla velký vliv na následný vývoj počítačových her.

Historicky první dochovanou hrou na hrdiny, ve které hráč procházel dungeony, bojoval s příšerami a vylepšoval svou postavu, je hra nazvaná jednoduše *The Dungeon* z roku 1975. Tato hra byla vytvořena pro výukový počítač PLATO provozovaný na Univerzitě Illinois v Urbana Champaign pod názvem *pedit5* a studenti si ji mohli zahrát na kterémkoliv ze 150 terminálů (i přes nevoli univerzitního systémového správce, který ji údajně často odstraňoval). Ačkoliv byly herní úrovně v této hře fixní, obsahovaly náhodně generované nepřátele.

Velikým milníkem v kontextu náhodně generovaných dungeonů byla hra *Beneath Apple Manor*. Tato hra z roku 1978 byla původně vytvořena pro sériově vyráběný počítač Apple II, takže si ji mohla zahrát širší veřejnost. V letech 1982 a 1983 se pak dočkala přepracování pro MS-DOS a Atari. Cílem hráče bylo projít procedurálně generované dungeony a najít v nich zlaté jablko. Po cestě sbíral různé předměty a bojoval s nepřáteli. Grafika hry byla velmi jednoduchá především kvůli omezeným možnostem systému Apple II.



Obrázek 2.2: Ukázka ze hry *Beneath Apple Manor*, která obsahovala první náhodně generované dungeony.

Koncem 70. let byla pro Unixové systémy vydána knihovna *curses*, která umožňovala vývoj textových uživatelských rozhraní. Důležitou funkcí byla možnost vkládat znaky na libovolné místo na obrazovce (podle souřadnic). Tato knihovna byla snadno použitelná a dala vzniknout prvním počítačovým hrám s ASCII grafikou. V roce 1980 vznikla hra *Rogue*, která odstartovala žánr počítačových her nazvaný Roguelike (v překladu „podobné hře Rogue“). Vznikalo mnoho her, které se více či méně podobaly původní hře *Rogue*. Spousta her přebírala pouze některé aspekty Roguelike her a proto vznikaly další dílčí žánry, které však nemají přesně vymezené hranice (např. Roguelite nebo Roguelike-like)[5]. V roce 2008 byla v Berlíně uspořádána konference, jejímž výsledkem byl pokus o definování žánru

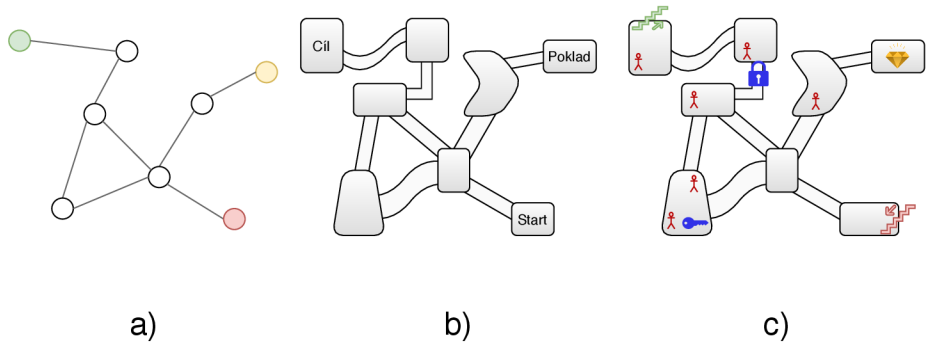
Roguelike. Vznikla tzv. Berlínská interpretace[1], která obsahuje základní charakteristiky tohoto žánru a přiřazuje jim různou důležitost.
Původní koncepce žánru Roguelike z 80. a 90. let se vyznačovala těmito prvky:

- Náhodně generované úrovně – místnosti, předměty, příšery a další prvky v náhodně generovaných úrovních zvyšujících její opakovanou hratelnost.
- Tahová hra – každý příkaz představuje jeden pohyb nebo akci. Na času nezávisí, hráč si může další tah v klidu promyslet.
- Jeden život – hráč v Roguelike hře má pouhý jeden život. Musí proto postupovat opatrně. Pokud umře, musí začít od první úrovně, jeho herní zážitek se však bude díky procedurálnímu generování lišit.
- ASCII grafika – hry z té doby byly reprezentovány jednoduchou ASCII grafikou, např. postavu hráče často reprezentoval znak zavináče (`'@'`).

Dnešní počítačové hry mohou být do tohoto žánru zařazeny, ačkoliv splňují pouze některá z pravidel. Vznikly různé variace a mutace na žánr Roguelike, jádro her však zůstává podobné – zkoumání (většinou) podzemních prostorů, boj proti nepřátelům, vylepšování postavy a stále jedinečný herní zážitek díky procedurálnímu generování.

2.2 Definice herní úrovně typu dungeon

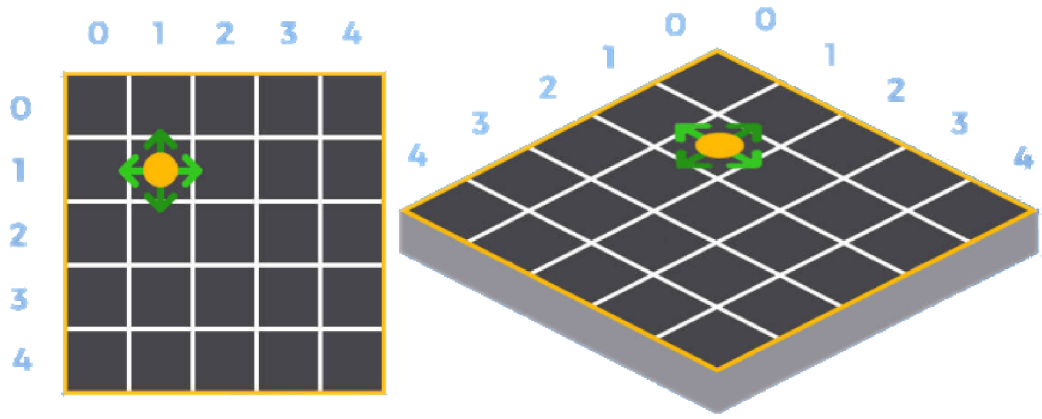
Dungeony jsou velmi častým a populárním prvkem mnoha her na hrdiny (RPG – Role Playing Game) a především Roguelike her. Z angličtiny se dungeon do češtiny překládá jako žalář, katakomba, kobka nebo hladomorna. Jedná se tedy o temný, uzavřený prostor složený z místností a chodeb. Svou strukturou a rozložením mohou dungeony připomínat labyrint. Chodby mohou být různě široké, mohou zatáčet, vytvářet smyčky atd. Důležité je, aby propojovaly všechny místnosti dungeonu a umožňovaly tak jeho průchodnost. Místnosti mohou být různých tvarů i velikostí a mohou obsahovat další místnosti či chodby. Za dungeony jsou často považovány i jeskynní komplexy, které se vyznačují méně strukturovaným tvarem a širšími chodbami, které mohou být zároveň místnostmi.



Obrázek 2.3: Postup při konstrukci dungeonu, a) topologická struktura obsahující graf s uzly (místnostmi), některé se speciálním významem, b) přiřazení geometrické podoby (rozměrů) chodbám a místnostem, c) rozmístění herních prvků dále upravujících postup hráče.

[1]Berlínská interpretace – http://roguebasin.roguelikedevelopment.org/...

Dungeony v počítačových hrách často obsahují začátek a jeden nebo více konců, které mohou vést do dalších úrovní, a v chodbách a místnostech mohou číhat nepřátelé nebo poklady, případně různé hádanky, spínače, dveře, pasti a další prvky. Dungeony jsou vysoce strukturované a uspořádání prvků v nich obsažených často podléhá různým pravidlům, které souvisí především s hratelností – například chodba plná nepřátel s pokladem na konci nebo místnost s cílem a silným protivníkem, která je v největší vzdálenosti od začátku dungeonu. Ačkoliv se hráč může po dungeonu volně pohybovat, jeho dosah může být omezen silnými nepřáteli nebo například zamčenými dveřmi. Procedurální generování dungeonů tedy znamená, že je nutné vytvořit topologickou strukturu, přidělit jí geometrickou podobu a rozmístit prvky týkající se postupu (viz obrázek 2.3)[17].



Obrázek 2.4: Porovnání klasického zobrazení shora (vlevo) oproti isometrickému zobrazení (vpravo). Správná transformace a speciální sada grafických dlaždic tvoří iluzi třetího rozměru. Převzato z https://themindstudios.com/....

Dungeony mohou být dvou i trojrozměrné, mohou být předem vygenerované a omezené nebo mohou být generovány dynamicky „za chodu“ a mohou být tedy potenciálně nekonečné. Dungeony mohou být vykresleny pohledem shora (z ptačí perspektivy), ale častěji se používá některá forma axonometrie[2], nejčastěji isometrické zobrazení. Pomocí správně transformovaných dlaždic a speciálně navržených grafických elementů tak mohou dungeony vypadat jako 3D, ačkoliv jsou pouze 2D (viz obrázek 2.4).

2.3 Techniky generování herního obsahu

V této části jsou probrány některé základní techniky generování herního obsahu, především dungeonů. V praxi se mohou využívat kombinace či modifikace různých technik.

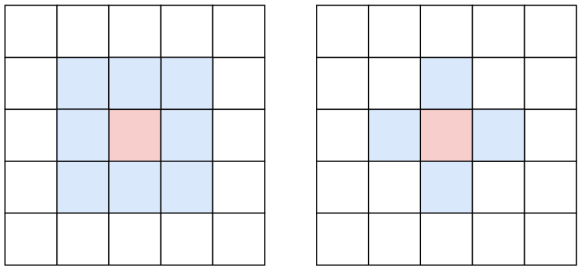
2.3.1 Celulární automaty

Celulární automat je diskrétní matematický model, který znázorňuje chování nelineárního systému založeného na jednoduché sadě podmínek. Motivací k vytvoření prvních celulárních automatů v polovině 20. století byla snaha John von Neumanna o vytvoření sebereplikujícího robota, který by dokázal sestavit druhého robota jako svou identickou kopii. Jeho kolega, Stanislaw Ulam, navrhl využít pro tvorbu takového modelu diskrétní přístup. Ten se mu dříve osvědčil při studiích růstu krystalů, kde jako svůj model používal diskrétní

[2] **axonometrická projekce** je jednoduchý způsob promítání prostorových struktur do roviny

prvky v mřížce. Následně společně podobný systém použili k výpočtům pohybů tekutin. Tato metoda spočívala v tom, že se hmota kapaliny rozdělila do diskrétních jednotek, kde se pohyb každé z nich počítal na základě jejich sousedů. Tím vznikl první z celulárních automatů. V současnosti se celulární automaty používají v různých oborech, především v již zmíněné fyzice, ale také v biologii a chemii, často v kombinaci s jinými systémy. Zajímavé je, že celulární automaty v určité přenesené formě nacházejí využití také v informatice a umění. Ve světě počítačů se používají jako základ pro simulace různých systému reálného světa (např. doprava), a v některých specifických případech také při procedurálním generování. Ve světě umění se pak používají k tvorbě abstraktních tvarů a vzorů.

Problematika celulárních automatů je dnes velice rozsáhlá a jejich typy lze dělit z hlediska různých kritérií (dimenzí, stavů, pravidel atd). V této práci se spokojíme s tím, že celulární automat lze definovat jako mřížku buněk, které jsou uspořádány v n-rozměrném prostoru (tzv. *celulární prostor*), přičemž všechny buňky mají stejný tvar (např. čtverec nebo šestiúhelník). Každá buňka v určitém diskrétním, časovém bodě nabývá jednoho z množiny možných stavů. Velmi důležitou součástí definice celulárního automatu je počáteční stav buněk v nulovém čase ($t = 0$). Stav buněk v následujícím časovém bodě $(t + 1)$ (též nazývaném další „generace“ buněk) lze vypočítat použitím příslušného pravidla na aktuální stav každé z nich. Souhrnu možných pravidel se říká *přechodová funkce*. Vstupem přechodové funkce je stav vyhodnocované buňky a stavy sousedních buněk a výstupem je konkrétní výsledný stav (deterministická funkce) nebo množina možných stavů (nedeterministická funkce). Pokud je funkce nedeterministická, obvykle bývají vrácené stavy doplněny o rozdělení pravděpodobnosti. U každého celulárního automatu je nutné určit typ použitého sousedství, tedy které sousední buňky se mají brát při vyhodnocování stavu v úvahu[15].



Obrázek 2.5: Dvě nejčastěji používaná sousedství u celulárních automatů – vlevo Moorovo a vpravo von Neumannovo. Modře znázorněné buňky se zohledňují při vyhodnocování následného stavu buňky uprostřed.

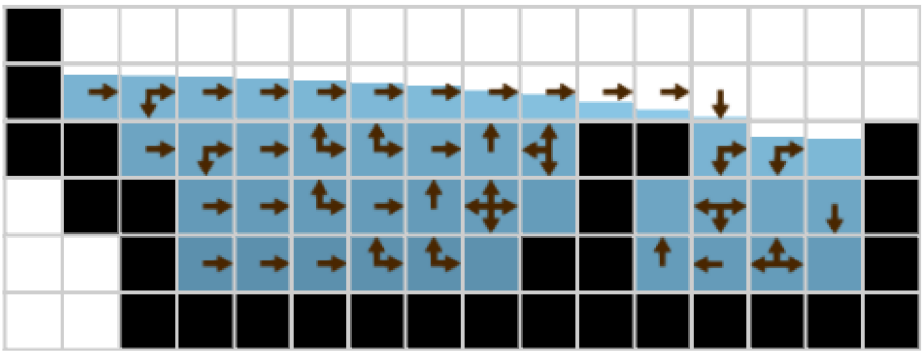
Po několika generacích lze pozorovat, že v závislosti na použitých pravidlech, stavech, sousedství a dalších parametrech vykazují buňky v mřížce (systém) určité rysy chování. Pro celulární automaty je typická jejich nelinearita a nepředvídatelnost. Často se vyhledávají automaty, které po spuštění vykazují chování, které je užitečné pro daný specifický účel.

V kontextu procedurálního generování obsahu do počítačových her se celulární automaty používají v některých komerčních titulech například k simulování šíření ohně nebo proudění kapalin. Použití celulárních automatů může být v tomto případě výhodné, protože taková simulace je mnohem méně náročná na výpočetní čas než analytická řešení, přičemž je pro herní účely dostatečně přesná.

Zajímavé řešení simulace toku kapaliny publikoval ve svém článku[8] Jon Gallant, který jej využil v systému Unity[3]. Toto řešení dělí herní plochu na čtvercovou mřížku, kde každá

[3]Unity je multiplatformní systém pro vývoj 2D a 3D aplikací. https://unity.com

buňka mřížky může být v jednom ze tří stavů – prázdná (nulový objem), kapalná (nenulový objem) a stěna. Během iterace se buňky s nenulovým objemem vyhodnocují a podle objemu a stavu sousedních buněk se objem kapaliny mezi buňkami přesouvá (v tomto případě bylo použito von Neumannovo sousedství, které minimalizuje „rozšiřování“ kapalin ve spádu, např. u vodopádů). Toto chování je ilustrováno na obrázku 2.6. Kapalina přirozeně proudí



Obrázek 2.6: Simulace kapaliny pomocí celulárního automatu. Šipky graficky znázorňují pravidlo použité pro danou buňku. Převzato z [8].

směrem dolů (simuluje vliv gravitace), případně se rozlévá do stran. Aby vypadala simulace kapaliny přirozeně, musí simulovat také hydrostatický tlak. Protože je účelem simulace její použití ve hrách, je důležitá především rychlost výpočtu. Z tohoto důvodu zavádí toto řešení zajímavý aspekt, kdy může buňka obsahovat o trochu více kapaliny, než je běžné maximum. Takto „stlačená“ kapalina poté vyhodnocuje i pravidlo, které interaguje s buňkou o jednu pozici výše, a snaží se kapalinu směrem vzhůru předat.

Toto řešení pro simulace kapalin má výhodu v tom, že jej lze parametrizovat podle požadovaných fyzikálních vlastností. Vhodným nastavením počtu iterací v herní smyčce lze dosáhnout požadovaného poměru mezi použitým výpočetním výkonem a rychlostí simulace. Nevýhodou je nutnost procházení všech buněk mřížky a fakt, že toto řešení kapalinu rozděluje na diskrétní jednotky. Pro dosažení vyšší přesnosti musí dojít ke zmenšení velikosti buněk, ale zároveň ke kvadratickému zvýšení jejich počtu a tím i složitosti algoritmu.

Některé výzkumné práce navrhují použití celulárních automatů ke generování terénu. Ty jsou však velice nepředvídatelné, nelze je dobře parametrizovat a proto se v praxi nepoužívají.

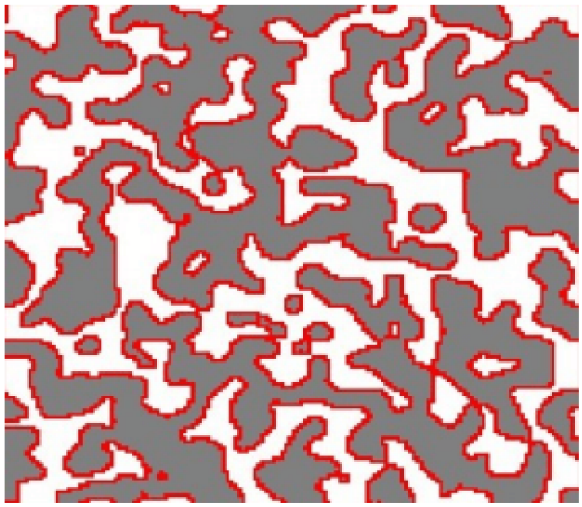
Dobře známy jsou celulární automaty, které vytvářejí struktury podobající se jeskynním komplexům. V roce 2010 proběhla v Kalifornii konference o procedurálním generování obsahu do počítačových her, kde byla představena metoda generování nekonečných jeskynních 2D komplexů pomocí celulárního automatu[11]. Herní prostor se v této metodě generuje po jednotlivých mřížkách, z nichž každá má 50x50 buněk. Díky tomu je možné mapu generovat postupně, jak se hráč pohybuje. Každá buňka může být v jednom ze tří stavů: skála (neprůchozí), podlaha (průchozí) nebo stěna (na rozhraní skály a podlahy). Na začátku algoritmu se všechny buňky nacházejí ve stavu podlahy a určité procento z nich je náhodně změněno do stavu skály. Při iteracích se u každé z buněk vyhodnocuje její okolí tak, že se sečte počet buněk v Moorově sousedství, které jsou ve stavu skály. Pokud součet překročí určitou hodnotu, v další iteraci se buňka změní na skálu, v opačném případě se změní na podlahu. Po provedení požadovaného počtu iterací se buňky na rozhraní skály a podlahy změní na stěny. Tato metoda je zajímavá tím, že nabízí sadu jednoduchých parametrů, pomocí kterých lze chování automatu modifikovat a získat tak jiné výsledky.

- r – procento buněk, které jsou na začátku ve stavu skály
- n – počet prováděných iterací, více iterací má za následek širší jeskyně
- T – počet sousedních buněk, které musí být ve stavu skály, aby byla buňka vyhodnocena také jako skála
- M – maximální vzdálenost buněk Moorova sousedství (obvykle 1)

Vliv těchto parametrů na výsledné struktury však nelze jednoznačně předpovědět. Pokud se však použití stejné parametry a stejné náhodného semínko[4], je možné vygenerovat stejně vypadající jeskyni. Toho se využívá v případě, kdy se hráč vrátí zpět do dřívější části jeskyně.

Stejným způsobem, jakým byla vygenerována první mřížka, se vygenerují také další čtyři sousední mřížky – pouze se použijí jiná náhodná semínka. Po vygenerování je potřeba zkontrolovat průchodnost jeskyní mezi první a každou ze sousedních mřížek. Uzavřené shluky buněk ve stavu podlahy jsou propojeny tunely o definované šířce tak, že se vyberou dvě buňky ve stavu podlahy nejblíže k hranicím mřížky. Na výsledných 5 mřížek se následně použije dalších n iterací celulárního automatu, které tunely vyhladí.

Autoři ve své práci zkoumali i výpočetní náročnost algoritmu. Vytvoření jeskyně skládající se z 9 mřížek na obrázku 2.7 jim na počítači se systémem Windows 7, procesorem Intel Pentium M 1.73 GHz a 1.50 GB RAM trvalo 349 milisekund, proto algoritmus vyhodnotili jako vhodný pro generování v reálném čase.



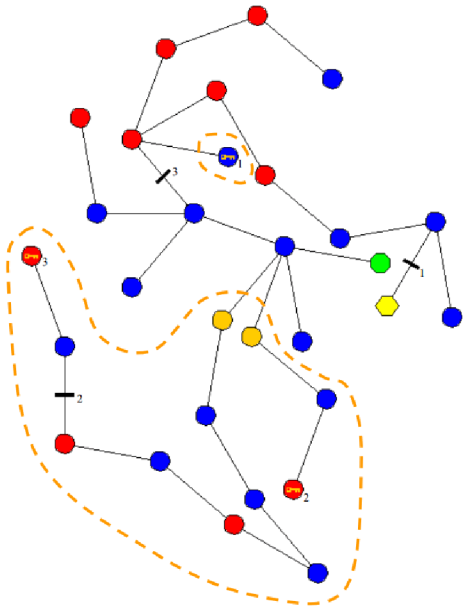
Obrázek 2.7: Jeskyně složená z 9 mřížek o velikosti 50x50 buněk a vygenerovaná pomocí celulárního automatu. Šedě znázorněné jsou skály, bíle podlahy a červeně stěny. Použité parametry: $r = 50\,\%$, $n = 4$, $T = 13$, $M = 2$. Převzato z [11].

Celulární automaty mají potenciál pro využití v počítačových hrách. Reálně se používají pro jednoduché simulace fyzikálních jevů a ačkoliv nejsou kvůli nedostatečným možnostem jejich ovládání příliš vhodné pro procedurální generování, lze je použít v kombinaci s dalšími metodami.

[4]Náhodné semínko (random seed) je číslo, které garantuje, že s použitím jiného semínka pseudonáhodný generátor vytvoří jinou sekvenci náhodných čísel.

2.3.2 Generativní gramatiky

Generativní gramatika je formální systém generující určitou množinu řetězců nad určitou abecedou. Pojem generativní gramatika v 50. letech 20. století poprvé definoval „otec moderní lingvistiky“, Američan Noam Chomsky, a sice jako gramatiku s pravidly, která umožňují vytvářet kombinace slov, ze kterých vznikají gramaticky správné věty v daném jazyce. Formálně je generativní gramatika čtveřice $G = (V, V_T, P, S)$, kde V je konečná množina neterminálních symbolů, V_T je konečná množina terminálních symbolů, S je tzv. výchozí symbol (platí, že $S \in V - V_T$) a P je konečná množina dvojic (nazývaných přepisovací pravidla) tvaru (u, v) kde $u \in (V - V_T)^*, v \in V^*$, přičemž u je neprázdný řetězec. Tato pravidla se zapisují v podobě $u \to v$ a čtou jako „přepiš u na v“. Postupnou aplikací přepisovacích pravidel na počáteční symbol S a z něj vzniklé řetězce se vytvářejí derivace (též posloupnosti řetězců nebo slovní formy), až dokud v řetězci nezůstanou pouze terminální symboly. Jazyk definovaný gramatikou $G = (V, V_T, P, S)$ je pak množina $L(G) = \{w : w \in V_T^*, S \Rightarrow^* w\}$[16]. Na základě generativních gramatik vznikly další druhy gramatik, například gramatiky grafů nebo tvarů, které se liší svou lingvistickou definicí, kdy např. jako terminální symboly používají uzly nebo tvary.



Obrázek 2.8: Graf herní úrovně vygenerovaný pomocí generativní gramatiky a parametrů pro velikost, obtížnost a zábavu. Převzato z [4].

David Adams ve své práci[4] představil systém, který generuje topologické celky ve formě grafů na základě číselných parametrů definujících hratelnost. Výsledné grafy jsou tvořeny uzly představujícími místnosti a hranami, které reprezentují propojující chodby (viz obrázek 2.8). Grafy jsou doplněny tzv. aktivátory (oblasti, kterými musí hráč projít, např. přepínač nebo klíč), dveřmi, které se pomocí aktivátorů otevírají, nepřáteli a odměnami. Generování grafu probíhá tak, že hledací algoritmus analyzuje výsledky přepisovacích pravidel a vybírá v danou chvíli to nejvhodnější. Toto chování (tzv. strategie) je pevně zakódované a neměnné. Pro tvorbu jiných typů úrovní by bylo nutné vytvořit novou sadu pravidel. Systém přijímá tři parametry, které souvisí s hratelností:

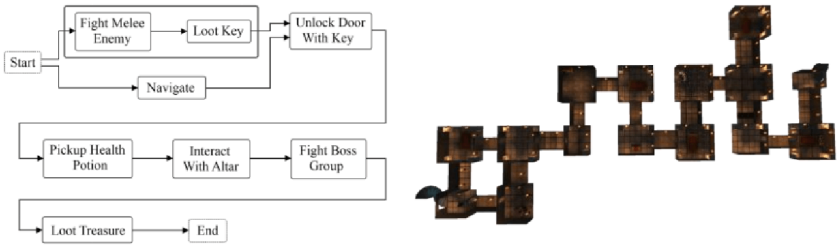
- velikost – průměrná vzdálenost od začátku do konce a počet objektů v grafu

- obtížnost – počet a koncentrace nepřátel, počet aktivátorů a dveří, rozsáhlost a rozvětvenost grafu
- zábava – variabilita úrovní, poměr velikosti úrovní a počtu objektů

Práce však nenabízí žádné řešení pro převod grafu do geometrické podoby. Autor v závěru přiznává, že převod grafu do podoby fyzické herní úrovně by byl velmi obtížný.

Zajímavý princip generování dungeonů na základě úkolů ve hře představil ve své práci Joris Dormans[7]. V této práci jsou použity grafové gramatiky, kde postupnou aplikací přepisovacích pravidel vzniká orientovaný graf mise, což je posloupnost úkolů, které musí hráč vykonat. Vygenerovaná kostra grafu je poté asociována s pravidly tvarové gramatiky. Rekurzivní aplikací těchto pravidel a úpravou velikostí tvarů pak vzniká herní úroveň, která je vhodná pro vygenerovaný graf mise. Výhodou je, že grafovou a tvarovou gramatiku lze používat nezávisle na sobě, to znamená, že pro graf mise lze vytvořit neomezené množství geometrických úrovní, a vygenerovanou geometrickou úroveň lze použít pro různé mise. Nevýhodou je, že vytvořené gramatiky jsou velmi specifické, jejich tvorba je složitá a neflexibilní. Autor do své práce nezahrnul žádné parametry, veškeré chování definoval přepisovacími pravidly.

Odpovědí na tento problém může být práce[12] představená v roce 2013 na konferenci Artificial Intelligence and Interactive Digital Entertainment. V ní vývojář na začátku určuje tzv. herní slovník akcí, který se skládá z dvojic **sloveso – obsah**, například **získat** (sloveso) **klíč** (obsah). Tyto akce jsou složené a skládají se z posloupnosti podakcí. Jednotlivé posloupnosti jsou v disjunktním vztahu – splněním kterékoliv posloupnosti se splní i akce samotná. Posloupnosti tedy představují předpisy, jakými lze akce splnit. Například akci získání klíče lze provést takto: otevřít truhlici → sebrat klíč *NEBO* porazit nepřítele → prohledat jeho tělo. Na základě vývojářem definovaného herního slovníku pak popisovaný algoritmus vytváří grafové gramatiky, které generují grafy akcí hráče podobné grafům v práci Jorise Dormanse [7]. Generování herního prostoru na základě vygenerovaných grafů je pak na vývojáři samotném. Autoři demonstrovali způsob, jak takový generátor vytvořit, na skutečné hře (Dwarf Quest), viz. obrázek 2.9.



Obrázek 2.9: Ukázka vygenerované herní úrovně (vpravo) na základě grafu herních úkonů (vlevo). Převzato z [12].

Moderní výzkum generativních gramatik v kontextu procedurálního generování se převážně týká generování struktur herních úrovní odpovídajících požadovaným parametrům hratelnosti. Tento přístup je jedinečný v tom, že vývojář nemusí řešit *způsob*, jakým bude herní úroveň vygenerována. Stačí mu charakterizovat herní prvky, jako jednotlivé lokace nebo akce, a generátor vytvoří odpovídající graf herní úrovně. Nevýhodou je složitá tvorba takových generátorů. Jakákoliv změna v herních prvcích může přepisovací pravidla dramaticky ovlivnit. Výzkumné práce zabývající se použitím generativních gramatik ke generování

herních úrovní navíc jejich fungování často demonstrují na velmi jednoduchých příkladech. Jimi uváděné parametry nebo pravidla jsou pro použití ve skutečných počítačových hrách často nedostačující. Jejich potenciál pro komerční využití je tudíž předmětem dalšího zkoumání. Použití grafových struktur jako obecné reprezentace místností se však ukazuje jako velmi užitečné a ve skutečnosti se také široce využívá. Vývojáři ale stále musí řešit problém, jak podle této abstraktní grafové reprezentace vytvořit geometrickou podobu herních úrovní.

2.3.3 Prohledávací metody

Prohledávací metody se pomocí evolučního nebo jiného stochastického vyhledávacího či optimalizačního algoritmu snaží najít řešení s požadovanými vlastnostmi. Předpokladem pro jejich využití je, že ve stavovém prostoru takové řešení skutečně existuje. Algoritmus iterativně prohledává stavový prostor tak, že postupně upravuje aktuální řešení. Pokud se nové řešení vlastnostmi přibližuje hledanému řešení, je zachováno, v opačném případě je zahozeno. Prohledávací metoda se neobejde bez základních komponent:

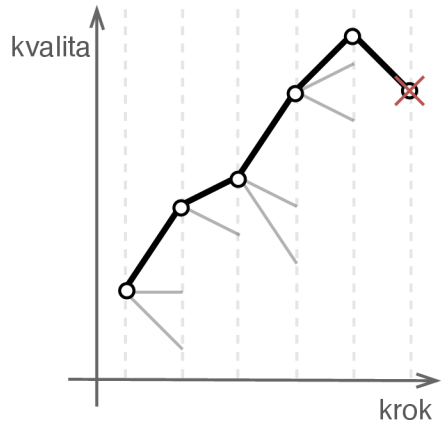
- Vyhledávací/optimalizační algoritmus – jádro prohledávací metody. Představuje způsob, jakým se metoda postupně přibližuje k výsledku.
- Reprezentace obsahu – jakým způsobem lze jednotlivá řešení popsat. To zahrnuje množinu všech vlastností, které jsou pro algoritmus důležité a „uchopitelné“. Může se jednat například o pole hodnot nebo graf.
- Jedna nebo více vyhodnocovacích funkcí – tato funkce dokáže převést určitou vlastnost obsahu na číselnou podobu, aby bylo možné jednoznačně určit, která je při posuzování výsledků lepší nebo horší.

Jedním z dobře známých vyhledávacích algoritmů je stochastický **evoluční algoritmus**. Tento algoritmus je inspirován principem Charlese Darwina o přirozeném výběru. V tomto algoritmu jsou vlastnosti a chování jedinců popsány pomocí *chromozomů*, což jsou nejčastěji řetězce nul a jedniček popisujících pozici jedince v prostoru možných řešení. Chromozom je složen z *genů*, což jsou nejmenší, dále nedělitelné jednotky. *Populace* je určitá konečná množina jedinců v dané generaci. *Fitness hodnota* představuje číselné hodnocení kvality jedince a je výstupem vyhodnocovací funkce. V každé generaci dojde k *selekci* jednotlivců (též nazývaných kandidáti) a zůstanou zachováni ti, kteří mají nejlepší hodnocení. Nejlépe hodnocení jedinci se mohou reprodukovat, nejhůře hodnocení jedinci jsou z populace vyřazeni. Pro selekci se běžně používá metoda, která lépe hodnoceným jedincům přiřadí větší pravděpodobnost, že budou vybráni k reprodukci pro další generaci. Samotná reprodukce, nebo též *křížení*, pak představuje výměnu částí chromozomů mezi dvěma vybranými jedinci (rodiči). Potomky se pak mohou stát oba dva nebo jeden z nich. Poslední fází je *mutace*. U potomků se projde celý chromozom a s velmi malou pravděpodobností dojde ke změně genu. K tomu dochází proto, aby byly do nové generace zaneseny vlastnosti, kterými nemuseli disponovat žádní jedinci z předchozí generace[18].

Zmíněný evoluční algoritmus dokáže vyhledávat ideální řešení bez nutnosti prohledávat celý stavový prostor. Nejčastějším problémem je však výběr vhodné reprezentace chromozomů a nalezení vyhodnocovací funkce, která dokáže správně zachytit požadované vlastnosti. Jednotlivé kvalitativní vlastnosti je však často obtížné reprezentovat jednoduchými číselnými hodnotami, proto se v praxi používají pokročilé algoritmy, které dokážou využít

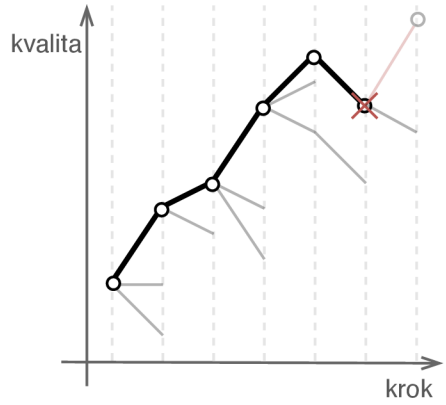
více vyhodnocovacích funkcí současně (například elitistický nedominovaný třídící genetický algoritmus NSGA-II).

Daleko jednodušší metodou lokálního prohledávání je metoda Hill-climbing (metoda lezení do kopce). Tato metoda využívá taktéž funkci pro ohodnocení kvality a jako další řešení vybírá nejlépe ohodnocenou možnost. Pokud je však ohodnocení kvality následného řešení horší než u aktuálního řešení, algoritmus končí a výsledkem je aktuální řešení (viz obrázek 2.10).



Obrázek 2.10: Znázornění algoritmu Hill-climbing. V jednotlivých krocích může být více možných následných řešení s různým ohodnocením kvality. Vybráno je vždy to nejkvalitnější. Pokud nelze v dalším kroku dosáhnout kvalitnějšího řešení, než je řešení aktuální, algoritmus končí.

Další využívanou stochastickou metodou lokálního prohledávání je **simulované žíhání**. Tato metoda vznikla za účelem, aby vyřešila problémy s lokálními extrémy, které vedly k častým neúspěchům metody Hill Climbing (viz obrázek 2.11). Jak již název napovídá,

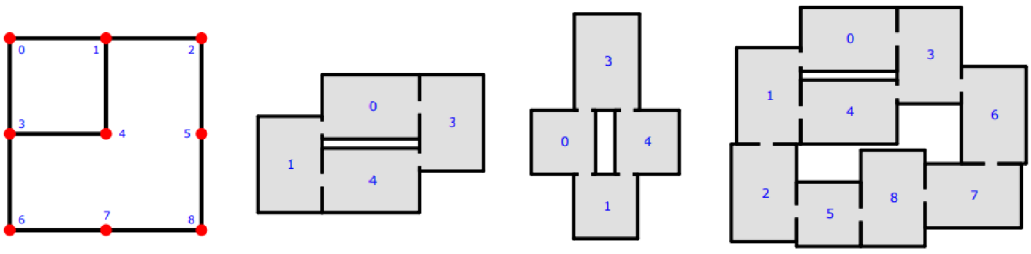


Obrázek 2.11: Znázornění algoritmu Hill-climbing, který byl ukončen předčasně z důvodu lokálního extrému, a tudíž se mu nepodařilo nalézt řešení s nejlepším ohodnocením.

metoda vychází z metody tepelného opracování kovů, kdy se kovy nejdříve zahřejí (čímž se atomy uvolní ze svých lokálních minim a začnou kmitat) a poté velmi pomalu ochlazují (čímž se atomy ustálí). Výsledkem je krystalická struktura s lepšími požadovanými vlast-

nostmi (vysoká integrita a minimum defektů). Simulované žíhání se snaží vyhledat globálně nejlepší řešení a vymanit se z případných lokálních extrémů. Na začátku algoritmu je stanovena hodnota teploty T, která se postupně snižuje (např. podle funkce nebo tabulky). Při výběru dalšího řešení se opět porovnává jeho kvalita s kvalitou současného řešení (rozdíl mezi těmito hodnotami se označuje rozdíl energií ΔE. Pokud platí, že $\Delta E > 0$, nové řešení se stane aktuálním podobně jako u metody Hill-climbing, jinak se nové (horší) řešení stane aktuálním s pravděpodobností $e^{\Delta E/T}$[9].

Simulované žíhání bylo například využito jako základ pro algoritmus hledající vyhovující umístění místností reprezentovaných podgrafy s uzly maximálně druhého stupně. Tyto podgrafy (nazývané řetězce) jsou po vyřešení následně propojeny v celkový dungeon. Simulované žíhání umožňuje s určitou pravděpodobností provést úpravu řešení již vyřešených podgrafů pro případ, kdy by lokální extrém neumožňoval jejich propojení ve výsledný graf. Vyhodnocovací funkce pak silně penalizuje případné průniky místností a chybějící propojení a naopak zvyšuje kvalitativní ohodnocení v případech, kdy k průniku nedojde a dojde k propojení místností či podgrafů[13].



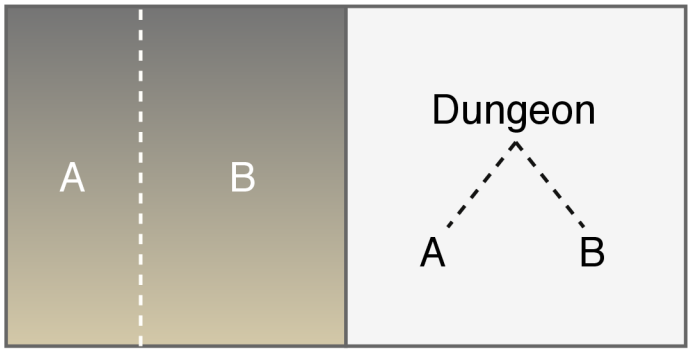
Obrázek 2.12: Ukázka generování dungeonu s přímo propojenými místnostmi. Při generování byla využita metoda simulovaného žíhání. Nalevo je vstupní graf, uprostřed pak dvě možné varianty, jak vygenerovat geometrii menší smyčky v grafu (řetězce), a vpravo je pak výsledný dungeon po propojení obou řetězců (smyček). Převzato z [13].

2.3.4 Dělicí algoritmy

Dělicí algoritmy rozdělují 2D nebo 3D prostor na menší části, přičemž každý bod v prostoru lze jednoznačně zařadit do jedné z těchto menších částí (též nazývaných *buňky*). Dělicí algoritmy často pracují hierarchicky a jsou volány rekurzivně i na již vytvořené celky. Výsledkem je stromová struktura částí, která je užitečná pro zpětné sestavování celku a také pro rychlé vyhledávání prvků. Tyto vlastnosti se využívají zejména v počítačové grafice.

Nejznámějším dělicím algoritmem je metoda **binárního rozdělování prostoru** (BSP – Binary Space Partitioning), která prostor rekurzivně dělí vždy na 2 části (ne nutně poloviny). Vzniká tím tzv. binární strom nebo též BSP strom, kdy každý uzel má přesně dva potomky (viz obrázek 2.13). Tento algoritmus nachází využití také při generování dungeonů. Protože prostor rozděluje na dvě části a žádná z částí ve stromu se nikdy nepřekrývá, opakovaným rozdělováním lze vytvářet místnosti a ty zpětným vyhledáváním v BSP stromu spojovat pomocí chodeb. Tento přístup zajišťuje, že se místnosti nebudou překrývat, že budou všechny spojené a že bude celý dungeon průchodný. Ve výsledných strukturách lze však vypozorovat jistá uspořádanost a některé společné vlastnosti, především fakt, že nikdy nedochází ke smyčkám.

Na začátku algoritmu existuje pouze kořenový uzel, který představuje celou herní oblast. Rekurzivním dělením herní oblasti poté vznikají podřazené uzly až po dosažení ukončovací



Obrázek 2.13: Ukázka první iterace binárního rozdělení prostoru (zde svisle), každá z části A a B je reprezentována jak v prostoru, tak ve stromu.

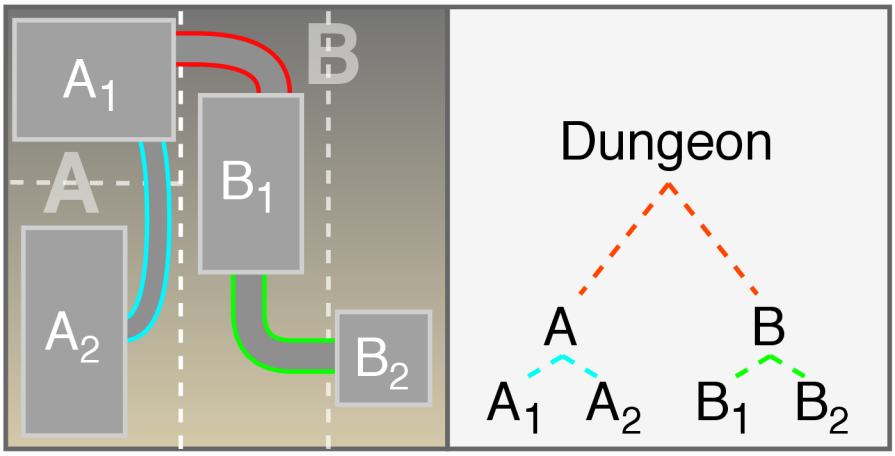
podmínky (např. počet uzlů nebo minimální plocha části). Nejčastěji tento algoritmus probíhá tak, že se daný prostor rozdělí nadrovinou[5] rovnoběžnou s jednou ze základních os. Například pokud je vstupem čtverec, pak se rozdělí na dvě části pomocí jedné vodorovné nebo svislé čáry. Algoritmus postupně dělí jednotlivé části, přičemž u vzniklých částí rekurzivně volá dělicí funkci také. Před rozdělením se vždy kontroluje, zda je vstup větší než minimální určená velikost. Pokud není, funkce končí. Jakmile rekurzivní volání dělicí funkce skončí, projdou se všechny uzly bez podřazených uzlů (listy) a v oblasti, kterou reprezentují, se vyberou dva náhodné body (rohy místnosti). U těchto rohů se zkontroluje, zda jimi tvořená místnost splňuje požadavky na minimální velikost, případně se vyberou body nové nebo se stávající posunou směrem od středu. Jakmile všechny části obsahují místnosti, dojde k jejich propojování pomocí chodeb. K tomu je využit BSP strom. Algoritmus projde všechny uzly stromu od nejnižších úrovní až po kořen, vybere vždy dva uzly se společným nadřazeným uzlem a mezi jejich místnostmi vytvoří chodbu. Tímto postupným spojováním dojde k propojení všech místností a celého dungeonu (viz obrázek 2.14)[17].

Metoda BSP je jednoduchý způsob, jak vytvářet hratelné dungeony. Problémem je, že takto vzniklé dungeony nejsou příliš přizpůsobitelné a jsou poněkud omezené v možnostech. Dalším problémem je jejich poměrně velká uspořádanost a jednotvárnost, kterou lze z části rozbít vytvářením menších místností v určených částech (tím se však sníží jejich „hustota“), případně upravit část algoritmu generujících místností tak, aby umožňovala místnosti různých tvarů nebo úplné vyplnění dostupného prostoru.

2.3.5 Generování pomocí agentů

Při procedurálním generování je důležitým požadavkem náhodnost, která zajistí jedinečný (ne repetitivní) obsah při každém spuštění generátoru – tato náhodnost však nesmí způsobit ztrátu požadované míry uspořádanosti a spolehlivosti, nesmí docházet k chybám a obsah musí dávat v daném kontextu smysl. Aby vygenerovaný obsah nevybočoval z požadovaných hranic a nedocházelo k nečekaným „nehodám“, je nutné do algoritmů generátoru zahrnout faktor, který těmto situacím zabrání. Možným řešením může být **generování pomocí agentů**, přičemž slovem agent se myslí autonomní prostředník s určitou mírou naprogramované inteligence nebo též vzorců chování. Agent dokáže snímat okolní prostředí, analyzovat jej a reagovat na něj v rámci naprogramovaného chování.

[5] **Nadrovina** v rozměru n představuje podprostor v rozměru $n-1$, například ve 3D je nadrovinou rovina, zatímco ve 2D je nadrovinou přímka.



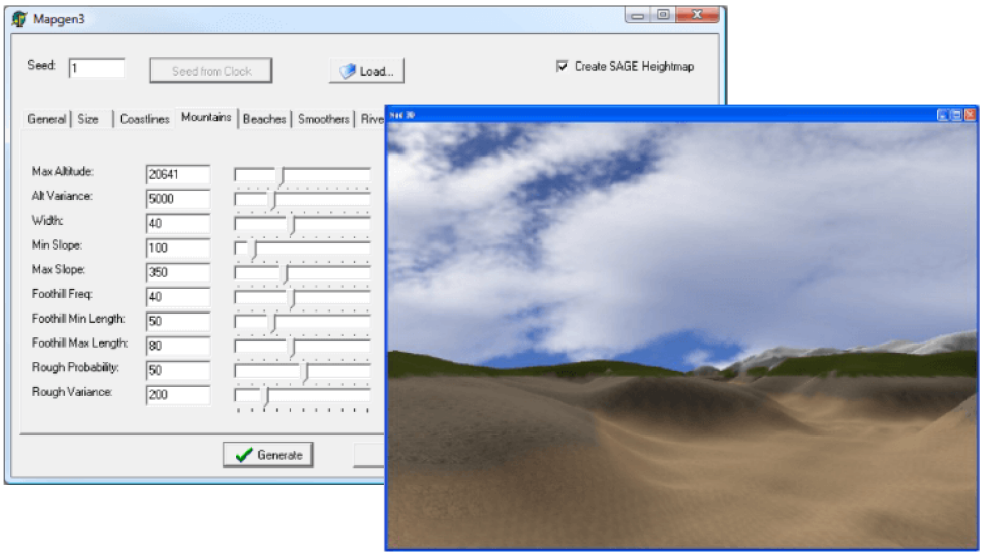
Obrázek 2.14: Jednoduchý dungeon vygenerovaný metodou BSP (vlevo). Dungeon se nejdříve rozdělí na dvě a poté na čtyři plochy, v těch se vytvoří místností zaujímající určitou část plochy a poté se spojí místnosti reprezentované uzly v BSP stromu (vpravo) tak, že se nejdříve propojí uzly A_1 a A_2, poté uzly B_1 a B_2 a poté se propojí jejich rodičovské uzly A a B.

Autoři článku[6] publikovaného v jednom z časopisů institutu IEEE v roce 2010 využili agentů k procedurálnímu generování 3D terénu. Snažili se přitom, aby se jejich generátor vyznačoval náhodností, rychlostí a aby jej bylo možné jednoduchým způsobem ovládat a modifikovat. Výsledný terén byl tvořen jednoduchou mřížkou bodů (v práci autoři použili rozměry 512x512 bodů), které určovaly výškový profil a byly od sebe vzdálené přibližně 1 metr, výsledný vygenerovaný terén tak pokrýval oblast cca 0,25 km^2. Ke generování použili různé druhy agentů, kteří měli různé úlohy, které se prováděly jedna po druhé (fáze generování):

1. Pevnina – tvorba základního tvaru pevniny, čímž přirozeně vzniká pobřežní čára, které může být poté obklopeno vodou
2. Vyhlazování – náhodné průchody, které průměrují výšku bodů v okolí
3. Pláže – zarovnávání oblastí na pobřežní čáře
4. Hory – tvorba kopců, hor a pohoří
5. Řeky – eroze terénu vytvářející řeky směrem od hor k pobřeží

Všichni agenti mají společnou sadu vlastností: pracují asynchronně a souběžně, mají přehled o výšce všech bodů na celé mapě a mohou je kdykoliv upravovat, protože jich však v každé fázi pracuje více, podmínky kolem každého z nich se mohou dynamicky měnit. U všech druhů agentů lze nastavit tzv. tokeny, které agenti spotřebovávají prováděním změn v terénu. Tokeny tedy určují jejich životnost a spolu s počtem agentů jsou druhým parametrem, který má významný vliv na výsledek. Agenti při své práci procházejí stanovenou částí terénu a upravují výškový profil podle požadovaných parametrů. V každé fázi je k dispozici mnoho parametrů, které jsou zakomponovány jako proměnné do algoritmů.

Agenty lze použít také pro generování dungeonů. Pro takové případy se obvykle používá jediný agent, který prochází herní plochou, „kope“ tunely a vytváří místnosti. Tím, že agent



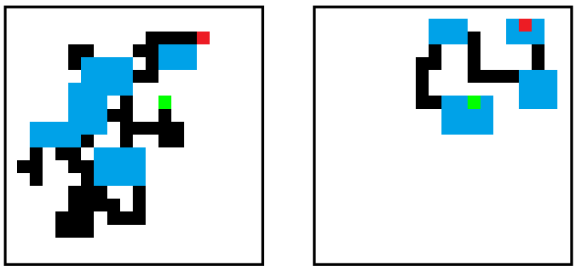
Obrázek 2.15: Ukázka parametrů pro generování hor pomocí agentů (vlevo) a výsledný terén (vpravo), převzato z [6].

musí celý dungeon projít, je zajištěno, že jej bude moci projít i hráč. Vzhled výsledného dungeonu pak závisí přímo na umělé inteligenci agenta a na informacích, které o dungeonu má. Problémem je, že je obtížné předpokládat, jaký vliv na výsledný vzhled dungeonu budou změny v chování agenta mít. Parametrem například může být, zda agent může protnout již existující chodby a místnosti. Pokud ano, pak může být výsledná struktura chaotická a nevzhledná. Pokud ne, pak se může agent zablokovat například v rohu herní plochy a výsledný dungeon bude velmi malý.

Nejjednodušším typem agenta je „slepý“ agent. Tento agent začíná na libovolném místě v dungeonu, odkud se pohybuje zcela náhodně bez ohledu na stav dungeonu, a při každém kroku mění dlaždici na jeho pozici na prostupnou (např. podlaha). S každým dalším krokem ve stejném směru se zvyšuje pravděpodobnost, že agent směr pohybu změní. Po změně směru se pravděpodobnost změny směru nastaví na nulovou. Obdobně funguje i vložení místnosti. Při každém kroku se pravděpodobnost vložení místnosti zvýší a pokud dojde k vložení místnosti (s náhodnými rozměry a středem na pozici agenta), pravděpodobnost se nastaví na nulovou. Tento přístup vede ke vzniku velmi chaotických dungeonů s překrývajícími se místnostmi a slepými uličkami (viz obrázek 2.16 vlevo).

Dalším pokročilejším, ale přesto jednoduchým typem agenta je „informovaný“ agent. Tento agent má informace o místnostech a chodbách a snaží se je pokládat tak, aby nedocházelo k jejich průniku. To zajišťuje tím, že když má položit místnost nebo chodbu, hlídá, aby tím nevznikla kolize. Pokud by ke kolizi došlo, zkusí místnost a chodbu s jinými rozměry (v rámci minimální a maximální definované velikosti). Agent se do místnosti nebo na konec chodby přesune až po jejich úspěšném vytvoření. Pokud ke kolizi dochází u všech možných kombinací velikostí, algoritmus končí. Tento agent vytváří dungeony, které jsou průchodné, nekříží se, ale jsou velmi lineární. Navíc může snadno dojít k tomu, že se agentovi nepodaří vložit žádnou místnost ani chodbu, a z tohoto důvodu algoritmus skončí, ačkoliv velká část herní plochy může zůstat nevyužita (viz obrázek 2.16 vpravo).

Ani jeden ze zmiňovaných přístupu není připraven pro použití v praxi. První přístup vytváří příliš chaotickou strukturu s překrývajícími se místnostmi, druhý přístup zase může



Obrázek 2.16: Ukázka dungeonů vytvořených pomocí agentů. Zeleně je označen počátek, červeně je označen konec. Dungeon vlevo vznikl pomocí „slepého“ agenta a obsahuje překrývající se místnosti a slepé uličky. Dungeon vpravo vznikl pomocí „informovaného“ agenta a v tomto případě skončil předčasně.

způsobit předčasné ukončení algoritmu. Přidávání dalších podmínek a „inteligence“ k agentovi může tyto problémy vyřešit, faktem však zůstává, že tvorba dungeonů tímto způsobem vyžaduje značné množství úsilí, času a experimentování s nepříliš jasnými výsledky. Tato metoda je tedy vhodná především v situacích, kdy je potřeba dosáhnout méně strukturované až chaotické podoby (např. jeskyně), nebo pokud je hlavní požadovanou vlastností nepředvídatelnost výsledného dungeonu[17].

Kapitola 3

Současná řešení generování dungeonů

Tato kapitola obsahuje náhled na existující, praktická řešení generování dungeonů. V první části jsou představeny existující hry, které pomocí některých z výše uvedených technik vytvářejí skutečné a hratelné dungeony. Druhá část se pak na již existující řešení dívá z perspektivy vývojářské – jsou prezentovány knihovny, které vývojářům umožňují dungeony generovat.

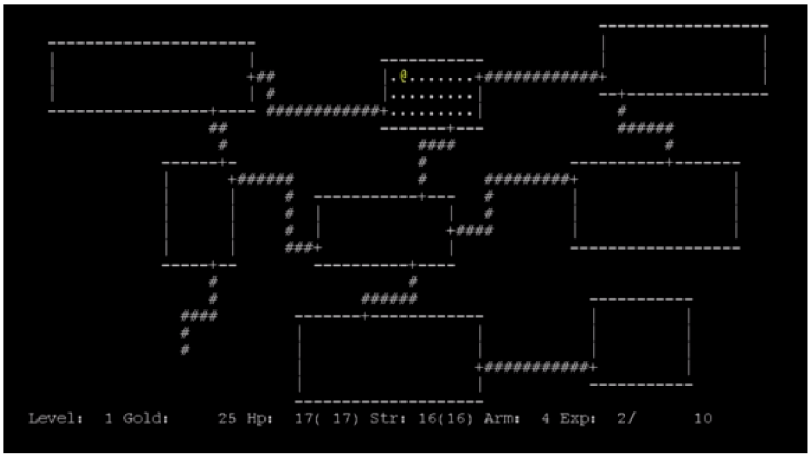
3.1 Generování dungeonů v existujících hrách

Tato část popisuje dungeony v některých známých existujících hrách. Ne vždy je jasné, jaké techniky vývojáři pro generování dungeonů použili. Zejména v komerčních hrách bez otevřeného zdrojového kódu lze pouze hádat, jak hra vnitřně funguje. Nicméně i samotným pozorováním topologické a geometrické stavby těchto úrovní si lze udělat dobrý obrázek o tom, jak generátory fungují. Analýza těchto herních úrovní může navíc sloužit jako předloha pro požadavky na vytvářenou knihovnu.

3.1.1 Rogue (1980)

Rogue je počítačová hra, která dala vzniknout žánru tzv. Roguelike her. Vyvinuli ji Michael Toy, Glenn Wichman a Ken Arnold pro Unixové systémy a vydali ji v roce 1980. Hra si rychle získala srdce spousty hráčů a programátorů, protože obsahovala tehdy revoluční (ale ne úplně první) procedurálně generované dungeony, ve kterých museli hráči bojovat proti příšerám, získávat lepší vybavení a vylepšovat svou postavu. Hra disponovala jednoduchou ASCII grafikou (viz obrázek 3.1). Pro hru bylo nutné vytvořit efektivní algoritmus generování dungeonů. Dungeony ve hře *Rogue* jsou tvořeny jednoduchými obdélníkovými místnostmi, které jsou propojeny chodbami. Dungeon bylo nutné vygenerovat ve velmi omezeném prostoru, protože herní plochu nebylo možné rolovat – vždy se zobrazovala celá úroveň a ta byla omezena staticky daným množstvím možných ASCII znaků na výšku a na šířku.

Algoritmus ve hře je poměrně jednoduchý. Celá herní plocha se rozdělí na mřížku 3x3 buněk. Každá buňka představuje jednu místnost, která může mít v rámci prostoru buňky náhodnou velikost. Místnosti obsahují informaci o tom, zda jsou připojeny k alespoň jedné jiné místnosti, a seznam místností, ke kterým jsou připojeny. Algoritmus začíná náhodným výběrem jedné začáteční buňky, ve které se hráč na začátku úrovně objeví. Následně se



Obrázek 3.1: Ukázka ze hry *Rogue*, místnosti a chodby dungeonu jsou tvořeny ASCII grafikou.

vybere sousední nepropojená místnost, vytvoří se mezi nimi chodba a začáteční místnost se označí jako propojená. Algoritmus se poté opakuje u této vybrané sousední místnosti, tedy místnost se propojí s jinou sousední nepropojenou místností a označí se jako propojená. Takto algoritmus v cyklu pokračuje, až dokud nelze vybrat žádnou sousední místnost, protože jsou všechny sousední místnosti propojeny. Následně algoritmus projde všechny zbylé nepropojené místnosti a propojí je s některou ze sousedních propojených místností. Nyní má každá místnost minimálně jedno propojení. Do naposledy propojené místnosti se přidá schodiště vedoucí do další úrovně – tato místnost je obvykle nejdále od začáteční místnosti. Algoritmus nakonec může vytvořit další náhodná propojení mezi místnostmi. V některých úrovních může dojít k tomu, že se v buňce mřížky místnost nevytvoří a tato buňka se použije pouze jako prodloužená chodba. Díky tomu nejsou výsledné dungeony tak jednotvárné[10].

Uvedený algoritmus je velmi jednoduchý, rychlý a je vždy úspěšný. Takto vytvořený dungeon je zcela průchodný. Nevýhodou je omezený počet místností, rozměry mřížky je sice možné zvětšit, ale dungeon zůstane stále velmi stereotypní a příliš uspořádaný.

3.1.2 Diablo 1 (1996)

Diablo 1 je hra vytvořená firmou Blizzard Entertainment a vydaná v roce 1996. Tato hra se stala velmi populární a dodnes je považována za důležitého předchůdce moderních her na hrdiny s dungeony. Hráč v této hře začíná v bezpečné statické úrovni, ve vesničce Tristram, odkud provádí výpravy do podzemních, náhodně generovaných úrovní (dungeonů). Hra obsahuje těchto úrovní celkem 16 ve čtyřech různých prostředích, z nichž každé je generováno jiným způsobem. Jako většina komerčních titulů neměla hra otevřený zdrojový kód a proto se fanoušci mohli jen dohadovat, jakým způsobem hra úrovně generovala. V roce 2018 programátor, který vystupuje pod přezdívkou GalaXyHaXz, zpětným inženýrstvím ladicích informací a symbolických odkazů omylem ponechaných v japonské verzi hry rekonstruoval přibližný původní zdrojový kód hry a spolu s dalšími autory jej vydali pod názvem Devilution[1]. Poté vznikl projekt DevilutionX[2], který si klade za cíl přizpůsobit hru moderním

[1] Původní projekt Devilution – https://github.com/diasurgical/devilution
[2] Verze projektu Devilution pro moderní OS – https://github.com/diasurgical/devilutionX

Obrázek 3.2: Ukázka ze hry *Diablo 1*.

operačním systémům a poskytnout fanouškům prostředky pro další modifikace této kultovní hry. Tyto verze ale stále slouží pouze jako nadstavby pro původní hru. Bez vlastnictví původní hry a jejích zdrojů (grafika, zvuky, animace atd.) nelze tyto rekonstruované hry spustit.

S možností nahlédnout do zdrojového kódu už nic nebránilo tomu, aby fanoušci začali zjišťovat, jaké vnitřní mechanismy hra využívala. V červenci roku 2019 se na webovém blogu objevil článek[1], který celkem podrobně analyzuje chování procedurálního generátoru úrovní v Diablu 1.

Jak již bylo zmíněno, úrovně Diabla 1 se dělí do čtyř prostředí a hráč se s nimi setkává postupně s tím, jak sestupuje do níže položených úrovní. Jsou to: **kostel**, **katakomby**, **jeskyně** a **peklo**. Ačkoliv každé prostředí má svá specifika, podle kterých byl generátor pro to či ono prostředí přizpůsoben, mají i mnoho společného. Každá úroveň se skládá ze 40x40 dlaždic, přičemž každá dlaždice se poté ještě dělí na 4 části, které slouží pro jemnější grafické detaily a určení průchodnosti postavou hráče. Generátor úrovní pracuje ve dvou fázích:

- Předběžný dungeon – Na konci této fáze je vytvořena kostra dungeonu. Veškerý prostor je rozdělen do jednoho ze dvou stavů – průchozí (stěna) nebo neprůchozí (podlaha). Na konci této fáze dungeon navíc obsahuje dveře a další vysokoúrovňové prvky.

- Konečný dungeon – V této fázi generátor řeší rozmístění grafických prvků, dodává do úrovně další stěny, sloupce, předem připravené části a další detaily.

Samotná tvorba místností a chodeb probíhá dvěma velmi odlišnými způsoby.

První metoda se používá pro úrovně v prostředí kostela, jeskyně a pekla (pouze s mírnými variacemi, jejichž výsledkem je, že každé další prostředí je otevřenější než to předchozí). Celý proces začíná vyplněním herní plochy neprůchozí stěnou a tvorbou místností

na této ploše. Místnost je čtvercová nebo mírně obdélníková konstrukce tvořena podlahou a stěnami na okrajích. V prostředí jeskyní pak má místnost nepravidelné okraje, generátor 50 % stěn nahradí za podlahu. V prostředí kostela se vkládají dvě velké místnosti propojené širokou chodbou se sloupy, v jeskyni a pekle pouze jedna místnost. V pekle je pak na okraji této hlavní místnosti vytvořena chodba táhnoucí se napříč a propojená se zbytkem místnosti.

Generování dalších místností probíhá voláním rekurzivní funkce, která napojuje další menší místnosti na okraje stávajících místností (pokud je tam dostatek volného místa). Průchodnost celého dungeonu je vždy zajištěna tím, že místnosti doléhají přímo na hranu, čímž dochází k jejich přímému spojení a odstranění této hrany. V prostředí kostela se tato funkce volá na samotnou místnost, ve které vybere náhodně osu X nebo osu Y a na okrajích místností po vybrané ose vytvoří dvě místnosti. Vytvořené místnosti se zarovnají se středy hran. Následně je funkce rekurzivně volána i pro nově vzniklé místnosti, pouze se použije osa opačná. Rekurzivní volání končí, jakmile není místo k vytváření nových místností. Stejný postup se používá i v prostředí pekla, místnosti však mají fixní velikost a generuje se jich pouze omezený počet daný součtem podlahových dlaždic. Tyto vygenerované místnosti se poté zrcadlově kopírují – svisle i vodorovně – díky čemuž jsou úrovně v pekle velmi symetrické. V prostředí jeskyní se funkce pro generování místností volá pro každou hranu místnosti. Místnosti vytvořené na hranách nejsou nijak zarovnávány, jejich umístění je náhodné, jsou však daleko menší, čímž je celá jeskynní úroveň členitější, ale ve výsledku otevřenější. V jeskynních úrovních mohou místnosti vytvářet také smyčky a napojovat se na již existující předchozí místnosti, což jen podporuje jejich otevřenost.

Generátor v prostředí kostela a v prostředí jeskyní poté dodatečně vytváří stěny, které vycházejí z jedné stěny a končí u jiné. V kostele se stěny táhnou vždy od rohu podél stěny do volného prostoru až k další stěně kde končí. Díky tomu je prostředí kostela velmi strukturované. Stěny jsou poté v 25 % případů nahrazeny průchodným sloupořadím, v 25 % případů mřížemi a jinak zůstávají stěnami. Do těchto stěn a mříží jsou poté přidány dveře nebo oblouky, které zajišťují průchodnost po celém dungeonu. V prostředí jeskyní nemusí generované stěny vycházet přímo z rohů, ale z jakéhokoliv místa jiné stěny. I tyto stěny jsou poté zprůchodněny dveřmi.

Jeskynní úroveň je dále specifická tím, že se úroveň „začišťuje“ algoritmem pro erozi. Osamostatněné stěny uprostřed volného prostoru jsou odstraněny, u dlouhých rovných stěn na okrajích se vytváří členitější struktura tím, že se 50 % stěn nahradí za podlahu. Větší celky stěn obklopené ze všech stran podlahou jsou nahrazeny za lávu (fakticky se jedná o stěnu jen s jinou texturou) a z těchto lávových jezer poté proudí řeky až ke stěnám. Aby řeky nenarušily průchodnost úrovně, generátor přes ně vytváří na vhodných místech mosty.

Druhá metoda se používá výhradně pro katakomby a je velmi speciální. Generátor katakomb si ukládá veškeré dlaždice do ASCII mapy (vývojáři společnosti Blizzard byli inspirování žánrem Roguelike, viz část 2.1.3). Toto řešení navíc nevyžaduje žádné pokročilé struktury pro uchování dat. Prázdné políčko (`' '`) znamená neprůchozí dlaždici. Na začátku algoritmu je zavolána rekurzivní funkce, které je předána jako vstup celá plocha úrovně bez okrajů. Tato funkce má za úkol vygenerovat místnosti ve vstupní oblasti, přičemž si uchovává referenci na místnost, která ji zavolala (na začátku algoritmu null). Nová místnost má náhodnou velikost (přizpůsobenou velikosti vstupní oblasti) a její pozice v rámci vstupní oblasti je také náhodná. Místnost se do ASCII mapy zakreslí následovně: okraje místnosti (stěny) se zapíší jako `'#'`, dlaždice podlah se zapíší jako `'.'` a rohy místnosti se zapíší písmeny `'A', 'B', 'C', 'E'`. Pokud není reference na volající místnost null, generátor na nejbližších stranách volající a nové místnosti vybere dvě náhodné rovnoběžné dlaždice

stěn a uloží je do seznamu chodeb pro pozdější využití. Zbytek vstupní oblasti se rozdělí na čtyři obdélníkové oblasti, jejichž velikost se zmenší o 2 (aby vznikla mezi místnostmi mezera) a poté se pro ně zavolá rekurzivní funkce pro tvorbu místností s referencí na právě vygenerovanou místnost. Po dokončení rekurzivních funkcí (které skončí, jakmile už nebude prostor pro nové místnosti) je nutné mezi vygenerovanými místnostmi vytvořit chodby. Ty vzniknou přímým propojením dvojic stěn uložených dříve. Chodbám se určí náhodná šířka 1 až 3 dlaždice a zarovnají se pomocí rohů propojovaných místností. Pokud chodba protne stěnu (`'#'`), znak se změní na `'D'` (door), pokud protne neprůchozí dlaždici (`' '`), znak se změní na `','`, což označuje chodbu samotnou. Tento mezi-zápis umožňuje křížení chodeb. Po zakreslení chodeb se rohy (`'A', 'B', 'C', 'E'`) změní na stěny (`'#'`), neprůchozí dlaždice (`' '`) vedle chodeb (`','`) se změní také na stěny (`'#'`) a samotné znaky chodeb (`','`) se nakonec nahradí podlahou (`'.'`). Na závěr se spustí algoritmus, který vyhledá nevyužitý prostor na mapě a vyplní jej protáhnutím existujících místností, případně vytvořením nových místností.

Generátor Diabla 1 kromě generování místností a chodeb provádí mnoho procedur, bez kterých by nebyly herní úrovně kompletní. Následuje jejich stručný výčet.

Předem připravené části

Hra obsahuje několik předem připravených částí, které slouží především pro úkoly v té dané úrovni. Tyto předem připravené části generátor jednoduše vloží podle určitých parametrů (v závislosti na prostředí dané úrovně). Může se jednat například o podstavec s knihou a svíčkami okolo nebo o místnost s obtížným protivníkem a specifickým grafickým zpracováním.

Obrázek 3.3: Typická předem připravená část ve hře *Diablo 1* – řezníkovo doupě. Místnost je pokaždé naprosto stejná. Převzato z [1].

Mini-části

Každá úroveň má připraveny desítky tzv. mini-částí, které generátor do úrovně vkládá na základě různých pravidel. Tato pravidla mohou zahrnovat např. výskyt určitých prvků v úrovni nebo vložení na základě pravděpodobnosti. Mini-části slouží k dekorování úrovní

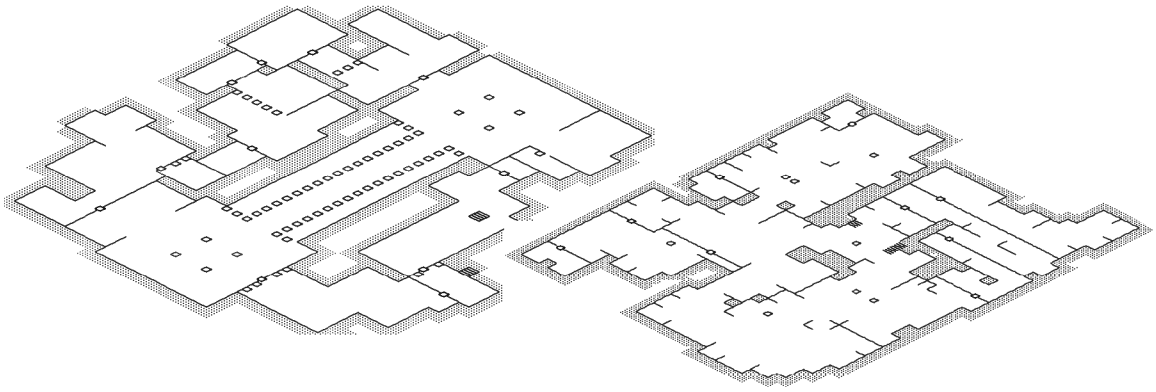
(např. výbušné sudy, světla, kaluže krve), k opravám chyb vzniklých generováním (je jednodušší tyto chyby opravovat v postprodukci než je opravovat v hlavní fázi generování) nebo třeba ke vkládání schodišť propojujících úrovně.

Speciální místnosti

Ve hře se vyskytují místnosti, které mají určité speciální zaměření. Tyto místnosti mají stanovené optimální rozměry a úrovně, ve kterých se mohou vygenerovat. Může se jednat například o knihovnu, ve které hráč nalezne vždy minimálně jeden svitek, nebo o místnost plnou příšer. V prostředí kostela generátor tyto místnosti vytváří tak, že si vybere nějakou z již existujících místností a tu přepracuje. V dalších prostředích je jednoduše vkládá do volného prostoru.

Kontrolní mechanismy

Generování úrovní se neobejde bez kontrolních mechanismů, které zjistí, zda vygenerovaná úroveň splňuje požadované parametry. V úrovních se kontroluje především průchodnost dungeonu napříč, zda se podařilo vložit předem připravenou část pro úkol dané úrovně a zda došlo k vygenerování dostatečného počtu průchozích dlaždic. Pokud některá z těchto kontrol selže, generování probíhá celé od začátku.

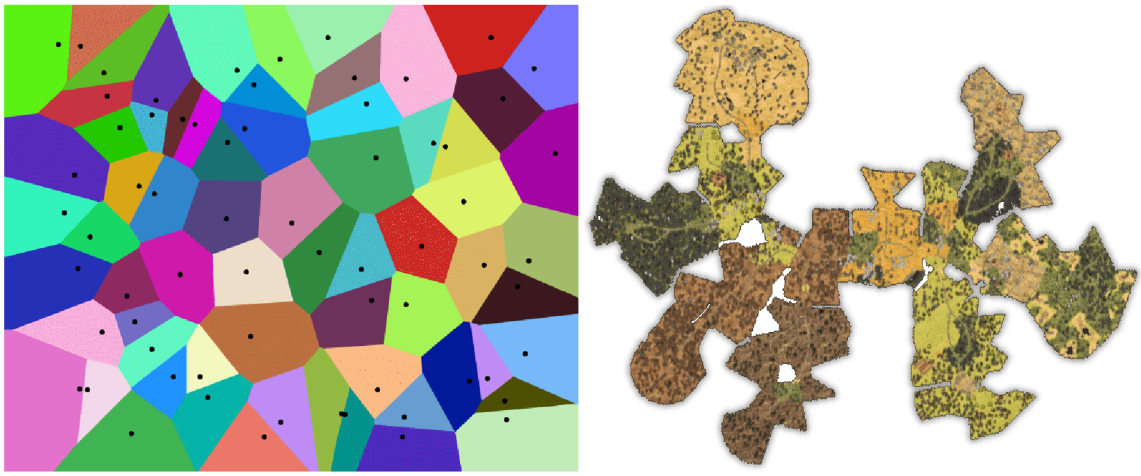


Obrázek 3.4: Ukázka struktur vygenerovaných úrovní ve hře *Diablo 1*, kostel (vlevo) je mnohem uspořádanější, zatímco jeskyně (vpravo) má více entropický charakter, převzato z [1].

3.1.3 Don't Starve (2013)

Don't Starve je počítačová hra vydaná v roce 2013 kanadskou společností Klei Entertainment. Jedná se o hru žánru survival (boj o přežití), ve které se hráč ujímá role protagonisty Wilsona, vědce, který se za podivných okolností objeví v tajemném a neprobádaném světě. Tento svět je náhodně generovaný a hráč v něm má volné pole působnosti – může vyrábět různé předměty, objevovat rozličná místa a musí se pokusit přežít (režim sandbox – pískoviště). V tomto světě se nachází předmět, který hráče přemístí do režimu dobrodružství. V tomto režimu má hráč jediný cíl – zjistit, co se mu stalo a proč a porazit antagonistu Maxwella. V režimu dobrodružství hráč postupně prochází pět náhodně generovaných světů, přičemž v každém z nich začíná téměř od nuly. Smrt v kterémkoliv světě znamená, že se

hráč přesune do úplně prvního světa a ztratí veškerý postup. Tento mechanismus „trvalé smrti“ se nápadně podobá žánru Roguelike.



Obrázek 3.5: Náhodný Voroného diagram (vlevo) a ukázka mapy ze hry *Don't Starve* vytvořené pomocí Voroného diagramu (vpravo). Herní plocha se rozdělí na menší části a těm se poté přiřadí specifický účel a vzhled. Části bez účelu budou zahozeny.

Jak již bylo zmíněno, světy v Don't Starve jsou procedurálně generované. Ačkoliv je svět otevřený, lze jej interpretovat jako síť propojených místností a tedy i jako dungeon. Jejich fyzická geometrická struktura je vytvořena na základě Voroného diagramu[3] (viz obrázek 3.5). Světy občas mohou obsahovat vchody do dalších světů. Každý svět má přiřazen motiv, který určuje, jak bude vypadat terén a jaké se ve světě vygenerují úlohy, předměty, nepřátelé a další prvky. Místnosti světa jako takové se generují na základě úloh, které hráč musí vykonat. Každá úloha obsahuje seznam tzv. zámků a klíčů, které abstraktně reprezentují posloupnost úkonů v dané úloze. Při generování se pro každou úlohu vytvářejí uzly místností (s případnými klíči nebo zámky), které se zařazují podle určitých pravidel do grafu (který je ze začátku lineární, poté může utvářet i smyčky). Úlohy se svými místnostmi se poté zařazují do stromové struktury podle toho, jaké klíče a zámky obsahují. Kontroluje se přitom, aby nedošlo k situaci, kdy by se hráč ocitl v bezvýchodné situaci. Graf může být na náhodných místech propojen, což usnadňuje průchod, a případně mohou být vloženy teleportační prvky, které fakticky graf spojují také. Nakonec je pro takto utvořený graf vytvořena geometrická podoba úrovně a přiřazena grafika[3].

Generátor úrovní ve hře Don't Starve ukazuje, že při vytváření úrovně zohledňující hráčův postup může být výhodné nejdříve vygenerovat abstraktní graf, kterému lze poté pomocí některé metody (například Voroného diagramu) přiřadit geometrickou podobu.

3.2 Dostupné knihovny pro generování dungeonů

Tato část popisuje některé existující knihovny pro generování dungeonů. U každé z nich je provedena analýza základních funkcí, vstupů, výstupů, případně také technik, které používají. Průzkum existujících knihoven může poskytnout představu o tom, kterým směrem by se měla nová knihovna ubírat.

[3] **Voroného diagram** je kolekce oblastí vzniklých dělením roviny na základě vzdáleností k dané diskrétní množině bodů.

3.2.1 DungeonGenerator (jongallant)

Zajímavý generátor pro systém Unity, který při tvorbě dungeonů využívá fyzikálního systému, algoritmu pro normální rozdělení v kružnici, delaunayovy triangulace a vyhledávání minimální kostry grafu[2].

Projekt na GitHubu: `https://github.com/jongallant/DungeonGenerator`

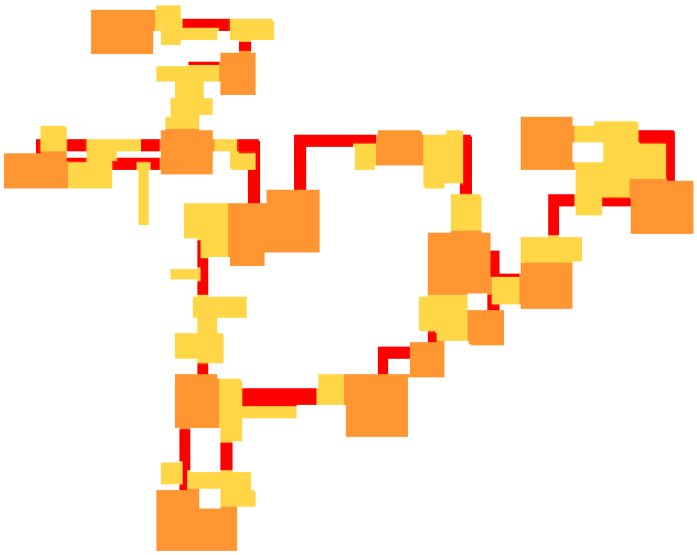
Tato knihovna má jediný účel: vygenerovat dungeon skládající se z místností a chodeb, přičemž i samotné chodby mohou být tvořeny vedlejšími místnostmi. Knihovna je určena pro použití v systému Unity a je napsána v jazyce C#. Dostupnými **vstupními parametry** knihovny jsou:

- Počet místností
- Poloměr kružnice, do které budou místnosti rovnoměrně vkládány (a následně pomocí fyzikálního systému rozprostřeny)
- Podíl hlavních místností
- Množství dodatečných propojení mezi místnostmi

Parametry jsou pevně zapsány v kódu a je nutné jejich hodnotu upravovat tam. To vyžaduje alespoň obecný přehled nad vnitřní strukturou knihovny. Knihovna nenabízí konfigurační soubor. **Výstupem knihovny** je:

- Seznam místností a jejich vlastností (pozice, rozměry)
- Seznam vytvořených chodeb mezi místnostmi
- 2D mřížka obsahující místnosti a chodby ve formě dlaždic
- Rastrový obrázek dungeonu.

Generátor lze spustit například přetažením skriptu `DungeonGenerator.cs` na příslušný `GameObject` Unity scény a následným stiskem tlačítka `Play`. Třída `DungeonGenerator` je potomkem standardní Unity třídy `MonoBehaviour`, která zajistí její spuštění. Po spuštění se nejdříve zavolá metoda `RoomGenerator.Generate`, která vytváří zadaný počet místností s náhodnými rozměry a vektorovou reprezentaci rovnoměrně na kružnici o zadaném poloměru. Největší místnosti se určí jako hlavní (ty jsou vždy součástí výsledného dungeonu a tvoří body kostry grafu), ostatní jako vedlejší (ty jsou součástí dungeonu jen pokud je protíná některá z chodeb mezi hlavními místnostmi). Všechny místnosti jsou asociovány s fyzikálními objekty Unity `Rigidbody2D`, které zajišťují, že se při překrývání rozpínají a odsunují od sebe. Fyzikální systém Unity zajistí, že se místnosti rozprostřou po herní ploše a nebudou se překrývat. Tento proces je nepředvídatelný a zajišťuje unikátní ráz dungeonu při každém vygenerování. Jakmile se místnosti ustálí, u všech hlavních místností se vyhledají středy a ty se použijí do algoritmu delaunayovy triangulace, který mezi nimi vytvoří trojúhelníkovou mřížku. Tato mřížka se poté převede na graf a pro takto vzniklý graf se vypočítá minimální kostra. Ta zajistí, že všechny hlavní místnosti budou v rámci dungeonu dosažitelné, ale nebudou všechny vzájemně propojené. Protože je minimální kostra grafu lineární struktura bez smyček, do grafu se náhodně přidají dodatečná propojení. Hlavní místnosti se poté propojí chodbami. V tomto případě se používají tři typy chodeb: rovné vodorovné, rovné svislé nebo chodby ve tvaru písmene L. Chodba samotná je také fyzikálním objektem a pokud protne fyzikální objekt vedlejší místnosti, vedlejší místnost se stane



Obrázek 3.6: Ukázka dungeonu vygenerovaného pomocí knihovny DungeonGenerator (jongallant). Oranžově jsou vyobrazeny hlavní místnosti, žlutě vedlejší místnosti a červeně chodby.

součástí dungeonu. Ostatní vedlejší místnosti se zahodí. Následně se vytvoří 2D mřížka a procesem podobný rasterizaci se matematicky zapsané místnosti a propojení převedou na dvourozměrné pole dlaždic, z nichž každá je reprezentována celým číslem: 0 = prázdné místo, 1 = propojení, 2 = stěna. Místnosti a chodby tak v dungeonu získají prostorovou charakteristiku. Knihovna nakonec vygeneruje také rastrový obrázek vytvořeného dungeonu.

Knihovna se nepoužívá příliš intuitivně, protože neobsahuje žádné veřejné rozhraní. Uživatel musí upravovat přímo vnitřní parametry a vyextrahovat si výstup sám. Její vstupní parametry jsou navíc velmi omezené. Kladnou stránkou je fakt, že knihovna vytváří jak matematickou (vektorovou) reprezentaci místností a chodeb, tak 2D mřížku a dokonce i obrázek, který může sloužit jako mapa. Místnosti jsou však obecné a je na vývojáři, aby jim dal jejich funkci až poté. Generování nelze přizpůsobit potřebám hry a herních prvků (například nelze určit, že jedna místnost má být počáteční a druhá koncová a tyto dvě od sebe při generování co nejvíce vzdálit). Knihovna je navíc omezena pouze na využití v systému Unity, protože využívá jeho mechanismů.

3.2.2 Rot Web API (MattMcFarland)

Webová služba vytvořená v systému Node.js, která nabízí rozhraní pro tvorbu Roguelike map, bludišť, jeskyní pomocí celulárních automatů a dungeonů pomocí agentů. Funguje jako webový server, který zpracovává HTTP GET požadavky a odpovídá JSON strukturami.

Projekt na GitHubu: `https://github.com/MattMcFarland/rot-web-api`

Knihovnu lze nainstalovat pomocí správce balíků *npm* příkazem `npm install` a poté ji lze spustit příkazem `npm start`. Knihovna nabízí několik typů generátorů, z nichž každý je dostupný prostřednictvím svého rozhraní. Každý generátor lze zavolat příslušným požadavkem HTTP GET na adresu webové služby s příslušnými parametry. Například generátor

`Rogue` nabízí parametry *width*, *height*, *roomWidth* a *roomHeight* a lze jej například zavolat tímto způsobem:

`http://localhost:8080/api/rogue?width=80&height=25&roomWidth=10&roomHeight=10`

Po obdržení požadavku server vygeneruje Roguelike dungeon a zašle zpět odpověď ve formátu JSON (viz obrázek 3.7).


```
{ "map": [ [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ], [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ], [ 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
], [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ]... ], "_width": "25", "_height": "80", "_options": { "cellWidth":
3, "cellHeight": 3, "width": "80", "height": "25", "roomWidth": "10", "roomHeight": "10" },
"rooms": [ [ { "x": 1, "y": 10, "width": 1, "height": 0, "connections": [ [ 0, 1 ] ], "cellx": 0,
"celly": 0 }, { "x": 4, "y": 32, "width": 1, "height": 0, "connections": [ [ 0, 2 ] ], "cellx": 0,
"celly": 1 }, { "x": 2, "y": 63, "width": 0, "height": 1, "connections": [ [ 1, 2 ] ], "cellx": 0,
"celly": 2 } ], [ { "x": 9, "y": 9, "width": 0, "height": 0, "connections": [ [ 1, 1 ] ], "cellx": 1,
"celly": 0 }, { "x": 9, "y": 33, "width": 0, "height": 1, "connections": [ [ 0, 1 ] ], "cellx": 1,
"celly": 1 }, { "x": 11, "y": 58, "width": 0, "height": 0, "connections": [ [ 2, 2 ] ], "cellx": 1,
"celly": 2 } ], [ { "x": 20, "y": 8, "width": 0, "height": 1, "connections": [ [ 2, 1 ] ], "cellx":
2, "celly": 0 }, { "x": 20, "y": 28, "width": 0, "height": 0, "connections": [ [ 1, 1 ] ], "cellx":
2, "celly": 1 }, { "x": 18, "y": 59, "width": 1, "height": 0, "connections": [ [ 2, 1 ] ], "cellx":
2, "celly": 2 } ] ], "connectedCells": [ [ 1, 1 ], [ 2, 2 ], [ 2, 1 ], [ 0, 1 ], [ 1, 2 ], [ 0, 2 ] ]}
```


Obrázek 3.7: Ukázka JSON odpovědi webového serveru Rot Web API (MattMcFarland) na HTTP GET požadavek pro generátor `Rogue`. Část `map` obsahuje 2D matici dlaždic (zkráceno), následné části s podtržítky obsahují obecnou charakteristiku dungeonu a další části představují obecný popis místností a propojení mezi nimi.

Tento projekt ukazuje zajímavý koncept generátoru herních úrovní v podobě webové služby. Jednoduchým voláním HTTP GET požadavků s příslušnými parametry lze vygenerovat požadovaný výstup. Server dokáže obsloužit spoustu klientů pomocí vygenerovaných dat nezávisle na programovacím jazyce, který používají. Nevýhodou je, že je potřeba server, na kterém bude služba trvale běžet. Dostupné algoritmy, jejich parametry a výstupy jsou navíc poměrně triviální.

3.2.3 Shrnutí existujících řešení

Knihovny představené v části 3.2 mají výhody i nevýhody, z nichž se lze při vývoji této knihovny poučit. Důležitou otázkou je **použitelnost knihovny** z hlediska programovacího jazyku nebo platformy. Ideální by bylo, kdyby byla knihovna použitelná univerzálně, například jako služba, která obslouží požadavky klientů, případně jako program spustitelný z libovolného systému či jazyku. Dalším důležitým faktorem je **uživatelská přívětivost**. Uživatel by měl mít k dispozici jasně definované rozhraní pro definování parametrů, které ovlivní výstup, bez nutnosti tyto parametry složitě nastavovat přímo v kódu knihovny. Neméně důležitý je pak **formát výstupu**. Důležitá není pouze podpora různých souborových formátů, ale i přizpůsobení výstupu podle toho, pro jaké případy užití bude výstup navržen. Například pokud bude uživatel chtít výstup vykreslit na rovinu, mřížka buněk je daleko vhodnější alternativou, než složité grafové struktury sestávající se z uzlů a hran.

Kapitola 4

Návrh knihovny pro generování dungeonů

Hlavním cílem této bakalářské práce je tvorba knihovny pro procedurální generování dungeonů. Před vytvořením samotné knihovny je vhodné tuto knihovnu obecně navrhnout – jaké jsou na ni kladeny požadavky, co musí knihovna dělat, jak ji bude uživatel používat a jaká tedy bude její celková podoba. O tom pojednává tato kapitola. V kapitole 5 pak bude knihovna podle návrhu implementována.

4.1 Požadavky na knihovnu

Před vytvořením samotného návrhu je potřeba si uvědomit, jak bude knihovna používána a jaké vlastnosti tedy musí splňovat. Knihovna je určena především pro vývojáře, kteří nechtějí herní úrovně vytvářet ručně a hledají jednoduchý způsob, jak je generovat procedurálně. Knihovna by jim tedy měla ušetřit čas, ale přitom vytvářet úrovně, které budou moci použít ve svých hrách. Vývojář knihovnu jednoduše zahrne do svého projektu a prostřednictvím jejího rozhraní určí požadované parametry. Pokud vývojář pracuje v jiném jazyce, měl by být schopen knihovnu i přesto využít. Po spuštění generátoru dojde k vytvoření struktury herní úrovně, která bude respektovat požadavky vývojáře.

4.1.1 Funkce knihovny

Knihovna musí uživateli umožnit vytvořit a vložit vstupní konfiguraci, tuto konfiguraci validovat a případně uživatele informovat o chybách, aby mohl konfiguraci opravit. Poté se tento vstup předá generátoru, který podle konfigurace vygeneruje herní úroveň. Pokud by došlo během generování k chybám, uživatel musí být o této skutečnosti informován. Následně musí být uživateli nabídnuty různé způsoby, jakými může k výsledné výstupní struktuře přistupovat a zpracovat ji.

4.1.2 Cílové vlastnosti

Při vývoji knihovny bude nutné brát zřetel na tyto vlastnosti:

- *Rychlost:* Knihovna by měla dosáhnout výsledku v rozumném čase, není jisté, kdy vývojář generátor spustí. Může to být na „načítací obrazovce“, nebo dynamicky přímo ve hře. Knihovna by neměla vývojáře nutit k tomu, aby herní úrovně postupně gene-

roval na samostatném vláknu, proto je rychlost klíčová, aby se hra „nezasekla“ a hráč nemusel čekat.

- *Přizpůsobitelnost:* Knihovna by měla nabízet takové parametry pro přizpůsobení výstupu, aby pokryla co největší množství použití. Samozřejmě nelze podchytit všechny případy užití, vzorem ale může být analýza existujících her. Knihovna by vývojáři zároveň měla umožnit, aby si vygenerovaný výstup mohl dále přizpůsobovat o prvky, které knihovna nepodporuje. Například doplnit do vygenerované obdélníkové místnosti sloupořadí.
- *Jednoduchost:* Knihovna musí ušetřit vývojářům čas. Její použití by proto mělo být jednoduché a intuitivní.
- *Spolehlivost:* Aby mohl vývojář knihovnu ve své hře použít, musí se spolehnout na její výstup. U prvků herní úrovně, které vývojář označí jako povinné, musí knihovna **zajistit**, že se budou ve vygenerovaném výstupu skutečně nacházet. Pokud bude vývojář například generovat úroveň, která bude obsahovat místnost s hlavním nepřítelem, nemůže dojít k tomu, že by ve výsledné úrovni chyběla.
- *Různorodost:* Vývojář knihovnu použije, protože chce, aby byly herní úrovně náhodně generované. S tím totiž úzce souvisí to, za jak dlouho je hráč začne vnímat jako repetitivní a stereotypní. Knihovna proto musí zajistit, že hráč pokaždé prožije jedinečný zážitek.
- *Opakovatelnost:* Knihovna by měla obsahovat mechanismus, který umožní při zadání stejných parametrů opakovaně získat tentýž výstup (je-li to vyžadováno vývojářem). Může se jednat například o klíč, který se použije jako semínko generátoru náhodných čísel.
- *Univerzálnost:* Knihovna, a především její výstup, by měly být univerzálně použitelné (do té míry, jak je to technicky možné a proveditelné).

Knihovna bude z hlediska těchto vlastností testována v kapitole 7.

4.1.3 Struktura a vlastnosti dungeonu

Herní úroveň generovaná touto knihovnou by měla obsahovat místnosti specifikované uživatelem a tyto místnosti musí být vzájemně vhodným způsobem propojeny na rovině. Každá místnost může být propojena s více místnostmi (ideálně nejbližšími), místnosti mohou tvořit smyčky ale i „slepé konce“. Je nutné nalézt ideální poměr mezi „propojeností“ místností a linearitou. Místnosti se nesmí vzájemně překrývat, ale zároveň by dungeon neměl obsahovat místnosti příliš vzdálené od ostatních místností. Herní úroveň nesmí obsahovat žádné osamostatněné místnosti, tzn. hráč by měl být vždy schopen dostat se z jednoho konce herní úrovně na druhý. V případě, že jsou místnosti od sebe vzdáleny, lze je propojit chodbou. Chodba by však neměla být příliš dlouhá, měla by respektovat velikosti místností a celkové herní úrovně. Pokud na sebe místnosti doléhají, chodba není třeba. Vývojář však musí být informován o tom, zda mají být dvě sousední místností propojeny či nikoliv.

Hry obsahující dungeony jsou velice často s isometrickou grafikou a jednotlivé grafické assety[1] herních úrovní reprezentují nějakou jejich diskrétní část (například asset znázorňující blok stěny). Proto je vhodné, aby byl dungeon rozčleněn do dlaždic a každý prvek

[1] **Grafický asset** – vizuální reprezentace objektu použitého ve hře, typicky se jedná například o 3D model s texturou nebo rastrový obrázek

dungeonu bude tvořen jednou nebo více dlaždicemi. Díky tomu bude moci vývojář dlaždicím přiřadit konkrétní grafickou podobu.

4.1.4 Způsoby použití knihovny

Knihovna musí nějakým způsobem převzít konfiguraci, tu zpracovat a předat na výstup výslednou strukturu. Aby byla co nejpoužitelnější, je vhodné, aby ji šlo používat různými způsoby, z nichž minimálně jeden nesmí být závislý na použitém jazyce. Existují různé způsoby, jakými se knihovny používají:

- Knihovna jako součást projektu. Standardní vložení knihovny do projektu a volání funkcí či metod jejího veřejného rozhraní, knihovna, nebo minimálně její volané části, jsou poté většinou součástí samotného projektu (nejsou-li již dostupné v prostředí, kde je projekt spouštěn). Taková knihovna může mít podobu binárního souboru (nutné další zpracování programem zvaným *linker*) nebo skriptu. Tento způsob redukuje možnost využití na jeden konkrétní programovací jazyk. K dispozici však mohou být specializované nástroje, které dokážou knihovnu transformovat pro využití v jiných jazycích. Příkladem může být projekt SWIG[2], který umožňuje propojit programy napsané v jazyce C a C++ s vyššími programovacími jazyky.
- Knihovna jako webová služba. Knihovna je spuštěna na webovém serveru a zpracovává příchozí požadavky klientů, které ji webový server předá. V současnosti se nejčastěji používá architektura rozhraní REST, která definuje komunikační protokol pro komunikaci mezi aplikacemi a klienty prostřednictvím HTTP volání.
- Knihovna jako program. Knihovna přijímá přímý nebo souborový vstup a vytváří výstup. Knihovnu lze tudíž spouštět jako kterýkoliv jiný program. S využitím funkcí operačního systému lze pak knihovnu spouštět jiným programem napsaným v libovolném jazyce.

Konkrétní způsoby, jakými se bude používat tato knihovna, budou záviset na implementačním jazyce.

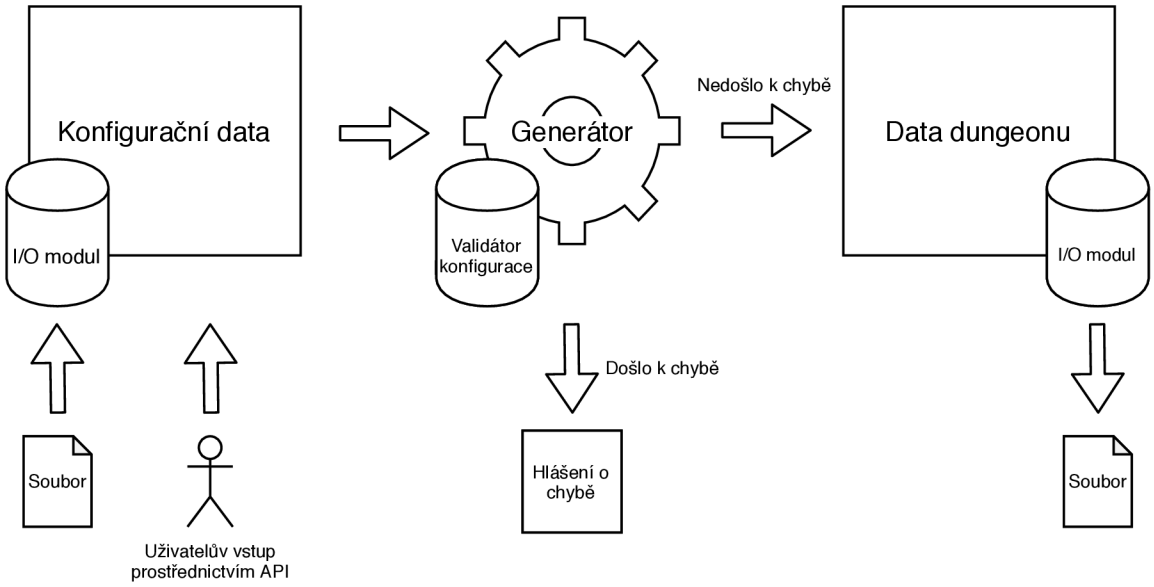
4.2 Struktura knihovny a popis komponent

Obecný návrh struktury knihovny se stručným popisem je uveden na obrázku 4.1. Tato část obsahuje popis jednotlivých komponent knihovny. Podrobnější popis navrhovaného algoritmu generátoru se nachází v části 4.3.

4.2.1 Vstupní konfigurace

Vstupní konfigurace je důležitý prvek, který generátoru říká, jakým způsobem má pracovat. Tuto konfiguraci bude moci uživatel určit pomocí přiloženého rozhraní, případně pomocí konfiguračního souboru. Konfigurace bude obsahovat informace o tvarech a počtech požadovaných místností a globální parametry.

[2] http://www.swig.org



Obrázek 4.1: Obecný návrh struktury knihovny – knihovna načte konfigurační data (ať už zadaná uživatelem prostřednictvím API nebo jako soubor), tato data poté předá generátoru, který konfiguraci prověří a vygeneruje dungeon, který lze poté také převést do souborové podoby.

Tvary místností a tvarové šablony

Každá místnost musí mít přiřazen tvar. Ty se budou definovat zvlášť. Tvary místností budou mít svůj jedinečný název (identifikátor), který lze poté používat v definicích místností. Každý tvar tedy půjde využít ve více místnostech. Tvar jednoznačně definuje, jak bude daná místnost velká a který prostor bude zaujímat. Uživatel bude moci tvar definovat dvěma způsoby:

1. Exaktním ručním zápisem se znaky reprezentujícími jednotlivé dlaždice, kdy každý z nich představuje prostor místnosti nebo prázdný prostor
2. Použitím předem připravené šablony s parametry, například šablona pro generování obdélníků s šířkou a výškou ze zadaného rozsahu, kdy generátor při každém vygenerování místnosti vytvoří různě veliký obdélníkový tvar

Místnosti

Uživatel musí definovat, které místnosti v dungeonu chce. Každá místnost bude mít svůj jedinečný název (identifikátor) a přiřazený tvar. Uživatel bude moci určit, kolik daných místností se má v dungeonu vygenerovat (lze zapsat i jako číselný rozsah od-do). U každé místnosti bude moci uživatel určit, zda ji může generátor náhodně otáčet (v krocích o velikosti 90°).

Globální parametry

Kromě místností a jejich tvarů bude moci uživatel určit některá obecná nastavení. Jedním z nich je například *semínko* (číselné nebo textové), které uživateli umožní vygenerovat

tentýž dungeon opakovaně. Pokud uživatel semínko vynechá, použije se náhodné. Dalším parametrem bude explicitní určení *formátu výstupního souboru*, který použije I/O modul v případě, že se má výstup uložit do souboru. Při vynechání se použije stejný formát, který byl použit ve vstupním souboru s konfigurací.

4.2.2 Výstupní data dungeonu

Hlavním důvodem, proč knihovnu uživatel využije, je získání dat reprezentujících dungeon. Aby tato data mohl uživatel využít, je důležité, aby na ně knihovna nabízela více pohledů, z nichž si uživatel vybere pro něj nejvhodnější. Cílem je uživateli co nejvíce usnadnit využití výstupních dat.

Graf místností

Dungeony nejsou nic víc, než kolekce místností propojené chodbami. Uživatel tedy musí vědět, které místnosti dungeon obsahuje a jak jsou propojeny. Výstupní data proto budou obsahovat seznam místností, které vyplývají ze vstupní konfigurace. Každá místnost bude obsahovat vygenerovaný jedinečný číselný *identifikátor*, který představuje identifikátor uzlu v pomyslném grafu. Dále bude u místností uveden *název* místnosti tak, jak jej uvedl uživatel ve vstupní konfiguraci (aby uživatel věděl, o jakou místnost jde, když jich je více různých). Každá místnost musí mít *tvar*, ten bude ve výstupních datech reprezentován maticí znaků reprezentujících prázdnou, resp. vyplněnou buňku. Tvar může vycházet z tvaru zadaného uživatelem nebo se může jednat o tvar vygenerovaný knihovnou v případě, že ve vstupní konfiguraci byl tvar místnosti definován šablonou. U místností nemůže chybět ani *pozice* v rámci dungeonu. Souřadnice místnosti jsou vždy kladné a určují pozici buňky, která je ve tvaru místnosti doplněném na obdélník v levém horním rohu. A konečně – u každé místnosti bude *seznam propojení* – tedy seznam místností, se kterými je místnost přímo propojena chodbou. Místnosti v tomto seznamu budou reprezentovány svými číselnými identifikátory, takže uživateli stačí vyhledat si je v seznamu místností. Díky tomu má uživatel veškeré informace potřebné pro sestavení grafu místností.

Chodby

Výstupní data budou obsahovat také seznam chodeb. Chodby budou ve výstupních datech reprezentovány jako speciální místnosti. U každé bude uveden její *číselný identifikátor*, *tvar* reprezentovaný maticí znaků, *pozice* v dungeonu a *místnosti, které propojuje.*

Mřížka buněk

Pro snadnější použití výstupních dat budou do výstupních dat účelově zanesena redundantní data v podobě mřížky buněk, která bude umožňovat například sekvenční zpracování v cyklu. Každá buňka bude obsahovat *pozici* v dungeonu, její *typ* (např. stěna, místnost, chodba, prázdný prostor atd.) a případně *identifikátor* příslušné místností nebo chodby, ke které náleží.

Další data

Ve výstupní konfiguraci bude uvedeno i *semínko*, podle kterého byl dungeon vygenerován. To je užitečné v případě, kdy uživatel semínko do konfigurace nezadal, a tudíž bylo vybráno náhodně.

4.2.3 I/O modul

Knihovna bude podporovat také souborový vstup a výstup. Uživatel bude moci načíst soubor obsahující vstupní konfiguraci. Soubor se zpracuje a vytvoří se z něj konfigurační struktura stejná, kterou by uživatel mohl vytvořit sám ručně prostřednictvím rozhraní knihovny. Konfigurační struktura se pak předá generátoru jako obvykle.

Do souboru bude možné uložit i dungeon samotný, což je důležité zejména pro použití knihovny nezávisle na programovacím jazyku. K tomuto účelu bude možné knihovnu spouštět jako program, kdy po spuštění bude knihovna očekávat cestu ke konfiguračnímu souboru na standardním vstupu. Na základě přípony tohoto souboru se obsah souboru zpracuje odpovídajícím způsobem. Budou se přitom kontrolovat veškeré náležitosti a syntaktická správnost souborového formátu. V případě chyby v konfiguraci, syntaktické chyby nebo chyby generátoru bude uživatel informován tím, že se místo výstupu vytvoří soubor s názvem `err-XX.err`, kde `XX` bude nahrazeno časovým razítkem spuštění generátoru. Pokud bude generování úspěšné, vytvoří se soubor s názvem `dungeon-XX.YY`, kde `XX` bude nahrazeno časovým razítkem spuštění generátoru a `YY` bude představovat příponu výstupního souboru, která bude určena podle přípony vstupního konfiguračního souboru nebo explicitním určením přípony uživatelem v konfiguraci.

Pro větší použitelnost by měla knihovna podporovat pokud možno více než jeden formát používaný pro přímou výměnu dat. Mělo by se jednat ideálně o snadno lidsky čitelné formáty, které lze programově zpracovat, umožňují zápis kolekcí (například v podobě seznamů) a nabízejí širokou podporu napříč programovacími jazyky, aby bylo možné výsledný dungeon zpracovat bez nutnosti ručního zpracovávání dat. Podle článku na blogu Nordic APIs, skandinávského pořadatele událostí pro uživatele a nadšence rozhraní API, jsou nejpoužívanějšími formáty pro přímou výměnu dat formáty XML, JSON a YAML[14]. I/O modul knihovny by tak měl podporovat alespoň tyto tři.

4.2.4 Generátor

Generátor je nejdůležitější komponentou celé knihovny. V knihovně se bude skládat ze dvou částí – validátoru konfigurace, který se ujistí, že konfigurace obsahuje informace, které jsou ke generování potřeba, a samotný generovací modul, který podle vstupu vygeneruje výslednou strukturu, případně oznámí chybu. Generátor bude obsahovat několik dílčích modulů, které budou mít na starost jednotlivé kroky algoritmu popsaného v části 4.3. Navržený generátor musí splňovat veškeré cílové vlastnosti stanovené v části 4.1.2, což bude ověřeno v kapitole 7.

4.3 Návrh algoritmu generátoru

Algoritmus generátoru, pomocí kterého dojde ke generování výsledné struktury dungeonu podle vstupní konfigurace, bude rozdělen do série kroků, z nichž v každém bude provedena funkčně a logicky oddělená činnost, které jsou však vzájemně závislé. Pokud dojde v kterémkoliv kroku k chybě, generování nemůže dále pokračovat. Krok by mělo být možné opakovat pro případ, kdy jsou k jeho úspěšnému provedení potřeba příznivé vstupně-výstupní podmínky (typicky situace, ve kterých hraje roli náhodný generátor čísel).

Algoritmus se bude podobat algoritmu představenému v části 3.2.1, protože vyhovuje požadavkům knihovny, ačkoliv některé jeho kroky budou přidány, pozměněny nebo úplně vynechány.

Veškeré funkce využívající náhodných jevů budou využívat společný generátor, který bude vytvořen podle stanoveného semínka. Taktéž budou využívány takové kolekce a struktury, které umožňují iterovat prvky v předvídatelném pořadí. Díky tomu bude moci být uživateli nabídnuta možnost vygenerovat tentýž dungeon opakovaně s totožným výsledkem.

4.3.1 Vkládání místností

Celý algoritmus začne vkládáním místností do dungeonu. Nejdříve se načtou požadované místnosti v konfiguraci a vkládací modul je projde jednu za druhou. U každé předepsané místnosti bude načten jejich počet a pokud se jedná o rozsah, vybere se náhodný počet z tohoto rozsahu (s rovnoměrným rozdělením pravděpodobnosti). Následně se místnostem přidělí tvar – buď z konfigurace, nebo se náhodně vygeneruje ze šablony. V případě, že je povolena rotace tvaru, dojde k náhodné rotaci o 0, 90, 180 nebo 270 stupňů. Nakonec budou místnosti umístěny tak, že jejich střed bude ležet na počátku souřadnic dungeonu (bod [0,0]) s mírnou odchylkou.

Obrázek 4.2: Znázornění podoby dungeonu po vložení místností – místnosti se nacházejí v oblasti kolem středu s mírnou odchylkou. Překrytí je znázorněno tmavší šedou barvou.

4.3.2 Rozprostření místností

Po vložení místností bude nutné místnosti rozložit po ploše tak, aby se žádná místnost nepřekrývala. Pro jednodušší kontrolu překrývání bude algoritmus doplňovat tvary místností na obdélník.

Kontrola, zda se dva obdélníky překrývají, je pak jednoduchá. Stačí zkontrolovat, zda je jeden obdélník vlevo od druhého, resp. zda je jeden obdélník pod druhým obdélníkem. Toho lze docílit kontrolou, zda je levá hrana prvního obdélníku více vpravo než pravá hrana druhého obdélníku a naopak, resp. zda je horní hrana prvního obdélníku více dole než dolní hrana druhého obdélníku.

Relativně náročnějším pak je algoritmus pro rozprostření obdélníků po herní ploše. Při implementaci lze vycházet z techniky Separating Axis Theorem (SAT), která říká, že pokud existuje osa, která se neprotíná s projekcemi obou objektů, tyto objekty se nepřekrývají[20]. Součástí této techniky je vyhledávání mezery mezi objekty pomocí projekce objektů na

obecnou osu. V tomto případě jsou obdélníky vždy rovnoběžné s osami souřadnic, a tudíž je situace výrazně jednodušší. Zjednodušeně řečeno stačí vyhledat takový vektor, o který je nutné jeden obdélník po určité ose posunout, aby se přestal překrývat s druhým obdélníkem. Lze přitom otestovat vodorovnou i svislou osu a vybrat nejmenší možný vektor, buď na jedné ose, nebo na obou současně. Tento vektor se poté přičte k souřadnicím obdélníku. Další možnou variantou je určit náhodný úhel, po kterém bude obdélník posouván, a poté vypočítat vektor, o který je nutné obdélník pod daným úhlem posunout, aby se přestal překrývat s jiným obdélníkem. Vzhledem k faktu, že souřadnice v dungeonu neobsahují desetinná místa, nebude tento úhel dodržován úplně přesně.

Důležitým faktorem je pak i způsob iterování mezi obdélníky. Jednou možnou variantou je sériové iterování obdélníků v náhodném pořadí a následné testování kolizí se všemi ostatními obdélníky. Při každé kontrole se získá vektor, o který je nutné první obdélník posunout po ose, aby nekolidoval s druhým. Pokud je velikost tohoto vektoru větší než velikost aktuálně uloženého vektoru, vektor se uloží. Nakonec se obdélník o největší vektor posune a jeho pozice se uzamkne. Takto budou iterovány všechny obdélníky, dokud se nebude žádný překrývat. Druhou možnou variantou je iterování obdélníků v náhodném pořadí, přičemž pro každý z nich se na začátku určí úhel, ve kterém se bude posouvat. Postupným posunováním obdélníku pod tímto úhlem a kontrolováním kolizí budou všechny obdélníky rozprostřeny. Aby však obdélníky lépe „zapadly“ do mezer mezi paprsky tvořenými z již rozprostřených obdélníků, je vhodné k úhlu přičíst nebo odečíst náhodnou odchylku.

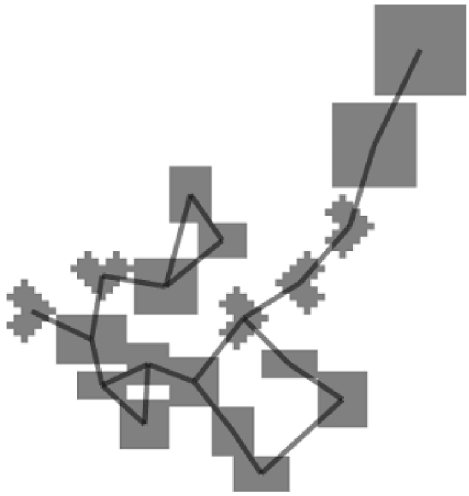
Ze všech zmiňovaných metod bude vybrána ta, která bude mít prostorově nejpřijatelnější výsledky a dosáhne klidového stavu v nejkratším čase. Číselné parametry, úhly a odchylky pro algoritmy se určí experimentálně.

Obrázek 4.3: Znázornění podoby dungeonu po rozprostření místností – místnosti se nikde nepřekrývají a mohou být mezi nimi menší mezery.

4.3.3 Tvorba grafu

Dalším krokem je propojení místností. Cílem je, aby byla každá místnost propojena alespoň s jednou další místností a aby bylo vždy možné najít cestu z jedné místnosti do kterékoliv jiné. K tomuto účelu bude vytvořena množina bodů grafu ze středů místností. Tyto středy budou následně propojeny trojúhelníky metodou Delaunayovy triangulace, která vychází

z podmínky, že uvnitř kružnice opsané libovolnému trojúhelníku se nebude nacházet žádný vrchol jiného trojúhelníku. Následně bude vyhledána minimální kostra grafu, přičemž hrany budou ohodnoceny podle vzdálenosti mezi místnostmi podle Pythagorovy věty (s vynechanou odmocninou pro vyšší výpočetní rychlost). Vzniklá kostra neobsahuje žádné smyčky, což je pro dungeon nezvyklé. Proto budou smyčky do dungeonu zaneseny. Algoritmus se přitom zaměří na vrcholy prvního stupně (tedy místnosti, které jsou propojeny právě s jednou další místností). Tyto místnosti budou s určitou pravděpodobností opět propojeny s jedním ze svých původních sousedů. Důležitá je však vzdálenost mezi těmito místnostmi – Delaunayova triangulace má tendenci vytvářet dlouhé hrany na okrajích grafu, což by způsobilo generování příliš dlouhých chodeb, proto se bude při propojování zohledňovat i vzdálenost – pokud bude mnohem větší než je průměrná vzdálenost mezi místnostmi v dungeonu, k propojení nedojde.



Obrázek 4.4: Znázornění podoby dungeonu po propojení místností – hrany grafu jsou znázorněny šedými čárami. Graf obsahuje tři smyčky, které byly vytvořeny uměle z minimální kostry.

4.3.4 Propojování chodbami

Místnosti, které jsou propojené v grafu, je nutné propojit i fyzicky v dungeonu chodbami. Mohou nastat dvě situace:

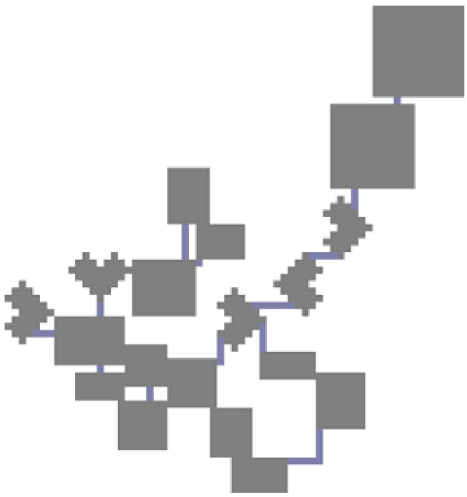
1. Místnosti mají společnou část vodorovné nebo svislé hrany (společné souřadnice X nebo Y v určitém rozsahu) – pokud se místnosti nedotýkají, stačí je propojit rovnou chodbou.

2. Místnosti nejsou pod sebou ani vedle sebe – tyto místnosti se propojí chodbou ve tvaru písmene L, přičemž začátek a konec chodby bude začínat u rohů místností tak, aby nemohlo dojít ke kolizi s jinou místností nebo chodbou. Chodba pak bude mít nejmenší možnou délku a obsah jejího tvaru doplněný na obdélník bude co nejmenší.

V některých případech však ke kolizím dojít může – například při křížení přímých chodeb propojujících dvě a dvě místností. V takových případech lze tento problém vyřešit několika způsoby:

1. Sloučením chodeb, kdy výsledná chodba bude propojovat všechny místnosti současně.
2. Implementací algoritmu pro hledání cesty, výsledná chodba by pak byla několikanásobně lomená čára.
3. Odstraněním propojení mezi místnostmi způsobujících kolizi, případně přepojením těchto místností k jiným místnostem vhodným způsobem.
4. Zahozením dungeonu a vygenerováním nového.

Protože tato knihovna neumožňuje určování pořadí místností za sebou a omezování místností, se kterými jsou propojeny, lze použít variantu číslo 1. Pokud by však v budoucnu došlo k implementaci těchto konfiguračních možností, musel by se problém křížících chodeb řešit alternativním způsobem.



Obrázek 4.5: Znázornění podoby dungeonu po propojení místností chodbami – pokud se místnosti dotýkají, chodba není potřeba. Na obrázku lze pozorovat i chodbu ve tvaru písmene L.

4.3.5 Normalizace souřadnic

Nakonec budou upraveny veškeré souřadnice tak, aby nebyly záporné. Jednoduše se projdou veškeré souřadnice objektů na ose X a na ose Y a uloží se nejvíce záporné hodnoty pro každou z těchto os. Jejich absolutní hodnoty se poté přičtou ke všem souřadnicím všech objektů v dungeonu, čímž dojde k posunutí objektů na rovině dungeonu vpravo dolů. Výsledné souřadnice budou vždy kladné.

Kapitola 5

Implementace navržené knihovny

Tato kapitola obsahuje popis implementačních detailů, zejména použité technologie, strukturu knihovny a informace o implementaci jednotlivých částí. V kapitole již nejsou uváděny teoretické detaily o tom, jak samotné generování funguje. To je vysvětleno zejména v kapitole 4.

5.1 Použité technologie

Knihovna byla vyvinuta v programovacím jazyce **Java** verze 8 SE. Java je široce využívaným programovacím jazykem s rozsáhlou podporou objektových paradigmat, pomocných funkcí a knihoven. Dalšími vhodnými alternativami by byl například jazyk C#, který se široce používá ve vývojářském systému Unity, nebo jazyk C++. Protože je však knihovna koncipována tak, aby byla použitelná z kteréhokoliv jazyka, Java je spíše volba plynoucí z osobní preference.

Pro automatizované sestavování projektu a správu závislostí byl využit systém Apache Maven[1], který na základě obsahu souboru `pom.xml` v kořenovém adresáři dokáže při sestavování automaticky stáhnout a připojit veškeré závislosti a pomocí příslušných pluginů také vytvořit spustitelný balík JAR obsahující veškeré závislosti.

Projekt využívá také množství knihoven. Za zmínku stojí knihovny projektu Apache Commons[2] s užitečnými funkcemi pro práci s řetězci, čísly a soubory, knihovna pro Delaunayovu triangulaci[3] z izraelské Ben Gurionova univerzity v Negevu, knihovna JGraphT[4] pro práci z grafy, ze které knihovna využívá algoritmus pro hledání minimální kostry grafu, a knihovna Jackson[5] pro práci se soubory ve formátu XML a její doplňkové moduly pro práci s formáty JSON a YAML.

5.2 Struktura knihovny

Projekt je rozčleněn do balíků podle jejich účelu, v seznamu je uvedena vždy pouze významová část názvu (bez společné části `cz.iwitrag.procdungen`).

[1] https://maven.apache.org
[2] https://commons.apache.org
[3] https://www.cs.bgu.ac.il/~benmoshe/DT/Delaunay%20Triangulation%20in%20Java.htm
[4] https://jgrapht.org
[5] https://github.com/FasterXML/jackson

- cz.iwitrag.procdungen – předpona všech ostatních balíků (dále bude vynechávána). Obsahuje třídu `Main`, která se spouští při spuštění balíku JAR knihovny.
- api – obsahuje vše, s čím přijde do kontaktu uživatel při práci s knihovnou. Důležitá je třída `ProcDunGenApi`, která je výchozím bodem pro předání vytvořené konfigurace a spuštění generátoru.
- api.configuration – třídy pro ruční (bez souboru) nastavení konfigurace generátoru.
- api.configuration.files – I/O modul konfigurace pro vytvoření konfigurace ze souboru, skládá se zejména z POJO[6] tříd, které se v tomto případě používají k uchovávaní dat načtených ze souboru před jejich zpracováním.
- api.data – třídy obsahující data vytvořeného dungeonu určené pro zpracování uživatelem.
- api.data.files – I/O modul dat s dungeonem pro jejich uložení do souboru.
- generator – obsahuje vše související se samotným generátorem.
- generator.implementations.physical – implementace jediného generátoru knihovny, obsahuje třídy pro zpracování jednotlivých fází generování.
- generator.templates – třídy pro generování tvarů místností podle šablon.
- util – pomocné třídy pro práci s tvary a čísly.

5.3 Konfigurační API

Pro vytvoření konfigurace předávané generátoru obsahuje knihovna třídy, které uživatel jednoduše naplní požadovanými údaji. Nejdůležitější je třída `DungeonConfiguration`, která sdružuje veškeré konfigurační náležitosti v sobě: tvary místností, místnosti samotné, požadovaný typ výstupního souboru (nepovinné) a explicitně zadané semínko (nepovinné). Kromě metod pro nastavování výše uvedeného obsahuje tato třída statickou metodu `fromFile()` pro vytvoření instance konfigurace ze souboru a metodu `validate()` pro ověření správnosti konfigurace. Tuto metodu knihovna volá před spuštěním generátoru automaticky.

Třída `RoomShape` představuje tvar místnosti. Tvar lze určit přímo jako matici znaků (pomocí třídy `TextualShape`) nebo pomocí šablony (třída `RoomTemplate`). V případě, že chce uživatel použít předem připravenou šablonu, musí zjistit její název (například z dokumentace) a parametry, které přijímá. Pokud budou tyto údaje chybět nebo budou neúplné či nesprávně zadané, uživatel bude informován.

Místnosti lze definovat pomocí třídy `Room`. Každá místnost má určitý název a tvar a dále musí mít uveden počet. Ten se uvádí jako `IntRange`, tedy celočíselný rozsah. Generátor poté vybere hodnotu z rozsahu, což zvyšuje variabilitu. U každé místnosti lze i nastavit, zda může generátor tvar náhodně otáčet.

[6] **POJO** – Plain old Java object, obyčejný Java objekt bez jakýchkoliv restrikcí ze strany návrhových vzorů, konvencí nebo frameworků

5.4 Zapouzdření výstupních dat

Výstupní data dungeonu jsou zapouzdřena ve třídě `Dungeon`, která obsahuje vygenerované místnosti (třída `Room`) a chodby (třída `Corridor`), mřížku buněk tvořících dungeon (třída `Grid`) a použité semínko (definované uživatelem nebo náhodně vybrané). Třída dále obsahuje metody `toYaml()`, `toJson()` a `toXml()`, které zapisují výstupní data do souboru v příslušném formátu.

Mřížka získaná voláním metody `getGrid()` se generuje, jakmile je potřeba, a je poté uložena. V případě, že by došlo k úpravě dungeonu (např. smazání místnosti), uložená mřížka se smaže, protože je neaktuální. Mřížka vzniká tak, že se v cyklu procházejí jednotlivé místnosti a chodby, získá se jejich pozice (levý horní roh tvaru doplněného na obdélník) a poté se v cyklu propisují buňky jejich tvaru do mřížky s relativními pozicemi. Mřížka je tvořena buňkami reprezentovanými třídou `GridCell`. Každá buňka obsahuje pozici v rámci mřížky, typ a případnou referenci na příslušnou místnost nebo chodbu. Třída `Grid` je univerzální a používá se kromě dungeonu i pro samotné tvary místností a chodeb na výstupu.

Místnosti a chodby jsou podtřídami třídy `DungeonRectangle`, která implementuje rozhraní `Rectangle`. To znamená, že obě třídy obsahují tvar reprezentovaný mřížkou, pozici v dungeonu a metody pro získání rohů a středu a detekci průniku. Kromě toho obsahují i přidělený náhodný číselný identifikátor a seznam propojujících místností (ve třídě `Room` to značí sousední připojené místnosti a ve třídě `Corridor` to značí místnosti, které chodba propojuje).

5.5 Práce se soubory

Vstupní konfigurace a výstupní data nabízejí podporu souborů. Ta je realizována prostřednictvím knihovny **Jackson** a jejích přídavných modulů. Podporovány jsou formáty JSON, YAML a XML. Data obsažená v souborech jsou reprezentována POJO třídami s příslušnou strukturou a anotacemi vyžadovanými knihovnou Jackson.

Vytvoření vstupní konfigurace ze souboru začíná zavoláním statické metody `fromFile()` ve třídě `DungeonConfiguration`. Podle přípony souboru se vytvoří příslušný `ObjectMapper` knihovny Jackson. Tento objekt je zodpovědný za čtení a ukládání dat v konkrétním formátu. Následně je `ObjectMapper` nakonfigurován, například aby nerozlišoval velikost písmen u jednotlivých položek (chceme, aby byla práce se soubory pro uživatele co nejpříjemnější). Nakonec je zavolána jeho metoda `readValue()` s parametrem v podobě třídy `FileConfigurationPojo`. Metoda vytvoří instanci tohoto objektu POJO a naplní jej daty. Dojde při tom k vytvoření podřazených objektů POJO, které třída obsahuje v sobě. Tyto objekty POJO jsou nakonfigurovány pomocí anotací tak, aby je knihovna Jackson úspěšně zpracovala. Pokud byla data úspěšně načtena, jsou transformována do podoby instance `DungeonConfiguration`, která je poté vrácena.

Uložení výstupních dat dungeonu do souboru funguje obdobným způsobem. Vše začíná zavoláním metody `toYaml()`, `toJson()` nebo `toXml()` (podle požadovaného formátu). Metoda vytvoří příslušnou instanci `ObjectMapper` pro daný formát, nakonfiguruje jej a následně převede veškerá výstupní data do POJO objektů, které lze poté zapsat do souborů.

Kromě volání příslušných metod lze I/O modulu využít také spuštěním JAR souboru knihovny. Tím se zavolá metoda `main()`, která pracuje následovně:

1. Čtení standardního vstupu pomocí čteček `InputStreamReader` a `BufferedReader`, na kterém knihovna očekává cestu ke konfiguračnímu souboru.

2. Vytvoření instance třídy `File` se souborem a její předání statické metodě `fromFile()` třídy `DungeonConfiguration` pro načtení konfigurace ze souboru.

3. Vygenerování dungeonu voláním metody `generateDungeon()` třídy `ProcDunGenApi`.

4. Uložení dungeonu do souboru. Výstupní formát se načte z konfigurace (pokud jej uživatel určil), případně se použije formát vstupního konfiguračního souboru.

Pokud během volání metody dojde k chybě, místo souboru s daty dungeonu se vytvoří soubor s chybou.

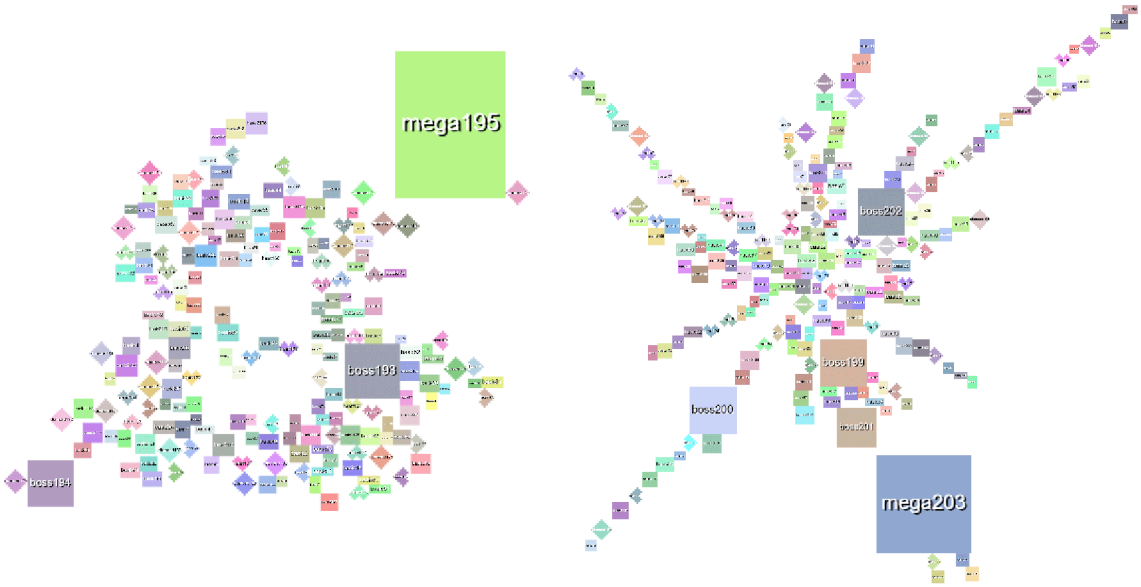
5.6 Implementace generátoru

Knihovna obsahuje jediný generátor, který načítá vstupní konfiguraci a na jejím základě generuje výstupní obsah. Každá instance generátoru reprezentovaného třídou `ConfiguredGenerator` obsahuje svou vlastní konfiguraci `DungeonConfiguration`, podle které po zavolání metody `generate()` pracuje. Generátor je možné zastavit po určité fázi generování nastavením proměnné `stopAfterPhase`, která uchovává instanci výčtového typu `ConfiguredGenerator.Phase`. Po spuštění generátoru si generátor vytvoří interní generátor náhodných čísel `RandomGenerator`, který je poté distribuován volaným komponentám zajišťujícím jednotlivé fáze generování. Tím je zajištěno, že veškeré náhodné jevy ve všech fázích budou podléhat určenému semínku. Pokud během kterékoliv fáze dojde k chybě, ze které se nelze zotavit, vyvolá se výjimka `GeneratorException` s příslušnou chybovou zprávou.

Generátor začíná svou činnost **tvorbou místností**, přičemž jako parametry přijímá počátek, kolem kterého budou místnosti vkládány, a odchylky na osách X a Y. Poté jsou načteny místnosti definované v konfiguraci a k nim přiřazené textové tvary nebo šablony. Pokud se jedná o textově zadaný tvar, místnost je vytvořena přímo, v opačném případě je zavolána třída `TemplateFactory` a jsou jí předány parametry z konfigurace. K dispozici jsou dvě základní šablony – „square“ a „rectangle“, které přijímají rozměry na jednotlivých osách s možností zadání číselného rozsahu od-do. V případě chyby je vyvolána výjimka `TemplateException`. Výsledkem je textový tvar `TextualShape`, se kterým se dále pracuje již běžným způsobem. Generátor vygeneruje počet místností, který je uveden v konfiguraci u dané definice. Pokud je pro danou místnost povoleno otáčení tvarů, tvar bude náhodně otočen o 0, 90, 180 nebo 270 stupňů. Nakonec jsou místnosti přidány do dungeonu, v této fázi mohou mít i záporné souřadnice. Ty budou normalizovány později.

Dalším krokem je **rozprostření místností** po herní úrovni. Na začátku je iterován seznam místností v náhodném pořadí. Pro každou místnost je vybrán náhodný směr od její počáteční pozice (0-359 stupňů). Následně se procházejí všechny ostatní místnosti a je kontrolováno, zda se s iterovanou místností neprotínají. Pokud ano, voláním metody `getDifferenceInGivenAngle()` je vypočítán vektor, o který se musí iterovaná místnost přesunout, aby v kolizi nebyla. Protože jsou souřadnice celá čísla, úhel vypočítaného vektoru nemusí být exaktně přesný. Následně je místnost v daném směru posunuta a kontrola kolizí probíhá od začátku.

Tento systém teoreticky zajišťuje rovnoměrné rozprostření místností po ploše. Realita je ovšem odlišná. Prvním problémem jsou příliš velké místnosti. Pokud se v konfiguraci nachází obrovská místnost, může dojít k tomu, že se přesune až jako jedna z posledních, a tudíž po ní na počáteční pozici zůstane velký nevyplněný prostor (viz obrázek 5.1 vlevo).



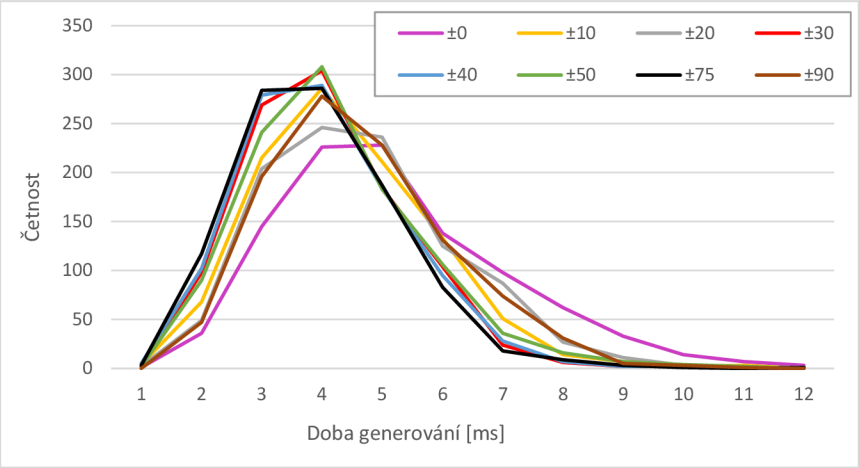
Obrázek 5.1: Vizualizace problémových situací po rozprostření místností. Na obrázku vlevo zanechala místnost *mega195* uprostřed prázdný prostor, který zůstal z velké části nevyplněn. Na obrázku vpravo je znázorněn problém, kdy posunováním obdélníkových místností pod fixním úhlem dochází k jejich řazení do paprsků za sebou.

Z tohoto důvodu byl zaveden seznam místností, které jsou již na své finální pozici, a kontrola kolizí probíhá pouze vůči těmto místnostem.

Dalším problémem je, že v rámci optimalizace rychlosti algoritmu probíhá kontrola kolizí s tvary doplněnými na obdélník. Metoda posunování obdélníkových místností pod náhodným, ale stále fixním úhlem pak má tendenci vytvářet „paprsky“ místností v řadě za sebou (viz obrázek 5.1 vpravo). Tento problém byl vyřešen tak, že se při kontrole kolizí a získávání kolizního vektoru použije náhodná odchylka úhlu v daném rozsahu. Díky tomu se může místnost při umísťování náhodně vychýlit ze svého kurzu a tím zaplnit prázdný prostor. Použitý rozsah pro úhlovou odchylku pak má vliv nejen na výsledný tvar dungeonu, ale také na dobu, za jakou se podaří místnosti rozprostřít bez kolizí (viz graf 5.2, který je pro lepší znázornění rozdílů prezentován jako histogram dob rozprostírání místností generátorem).

Z grafu je patrné, že ideální odchylkou pro úhel z hlediska rychlosti bude ±30, 40 nebo 75 stupňů, přičemž testováním bylo zjištěno, že hodnota 30 stupňů v malé míře vykazuje tentýž problém s paprsky a hodnota 75 stupňů už je zase příliš nezajímavá, protože sdružuje všechny místností příliš striktně uprostřed v kruhu. Proto byla vybrána úhlová odchylka ±40 stupňů.

Rozprostřené místnosti je dále nutno **propojit**. Nejdříve je vytvořena nová instance třídy `Delaunay_Triangulation` z knihovny pro Delaunayovu triangulaci a do jejího konstruktoru jsou předány středy místností jakožto uzly. Tyto uzly jsou reprezentovány polem instancí `Point_dt` pro trojrozměrné body – třetí rozměr je nevyužit. Následně je využit iterátor získaný metodou `trianglesIterator()`, který umožňuje iterovat vytvořenými trojúhelníky (třída `Triangle_dt`). Tyto trojúhelníky jsou poté převedeny na sadu trojic vzájemně propojených místností. Každá místnost přitom již v sobě obsahuje informace o tom, se kterými místnostmi je propojena.



Obrázek 5.2: Histogramy dob rozprostírání místností v závislosti na použité odchylce úhlu. Odchylky uváděné v legendě jsou ve stupních. Pro každou odchylku z legendy bylo realizováno 1000 generování s konfigurací čítající 150 až 300 místností. Měřeno na počítači s procesorem AMD Ryzen 5 2600, 8 GB RAM, v prostředí Java 11 s přidělenými 2 GB RAM a na systému Windows 10.

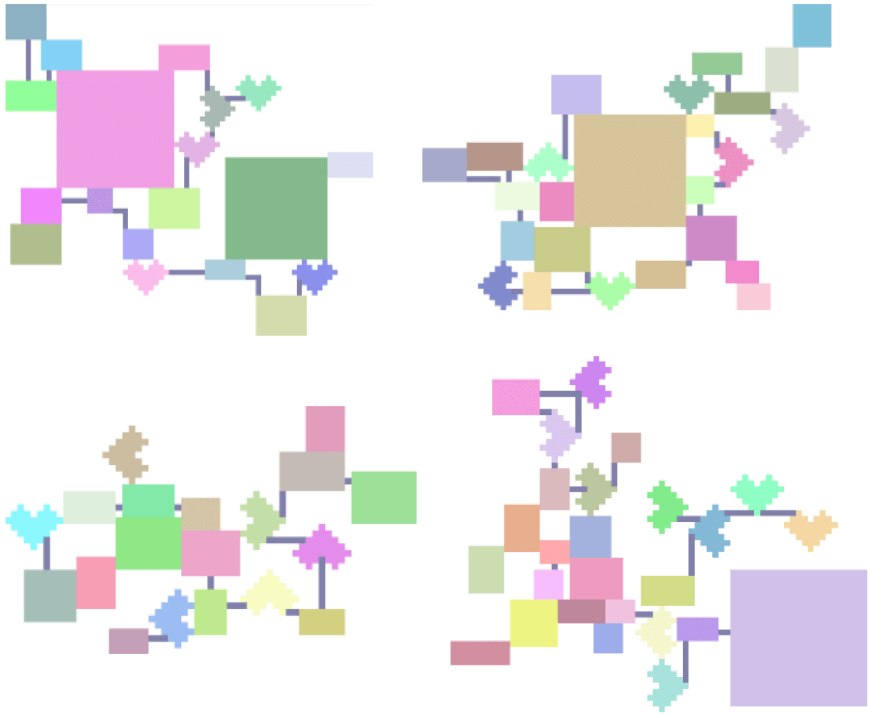
Následně je spuštěn algoritmus pro hledání minimální kostry grafu z knihovny JGraphT. Nejdříve je vytvořena interní reprezentace grafu, se kterým knihovna pracuje (třída `SimpleWeightedGraph`). Ten je naplněn místnostmi dungeonu jakožto body. Poté jsou přidány hrany mezi body podle toho, jak byly propojeny místnosti v předchozím kroku. Hrany jsou následně ohodnoceny podle vzájemné vzdáleností středů místností (bez druhé odmocniny). Takto připravený graf je předán implementaci algoritmu (třída `PrimMinimumSpanningTree`) českého matematika Vojtěcha Jarníka (též znám jako Primův algoritmus). Poté jsou zahozena existující propojení místností, jsou iterovány hrany výsledné struktury a místnosti jsou poté opětovně propojeny.

V poslední propojovací fázi jsou s určitou pravděpodobností (50 %) propojovány místnosti, které jsou propojeny pouze s jednou další místností. K tomuto účelu jsou vybrány místnosti, se kterými byla místnost propojena po Delaunayově triangulaci. Tím jsou do grafu uměle zaneseny smyčky. K propojení ovšem dojde pouze tehdy, je-li ohodnocení hrany mezi místnostmi menší nebo rovna průměrnému ohodnocení (aby posléze nedocházelo k tvorbě příliš dlouhých chodeb). Protože však ohodnocení nepředstavuje lineární vzdálenost mezi uzly grafu (nebyla využita odmocnina), jako mezní hodnota se používá suma všech ohodnocení vydělená počtem hran vynásobeným třemi. Tato hodnota byla určena experimentálně a vykazuje dobré výsledky pro malé i velké dungeony.

Poslední důležitou fází generátoru je **tvorba chodeb**. Chodby se vytvářejí mezi dvojicemi v grafu propojených místností. K tomu slouží metoda `createCorridor()`, která má za úkol místnosti propojit chodbou. Nejdříve je vypočítán přesah hran tvarů místností doplněných na obdélník po ose X a po ose Y. Pokud přesah existuje, je proveden test toho, zda nejsou místnosti propojeny přirozeně (vzájemnou blízkostí). Pokud nejsou, jsou stanoveny hrany, mezi kterými se chodba vytvoří. Chodba se vytvoří ve středu rozsahu, ve kterém mají místnosti přesah. Před zavoláním metody `createDirectCorridor()`, která vytváří chodbu mezi dvěma body, jsou tyto body protaženy k samotným tvarům místností (a nikoliv pouze k tvarům doplněných na obdélníky). Pokud místnosti přesah nemají, bude

vytvořena chodba ve tvaru písmene L. Nejdříve je zjištěna vzájemná poloha místností. Aby se minimalizovalo riziko kolize s jinou místností nebo chodbou, jsou vybrány nejbližší rohy místností. Chodba je tak vždy pokud možno co nejkratší. Následně jsou vytvořeny dvě přímé chodby, které se potkávají ve společném bodě mezi místnostmi (střední vrchol písmene L). Tyto přímo chodby jsou poté sloučeny v jednu.

Po vygenerování chodeb je celý dungeon **normalizován** – veškeré místnosti i chodby jsou posunuty směrem vpravo dolů do kladných souřadnic. Bez této úpravy by nebylo možné pracovat s mřížkou dungeonu reprezentovanou třídou `Grid`, která podporuje pouze nezáporné souřadnice. Na konci generátor vrátí strukturu herní úrovně reprezentovanou třídou `Dungeon`. Na obrázku 5.3 jsou znázorněny různé dungeony, pro jejichž generování byla použita stejná konfigurace.



Obrázek 5.3: Ukázka dungeonů po dokončení generování. Ve všech případech byla použita totožná konfigurace (největší čtvercová místnost byla označena jako nepovinná). Dungeony vykazují různorodou propojenost i tvary.

Kapitola 6

Tvorba vizualizační webové aplikace

Souběžně s vývojem knihovny probíhal také vývoj jednoduché webové aplikace, která dokáže její výstupy vizualizovat. Některé obrázky publikované v této práci pochází právě z této webové aplikace. Možnost vizualizovat výsledky generované knihovnou její vývoj značně usnadnil, protože vizuální kontrola je lidsky přijatelnější a efektivnější než kontrola výstupu v textové nebo objektové podobě. Vývojář si může díky této vizualizaci určit konfiguraci dle libosti a udělat si obrázek o tom, jak výsledné úrovně vypadají. Tato kapitola popisuje, jak byla tato jednoduchá webová aplikace navržena, obsahuje souhrn použitých technologií a implementační detaily.

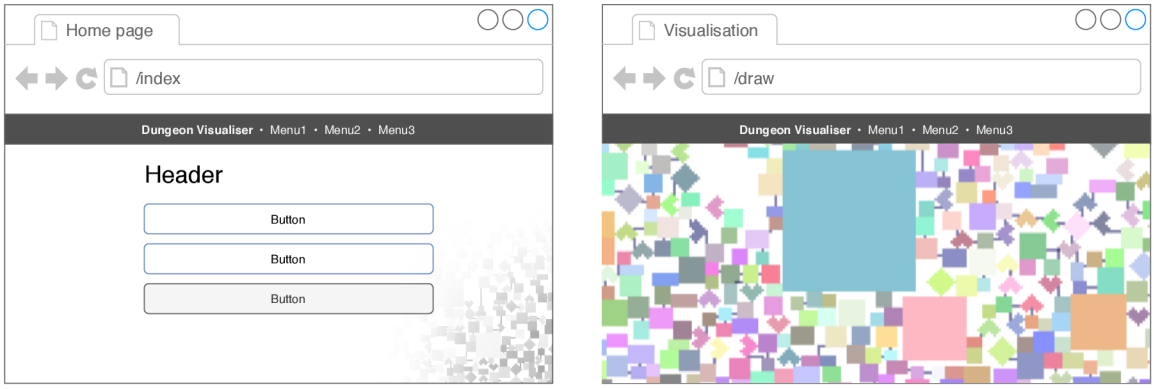
6.1 Analýza požadavků a návrh

Primárním účelem webové aplikace bude **vizualizace výstupních dat** z knihovny vytvořené v předchozích kapitolách. Webová aplikace proto musí umět zpracovat výstupní data a využít je k vykreslení místností a chodeb dungeonu. Místnosti dungeonu však nejsou pouhé čtverce nebo obdélníky – jsou reprezentovány tvary složenými z buněk, které musí být přesně vykresleny. Nad danou místností pak bude uveden název místnosti a její jedinečný číselný identifikátor. Uživatel by měl být schopen dungeon jednoduše **procházet** a **přibližovat či oddalovat**. Neméně důležitým aspektem je pak **vykreslení souřadnic** myši, které musí být přepočítány do souřadnic dungeonu.

Webová aplikace bude spouštět generátor a vizualizovat jeho výstup. Pro spuštění generátoru je potřeba konfigurace, kterou webová aplikace umožní **zadat přímo** (a vybrat typ souboru ručně) nebo ji **nahrát ze souboru**. O všech chybách musí být uživatel informován. Uživatel si bude moci určit **nastavení generátoru**, například které fáze má vykonat nebo zda má vynechat vykreslování některých prvků dungeonu.

Protože se jedná spíše o webovou pomůcku než o plnohodnotnou webovou aplikaci, není potřeba implementovat žádný systém přihlašování. Aplikaci by mělo jít **snadno spustit** a provádět vizualizace bez zdlouhavých procedur.

Na obrázku 6.1 je vyobrazen jednoduchý návrh. Webová aplikace bude velmi jednoduchá – bude obsahovat úvodní stránku s nabídkou, dva formuláře (pro souborový a ruční vstup konfigurace), stránku se základními informacemi o webové stránce a samotnou stránku s vizualizací.



Obrázek 6.1: Návrh webové aplikace pro vizualizaci dungeonů. Vlevo je znázorněno základní rozložení úvodní stránky a vpravo vizualizace dungeonu. Vizualizace bude využívat největší možný dostupný prostor.

6.2 Implementační detaily

Pro implementaci byl vybrán jazyk Java a technologie Java EE – konkrétně aplikační Spring 5[1] a podprojekt Spring MVC. Pro prezentační vrstvu byly použity technologie HTML5, CSS3 a sada Bootstrap 4[2]. Pro vývoj šablon byla použita technologie JSP (JavaServer Pages)[3], která umožňuje jejich kompilování do servletů. Tato technologie byla doplněna o knihovnu dalších standardních značek pro šablony (JSTL – Standard Tag Library), například cyklus `foreach`. Zvažován byl také systém Thymeleaf[4], ukázalo se však, že je pro potřeby aplikace nevhodný, protože vykazuje značný pokles výkonu při vpisování dat do šablon s kódem v jazyce JavaScript. Jazyk JavaScript byl totiž základem samotné vizualizace. Ta je realizována prostřednictvím HTML5 prvku `Canvas`, na který probíhá vykreslování. Samotné vykreslování byla provedeno pomocí knihovny PixiJS[5], která veškerou práci usnadňuje a utilizací JavaScriptového API WebGL (Web Graphics Library) pro rendrování na grafické kartě podstatným způsobem zvyšuje výkon. Pohyb po vizualizaci, její přibližování a oddalování a podporu gest na mobilních zařízeních přidává knihovna Pixi Viewport[6]. Veškeré uváděné závislosti byly spravovány systémem Apache Maven[7]. Stejným způsobem byla do projektu vložena také vyvinutá knihovna pro procedurální generování dungeonů.

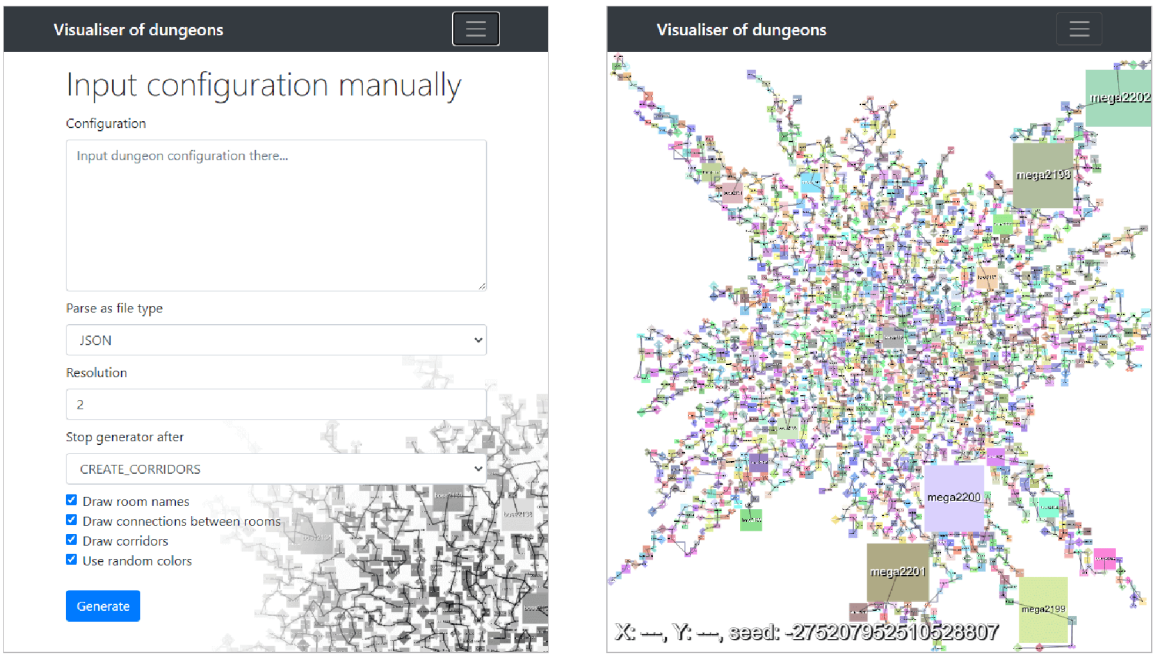
Po otevření úvodní stránky v prohlížeči a po obdržení HTTP požadavku se aktivuje dispečer `DispatcherServlet`, který podle URL adresy vyvolá příslušný ovladač (angl. controller), jehož úkolem je požadavek zpracovat a navrátit příslušný pohled (angl. view). V případě úvodní stránky bude zavolán `IndexController`, který vrátí pohled `index`. Tento pohled obsahuje pouze tělo samotné stránky (záhlaví a navigační tlačítka v těle stránky), přičemž záhlaví a zápatí jsou do stránky vloženy ze složky `views/parts`, která je určena pro dynamicky vkládané části. Takto složená odpověď je poté zaslána uživateli. Jakmile uživatel přejde na stránku pro vložení konfigurace, vyplní potřebné údaje a nastavení a data odešle, dojde k zavolání ovladače `ManualController`, resp. `FromFileController` s modelem na-

[1] https://spring.io
[2] https://getbootstrap.com
[3] https://www.oracle.com/java/technologies/jspt.html
[4] https://www.thymeleaf.org
[5] https://www.pixijs.com
[6] https://github.com/davidfig/pixi-viewport
[7] https://maven.apache.org

plněným daty. Po základním ověření (např. případných chybějících údajů) webová aplikace vyvolá metodu `GenerateDungeonPartially()` a získá výstupní strukturu. Pokud dojde k chybě v konfiguraci nebo generátoru, ovladač vrátí pohled `error` s chybovou zprávou a výpisem zásobníku získaným z výjimky. Pokud bude generování úspěšné, dojde k vyvolání pohledu `draw`, do kterého bude vložen obsah souboru `visualisation.jsp`. Tento soubor obsahuje kód v jazyce JavaScript zodpovědný za vizualizaci. Do pohledu bude i předán vygenerovaný dungeon, který bude sloužit jako zdroj dat ve vizualizaci. Uživateli se zobrazí očekávaná stránka a většinu obsahu bude tvořit zmiňovaná vizualizace.

6.3 Výsledná podoba a vyhodnocení

Výsledná webová stránka splňuje všechny požadavky uvedené v části 6.1. Její podoba je znázorněna na obrázku 6.2. Webová stránka umožňuje vizualizovat dungeony generované knihovnou a umožňuje uživateli provést konfiguraci před samotnou vizualizací. Jako možná vylepšení by bylo možné přidat konfiguraci vizualizace bez nutného opětovného generování, ukládání výstupní konfigurace (do textové nebo obrázkové podoby), případně ukládání samotných konfigurací a jejich snadnou úpravu (například možnost určit tvar místností klikáním do matice bodů). Aplikace umožňuje vizualizaci i na mobilních zařízeních a obsahuje podporu gest, ačkoliv u větších dungeonů může vizualizace na mobilních zařízeních trvat i desítky sekund. Aplikace je dodávána v souboru WAR (Web Application Resource), který umožňuje její velmi snadné zprovoznění na webovém serveru. Lze ji tak snadno využít k rychlé a efektivní vizualizaci výstupních struktur knihovny.



Obrázek 6.2: Výsledná podoba webové stránky pro vizualizaci dungeonů. Vlevo je znázorněn formulář pro ruční zadání konfigurace a nastavení generátoru, vpravo je pak samotná vizualizace.

Kapitola 7

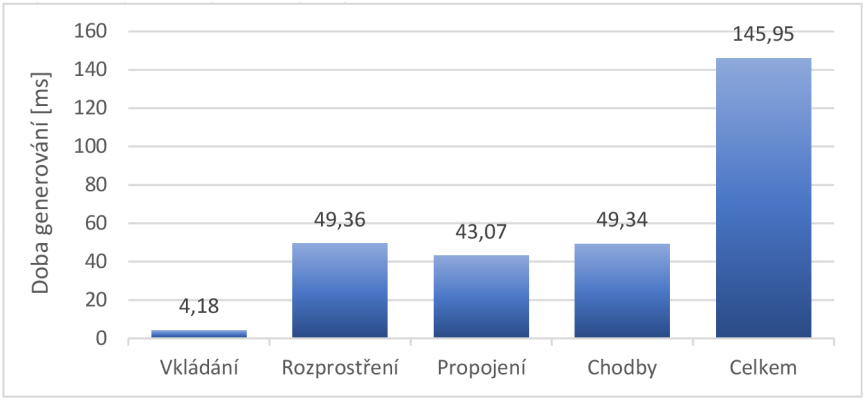
Testování vytvořené knihovny

Při návrhu knihovny byly v části 4.1.2 stanoveny cílové vlastnosti, které by měla vytvořená knihovna splňovat. V této kapitole bude knihovna posouzena z hlediska těchto vlastností. Prověřena bude především praktická využitelnost výstupu generátoru. Nakonec bude provedeno srovnání s existujícími řešeními uvedenými v kapitole 3.

7.1 Rychlost generování výstupu

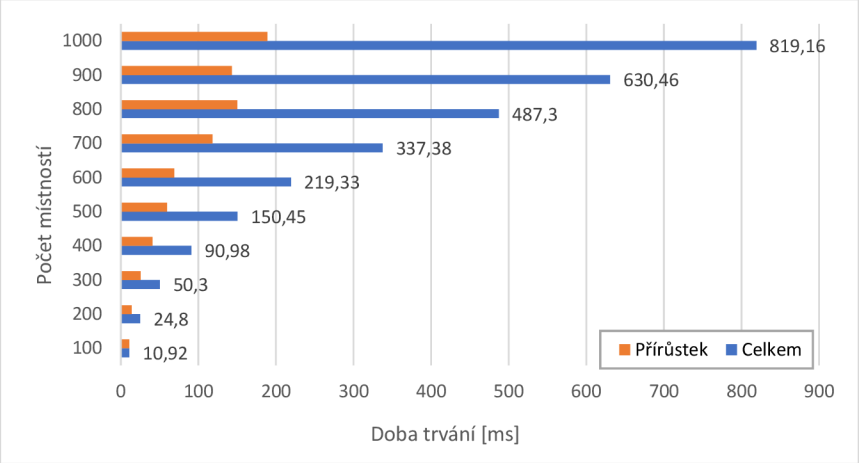
Rychlost generování hraje důležitou roli. Představuje totiž čas, který může hráč potenciálně strávit sledováním načítací obrazovky. Vývojář si bude s vysokou pravděpodobností výstup dále upravovat a přidávat další prvky, což bude dobu přípravy herní úrovně ještě více prodlužovat. Proto je nutné, aby knihovna svou činnost dokončila co nejrychleji. Veškerá měření rychlosti v této části probíhala na počítači s procesorem AMD Ryzen 5 2600, 8 GB RAM, v prostředí Java 11 s přidělenými 2 GB RAM a na systému Windows 10.

Při každém měření bylo sledováno trvání jednotlivých fází generátoru zvlášť a celkové trvání. Bylo provedeno 100 generování dungeonu s konfigurací čítající přesně 500 místností. Každé generování bylo provedeno s náhodným semínkem a výsledky (průměrné doby trvání) jsou znázorněny v grafu 7.1. Navzdory vysokému počtu místností (500 místností uživatel pravděpodobně nebude nikdy potřebovat, typické dungeony obvykle obsahují maximálně desítky místností) trvalo generování průměrně desetinu sekundy.



Obrázek 7.1: Rychlost generování dungeonů během jednotlivých fází a celkem. Jedná se průměrné doby při generování 100 dungeonů s konfigurací čítající přesně 500 místností.

Dále bylo analyzováno celkové trvání generování v závislosti na počtu místností. Opět bylo vždy vygenerováno 100 dungeonů a počty místností byly navyšovány po stovkách. Průměrné doby trvání jsou uvedeny v grafu 7.2. Z grafu je patrné, že se doba trvání spolu s počtem místností zvyšuje exponenciálně a nikoliv lineárně. To je očekávané chování, protože například při rozprostírání místností se musí při přemísťování každé místnosti kontrolovat kolize se všemi ostatními.



Obrázek 7.2: Rychlost generování dungeonů v závislosti na počtu místností. Jedná se průměrné doby při generování 100 dungeonů.

7.2 Parametry a přizpůsobitelnost výstupu

Knihovna umožňuje definovat počet a tvar místností, které jsou poté rozmístěny náhodně a náhodně propojeny chodbami. Z tohoto důvodu je knihovna vhodná spíše pro generování strukturovaných dungeonů, a nikoliv pro generování úrovní s menší mírou uspořádanosti (například jeskyní).

Výstupní struktura knihovny je fixně určena, nabízí však uživateli možnost doplnit vygenerované místnosti a chodby o prvky podle libosti (například nepřátele nebo poklady pro hráče). Uživatel má k dispozici výsledné tvary místností i chodeb v přehledné maticové struktuře.

7.3 Intuitivnost a jednoduchost použití

Pro vygenerování dungeonu je potřeba spustit jedinou metodu a předat jí konfiguraci. Uživatel si může vybrat, zda použije nabízené rozhraní API nebo zadá konfiguraci v souboru. Další možností je přímé spuštění knihovny a zadání vstupní souborové konfigurace, kdy knihovna vygeneruje výstupní soubor, případně soubor s chybovou zprávou. Uživatel nejdříve určí tvary místností a ty přiřadí místnostem. Pro zjednodušení má uživatel k dispozici šablony generující náhodné tvary. Tvary však může vytvářet i ručně jednoduchým zapsáním matice znaků. Další ukázkou jednoduchosti použití může být výpis 7.1, kde je demonstrováno generování dungeonu v počítačové hře pomocí několika málo řádků kódu. Výstupní struktura pak obsahuje samotné místnosti a chodby. Uživatel získá informace o jejich pozici, tvaru a vzájemného propojení. Dalším možným způsobem, jak přistupovat k výstupním datům, je

mřížka buněk dungeonu, přičemž každá buňka obsahuje pozici, typ a případně i odkaz na příslušnou místnost či chodbu, kterou reprezentuje.

Knihovna je doplněna o JavaDoc komentáře, které umožňují vygenerování textové dokumentace pro koncové uživatele. Díky tomu má uživatel přehled nad tím, jak knihovnu používat. Co se týče případných chyb, uživatel je informován o chybách v konfiguraci, chybách v generátoru, případně ostatních výjimkách (např. vstupně-výstupní chyby při práci se soubory) a měl by tak být schopen chybu opravit.

7.4 Praktická využitelnost

Funkčnost knihovny byla ověřena i prakticky přímo v počítačové hře. K tomuto účelu byla vybrána hra *Minecraft*, která umožňuje přetvářet herní svět ze stavebních bloků, což z ní činí dobrou volbu, vzhledem k povaze výstupní struktury knihovny, která je složena z diskrétních částí (buněk). Knihovna byla do hry implementována jako plugin serverového API Spigot MC[1]. Spigot MC funguje jako prostředník mezi serverovou částí jádra hry Minecraft a vývojáři a umožňuje jim prostřednictvím pluginů modifikovat jeho funkčnost. Plugin napsaný pro tuto práci funguje následovně:

1. Vytvoř instanci třídy `WorldCreator` a nastav jej tak, aby vytvářel ploché světy.
2. Pomocí třídy `Bukkit` a statické metody `createWorld()` vytvoř nový svět (pokud již existuje, pouze se načte a tím činnost pluginu končí).
3. Pokud svět předtím neexistoval, vytvoř jednoduchou konfiguraci dungeonu a ten vygeneruj.
4. Projdi mřížku ve výstupní struktuře a případné místností a chodby umísti do nově vygenerovaného světa.

Tímto jednoduchým způsobem je možné vytvořit funkční dungeon přímo ve hře (viz výpis kódu 7.1).

```
1  // Generate dungeon
2  DungeonConfiguration dungeonConfiguration = createConfiguration();
3  Dungeon dungeon = null;
4  try {
5      dungeon = new ProcDunGenApi().generateDungeon(dungeonConfiguration);
6  } catch (InvalidConfigurationException | GeneratorException ex) {
7      ex.printStackTrace();
8  }
9
10 // Place dungeon
11 if (dungeon != null && newWorld != null) {
12     for (GridCell gridCell : dungeon.getGrid()) {
13         if (gridCell.getType() == GridCell.Type.NONE)
14             continue;
15         Material material = Material.GOLD_BLOCK;
16         int iterations = 1;
17         if (gridCell.getType() == GridCell.Type.ROOM) {
18             material = Material.IRON_BLOCK;
```

[1] https://www.spigotmc.org

```
19              iterations = 5;
20          }
21          else if (gridCell.getType() == GridCell.Type.CORRIDOR) {
22              material = Material.LAPIS_BLOCK;
23              iterations = 2;
24          }
25          for (int i = 0; i < iterations; i++) {
26              Block block = newWorld.getBlockAt(gridCell.getPosition().getX(),
                    80 + i, gridCell.getPosition().getY());
27              block.setType(material, false);
28          }
29      }
30  }
```

Výpis 7.1: Ukázka praktického použití knihovny ve hře Minecraft. Nejdříve je jednoduše vygenerován dungeon a ten je poté za pomoci mřížky vložen do herní úrovně.

Výsledek je znázorněn na obrázku 7.3. Demonstrace je velmi jednoduchá a slouží pro účely této práce. Aby bylo možné dungeon projít, bylo by nutné algoritmus pluginu dále upravit, aby vytvářel „duté“ místnosti a aby byly chodby širší. K tomuto účelu stačí upravit měřítko algoritmu, aby pro jednu buňku mřížky ve výstupní struktuře vytvořil nikoliv jeden blok, ale 2x2 nebo více bloků, čímž vznikne prostor pro další manipulaci.

Obrázek 7.3: Náhodný dungeon vygenerovaný pomocí vytvořené knihovny ve hře *Minecraft*.

7.5 Spolehlivost generátoru

Výstup generátoru vždy odpovídá zadané vstupní konfiguraci. Uživatel může nepovinné místnosti označit tak, že při zadávání jejich počtu připustí nulový počet. Generátor zajišťuje, že celý dungeon bude možné projít a že žádná místnost nezůstane nepropojena.

Při testování bylo odhaleno, že u velmi velikých dungeonů občas dochází k protnutí chodby a cizí místnosti. Tuto chybu lze opravit implementací složitějšího algoritmu pro hledání cesty chodby, což však pravděpodobně bude mít negativní dopad na celkovou rychlost

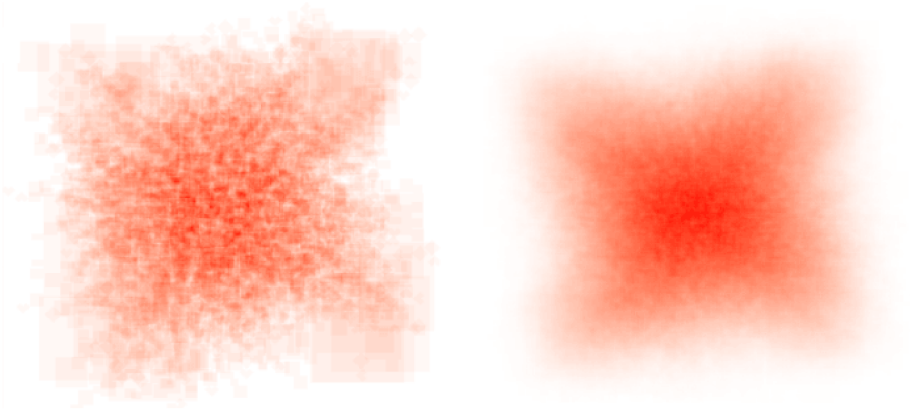
generátoru. Dalším zjištěným problémem je, že větší místnosti mají tendenci být „vytlačeny“ k okrajům místnostmi menšími. Tento problém se dá z části vyřešit tím, že se před rozprostřením místností stanoví takové pořadí místností, ve kterém budou místnosti zaujímající větší plochu upřednostněny před menšími.

7.6 Různorodost výsledných struktur

Základem knihovny je procedurální generování. Ačkoliv se v tomto případě jedná o pouhé generování místností a chodeb, i přesto může mít velký vliv na celkovou kompozici herní úrovně a její vnímání hráčem. Na repetitivnost a stereotyp mají v tomto případě vliv tři zásadní faktory:

1. Konfigurace vytvořené vývojářem a předané knihovně.
2. Schopnost knihovny generovat různorodé herní úrovně.
3. Obsah herních úrovní a ostatní aspekty hry, například příběhová linie.

Z pohledu této práce se první a poslední bod nedají nijak zvlášť ovlivnit a jsou v rukou vývojáře. Druhý bod tato knihovna splňuje pouze do té míry, že v případě použití náhodného semínka má vývojář jistotu, že bude rozložení herní úrovně vždy jedinečné. Jediným výraznějším prvkem různorodosti v konfiguraci jsou **číselné rozsahy** u parametrů šablon a u počtu místností. Co se týče rozložení místností, použitý algoritmus podle očekávání soustřeďuje většinu místností do středu. Vytvořením jednoduché teplotní mapy pozic buněk dungeonu (obrázek 7.4) lze však vysledovat, že u okrajů se místnosti sdružují spíše v rozích, a nikoliv rovnoměrně. Po bližším přezkoumání lze v teplotní mapě vypozorovat jev, kdy se větší místnosti častěji nacházejí u okrajů herní úrovně. Použitá metoda rozložení místností tak vykazuje určité nedostatky, které se projevují v určitých mezních situacích (velmi velké množství generovaných místností nebo definice místností, které svou plochou výrazně přesahují plochy ostatních místností).



Obrázek 7.4: Teplotní mapa reprezentující četnost místností na určitých místech dungeonu. Červenější odstín značí větší četnost. Na obrázku vlevo bylo vygenerováno 30 dungeonů, na obrázku vpravo 750 dungeonů. Konfigurace obsahovala okolo 250 místností.

7.7 Opakovatelnost výstupu

Celá knihovna byla navržena tak, aby umožňovala zadání semínka generátoru, podle kterého generátor pracuje. Byly použity Java kolekce s konstantním pořadím iterování (například `LinkedHashSet`, což je množina, která interně využívá dvoucestného spojového seznamu a zachovává tak pořadí prvků podle toho, jak byly vloženy) a globálně sdílený generátor náhodných čísel.

Semínko lze zadat do konfigurace prostřednictvím souboru nebo přímo v rozhraní API. Knihovna garantuje, že pokud bude pro generování opakovaně zadána kombinace totožných parametrů a semínka, výstup bude vždy stejný. Tento jev byl také otestován a nepodařilo se najít žádnou konfiguraci, ve které by neplatil.

7.8 Srovnání knihovny s existujícími řešeními

V kapitole 3 byly představeny knihovny a hry, které souvisí s problematikou procedurálního generování dungeonů. V této části bude knihovna ve stručnosti s těmito řešeními porovnána.

Herní úrovně v představené hře Rogue z roku 1980 se skládaly z 9 obdélníkových místností s náhodnými rozměry a tyto místnosti byly propojeny chodbami. Vytvořená knihovna dokáže díky tvarovým šablonám generovat náhodné čtvercové i obdélníkové místnosti, přičemž uživatel může určit jejich počet. Nemůže však nijak ovlivnit jejich pozici, která je určena náhodně. Generování takto malých dungeonů je však velmi rychlé a proto není problém vygenerovat například 100 dungeonů a vybrat si z nich ten nejvhodnější. Díky metodám poskytovaným knihovnou může vývojář velmi rychle určit vzájemnou polohu místností.

Generátor herních úrovní ve hře Diablo 1 z roku 1996 je typickou ukázkou generátoru, jimž vytvářené úrovně jsou příliš specifické, než aby se daly pokrýt univerzálně použitelnou knihovnou. V počátečních herních prostředích by však bylo možné úrovně vygenerované touto knihovnou použít jako základ a dále s nimi pracovat, například postupným rozšiřováním o přiléhající místnosti a rozšířené chodby a další požadované detaily. Vytvořenou knihovnu by však nebylo možné použít na generování jeskyní, protože je její výstup příliš uspořádaný – to však ani nebylo cílem, jeskyně jako taková nesplňuje charakteristiku dungeonu.

Světy vytvářené ve hře Don't Starve jsou sekvenčně řazené podle úloh a tudíž je vytvořená knihovna nedokáže nijak napodobit. Vývojář ovšem může implementovat generátor vlastních tvarů pro místnosti v textové podobě a vytvořit v konfiguraci místnosti, kterým tvary přiřadí. Generátor je poté schopen tyto místnosti vhodným způsobem vložit a propojit a vývojář může některé z nich použít a přiřadit jim účel, vlastnosti a vizuální prvky až posléze.

V části 3.2.1 byl představen generátor *DungeonGenerator*, kterým byly inspirovány některé fáze generátoru použitého v této knihovně. Tento generátor pro systém Unity nabízí pouze velmi omezenou sadu vstupních parametrů, které jsou navíc pevně zapsány v knihovně. Nenabízí žádné rozhraní ani možnost použít knihovnu mimo systém Unity, protože je závislý na fyzikálním systému Unity a využívá jeho algoritmus pro rozprostírání místností. Uživatel nemá navíc žádnou kontrolu nad tím, jaké místnosti generátor vygeneruje a vytvářené chodby mezi místnostmi jsou doplňovány jako izolované buňky bez reference na místnosti, které chodba propojuje. Knihovna vytvořená v této práci veškeré tyto problémy řeší a nabízí i další funkce, například možnost semínka, určování tvarů a počtů místností, plnou souborovou podporu, možnost spouštění jako program a plně doku-

mentované rozhraní. Obě knihovny na výstupu uživateli poskytují jak grafovou reprezentaci místností, tak dvourozměrnou mřížku.

Knihovna *Rot Web API* představená v části 3.2.2 je značně omezená co se parametrů a výstupu týče, ukazuje však velmi zajímavý koncept generátoru jakožto webové služby, díky čemuž lze tuto knihovnu používat z kteréhokoliv systému nezávisle na programovacím jazyku. Pro použití je však nutné knihovnu zprovoznit jako webový server. Knihovna odpovídá ve formátu JSON, který je široce používán například v architektuře REST. Knihovna vytvořená v této práci podporuje jak formát JSON, tak formáty YAML a XML a umožňuje používání nezávisle na jazyku díky tomu, že je spustitelná jako program a umožňuje souborovou komunikaci. Výhodou je, že ji uživatel nemusí složitě nastavovat a nepotřebuje webový server. Nevýhodou je nutnost mít na cílovém počítači nainstalováno běhové prostředí Javy.

Kapitola 8

Zhodnocení výsledků práce a závěr

Cílem této práce bylo analyzovat existující metody procedurálního generování, zejména metody pro generování herních úrovní typu dungeon, navrhnout a implementovat prakticky použitelnou knihovnu umožňující zadání konfigurace a vygenerování herní úrovně a následně vytvořit webovou aplikaci, ve které bude knihovna prakticky využita a zároveň bude vizualizací demonstrovat i funkčnost knihovny.

Vytvořená knihovna generuje strukturované místnosti propojené chodbami a je připravena pro použití ve hrách. Hlavními přednostmi je rychlost generování, jednoduchost použití a dobrá konfigurovatelnost. Nevýhodou je omezená použitelnost v případě, že má vývojář specifičtější požadavky na uspořádání dungeonů. Knihovna umožňuje zadání konfigurace prostřednictvím poskytnutého rozhraní, případně načtením ze souboru, a vygenerovaný dungeon lze rovněž uložit do souboru. Knihovna je využitelná i jako spustitelný program, což umožňuje její použití nezávisle na programovacím jazyku. Navržená webová aplikace umožňuje tyto dungeony jednoduchým způsobem vizualizovat a usnadňuje tak testování konfigurací a výstupů. Funkčnost knihovny byla dále prověřena testováním a jejím použitím v počítačové hře.

Během vytváření knihovny bylo potřeba zvážit uživatelské požadavky a aplikovat techniky, které je dokážou uspokojit. Knihovna generuje místnosti, ty zařazuje do kontextu grafů a prostřednictvím Delaunayovy triangulace a hledání minimální kostry s grafem dále pracuje. Nezanedbatelnou výzvou byl i výběr vhodné techniky pro rozprostření místností na rovině, kde se ukázalo, že drobné změny mohou mít velký vliv na výsledek.

Při práci na knihovně a webové aplikaci jsem si procvičil práci s literaturou i internetovými zdroji, osvojil jsem si princip fungování různých technik procedurálního generování a musel jsem se zamyslet nad tvorbou knihovny z hlediska její použitelnosti, vytvořit důkladný návrh a ten realizovat.

Knihovna nabízí spoustu prostoru k vylepšování a rozšiřování. Systém generování tvarů na základě šablon se ukázal jako velmi užitečný, proto by bylo jistě vhodné, aby knihovna umožňovala naprogramovat šablony vlastní. Knihovna obsahuje jeden generátor, který pracuje v několika fázích. Tyto fáze by bylo vhodné proměnit na univerzálně použitelné a zvláště konfigurovatelné komponenty, které by se daly použít ve vlastních generátorech. S tím by byla spojena možnost upravovat strukturu vstupní konfigurace a výstupních dat. Zajímavou možností by byla implementace generátoru, který by umožnil tvorbu místností v určitém sledu. Například místnost s nepřítelem, za kterou by vždy následovala místnost s pokladem. K tomu by byl ovšem potřeba dodatečný výzkum.

Literatura

[1] Dungeon Generation in Diablo 1. *BorisTheBrave.Com* [online]. 2019. Publikováno 14. 7. 2019 [cit. 22. ledna 2020]. Dostupné z: https://www.boristhebrave.com/2019/07/14/dungeon-generation-in-diablo-1.

[2] Procedural Dungeon Generation Algorithm. *Gamasutra* [online]. 2015. Publikováno 9. 3. 2015 [cit. 5. února 2020]. Dostupné z: https://www.gamasutra.com/blogs/AAdonaac/20150903/252889/Procedural_Dungeon_Generation_Algorithm.php.

[3] Kolektiv autorů. World Generation. *Don't Starve Wiki* [online]. 2015. Publikováno 27. 6. 2015 [cit. 26. ledna 2020]. Dostupné z: https://dontstarve.fandom.com/wiki/World_Generation.

[4] Adams, D. *Automatic Generation of Dungeons for Computer Games*. Velká Británie, 2002. Bakalářská práce. University of Sheffield.

[5] Brewer, N. Going Rogue: A Brief History of the Computerized Dungeon Crawl. *Insights. IEEE USA* [online]. 2016. Publikováno 7. 7. 2016 [cit. 6. prosince 2019]. Dostupné z: https://insight.ieeeusa.org/articles/going-rogue-a-brief-history-of-the-computerized-dungeon-crawl/.

[6] Doran, J. a Parberry, I. Controlled Procedural Terrain Generation Using Software Agents. *IEEE Transactions on Computational Intelligence and AI in Games*. 1. vyd. Červen 2010, sv. 2, č. 2, s. 111–119. DOI: 10.1109/TCIAIG.2010.2049020. Dostupné z: https://doi.org/10.1109/TCIAIG.2010.2049020.

[7] Dormans, J. Adventures in level design: Generating missions and spaces for action adventure games. In: PCGames '10. *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. Monterey, California: Association for Computing Machinery, červen 2010, č. 1. ISBN 978-1-4503-0023-0.

[8] Gallant, J. 2D Liquid Simulator With Cellular Automaton in Unity. *Jgallant's Indie Game Developer Homepage* [online]. 2017. Publikováno 30. 1. 2017 [cit. 7. ledna 2020]. Dostupné z: http://www.jgallant.com/2d-liquid-simulator-with-cellular-automaton-in-unity.

[9] Henderson, D., Jacobson, S. H. a Johnson, A. W. The theory and practice of simulated annealing. In: Glover, F. a Kochenberger, G., ed. *Handbook of metaheuristics*. Springer, Boston, MA, 2003, sv. 57, kap. 10, s. 287–319. International Series in Operations Research & Management Science. DOI: 10.1007/0-306-48056-5_10. ISBN 978-0-306-48056-0. Dostupné z: https://doi.org/10.1007/0-306-48056-5_10.

[10] Hughes, M. D. Game Design: Article 07: Roguelike Dungeon Generation. *Mark Damon Hughes* [online]. 2019. Webový snímek z 25. 10. 2013 [cit. 21. ledna 2020]. Dostupné z: https://web.archive.org/web/20131025132021/http://kuoi.org/~kamikaze/GameDesign/art07_rogue_dungeon.php.

[11] Johnson, L., Yannakakis, N. G. a Togelius, J. Cellular automata for real-time generation of infinite cave levels. In: PCGames '10. *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. Monterey, California: Association for Computing Machinery, červen 2010, č. 10. ISBN 978-1-4503-0023-0.

[12] Linden, R., Lopes, R. a Bidarra, R. Designing procedurally generated levels. In: Association for the Advancement of Artificial Intelligence. *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Boston, Massachusetts: AAAI Press, říjen 2013, s. 41–47. ISBN 978-1-57735-607-3.

[13] Ma, C., Vining, N., Lefebvre, S. a Sheffer, A. Game level layout from design specification. *Computer Graphics Forum*. 1. vyd. Červen 2014, sv. 33, č. 2, s. 95–104. DOI: 10.1111/cgf.12314. ISSN 1467-8659. Dostupné z: https://doi.org/10.1111/cgf.12314.

[14] Sandoval, K. What Data Formats Should My API Support? *Nordic APIs* [online]. 2016. Publikováno 2. 2. 2016 [cit. 10. dubna 2020]. Dostupné z: https://nordicapis.com/what-data-formats-should-my-api-support/.

[15] Sarkar, P. A Brief History of Cellular Automata. *ACM Comput. Surv.* 1. vyd. New York, NY, USA: Association for Computing Machinery. březen 2000, sv. 32, č. 1, s. 80–107. DOI: 10.1145/349194.349202. ISSN 0360-0300. Dostupné z: https://doi.org/10.1145/349194.349202.

[16] Sgall, P. Matematický popis přirozených jazyků. *Pokroky matematiky, fyziky a astronomie*. 1. vyd. Jednota českých matematiků a fyziků. 1978, sv. 23, č. 3, s. 140–148. ISSN 0032-2423.

[17] Shaker, N., Liapis, A., Togelius, J., Lopes, R. a Bidarra, R. Constructive generation methods for dungeons and levels. In: Shaker, N., Togelius, J. a J. Nelson, M., ed. *Procedural Content Generation in Games*. Springer, Cham, 2016, s. 31–55. ISBN 978-3-319-42716-4.

[18] Shaker, N. a Togelius, J. The search-based approach. In: Shaker, N., Togelius, J. a J. Nelson, M., ed. *Procedural Content Generation in Games*. Springer, Cham, 2016, s. 17–30. ISBN 978-3-319-42716-4.

[19] Shaker, N., Togelius, J. a J. Nelson, M. What is procedural content generation? In: Shaker, N., Togelius, J. a J. Nelson, M., ed. *Procedural Content Generation in Games*. Springer, Cham, 2016, s. 1–3. ISBN 978-3-319-42716-4.

[20] Shiu Chong, K. Collision Detection Using the Separating Axis Theorem. *Envato Tuts+* [online]. 2012. Publikováno 6. 8. 2012 [cit. 12. dubna 2020]. Dostupné z: https://gamedevelopment.tutsplus.com/tutorials/collision-detection-using-the-separating-axis-theorem--gamedev-169.

Příloha A

Obsah přiloženého média

- `/`
  - `knihovna/`
    - `readme.pdf` ..... Pokyny pro kompilaci a použití
    - `src/`
      - `main/`
        - `java/` ..... zdrojové kódy knihovny
        - `resources/` ..... statické zdroje a knihovny
      - `test/` ..... zdrojové kódy testů
    - `target/` ..... přeložené soubory
      - `jar/` ..... spustitelný JAR soubor
      - `docs/` ..... vygenerovaná Java dokumentace
    - `examples/` ..... ukázkové konfigurační soubory
  - `web/`
    - `readme.pdf` ..... Pokyny pro kompilaci a použití
    - `src/`
      - `main/`
        - `java/` ..... zdrojové kódy webové aplikace
        - `webapp/`
          - `resources/` ..... statické zdroje a knihovny
          - `WEB-INF/` ..... definiční soubory Java EE a Spring a šablony
    - `target/` ..... archív WAR s webovou aplikací
  - `zprava/`
    - `bib-styles/` ..... bibliografické styly
    - `kody/` ..... úryvky kódu použité ve zprávě
    - `obrazky/` ..... obrázky použité ve zprávě
    - `template-fig/` ..... loga VUT
    - `fitthesis.cls` ..... šablona pro BP
    - `Makefile` ..... soubor pro překlad zprávy
    - `xsipos03-proc-gen-dungeonu-kapitoly.tex` ..... hlavní text práce
    - `xsipos03-proc-gen-dungeonu-literatura.bib` ..... bibliografická databáze
    - `xsipos03-proc-gen-dungeonu-prilohy.tex` ..... přílohy
    - `xsipos03-proc-gen-dungeonu.pdf` ..... zpráva v PDF
    - `xsipos03-proc-gen-dungeonu-tisk.pdf` ..... zpráva v PDF – verze pro tisk
    - `xsipos03-proc-gen-dungeonu.tex` ..... hlavní .tex soubor
    - `zadani.pdf` ..... zadání práce