VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GENERÁTOR ŽALÁŘŮ PRO UNREAL ENGINE

DUNGEON GENERATOR FOR UNREAL ENGINE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **SAMUEL MOŘKOVSKÝ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. TOMÁŠ STARKA**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Mořkovský Samuel**
Program: Informační technologie
Název: **Generátor žalářů pro Unreal Engine**
Dungeon Generator for Unreal Engine
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte techniky procedurálního generování herních "dungeonů" a API Unreal Enginu.
2. Implementujte generátor jako plugin pro UE.
3. Implementujte 3D vizualizaci pomocí UE.
4. Vytvořte krátké video prezentující výsledek.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Bod 1, část bodu 2 a část bodu 3.

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

Vedoucí práce: **Starka Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 1. listopadu 2021

Abstrakt

Práce se zabývá návrhem a implementací generátoru dungeonů pro herní engine Unreal Engine verze 4. Pro výsledek byla zvolena forma pluginu, která umožňuje generátor vložit do již existujícího projektu bez nutnosti zásahu do zdrojového kódu. Parametry modifikující proces generování jsou předávány z uživatelského rozhraní Unreal Enginu. Výsledkem procesu je množina místností a chodeb, které místnosti propojují tak, aby bylo možné každou místnost navštívit. Všechny tyto objekty jsou reprezentovány 3D modely v aktuální úrovni.

Abstract

The work's purpose is to design and implement dungeon generator for a game engine Unreal Engine version 4. Plugin was selected as a form of work's result, which allows to integrate generator into an already existing project without the need to modify any source code. Parameters modifying the process of generation are passed from the user interface of Unreal Engine. Result of the process is a set of rooms and corridors connecting the rooms in a way that every room can be visited. All these objects are represented by 3D models in current level.

Klíčová slova

procedurální generování, Unreal Engine 4, žalář, plugin, herní engine, generátor žalářů, delaunayova triangulace

Keywords

procedural generation, Unreal Engine 4, dungeon, plugin, game engine, dungeon generator, delaunay triangulation

Citace

MOŘKOVSKÝ, Samuel. *Generátor žalářů pro Unreal Engine*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Starka

Generátor žalářů pro Unreal Engine

Prohlášení

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

.....
Samuel Mořkovský
11. května 2022

Poděkování

Děkuji panu Ing. Starkovi za řádný dohled, projevenou trpělivost a svižnou komunikaci po celou dobu výkonu práce.

Obsah

Kapitola 1

Úvod

Cílem této práce je procedurální generování herních úrovní typu dungeon (z angl. žalář či kobka). Jedná se o herní úroveň, která je uzavřená a obsahuje místnosti s chodbami, které je propojují. Výsledek práce umožní uživateli generování dungeonů různých velikostí a rozložení na základě několika nastavovatelných parametrů. Výsledek bude spustitelný v programu Unreal Engine 4 tak, aby v případě libosti mohl uživatel vygenerovaný dungeon uložit či zreplikovat a dále jej jednoduše upravovat pro specifické potřeby své hry.

Jednoduše řečeno, procedurální generování je technika, kdy je herní obsah tvořen automaticky počítačem namísto ručně vývojářem. Takto tvořený obsah umožňuje mít ve hrách obsáhlou, někdy až zdánlivě nevyčerpatelnou množinu kombinací jednoho či více prvků. U hry No Man's Sky se jedná například o generování povrchů planet, které hráč může navštívit, a tvorů, kteří tyto planety obývají. V jiné hře, Minecraftu, je primárně generován terén, který hráč může prozkoumávat a upravovat. Dwarf Fortress generuje celý svět, do kterého jsou zasazeni obyvatelé mající náhodně generované povahové rysy. V Diablu 1 a Binding of Isaac se vytvářejí herní úrovně, dungeony, do kterých jsou po dokončení generování zasazeni nepřátelé s předdefinovanou podobou a chováním.

Procedurální generování představuje pro vývojáře příležitost vytvářet typ herního obsahu za kratší časový úsek, než kdyby jej musel připravovat ručně. Vývojář v tomto případě není ten, kdo obsah tvoří, ale kdo určuje, jak má být tvořen, a práci odvádí algoritmus. Takto vytvořený obsah potom nabízí hráčům možnost opakovaně nacházet unikátní kombinace herních prvků, hra může vypadat vždy jinak a představovat ojedinělý zážitek.
Na druhou stranu se generovaný obsah může pro hráče stát i zklamáním. Pokud každý nepřítel ve hře vypadá odlišně, ale má stejný vzorec chování, objevuje se pokaždé ve stejné situaci a dělá stále stejné zvuky, začne být pro hráče nudný a monotónní. Proto se tvůrci her snaží vyvážit náhodně generovaný a manuálně tvořený obsah tak, aby se hráč bavil, měl stále co objevovat a nenabyl pocitu, že se hra někde až příliš opakuje.

Text práce je rozčleněn do kapitol tak, že v kapitole 2 je popsáno, co se dnes proceduráně generuje, jaké techniky se k tomu používají a konkrétní příklady významných herních titulů, které svůj obsah alespoň částečně generují. Dále jsou zde uvedena některá aktuálně dostupná řešení pro využití s různými technologii a potom řešení konkrétně pro Unreal Engine 4. Kapitola 3 obsahuje návrh a očekávaný výsledek řešení. Implementace návrhu a případné odchylky od něj jsou popsány v kapitole 4.

Kapitola 2

Generování herního obsahu nyní

V této kapitole bude popsáno co znamená procedurální generování, dále bude naznačeno, jaký různý obsah může být dnes touto technikou v odlišných hrách vytvářen a z jakých známých technik a algoritmů se v procesech generování vychází. Dále budou zmíněny konkrétní herní tituly, ve kterých vývojáři v menší či větší míře vhodně zakomponovali procedurální generování, jaký konkrétní obsah se tímto způsobem tvoří a konečně jaké techniky, pokud jsou známé, se k tomuto procesu použilo.

2.1 Procedurální generování

Procedurální generování se dá definovat jako algoritmické vytváření herního obsahu s omezeným nebo nepřímým vstupem od uživatele. Lze se setkat se využitím generátorů pseudonáhodných čísel, což může nabízet více rozmanité a méně očekávatelné výsledky.

Kromě množství obsahu, který je procedurální generování schopné vytvořit, může představovat formu komprese, neboť generovaný obsah není zakomponován v souborech hry, ale potřebné množství tohoto obsahu bude teprve vytvořeno po spuštění a dočasně uloženo v paměti [10].

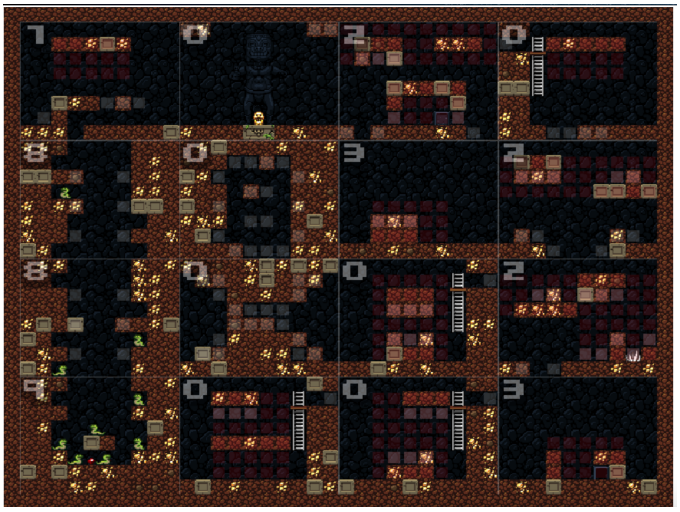
2.2 Rozmanitost generování obsahu

U her, které se skládají z několika herních úrovní, můžou být generovány celé tyto úrovně. Za zmínku stojí hra Spelunky, která úroveň rozdělí na 2D mřížku o velikosti 4×4, kterou následně naplní předdefinovanými průchody a místnostmi tak, aby hráč vždy začal v místnosti ve vrchním řádku a měl možnost se dostat až do místnosti v řádku nejnižším, která je určena jako cíl. Tyto místnosti jsou také vkládány s určitou variací.[1]

V počítačových hrách se může využívat procedurálního generování na vytváření terénu. Tímto způsobem vytváří svůj terén například hra Minecraft. Využívá k tomu algoritmus známý jako Perlinův šum, který používá tzv. seed (z angl. semínko). Seed je hodnota obecně užívaná pro inicializaci generátorů pseudonáhodných čísel. Hráč má přístup k seedu vygenerovaného světa a může jej znovu použít, čímž si vytvoří ten samý svět, protože tento algoritmus, popsaný v části 2.3, vytváří vždy stejný výsledek pro stejný seed.

[1] Princip generování úrovně pro hru Spelunky popsán na blogu:
http://tinysubversions.com/spelunkyGen/

[2] https://shanemartin2797blog.wordpress.com/...



Obrázek 2.1: Ukázka úrovně s mřížkou ze hry Spelunky, převzato.[2]

Ve hře Spore se, kromě jiného, generují druhy tvorů, na které hráč může narazit. Tito tvorové mohou mít různý počet končetin, obratlů v páteři či další detaily, které mají větší či menší dopad na chování a vlastnosti druhu. Pro vytvořený druh jsou na základě rysů dále vygenerovány i animace pohybu. Co se postav a tvorů týče, hra Dwarf Fortress automaticky generuje například povahové rysy, popis vzhledu i povolání.

Obrázek 2.2: Ukázka různě procedurálně vygenerovaných druhů tvorů ze hry Spore, převzato.[3]

Automaticky vkládaný obsah může být i záležitostí rozmisťování nepřátel, bonusů či zbraní pro hráče, jako to dělají některé hry od společnosti Valve. Ve hře Left 4 Dead je v každé úrovni zvláštní objekt, tzv. AI Director, který umisťuje nepřátele průběžně a vybírá jejich počáteční lokaci tak, aby hráči neviděli moment jejich umístění. Často tedy nepřátelé přicházejí z různých úhlů, což přidává na dynamičnosti a obtížnosti. Kromě nepřátel AI Director může umisťovat i zbraně či ozdravovací balíčky. Podobně funguje také ve hře Alien Swarm.

[3] https://www.engadget.com/...

Procedurální generování má pro herní obsah rozsáhlé možnosti použití. Kromě zmíněných lze generovat odměny pro hráče, typy zbraní (Borderlands) nebo třeba hudbu. Taková technika je použita například ve hře .kkrieger, kde namísto běžného formátu audio (přípony mp3, wav, raw...) jsou použity MIDI data, ze kterých čerpá algoritmus, jehož účel je přeložit tato data na audio.

2.3 Techniky generování využívané v herním průmyslu

Existuje mnoho různě složitých algoritmů zakomponovaných v počítačových hrách, které obstarávají procedurální generování. Následující techniky jsou často spojovány s generováním dungeonů, ale mohou být uplatněny i pro jiný typ obsahu.

Celulární automaty

Za prvního člověka zabývajícím se celulárními automaty je považován John von Neumann. Snažil se o vytvoření výpočetního systému, který by produkoval kopie sám sebe. Později, fyzikové a biologové začali studovat celulární automaty pro potřeby modelování v jejich příslušné oblasti. V aktuální době jsou celulární automaty studovány z mnoha různých úhlů a jsou hledány a objevovány vztahy k mnoha existujícím problémům [8].

Celulární automat je dynamický systém a matematický model, který modeluje živé okolí. Jeho základní prvky jsou buňky, pole buněk, okolí a pravidla. Buňka nabývá jednoho z konečného počtu stavů. V nejjednodušším případě se jedná o stavy $0 \check{} vypl$ a $1 \check{} zapl$. Pole buněk bývá běžně 1D (pole) nebo 2D (mřížka), mohou však být i n-rozměrné struktury. Okolí buňky c je u 1D určeno počtem sousedních buněk po obou stranách buňky. U 2D se nejčastěji jedná o Moorovo či von Neumannovo okolí. Obě okolí mohou mít velikost 1 či více, přičemž velikost 1 v případě von Neumannova okolí to znamená 4 okolní buňky od c, u Moorova okolí je to 8 buněk, jak lze vidět na obrázku 2.3.

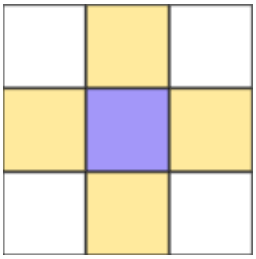
Souhrn pravidel pro změnu stavu buňky při dalším kroku tvoří přechodovou funkci. Argumentem této funkce je právě stav okolních buněk a buňky samotné. Pro čas $t = 0$ je podstatné zvolit počáteční konfiguraci – tedy stav všech buněk. Počet možných konfigurací v okolí lze vypočítat jako s^n, kde s je počet stavů a n je počet sousedů včetně buňky c [3].

V rámci procedurálního generování byl popsán Johnsonem L. et al. [4] systém pro generování jeskyní s chodbami. Motivací bylo vytvoření hry, ve které se hráč může nekonečně a plynule pohybovat. Dalším požadavkem bylo, aby jeskyně vypadaly víc organicky a erodovaně, spíše než aby měly rovné hrany a nepřirozené úhly. Jelikož nelze uchovat v žádné paměti nekonečně velkou jeskyni, bylo nutno ji generovat za běhu. Když tedy hráč odešel z jedné lokace jeskyně do jiné, byla mu ihned vytvořena lokace další a předchozí zmizela (nešlo o plynulý přechod pohledu hráče, vidět byla vždy pouze 1 lokace).

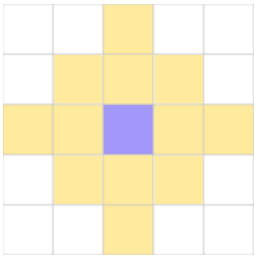
Tato metoda používá následující parametry pro generování lokace:

- r = Poměr buněk ve stavu kamene (prostor, kam se nedá vejít)
- n = Počet provedených iterací celulárního automatu
- T = Počet sousedů, kteří musí mít stav kamene, aby se samotná buňka stala kamenem
- M = Velikost sousedství

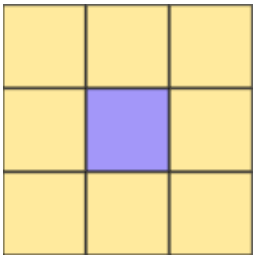
Každá lokace je mřížka 50×50 a buňka může nabývat stavu *prázdná, kámen* nebo *stěna*. Na počátku je mřížka prázdná a generování probíhá následovně:



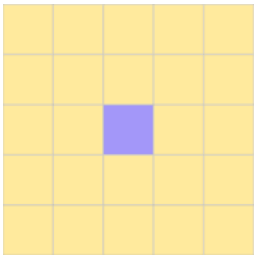
(a) von Neumannovo okolí o velikosti 1



(b) von Neumannovo okolí o velikosti 2



(c) Moorovo okolí o velikosti 1



(d) Moorovo okolí o velikosti 2

Obrázek 2.3: Znázornění Moorova a von Neumannova okolí pro 2D mřížku. Uprostřed se nachází modrá buňka c, ke které se okolí vztahuje.

- Každá buňka vyhodnotí, jestli se přemění na kámen dle pravděpodobnosti r. Výsledek je poměrně uniformní rozložení kamenných buněk.
- Provede se n kroků celulárního automatu. Základní pravidlo je, že pokud je buňka obklopena alespoň T sousedy ve stavu kamene, změní se v dalším kroku na kámen.
- Kamenné buňky, které hraničí s volným prostorem jsou změněny na stav stěny. To umožňuje vizuálně odlišit hraniční buňku od kamene, což působí esteticky lépe. Buňka ve stavu stěny je stále neprůchozí.

Takto se vygenerují lokace sousedící s lokací, kde se hráč aktuálně nachází, a v místech, kde je stěna nejužší, se vytvoří tunel, provedou se další iterace celulárního automatu kvůli zahlazení a je hotovo.

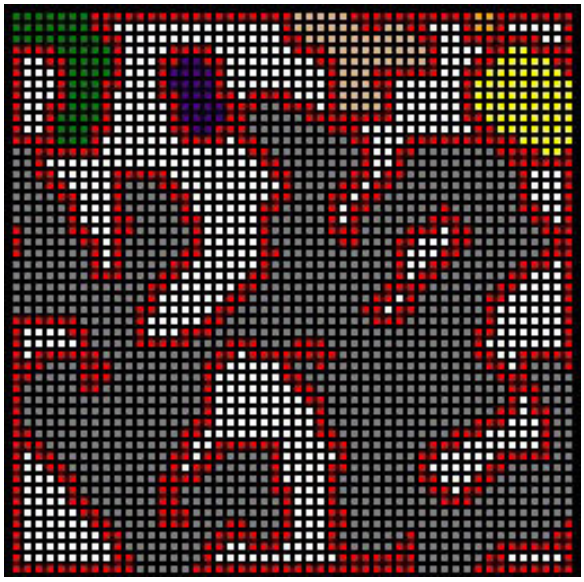
Na obrázku ?? je vidět, že na základě pouhých 4 parametrů lze generovat poměrně uspokojivě vypadající jeskyni, ve které se hráč může libovolně pohybovat a procházet do dalších lokací. Nevýhodou samozřejmě je, že méně parametrů v tomto případě znamená i menší kontrolu nad výsledkem, což například znemožňuje vytvořit specificky vypadající jeskyni.

Gramatiky

Gramatika je nástroj pro popis a analýzu jazyků. Obsahuje soubor pravidel, dle kterého jsou konstruovány platné věty daného jazyka.[4] Gramatiku lze definovat jako systém skládajícího se z:

- konečné abecedy Σ, přičemž abeceda může znamenat například specifické znaky, klíčová slova programovacího jazyka, elementy a atributy pro XML dokument a další.

[4] Čerpáno z: https://web.stanford.edu/...



Obrázek 2.4: Výsledná jeskyně popsaného algoritmu. Parametry: $n = 4, M = 1, T = 5$. Buňky typu kámen a stěna jsou reprezentovány bílou, respektive červenou barvou. Obarvené oblasti představují rozdílné tunely [4].

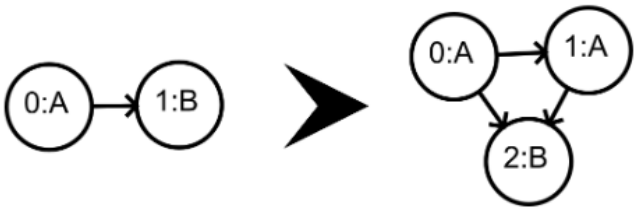
- konečná množina neterminálních symbolů N, které nebudou součástí výsledku, ale budou použity při jeho vytváření.
- počáteční symbol S, který náleží do neterminálních symbolů N.
- konečná množina pravidel R, kde každé pravidlo transformuje výraz neterminálních a terminálních symbolů do výrazu jiného. [1]

Gramatiky se při generování herního obsahu dají využít například pro mise či prostor. Mise popisuje, co může hráč v úrovni dělat a co musí splnit, aby úroveň dokončil. Prostor popisuje geometrické uspořádání prostředí. Pro popis takového obsahu se dá použít například graf, na který se postupně aplikují transformační pravidla předepsané gramatiky. Transformace grafu pak funguje následovně:

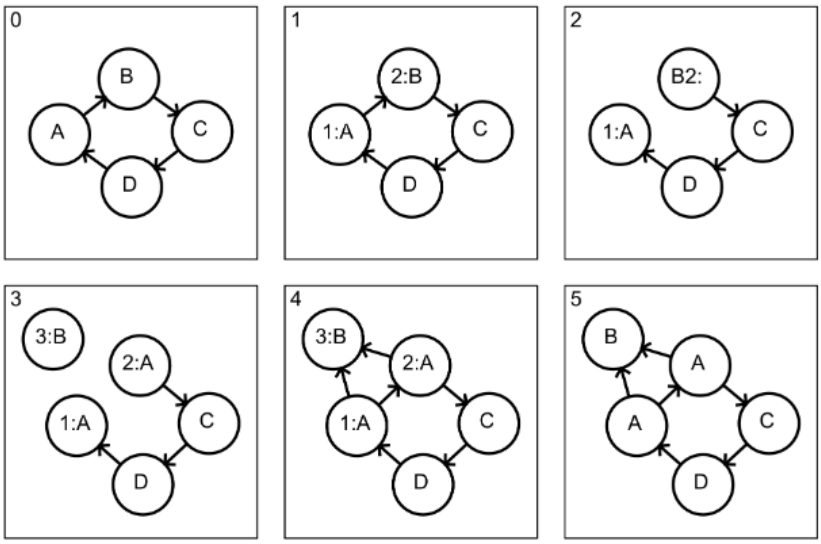
1. Je nalezen podgraf v cílovém grafu, který se shoduje s levou stranou pravidla, a podgraf se označí zkopírováním identifikátorů uzlů.
2. Odstraní se všechny hrany mezi označenými uzly.
3. Graf se transformuje tak, že označené uzly se přetransformují na odpovídající uzly na pravé straně pravidla, přičemž jsou přidány uzly z pravé strany, které nemají shodné uzly na cílovém grafu, a jsou odebrány uzly, které nemají shodné uzly na pravé straně pravidla.
4. Zkopírují se hrany dle pravé strany pravidla.
5. Odstraní se všechna označení.

Na obrázku 2.6 jsou znázorněny jednotlivé kroky aplikování pravidla.

S tímto postupem a upravenou abecedou pro příslušnou gramatiku lze generovat již zmíněné mise. Jednoduchá abeceda by mohla vypadat například takto:



Obrázek 2.5: Jednoduché pravidlo pro transformaci grafu.

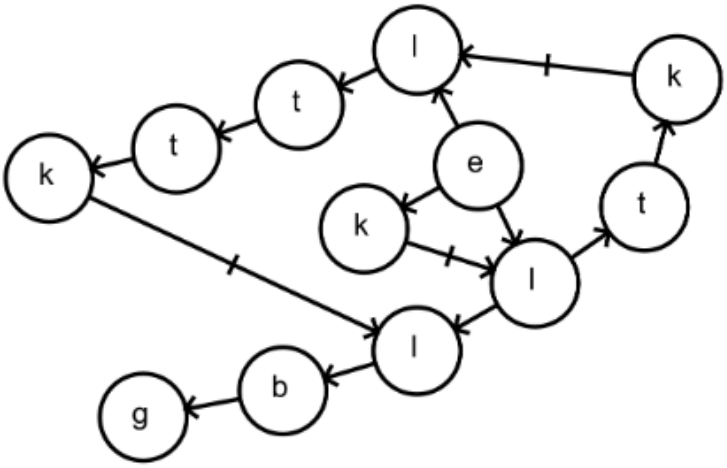


Obrázek 2.6: Příslušné kroky transformace grafu dle pravidla ??.

- Start (S) – počáteční symbol, ze kterého se bude generovat mise
- Vstup (e) – počáteční pozice hráče
- Úkoly (t) – libovolné, nespecifikované úkoly (mohou být například nepřátelé, hádanka k bonusové odměně, nový předmět, ...)
- Cíle (g) – úkol, jehož splnění ukončí úroveň
- Zámky (l) – úkol pro jehož dokončení je třeba klíče
- Klíče (k) – klíče odemykající zámky, přičemž jím může být typický klíč od dveří, ale i páka pro padací most či vyřešení hádanky, která otevře průchod
- Neterminální uzly s úkoly (T) – tyto uzly zajistí další rozvíjení v průběhu generování
- Běžné hrany – propojují uzly a určují, který úkol na který navazuje
- Odemykací hrany – propojují klíče a zámky, které odemykají

Spuštění transformace grafu s využitím takové abecedy s patřičnými pravidly by fungovalo, ale pravděpodobně by podávalo až příliš náhodné výsledky (generování by mohlo skončit například po 1. iteraci a při vysokém počtu iterací by graf už mohl být nežádoucně

komplikovaný). Proto se využívá sekvenčního spouštění pravidel, kdy se pravidla na graf aplikují kontrolovaně. Lze ale například použít rozptyl, z jehož rozsahu se náhodně určí počet aplikací daného pravidla. Tím lze získat unikátní výsledek a zároveň přijatelně složitý graf. Možný výsledek vyobrazuje graf na obrázku 2.7. Graf misí je však jen první část. K tomuto grafu je potřeba vytvořit patřičné prostředí tak, aby jej hráč mohl procházet dle logiky vycházející z grafu misí. Problém se nyní nachází v tom, že graf misí je pouze abstraktní struktura určující pořadí různých úkonů v úrovni a nemá nic společného s prostředím vygenerované úrovně. Tento problém lze řešit například některým z následujících způsobů:



Obrázek 2.7: Ukázka možného výstupu po aplikování kontrolovaného počtu pravidel na počáteční uzel S [9].

1. Transformovat graf misí na prostředí užitím takových pravidel, které výsledný graf zjednoduší na 1 či více přímých cest od vstupu (e) po cíl (g). Výsledek je stále příliš abstraktní, ale použitím algoritmů pro automatické rozložení grafů (z angl. automatic graph layout algorithms) v kombinaci se vzorkováním (z angl. sampling) lze docílit použitelné geometrie pro danou úroveň.

2. Přiřadit misím instrukce pro stavění prostředí. Namísto přímé transformace grafu misí na prostor má každá mise přiřazenou sadu instrukcí, pomocí které vytvoří geometrii v úrovni tak, aby se shodovala s nároky a logikou grafu mise. Nevýhoda je, že je obtížné tímto způsobem vytvořit více cest vedoucích ke stejnému cíli.

3. Postavit geometrii pro úroveň, z toho vytvořit abstraktní reprezentaci herního prostoru a z něj vygenerovat mise. Ke generování úrovně může být použito celulárního automatu, gramatik, evolučních algoritmů nebo jiných způsobů. Nevýhoda zde je, že úroveň může mít nedostatek potenciálu pro vložení misí (například nedostatek dveří, které by mohly být zamknuté, nedostatek pastí, staveb, prostoru pro nepřátele, ...). [9]

L-systémy

L-systémy (nebo také Lindenmayerovy systémy) jsou paralelně se přepisující systémy, které byly původně vytvořeny za účelem modelování růstu jednoduchých mnohobuněčných orga-

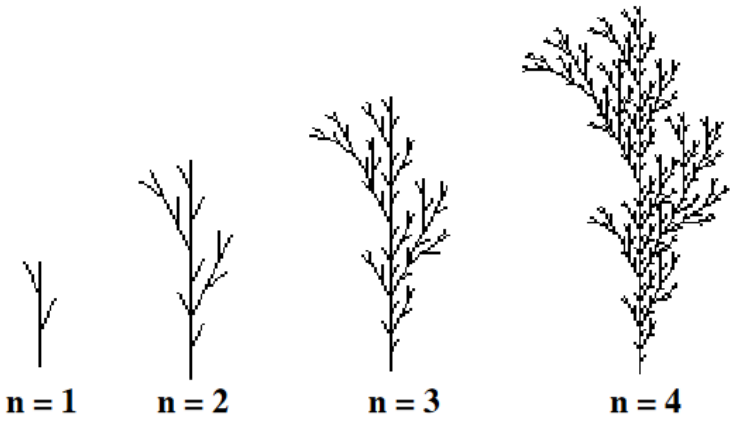
nismů [7]. Nyní jsou L-systémy aplikovány například pro biologické modelování, generování fraktálů, umělý život (z angl. artificial life) a krom mnoha dalších i pro vytváření herního obsahu.

L-systémy vycházejí z formálních gramatik 2.3 mají tedy také předepsaná pravidla pro přepisování řetězce a mají také počáteční hodnotu (u L-systému nazývaná axiom), která je danými pravidly rozvíjena. Důležitou vlastností L-systémů je paralelní přepisování, kdy řetězec není přepisován sekvenčně (například zleva doprava) jako celek, ale všechno přepisování se děje ve stejnou chvíli.

Jeden z možných způsobů grafické interpretace výsledku L-systému je želví grafika (z angl. turtle graphics), kde želvou je myšlen kurzor, který se pohybuje dle instrukcí po plátně a za sebou zanechává čáru. Instrukce ve spojitosti s abecedou mohou vypadat například takto:

- F: posun vpřed o 5 pixelů
- $-$: otočení doleva o 90°
- $+$: otočení doprava o 90°

V tuto chvíli je však omezením to, že vykreslovaná grafika musí být vykreslena jedním tahem. Ale mnoho zajímavých tvarů tímto způsobem vykresleno být nelze; například rostliny mají větve, které je třeba dokreslit a poté se vrátit zpět ke stonku. Pro takové případy se využívají tzv. závorkové L-systémy (z angl. bracketed L-systems), které obsahují 2 zvláštní symboly, [a], jejichž význam je vkládání, respektive vyjmutí, pozice a orientace kurzoru ze zásobníku. Nyní tedy lze před začátkem vykreslování větve rostliny uložit pozici a orientaci do zásobníku, vykreslit větev a z vrcholu zásobníku pozici a orientaci zase vyjmout a aplikovat je na kurzor. Jediným pravidlem $F \rightarrow F[-F]F[+F][F]$ lze vykreslit strukturu věrně připomínající rostlinu, jak je ukázáno na obrázku 2.8 [9].



Obrázek 2.8: Postupné rozvíjení rostliny pomocí iterací L-systému, počet iterací je znázorněn proměnnou n. Obrázek převzat [9].

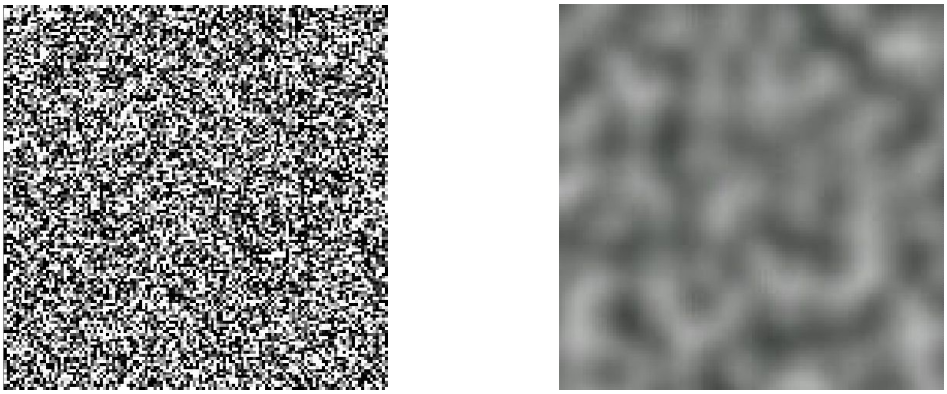
Tímto způsobem lze generovat 2D i 3D rostliny pro herní úroveň. Například jeden projekt využívající L-systémy je SpeedTree[5][5], který je běžně využíván významnými spo-

[5]Více o systému SpeedTree lze nalézt na https://store.speedtree.com/

lečnostmi nejen v herním průmyslu (Ubisoft, Epic Games, CD Projekt Red a další), ale i ve filmovém (například filmy Avatar, Warcraft a Avengers: Infinity War).

Perlinův šum

Perlinův šum je pseudonáhodný vzor čísel s desetinnou hodnotou, které jsou generovány napříč daným prostorem (běžně 1 až 3 dimenze, ale může být implementován i na n dimenzí). Výsledek neobsahuje náhodné číslo pro každou pozici, ale je tvořen plynulými přechody skrze prostor[6] (obr. 2.9). Takový výsledek může být vhodná počáteční heightmapa[7] pro generování terénu.



(a) Bílý šum (b) Perlinův šum

Obrázek 2.9: Porovnání výsledku tzv. bílého[8]a Perlinova šumu.

V porovnání s heightmapou skutečného světa však není výsledek příliš vyhovující, jde v podstatě jen o plynule se zvyšující a snižující povrch. První možný prvek pro úpravu výsledku je frekvence. Změnou frekvence (vynásobením vstupních parametrů pro funkci Perlinova šumu) lze docílit urychlení funkce a tím získat více přechodů pro stejně velkou heightmapu (obrázek 2.10).

(a) $f = 1$ (b) $f = 3$

Obrázek 2.10: Porovnání heightmap pro rozdílné frekvence Perlinova šumu

[6] https://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html

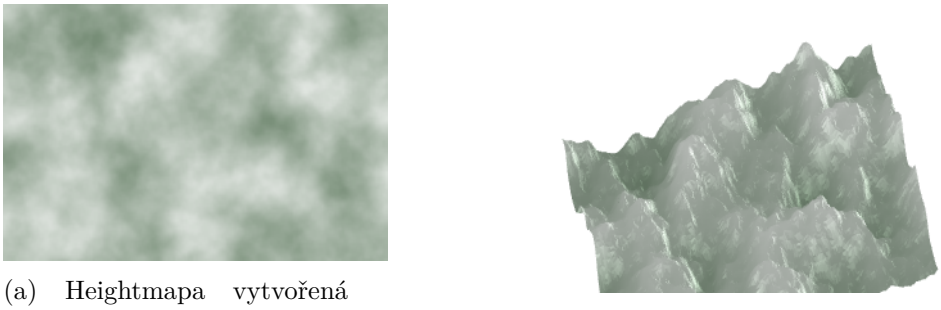
[7] Heightmapu můžeme zjednodušeně označit jako mřížku, kde každý bod (v případě heightmapy pixel) uchovává hodnotu, která reprezentuje vyvýšení terénu (elevace) v daném bodě https://en.wikipedia.org/wiki/Heightmap

[8] Každá pozice má náhodnou hodnotu, která není závislá na hodnotách sousedních pozic

Při součtu více výsledků funkce s různými poměry frekvencí lze dosáhnout realističtěji vypadající heightmapy – terén vypadá detailněji, kromě plynulých kopců se tvoří menší nerovnosti. Pro následující kód

```
e =    1 * noise(1 * nx, 1 * ny)
  +  0.5 * noise(2 * nx, 2 * ny)
  + 0.25 * noise(4 * nx, 4 * ny);
```

kde `noise()` je funkce Perlinova šumu, nx a ny jsou souřadnice aktuální pozice v mřížce a e je elevace této pozice, lze získat heightmapu z obr. 2.11. Výstupní heightmapa působí



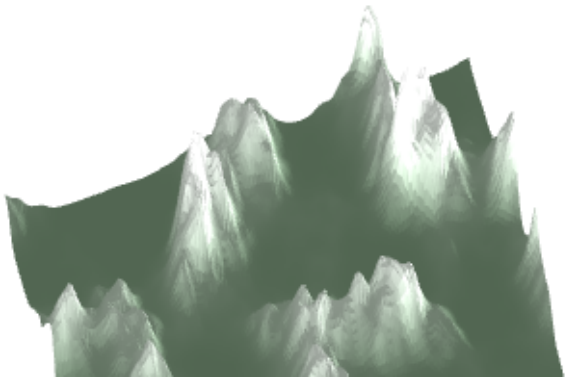
(a) Heightmapa vytvořená kombinací více frekvencí

(b) 3D vizualizace

Obrázek 2.11: Heightmapa a její 3D vizualizace. Čím světlejší bod, tím výše je položen.

detailně, nicméně povrch je až příliš hornatý, nevhodný pro použití jinde než v hrubém typu terénu jako typu hory, části jeskyně nebo třeba různé typy lomů. Toto lze řešit například umocněním elevace, což zapříčiní (za předpokladu, že elevace je v rozmezí 0–1), že čím je exponent vyšší, tím bude průměrná elevace nižší a vznikne více údolí a rovných ploch. Naopak čím nižší exponent, tím více se výsledky funkce Perlinova šumu blíží k číslu 1[9]. Po provedení tohoto kroku již vzniká povrch, kde se vyskytují kopce a rovinatější údolí. Výsledek, který je zobrazen na obrázku 2.12 lze použít například jako lokaci mezi skalami, kde v nižší oblasti může téct řeka či zde mohou být budovy a tvořit vesnici.

Obrázky 2.10, 2.11, 2.12 byly vytvořeny pomocí webové stránky redblobgames.com.



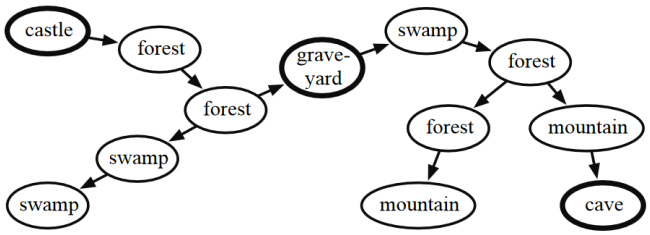
Obrázek 2.12: Heightmapa převedená na terén. Její body byly vygenerovány za pomocí součtu více frekvencí v různých poměrech a následně umocněny exponentem o hodnotě 5.

[9]Zdroj popsaného postupu: https://www.redblobgames.com

Genetické algoritmy

Genetické algoritmy jsou evoluční algoritmy založené na prohledávání, které se snaží nalézt optimální řešení pro optimalizační problém. Pro použití je nutná genetická reprezentace a tzv. fitness funkce. Genetická reprezentace je použita k zakódování možných řešení do řetězců, které se nazývají geny či chromozomy. Fitness funkce dokáže změřit kvalitu těchto chromozomů. Genetický algoritmus prochází iterativním procesem, ve kterém se aplikuje fitness funkce na jednotlivé chromozomy a vybírá několik nejlepších z této populace (proces selekce), zkombinuje je a vytvoří z nich jedince pro další populaci. Pro proces selekce lze použít metodu, která na základě zvolené pravděpodobnosti vybere řetězec pro kombinaci s jiným v závislosti na proporcionalitě jejich fitness: čím vyšší je jejich fitness, tím větší šance, že budou zkombinovány. Samotný proces kombinace se nazývá křížení. Dále lze použít mutace, což je proces, kdy je s nějakou malou pravděpodobností chromozom náhodně upraven. Mutace zaručují, že v nekonečném čase bude vždy nalezeno optimální řešení.

V rámci procedurálního generování bylo představeno mnoho metod, jednu z nich lze nalézt v knize *Toward Supporting Stories with Procedurally Generated Game Worlds* [2], kde je představena metoda automackého generování světů pro hry žánru RPG[10], přičemž výsledné světy vycházejí z dodaného příběhu. Příběh je k hernímu prostoru mapován za použití metafor mostů a ostrovů, které jsou uloženy v datové struktuře stromu, zde označeného jako *prostorový*. Ostrovy jsou oblasti, ve kterých se odehrává zápletka příběhu. Mosty pak spojují ostrovy, přesto jsou to stále lokace a ne nutně cesty ve smyslu průchodů, chodeb, silnic a podobně. Kromě propojení mezi mosty a ostrovy drží prostorový strom i informace o typu prostředí lokace, příklad na obrázku 2.13.

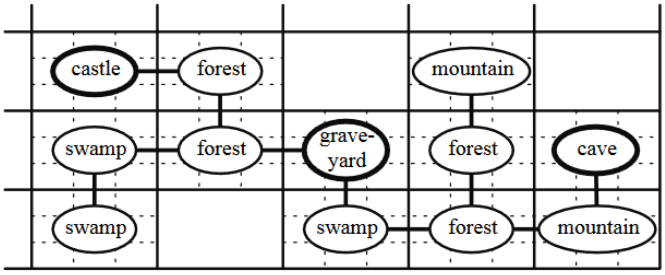


Obrázek 2.13: Příklad stromu propojující ostrovy (uzly s tučným obrysem) a mosty. Ostrovní uzly se vždy vyskytují podél jedné cesty stromu a jsou vloženy v pořadí, ve kterém se na ně ve vstupním příběhu poprvé odkazuje [2].

Tento strom je vytvářen genetickým algoritmem. Křížení a mutace se řeší přidáváním a odstraňováním uzlů a hran ve stromu. Fitness funkce zde pracuje s hodnocením propojených typů prostředí a data o herním stylu hráče (pro určení správného počtu a délek větví stromu). Nastavení pak zahrnuje velikost a lineárnost světa, pravděpodobnost na setkání s nepřáteli a nalezení pokladu. Proces generování optimálního stromu končí po specifikovaném počtu iterací. Výsledný strom je pak použit pro vygenerování 2D herního světa, kde uzly stromu jsou mapovány na mřížku (obr. 2.14) za pomocí algoritmu pro prohledávání do hloubky. Pokud není nalezeno řešení pro toto mapování, je strom zahozen a sestaven nový.

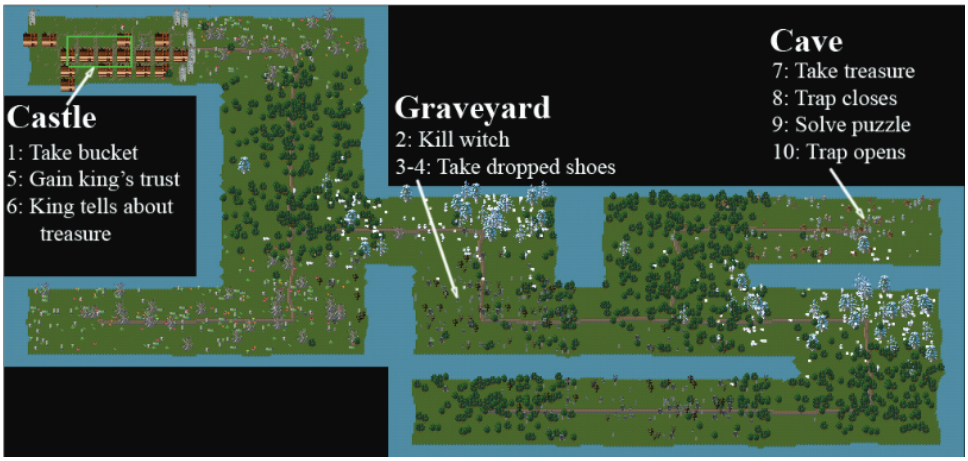
Jakmile je každému uzlu přiřazena oblast v mřížce, začne proces tvorby grafické reprezentace světa. Dle typu lokace jsou vybrány správné dekorace pro tento uzel (například pro les jsou dekorací stromy). Dekorace překrývají dlaždice (z ang. tiles), které jsou za pomocí

[10]Role playing games – žánr videoher, ve kterém hráč prochází hrou na základě úkolů a jejich postavě se průběžně zvyšují různé atributy a schopnosti



Obrázek 2.14: Strom z obrázku 2.13 převedený na mřížku. Pro zabránění sklonu algoritmu ke stavbě světa v jednom směru (například zleva doprava) je směr vkládání každé buňky náhodně upraven [2].

Gaussova rozdělení (a dalších metod) do oblasti rozptýleny vloženy. Grafická reprezentace je potom dokončena vyznačením hranic kolem průchozích regionů. Ty jsou reprezentovány vodou, do které hráč nemůže vstoupit, a tím mu zabrání odejít z mezí světa.



Obrázek 2.15: Výsledný svět pro mřížku 2.14. Zvýraznění lokace jsou typu ostrov, tedy uzly, ve kterých se odehrává zápletka příběhu [2].

Výhoda této metody je v tom, že svět není postaven náhodně, ale dokud je vstupní příběh správně sestaven, bude pro něj vygenerován svět. Výsledek tohoto přístupu také více připomíná manuálně vytvořený svět, což je při procedurálním generování vítanou vlastností. Popsaná metoda však přijímá jen jednoduché příběhy, není tedy jasné, jak snadné by bylo kontrolovat tuto metodu po přidání více vlastností příběhu a dalších nastavení [6].

2.4 Herní tituly s prvkem generování dungeonů

V této podkapitole jsou představeny některé významnější herní tituly, které při tvorbě obsahu používají procedurálně generované dungeony. Pokud je u dané hry známa metoda, kterou pro generování používá, bude zmíněna a stručně popsána.

Kromě těchto her existuje mnoho dalších a významných, které alespoň částečně procedurálně generují své prostředí a dungeony. Je zde nutno opět zmínit hru Minecraft, která generuje mnoho typů dungeonů (například jeskyně, podzemní komplexy měst a dolů, ruiny

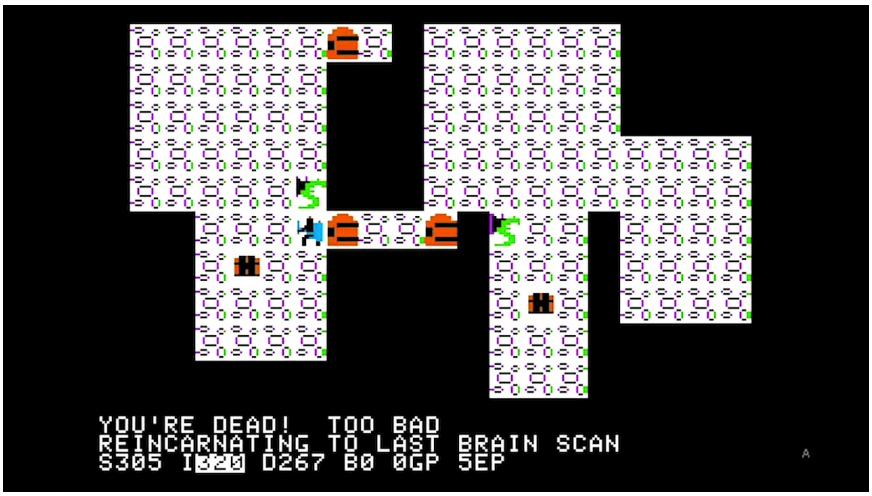
podvodní města, ...) a přitom je každou část možno hraním upravit. Binding of Isaac generuje pro své úrovně navzájem propojené místnosti, kterým je přidělen význam. Mohou se v nich nacházet různě silní nepřátelé, bedny s poklady, obchod, tajný prostor a mnoho dalších. Počet místností v každé úrovni postupně narůstá[11]. Dalšími z těchto her jsou Dwarf Fortress, Don't Starve, Diablo 1 nebo Terraria.

Beneath Apple Manor (1978)

Jednou z prvních her, která využila procedurální generování pro tvorbu herního obsahu, je hra Beneath Apple Manor. Hra byla vytvořena pro počítače Apple II programátorem Donem Worthem. Projekt byl napsán v jazyce BASIC a motivací bylo napsat simulaci populární stolní hry Dungeons & Dragons (zkr. D&D)[12].

Hráč má za úkol projít všech 10 úrovní a získat zlaté jablko. V pozdějším vydání s vylepšenou grafikou bylo úrovní pouze 5. Hráč v úrovni začíná na jediném poli a vidí pouze to, co se nachází v sousedních polích, zbytek úrovně je zakryt. Jak hráč postupuje, mapu postupně odkrývá a objevuje prostory a průchody mezi nimi.

Herní úrovně jsou generovány náhodně. Místnosti jsou vkládány tak, že jsou náhodně nastavovány souřadnice X a Y a na těchto souřadnicích se v herním prostoru vytvoří místnost. Takto vložené místnosti se často překrývají nebo leží těsně vedle sebe. Průchody jsou potom vytvořeny náhodným procházením herního prostoru z počátečních souřadnic jedné z místností, dokud není nalezena další místnost a proces se opakuje. Po dokončení generování dispozice dungeonu (obrázek 2.16) jsou vloženy bloky reprezentující dveře, poklad a nepřátele, kterých je několik typů a jejich síla se odvíjí od aktuální úrovně hry a počet zkušenostních bodů hráče



Obrázek 2.16: Ukázka herní úrovně ze hry Beneath Apple Manor. Ve vrchní části uživatelského rozhraní se nachází mapa s objevenými prostory. Hráč dále vidí aktuální pozici svou, nepřátel a interaktivní bloky s pokladem či dveřmi. Černé bloky jsou hranice úrovně.

Atributy hráčovy postavy jsou inspirované D&D a jsou to síla, inteligence, výdrž a tělo (reprezentující život). S těmito atributy je zacházeno jako se zásobou pro akce hráče. Ničení dveří a boj s nepřáteli ubírá sílu, kouzlení inteligenci, pohyb výdrž a obdržení poškození

[11] Proces procedurálního generování úrovně přiblížen na https://bindingofisaacrebirth.fandom.com
[12] Z rozhovoru s autorem na https://spillhistorie.no/...

snižuje zásobu těla. Úrovně také mohou obsahovat tajné průchody, které lze najít údery do zdi či vykouzlením rentgenového paprsku. Motivace hráče k zabíjení nepřátel je získání zkušenostních bodů, které na konci každé úrovně může vyměnit za navýšení zásoby některého atributu. Hráč tak může strategicky promýšlet, zda se mu vyplatí měřit síly s nepřáteli anebo se jim raději pokusit vyhnout. Při smrti hráč přijde o část svých atributů. Tomu lze předejít oskenováním charakteru mezi úrovněmi, čímž si hráč svůj postup uloží[13].

Unexplored

Unexplored je roguelite[14] hra, ve které hráč prochází procedurálně vygenerovanou úroveň, bojuje s nepřáteli, řeší hlavolamy a hledá odměny. Co je na hře zajímavého, je způsob jakým vytváří své úrovně – využívá totiž cyklického generování herního prostředí.

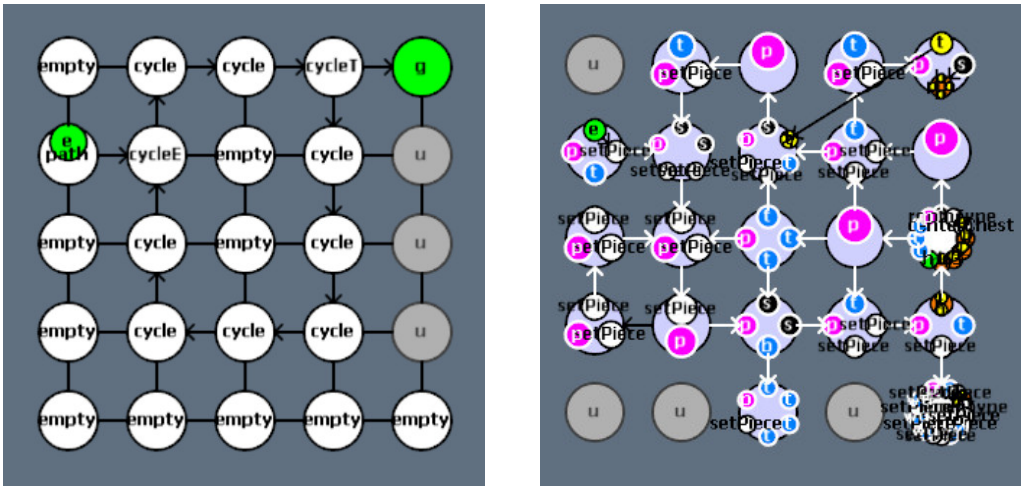
Cyklické generování v Unexplored[15] vychází z techniky transformace grafu. Každá úroveň má určený typ prostředí (prostředí je zde široký pojem, může znamenat například *oheň, voda, jeskyně, vodopády, místnosti*), které fungují jako rozhodovací reference pro některé kroky generování. Na počátku generování úrovně se vytvoří mřížka s uzly (transformovaný graf). Tyto uzly nejsou v průběhu generování nikdy odstraněny či přesunuty, pouze označovány různými informacemi. Z uzlů se označí začátek a konec a propojí se tak, aby vznikla smyčka. Smysl tohoto je, že vzniknou 2 cesty vedoucí mezi začátkem a koncem. Tyto cykly mohou být různých druhů (například cyklus se skrytou zkratkou, s jednou cestou velmi nebezpečnou a druhou velmi dlouhou, cesta bez možnosti návratu apod.), ale jsou pouze velmi obecné a kontrolují hlavní směr dungeonu, nikoli konkrétní informace o větvení či rozložení místností. Dále jsou přidávány menší cykly, překážky (může se později stát například nepřítelem, ale i hlavolamem či pastí) a slepé konce dokud nemá dungeon požadovanou složitost. Následně dojde k postupnému snižování abstrakce uzlů (obrázek 2.17) a informací – například z uzlu typu *cesta* se může stát jeskyně, místnost nebo tunel. Místnost potom může být knihovna, kovárna nebo třeba vězení a v dalším kroku jsou již vloženy dekorace pro konkrétní typ místnosti.

Z mřížky naplněné uzly s různými informacemi je potřeba vyrobit graficky reprezentované prostředí. To probíhá, zjednodušeně řečeno, zvětšením mřížky na rozlišení, které je dostatečně detailní pro specifikování tvarů oblastí a průchodů. Na každý blok jsou nyní aplikována pravidla a vznikne výsledný tvar úrovně. Všechny buňky ze zvětšené mřížky dostanou přidělený typ terénu, který vychází z typu prostředí. Takto připravená detailní mřížka (obrázek 2.18) může být předána hernímu enginu, který z ní již vytvoří hratelnou úroveň.

[13] https://episodiccontentmag.com/...

[14] Roguelite je podžánr RPG videoher charakteristický pro procházení procedurálně generovaných dungeonů a přenášení nějaké části postupu i po smrti hráče.

[15] Princip algoritmu i s obrázky dostupný na na https://www.boristhebrave.com/...



(a) Počáteční graf

(b) Finální podoba grafu

Obrázek 2.17: Na obrázku 2.17a Vlevo ukázka počátečního stavu grafu. Jsou vyznačeny uzly se začátkem (zelená ikonka se znakem e) a koncem (zelený uzel se znakem g), směr a cestu hlavního cyklu (uzly s označením *cycle*. Na obrázku 2.17b je příklad dokončeného grafu, který v sobě obsahuje množství informací o konkrétním stavu každého uzlu. Transformacemi grafu se výsledek změnil natolik, že je jen velmi těžko srovnatelný s jeho počátečním stavem. To je však jen výsledek mnoha aplikovaných pravidel na transformovaný graf, aby byla herní úroveň co nejzajímavější a rozmanitá. Obrázky převzaty a upraveny[16]

2.5 Aktuálně dostupná řešení pro obecné využití

Tato podkapitola představuje několik řešení pro procedurální generování dungeonů, které mohou být, mimo jiné, určitým způsobem použity pro Unreal Engine, ale jsou zaměřeny pro univerzální použití. Tato řešení mají často výstup v běžně užívaném formátu dat (JSON, XML), který lze snadno zpracovat do jiných technologií, jazyků a frameworků. Nevýhodou naopak může být, že pro opakované a přímé fungování s jinými technologiemi je třeba spouštět či komunikovat s externími aplikacemi, což může zdržovat běh celého procesu. Jedním možným řešením je požadovaná data připravit dopředu a přiložit ke zdrojovým souborům projektu, čímž se může omezit počet použitelných výsledků procedurálně generovaného obsahu a mohou se zvýšit požadavky na velikost úložiště.

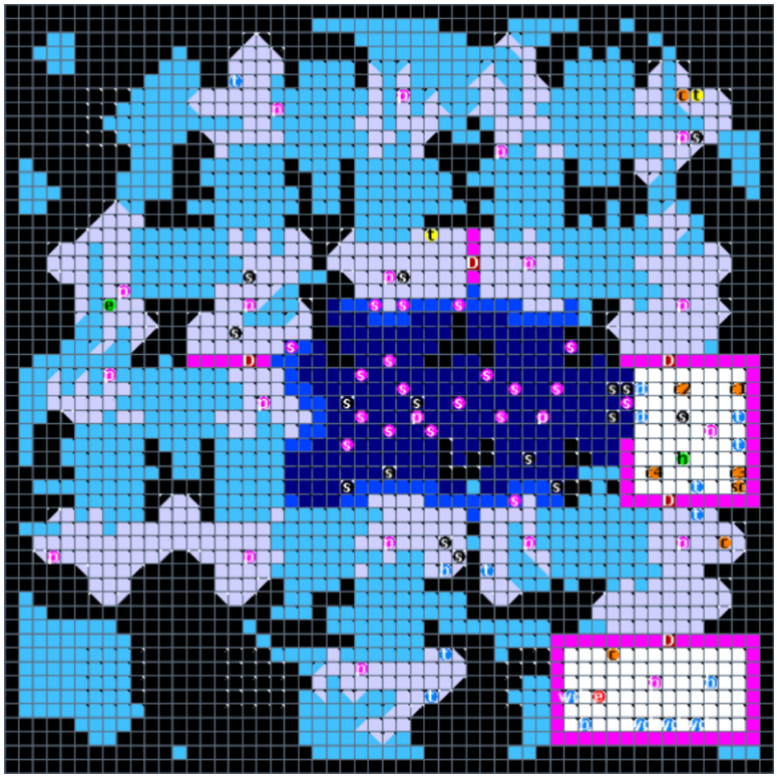
Edgar Dotnet

Knihovna je pro framework .NET a herní engine Unity[16] a umožňuje generovat 2D dungeony a mapy pro hry žánru platformer[17]. Cílem tohoto generátoru je přenechat co možná největší kontrolu nad generovanými úrovněmi uživatelům knihovny. Využívají se zde předdefinované šablony pro místnosti, ze kterých algoritmus vybírá a skládá strukturu dungeonu.

[16]Projekt v obou variantách lze nalézt na webových stránkách https://ondrejnepozitek.github.io/Edgar-DotNet/.

[17]Plaformer je žánr videoher, ve kterém se hráč z pravidla pohybuje ve 2D prostoru směrem do stran a nahoru/dolů po různých platformách. Základem těchto her je často mechanika skákáni či šplhání - zdroj: https://www.musicgateway.com/blog/...

Příkladem může být hra Super Mario Bros ze série Mario nebo hra Jump King.



Obrázek 2.18: Graf byl převeden na detailní mřížku, ve které se připravovala podrobná a finální verze reprezentace úrovně dungeonu. Tato mřížka je nyní připravena pro převedení do herního prostoru herním enginem.

Princip knihovny funguje na bázi upravování grafu, který vyjadřuje kolik propojených místností má výsledek mít a jak mají být propojeny. S tímto grafem může uživatel přímo pracovat a upravovat pokročilé parametry, jako například kolik cest má vést ke konečné místnosti, jestli mají existovat cesty se slepým koncem, jaké místnosti jsou se kterými přímo propojeny, minimální vzdálenost mezi místnostmi apod. Výsledek lze potom uložit ve formátu JSON nebo jako obrázek (příklad na obrázku 2.19).

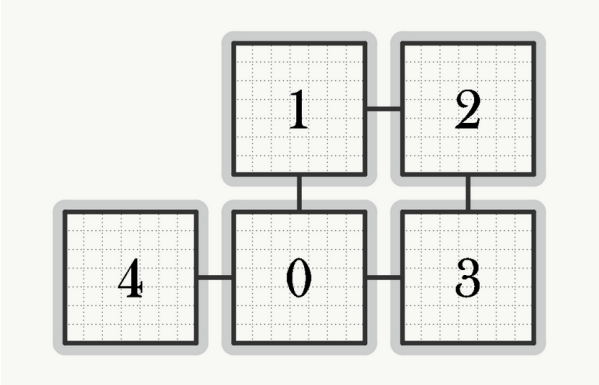
Mnohostrannost a flexibilita knihovny je zde na úkor výkonnosti. V dokumentaci je doporučeno nepřesahovat více než 30 místností a některé grafy mohou být příliš složité pro algoritmus na vygenerování – především se jedná o smyčky, případně vzájemně propojené smyčky. Další nevýhodou je, že knihovna nemá uživatelské prostředí, uživatel tedy musí mít dostatečné znalosti daného jazyka. Varianta balíčku pro Unity má po nainstalování uživatelsky přívětivé rozhraní v rámci prostředí enginu.

Roguelike web api

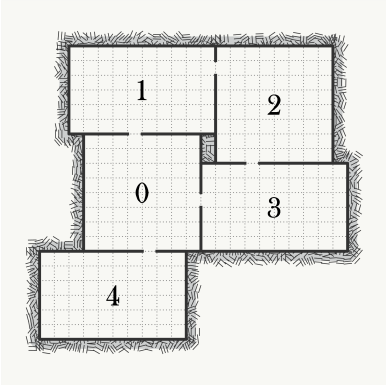
Jedná se o aplikaci[18] fungující na technologii Node.js[19] jako server s přístupným API pro HTTP požadavky metody GET. Po odeslání požadavku přichází odpověď ve formátu JSON se souřadnicemi popisující pozice stěn nově vygenerovaného dungeonu.

[18] Projekt, dokumentace a demo aplikace dostupné z https://github.com/MattMcFarland/rot-web-api

[19] Node.js je open source serverové prostředí pro jazyk JavaScript. Dokumentaci a bližší popis lze nalézt na https://nodejs.org/



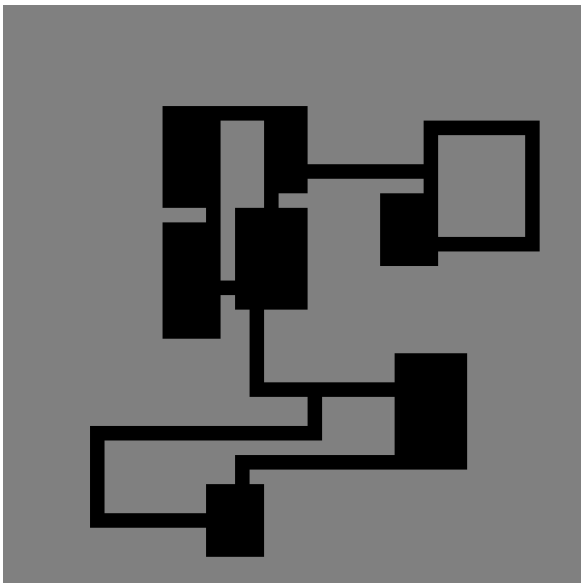
(a) Prezentace grafu



(b) Příklad výstupu

Obrázek 2.19: Vlevo je zjednodušený graf propojení místností, vpravo pak jeden z možných výstupů, přičemž šablony místností jsou čtverec a obdélník o rozměru 8×8 a 6×10 buněk[17].

Knihovna nabízí, dle dokumentace generování 7 typů úrovní, kterými jsou *Rogue*, *Digger* (obrázek 2.20), *IceyMaze*, *EllerMaze*, *Cellular Automation*, *Uniform* a *Arena*. Většina těchto typů přijímá pouze parametr určující výšku a šířku úrovně – *Rogue* přijímá i výšku a šířku místnosti a *Cellular Automation* topologii (4, 6 nebo 8).



Obrázek 2.20: Ukázka výstupu pro dungeon typu *Digger* s oběma parametry (šířka a výška) mající hodnotu 40. Výstup byl vizualizován pomocí demo aplikace odkazované v dokumentaci projektu.

Výhody této aplikace jsou jednoduchost použití aplikace a jejích parametrů – kroky instalace jsou popsány dostatečně a GET požadavky jsou dostatečně dokumentovány a demonstrovány v demo aplikaci. Forma serveru umožňuje obsluhovat více uživatelů současně a ti mohou na server posílat požadavky jakýmkoli způsobem, není tedy nutná znalost konkrétního programovacího jazyka. Nevýhodami jsou nemožnost záměrně vyvolat stejný výsledek vícekrát (parametry nepřijímají seed) a málo parametrů neumožňuje téměř žádné přizpůsobení generování úrovně pro potřeby uživatele.

Další projekty

Kromě zmíněných existuje nespočet dalších volně dostupných projektů (jen ve službě GitHub bylo ke dni 9. května 2022 pod vyhledáním pojmu „dungeon generator“ nalezeno 1484 repozitářů), které nějakým způsobem řeší procedurální generování dungeonu nebo herní úrovně. Mnohdy však nejsou dostatečně dokumentovány (často ani základní informace o formě výsledku či dokonce způsobu použití), jejich výstup nelze jednoduše převést do snadno automaticky zpracovatelného formátu (například výstup formou obrázku), jsou připraveny pro použití pouze ve specifické aplikaci (jako zásuvný modul do některé hry či herního enginu) nebo jsou samostatně fungující aplikací, kterou nelze využít pro vývoj jiného projektu.

Některé zmíněné nedostatky se dají částečně vyřešit upravením dané knihovny a zásahem do zdrojových souborů. Avšak to vyžaduje dostatečné znalosti technologie či programovacího jazyka a je nutno pochopit strukturu a fungování projektu. Krom toho může být porušena licenční smlouva.

2.6 Aktuálně dostupná řešení pro Unreal Engine

V této podkapitole jsou představeny některá z dostupných řešení pro Unreal Engine. Jsou vypsány jejich možnosti, přednosti a způsoby použití.

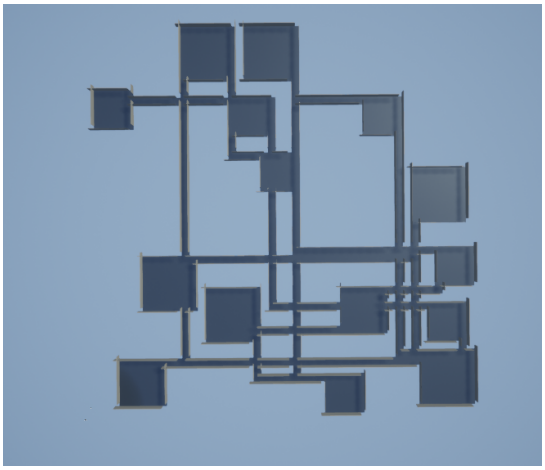
Orfeas Dungeon Generator

Jedná se o open source plugin[20] pro Unreal Engine verze 4 a 5. Funguje tak, že nejdřív vytvoří matici o rozměru, který nastaví uživatel, a poté se do náhodných pozic pokouší vložit prvky reprezentující místnost o rozměrech v mezích z parametrů od uživatele. Pokud se prvky povede vložit, jsou prvky matice označeny jako obsazené a vytvoří se co nejkratší chodba k naposledy přidané místnosti. Prvky chodby jsou označeny také jako obsazené. Pokud místnost nelze vložit, vybere se jiná náhodná lokace a tento proces se opakuje, dokud nepřesáhne uživatelem zadaný počet možných neúspěšných pokusů, než se přejde k další místnosti. Zdi jsou vkládány na prvky matice okrajů místností v případě, že je sousední prvek směrem ven z místnosti prázdný. Výsledná matice potom poskytne finální pozice a rotace pro objekty stěn a podlah. Jako parametry plugin přijímá hranice velikosti místnosti, rozměry matice a nastavení objektů, které budou pro výsledný dungeon použity.

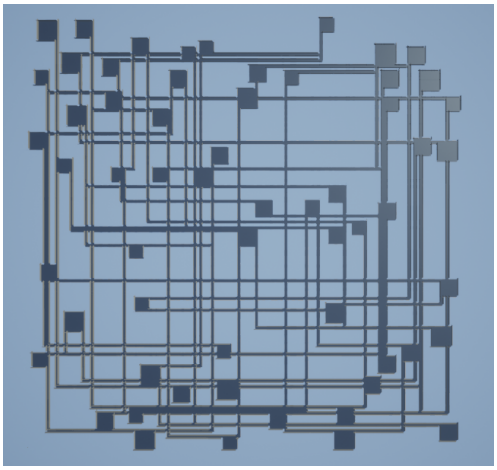
Výhodou je formát řešení ve formě pluginu (popsáno dále v práci v kapitole 3.1). Použití je pro nižší počet parametrů velmi jednoduché a při nižším počtu místností a menším rozsahu mřížky se uživatel snadno dopracuje uspokojivého výsledku. Čím větší počet místností a velikost matice, tím značněji algoritmus zatěžuje systém[21] (krom algoritmu samotného jde i o počet objektů, které vytváří), proces generování se prodlužuje a výsledné chodby tvoří nepřirozeně propojenou strukturu. Výsledek navíc nelze vygenerovat znovu při stejné konfiguraci – parametry neobsahují seed.

[20] Dostupný z: https://github.com/orfeasel/DungeonGenerator

[21] Testováno na mřížce o rozměrech 2000×2000 a počtu místností 150. Proces byl ukončen uživatelem po 7 minutách běhu bez výsledku. Počítač použitý na testování disponuje procesorem AMD Ryzen 3600x (3,6GHz) a 16 GB RAM.



(a) mřížka 50 × 50, 15 místností



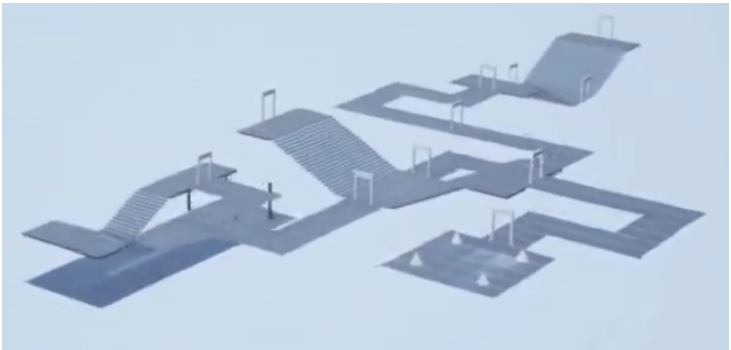
(b) mřížka 150 × 150, 60 místností

Obrázek 2.21: Porovnání výsledku pro rozdílná nastavení velikosti mřížky a počtu místností.

Procedural Dungeon (BenPyton)

Plugin[22], který umí procedurálně generovat dungeony podobající se těm ze hry Binding of Isaac. Výsledek je však ve 3D a místnosti mohou být vkládány nad sebe. Uživatel si navíc sám může naplnit seznam objektů místností, ze kterých se výsledný dungeon bude skládat.

Proces generování začíná odstraněním všech místností, které již v úrovni jsou. Následně se spustí nové generování, kdy je vybrána jedna místnost, ke které se vloží jedna nebo více místností. Jakmile je dosaženo požadovaného počtu sousedních místností, vybere se jiná místnost a proces se opakuje dokud není splněna podmínka pro konec generování dungeonu. Vše probíhá mimo herní prostor, do toho se místnosti opravdu vloží až ve chvíli, kdy je dokončen proces uspořádávání místností.



Obrázek 2.22: Příklad výstupu pro 4 různé objekty místností, převzato z videa odkazovaného v dokumentaci[23].

Konfigurace pluginu probíhá pomocí Blueprints[23] a před spuštěním generování je potřeba upravit minimálně 5 funkcí, které například získávají speciální data o místnosti či

[22] Dostupný z https://github.com/BenPyton/ProceduralDungeon

[23] Blueprints je systém vizuálního skriptování v Unreal Enginu. Jednou z výhod tohoto systému je větší vizuální přívětivost pro vývojáře, kteří nejsou zběhlí v kódování v C++, kdy zároveň nepřichází o téměř žádnou běžnou funkcionalitu. Nevýhodou je přehlednost, která se snižuje při přidávání více funkcionality rychleji než v kódu.

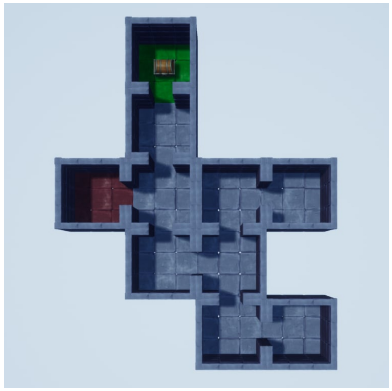
ověřují, jestli je již dungeon dokončen (může jít o například o kontrolu, zda byla vložena místnost pro přechod do další úrovně nebo jestli počet místností je již postačující). Uživatel může zvolit mezi 2 typy generování:

1. *depth first* – bude prioritizovat naposledy vloženou místnost pro přidávání další a povede tedy k více lineárnímu výsledku
2. *breadth first* – prioritní je první vložená místnost a vede k více rozvětvenému dungeonu

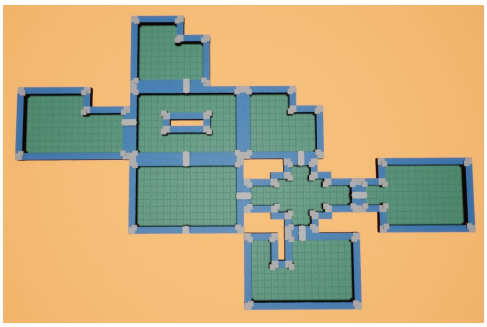
Z obrázku 2.22 je patrné, že místnosti na sebe vhodně navazují a lze si i představit možnosti dalšího rozmístění nepřátel do úzkých prostorů či vložení pokladů do slepých cest dungeonu. Rychlost generování závisí především na použitých objektech místností a podmínkách pro ukončení generování, které stanovuje hráč.

Další komerční řešení z Unreal Engine Marketplace

Další zajímavé, často zpoplatněné, generátory dungeonů lze nalézt přímo na Unreal Engine Marketplace. Jedním z nich je *Dungeon Generator* od autora *Koromelo*[24], který vytváří všechny místnosti ve stejném rozměru, jsou přímo průchozí mezi sebou (za chodby se dají považovat místnosti bez dalšího významu) a umí vložit místnost s bossem[25] a pokladem. Parametry jsou velmi jednoduché – nastavuje se jen počet místností, jejich velikost a modely, ze kterých se staví. Právě díky své jednoduchosti může být vhodný například pro minihry nebo pro dungeony s náměty bludiště (obrázek 2.23).



(a) Koromelo Dungeon Generator



(b) Simple Dungeon Generator

Obrázek 2.23: Vlevo příklad výstupu Dungeon Generatoru od autora Koromelo[24]. Vpravo ukázka výstupu Simple Dungeon Generatoru[26].

Na závěr kapitoly o existujících řešeních je vhodné zmínit *Simple Dungeon Generator*[26], který umí generovat rozložení místností na základě uživatelem vložených objektů, ukládat stav hry po projití hráče místností, automaticky následovat hráčův pohyb kamerou s plynulým přecházením mezi místnostmi, vkládat speciální objekty či nepřátele do místností s určitou šancí a rozsahem, využívat mnoho modelů pro tvorbu prostředí, měnit hudbu v závislosti na typu místnosti a herní úrovně a další. Nevýhodou může být velký počet parametrů, který je třeba upravit, než uživatel přizpůsobí tento systém pro své potřeby.

[24] Dungeon Generator dostupný z https://www.unrealengine.com/marketplace/...

[25] Jako *Boss* se v počítačových hrách zpravidla nazývají důležití (a často mnohem silnější) nepřátelé, které je nutné porazit pro získání pokladu či dokončení úkolu.

[26] Simple Dungeon Generator dostupný z https://www.unrealengine.com/marketplace/...

Kapitola 3

Návrh generátoru ve formě pluginu

V této kapitole je rozepsán návrh řešení práce. Nejprve vysvětluje důvod použití řešení typu plugin a příklady jiných možností řešení pro Unreal Engine 4. Dále jsou popsány požadavky na ovládání a chování pluginu, což zahrnuje i požadavky na výsledné dungeony. V neposlední řadě je popsán algoritmus jak pro proces vytváření a rozmisťování místností a chodeb, tak pro vytváření příkladových místností.

3.1 Požadavky

Požadavky na formát řešení

Prvním podstatným požadavkem je, aby výsledek bylo možné spustit v herním enginu Unreal Engine 4. Jednou možností je vytvoření požadové logiky v C++, případně v systému Blueprints, a tyto soubory pak distribuovat například přes službu GitHub. Nevýhodou je, že soubory Blueprints jsou typu `uasset`, které lze přečíst pouze v samotném editoru a nelze do nich, například ve zmíněném GitHubu, z toho důvodu nahlédnout před stažením a vložením do Unreal Engine. Pokud je k takto napsané logice také nutno dodat další soubory, přestává být stahování a vkládání souborů do příslušných adresářů pohodlné.

Další možností je vytvoření celého projektu s požadovanou logikou i herními objekty. Tento způsob je vhodný, pokud jde například o šablonu, která je určená k tomu, že z ní vývojář bude vycházet a přizpůsobovat potřebám svého projektu. Pakliže je ale toto řešení zamýšleno k vložení do jiného projektu, je nutno projekty sloučit a kromě nepohodlnosti je zde třeba brát i v potaz možnou rozdílnost verzí projektů.

Zásuvný modul, neboli plugin, lze jednoduše nainstalovat přímo do existujícího projektu vložením pluginu do podadresáře Plugins v kořenovém adresáři projektu. Druhou možností je instalace pomocí Epic Games Launcheru[1], kterým lze přistoupit na Unreal Engine Marketplace[2], kde jsou i vypsány podporované verze enginu. Pluginy v sobě mohou obsahovat kromě kódu i kterékoli jiné soubory užívané v projektech Unreal Enginu.

[1] Epic Games Launcher je aplikace společnosti Epic Games, která, kromě jiného, vyvíjí Unreal Engine. Díky této aplikaci lze jednoduše spravovat nainstalované verze Unreal Enginu a jeho projekty v počítači uživatele.

[2] Unreal Engine Marketplace je online služba, jejíž uživatelé zde mohou prodávat své díla, ať jde o celý projekt, plugin, model, kód, animace aj.

Požadavky na plugin a strukturu dungeonu

Další požadavek na plugin je, aby neočekávaně neskončil. Kromě vývojového prostředí může být použit i v projektu distribuovaného hráčům. Je proto vhodné, aby místo toho vypsal chybovou hlášku do případně dostupné konzole a bezpečně ukončil činnost.

Mezi další požadavky patří **náhodnost**, **opakovatelnost** a **rychlost**. Výsledek generování bude být při každém spuštění alespoň částečně odlišný, aby si uživatel mohl vybírat z co možná největšího vzorku vygenerovaných žalářů. Nastavitelností se rozumí přítomnost parametrů, které ovlivňují proces generování. Rozhraní pro parametry by mělo být patřičně popsané a intuitivní. Mezi takové parametry bude patřit například počáteční seed či velikost a počet místností. Opakovatelnost pak zaručí, že uživatel může zopakovat aktuální výsledek generování uložením stavu parametrů a za pomocí toho později vyvolat stejný výsledek. Průměrně velký dungeon (v jednotkách desítek až stovek místností) by se měl vygenerovat dostatečně rychle (maximálně v jednotkách), aby plugin příliš nebrzdil projekt, který jej využívá.

Výsledný dungeon by měl mít takovou strukturu, aby se dalo vejít do všech vzniklých místností a projít všemi chodbami. Kromě lineárního průchodu by měly vzniknout i smyčky, tedy možnost se dostat do místnosti více chodbami, ale zároveň i slepé konce, kde může být uživatelem generátoru později například umístěn vstup do další úrovně, například střežený silným nepřítelem. Místnosti by dále neměly být příliš daleko od sebe, aby nevznikaly příliš dlouhé chodby, což by mohlo při procházení dungeonu působit nudně. Dále se mohou místnosti lišit ve velikost a měly by se dát upravovat objekty, které místnosti či chodby tvoří.

Požadavky na příkladově vybavené místnosti

Koncept těchto místností bude spočívat v účelech testování pro uživatele. Například by měl uživateli umožnit odhadnout, kolik objektů zde může umístit. Mnoho místností plně vybavených objekty bude mít navíc dopad na výkon herního enginu, s čímž může uživatel pracovat a před ručním vybavováním a upravováním místností vzít tuto skutečnost v potaz. Dále mohou sloužit jako jednoduché dynamické prostředí pro rychlejší testování vyvíjené hry či projektu. Uživatel by měl mít možnost vybavování vypnout, popřípadě ponechat pouze stavbu těch objektů, které požaduje.

Vybavené místnosti by měly být vždy průchozí i s vloženým vybavením. Toto vybavení by zároveň mělo jít uživatelem pluginu upravovat, ať už jde o objekty či jejich velikost a počet polí, které tento objekt zabere. Každý typ objektu bude při umisťování dodržovat pravidla, která zajistí, že objekt bude umístěn logicky (například objekt knihovny bude směřovat směrem k chodbě, nikoli ke zdi či jiné knihovně, a zároveň nebude kolidovat s jiným objektem). Místnosti by měly obsahovat různý počet vybavení a při stejné konfiguraci a seedu by měly být výsledné místnosti v různých běžích generátoru vždy stejné.

Konkrétní typy místností s jejich požadavky:

- **Knihovna** – průchody a chodby by měly být pravidelné a málo členité. Četnost průchodů ve směru po ose X a Y by mělo být možné upravovat parametrem. Kolem vzniklých chodeb budou vsazeny objekty reprezentující objekt knihovny tak, aby byly otočeny směrem k chodbě. Dále mohou být vloženy objekty lamp pro částečné osvětlení místnosti a přidání dalšího typu nábytku. Pro dynamičnost je možné vložené objekty rotovat či měnit jejich velikost.

- **Sál** – měl by prezentovat stroze vybavenou místnost se sloupy a lampami. Sloupy by měly být stavěny pravidelně a jejich četnost by se měla dát měnit. Pro menší pravidelnost bude možno sloupům nastavit odchylku, o kterou se mohou náhodně posunout před vložením do sálu. Pokud bude v sálu dostatek místa, může být vložen trůn.
- **Skladiště** – chodby by měly být členité, ale stále by mělo být možné jimi projít skrze místnost. Na chodby budou navazovat volné placy, neboli nevyužité části skladiště. Mimo chodby a volné placy budou náhodně vloženy různé objekty, které uživatel specifikuje v parametrech pluginu.

3.2 Návrh algoritmu pro vytvoření struktury dungeonu

Návrh vychází z algoritmu, který byl použit ve hře TinyKeep a zveřejněn samotným autorem hry[3]. Tento algoritmus využívá technik, jako je Delaunayova triangulace, minimální kostry grafu či detekování kolizí. Výsledkem jsou místnosti a chodby rozmístěné dostatečně blízko u sebe, přičemž samotné chodby se mohou částečně skládat z menších místností. Zároveň proces umožňuje dosazení a doplnění parametrů, kreté lze zpřístupnit uživateli. Rozmístěné objekty by po výsledném generování měly být zarovnány do mřížky tak, aby byly dodrženy minimální rozestupy mezi místnostmi. Všechny náhodné kroky v průběhu generování by měly vycházet z generátoru pseudonáhodných čísel, který pracuje se seedem. Tento fakt je důležitý pro opakovatelnost procesu.

Vytvoření a rozložení místností

Algoritmus vkládá místnosti na náhodně vybrané body z kruhu. Jelikož objekty výsledného dungeonu mají být zarovnány do mřížky, místo náhodných bodů z kruhu budou vybrány náhodné body z mřížky, jejíž rozměr bude určen parametrem v kombinaci s velikostí podlahy místnosti (velikost podlahy určuje i rozměr buněk mřížky). Na vybrané body se následně vloží místnosti s náhodnou velikostí.

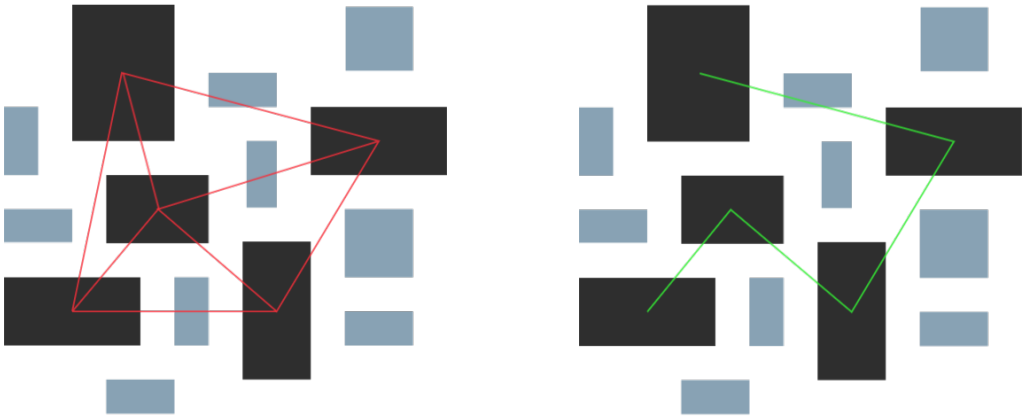
Takto umístěné místnosti se budou v prvotním stavu pravděpodobně překrývat a je potřeba je rozprostřít v herním prostoru. Na tento proces stačí jednoduchá metoda, která vypočítá směrový vektor pro posunovanou místnost v závislosti na místnostech, které se s ní překrývají. Tento vektor se následně zarovná tak, aby posunutá místnost opět seděla do mřížky a tento proces se opakuje, dokud se žádná místnost nepřekrývá s jinou. Zde lze také přidat prvek náhodnosti a vektor mírně pozměnit před zarovnáním do mřížky. Pro detekci kolize a překrývání místností lze využít komponentu Unreal Enginu zvanou Static Mesh Component[4], která umí reprezentovat objekt složený ze statického počtu polygonů.

Propojení místností

Další část algoritmu popisuje vytvoření grafu pomocí Delaunayovy triangulace mezi těmi nejdůležitějšími místnostmi, pro tuto práci to budou ty největší. Největší místnosti lze vybrat vypočítáním průměrného obsahu a místnost, která tento průměr převyšuje, bude označena jako *hlavní*. Tento obsah lze také vynásobit parametrem, což uživateli umožní rozšířit či zúžit výběr *hlavních* místností. Ostatní místnosti budou zatím označeny jako *nepoužité*. Po vybrání *hlavních* místností bude mezi jejich počátečními souřadnicemi vytvořen

[3] Zveřejněno na https://www.gamedev.net/
[4] Detailnější popis lze nalézt v dokumentaci https://docs.unrealengine.com/4.26/...

graf pomocí Delaunayovy triangulace. Takový graf sice splňuje, že z každé místnosti se lze dostat do jiné, avšak tato struktura by při vložení chodeb byla až příliš propojená a mohla by působit chaoticky. Proto se ve vytvořeném grafu nalezne minimální kostra, například pomocí Jarníkova algoritmu. Ta zaručí, že graf a výsledný dungeon budou lineárně průchozí (znázorněno na obrázku 3.1). V rámci vytvoření smyček mezi uzly grafu lze obnovit určitý poměr zahozených hran. Tento graf bude považován za výsledný a bude z něj vycházet stavba chodeb.



(a) Delaunayova triangulace

(b) Minimální kostra

Obrázek 3.1: Vlevo znázornění propojení hlavních místností pomocí Delaunayovy triangulace. Vpravo je již pouze minimální kostra. Některá původní propojení mohou být pro přidání komplexity obnovena.

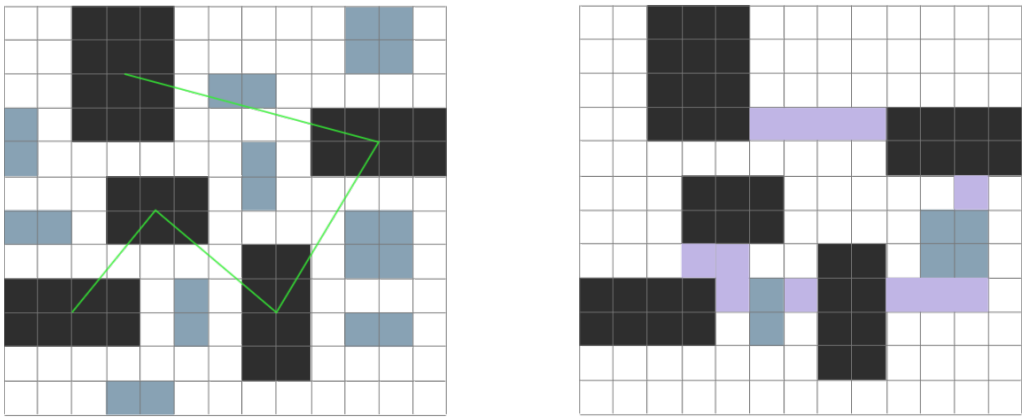
Aplikování mřížky a vytvoření chodeb

Všechny místnosti jsou nyní zarovnány v prostoru na mřížku, jejíž buňky mají rozměr velikosti modelu podlahy. Kromě zarovnání však tato mřížka nemá žádný význam a nelze se odkazovat na jednotlivé buňky. Proto před stavěním chodeb bude potřeba vytvořit reprezentaci mřížky i interně, aby jednotlivé buňky mohly obsahovat patřičné informace (pozici, zda a co se v nich nachází, jaký má buňka účel, zda do ní lze umístit chodbu či nikoli apod.). Tuto mřížku lze reprezentovat jako jednorozměrné pole, jehož velikost bude záležet na minimální a maximální souřadnici, ke které se místnosti rozmístily (lze vidět na obrázku 3.2). Tato mřížka a její buňky se mohou využít například i ke stavbě zdí a příkladových místností. Při vytváření mřížky bude nutné vzít v úvahu existující místnosti a dle jejich pozice a velikosti patřičné buňky označit jako buňky obsahující místnost.

S interně vytvořenou reprezentací mřížky a již předtím vytvořeným grafem propojení místností bude možno začít stavět chodby. V případě, že se propojené místnosti po ose X nebo Y překrývají, mohou vést chodby z jedné do druhé místnosti rovně, tedy jen po jedné ose. V opačném případě bude nutné vytvořit chodbu do tvaru písmene L. Kolize s jinými místnostmi či chodbami lze řešit následovně:

1. Pokud jde o kolizi s místností, která je označená jako *nepoužitá*, označí se tato místnost jako *vedlejší*, úsek buněk obsazené touto místností se přeskočí

2. Při kolizi s *hlavní* místností či s chodbami se přeskočí buňky obsazené touto místností



(a) Aplikována mřížka

(b) Vsazeny chodby

Obrázek 3.2: Obrázek nalevo znázorňuje interní reprezantaci mřížky. Každá buňka bude obsahovat informace o místnosti (a později chodbě), kterou je obsazena. V pravém obrázku je již finální rozložení s vloženými chodbami a odstraněnými nepoužitými místnostmi.

V neposlední řadě se odstraní všechny místnosti, které byly označeny jako *nepoužité* z prostoru i z interní reprezentace mřížky. Tyto místnosti nebyly propojeny žádnou chodbou s kteroukoli *hlavní* místností a nenesly tedy žádný význam.

V tomto stavu by měl být dokončen základní proces generování, místnosti by měly být náležitě propojeny chodbami a všechny nepoužité místnosti by již neměly existovat, jak je znázorněno na obrázku 3.2.

3.3 Návrh algoritmu pro vybavování místností

Vybavení každé místnosti se může vkládat po směru osy X nebo Y. Tento směr bude před začátkem označování chodeb a vkládání objektů náhodně určen pro každou místnost. Pro všechny typy místností také platí, že jako první se budou vkládat zdi podél stran místnosti. Je však nutné řešit situace, kdy buňka, na kterou se zeď vkládá, sousedí s chodbou. V tomto případě je možné nestavět zdi vůbec. Výhodou je, že (například u chodeb ve tvaru L) by mohla část místnosti mít zeď a část nikoli. To by přidalo na zajímavosti a otevřenosti prostoru. Nevýhodou je, že by prostor mohl být zase málo oddělený. Pro tento případ bude zaveden parametr pro uživatele.

Knihovna

Pro knihovnu se budou nejprve vyznačovat průchody, které musí zaručit průchodnost místnosti. V tomto případě může být vhodné navázat na vstupy do místnosti, pokračovat v jejich směru skrze celou místnost. Následně se vytvoří další průchody po směru osy X a Y s náhodnými rozestupy tak, aby vznikl prostor pro vkládání objektů knihoven. Ty se vloží tím způsobem, že se vyberou buňky v mezích místnosti, které jsou stále volné, vytvoří se objekt knihovny a po úpravě rotace a pozice se na pozici buněk vloží (objekty mohou zabírat i více buněk). Dále se budou vkládat lampy. Kromě volných buněk mohou být vloženy na buňky průchodu tak, aby ho zároveň neblokovaly (například situace kdy je více průchodů vedle sebe či nad sebou).

Sál

Počet sloupů v sále bude záviset od velikosti místnosti a parametru, který určí jaký rozestup mají mezi sebou sloupy mít. Sloupy se vloží tak, aby měly mezi sebou daný rozestup a zároveň měly co nejpodobnější odstup od zdi. Zároveň se bude kontrolovat, aby sloup nezablokoval vstup do místnosti. Kromě sloupů se místnost vybaví i lampami, které budou nejprve náhodně rozloženy do rohů místnosti a poté kolem sloupů. Opět bude kontrolováno, že lampa nezablokovala vchod.

Skladiště

Skladiště má mít členité a nepravidelné průchody. Toho lze docílit, že se některým algoritmem pro prohledávání prostoru naleznou cesty mezi vstupy. Pro tento případ lze použít například Depth First Search[5], který nezaručuje nalezení nejkratší cesty, což je vyhovující pro nepravidelnost průchodů. Od označených průchodů budou rozvinuty volné prostory, tedy několik sousedících buněk, které budou ve skladišti představovat nevyužité místo. Kolem průchodů se pak vloží lampy a na zbytek buněk místnosti budou vloženy uživatelem vybrané objekty.

[5]Depth First Search je prohledávací algoritmus, který postupně nalézá všechny listy datové struktury typu strom https://www.programiz.com/dsa/graph-dfs

Kapitola 4

Implementace pluginu

Tato kapitola popisuje implementační detaily práce, jako je její struktura či použité technologie a knihovny. Dále bude popisovat jakým způsobem jsou vyřešeny navrhnuté části softwaru a jak moc se od návrhu odklánějí.

4.1 Základní struktura

Jako formát práce byl vybrán zásuvný modul, neboli plugin, pro Unreal Engine verze 4.26. V kořenovém adresáři pluginu se nacházejí podadresáře, které obsahují:

- `Binaries` – kompilovaný kód pro plugin
- `Intermediate` – dočasné soubory pro sestavení
- `Content` – soubory s herním obsahem, jako jsou materiály, soubory Blueprints, modely, animace apod., případně může obsahovat podadresáře s těmito soubory. V rámci tohoto řešení se zde nacházejí příkladové objekty podlah, zdí, knihoven apod. s jejich materiály
- `Config` – konfigurační soubory
- `Resources` – další soubory pro plugin (například ikonka pluginu)
- `Source` – moduly se zdrojovými kódy v C++. V podadresáři `DungeonGenerator` se nacházejí další 2 adresáře – `Public` a `Private`, pro hlavičkové, respektive zdrojové soubory

Veškerá funkcionalita byla vytvořena v jazyce C++, Blueprints byl použit pouze na vytvoření příkladových objektů a herní objekt C++ třídy generátoru, který může být díky tomu umístěn do herní úrovně a má svou pozici, která se využívá jako výchozí bod pro generování dungeonu.

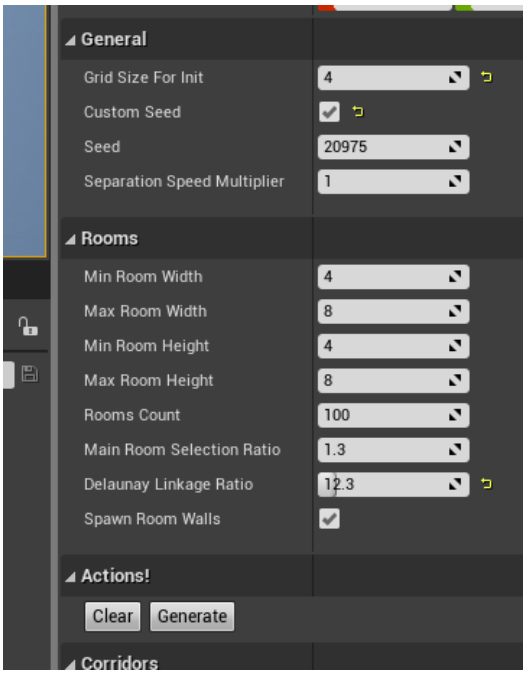
4.2 Konfigurace

Dle návrhu byly vytvořeny parametry, které může uživatel upravovat. Tyto parametry jsou rozděleny do sekcí dle jejich významu a souvislostí. Základní parametry jsou v sekci *General* a *Rooms*, jak lze vidět na obrázku 4.1. Tyto parametry ovlivňují výslednou strukturu a

rozložení dungeonu a jsou definovány v hlavičkovém souboru `DGGenerator.h`. Ve stejném souboru lze nalézt i konfiguraci objektů pro tvorbu chodeb.

Základní parametry pro generování příkladových místnosti jsou ve struktuře v souboru `DGRoomStruct.h`. Z této struktury poté dědí i struktura knihovny a sálu nacházející se v souborech `DGLibraryStruct.h` a `DGHallStruct.h`. Mezi tyto parametry patří možnost negenerovat tento typ místnosti a dále to jsou modely pro podlahy, zdi a lampy. Lampě lze také nastavit minimální a maximální měřítko a minimální a maximální počet výskytů v místnosti. Pro specifické místnosti a jejich parametry platí následující:

- *knihovna* – lze nastavit objekt knihovny, počet polí na ose X a Y, které tento objekt zabere, rozptyl v měřítku, které se náhodně na knihovny aplikuje (pro stejnou velikost objektů lze nastavit minimální a maximální hranici stejnou), obsahuje také parametry na šanci vytvoření objektu knihovny
- *sál* – výběr objektu trůnu a počet zabraných polí na ose X a Y, rozptyl v měřítku, stejné parametry pro objekt sloupu, navíc s možnou odchylkou – sloupy se tak mohou vytvořit v jiné buňce a zmírní se pravidelnost. Stejně tak se dá nastavit odchylka pro objekt lampy
- *skladiště* – tento typ místnosti nebyl implementován



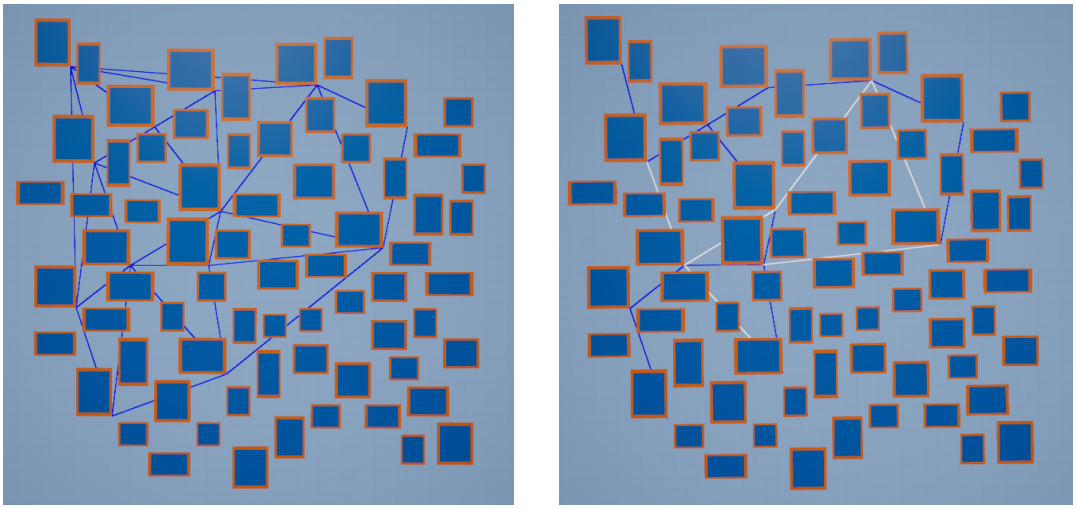
Obrázek 4.1: Ukázka uživatelského rozhraní pro úpravu parametrů v editoru.

4.3 Generování struktury dungeonu

Na obrázku 4.1 se ve spodní části nachází tlačítko s nápisem *Generate*. Tímto tlačítkem se proces generování spouští a zavolá se funkce `Generate()` třídy `ADGGenerator`.Funkce nejprve zkontroluje zadané parametry (nesmí například chybět model podlahy pro příkladové místnosti, pokud jsou generovány), které pak postupně používá za běhu. Generátor

si dále vytvoří strukturu typu `FRandomStream`, která umí využívat seed pro generování pseudonáhodných čísel, což umožňuje využít uživatelské parametry `CustomSeed` a `Seed` a v jiném běhu programu zopakovat výsledek generování. Dalším krokem je výběr pozic pro vložení místností, které probíhá, jak bylo navrhnuto, v mřížce. Tyto pozice jsou následně použity pro vložení místností reprezentovaných třídou `ADGRoom`, přičemž každá místnost je inicializována s náhodnou výškou, šířkou a je jim přidělen vybraný model podlahy v závislosti na vybraném typu místnosti. Při vkládání místností se postupně ukládají ukazatele na tyto objekty do pole ve třídě `ADGGenerator`. Následně se přes zmíněné pole iteruje a podle uložené výšky a šířky místnosti se počítá průměrný obsah a vynásobí se parametrem `MainRoomSelectionRatio`. Místnostem s vyšším obsahem než právě získaným je nastaven typ *hlavní*.

V tomto stavu jsou všechny místnosti stále na původním místě, pravděpodobně vzájemně se překrývající (nemusí se stát, například když uživatel zadá příliš velký rozsah počáteční mřížky pro vybírání náhodných bodů – parametr `GridSizeForInit`) a je třeba je rozdělit. Tento proces obstarává funkce `SeparateRooms()`, která prochází místnosti a u každé se pomocí komponenty `UStaticMeshComponent` dotáže na kolidující místnosti. Vrácený seznam je procházen a je vypočítán vektor, následně zaokrouhlen na násobky mřížky a místnost je o daný vektor posunuta. Takto se místnosti posunují dokud některá hlásí alespoň 1 kolizi s jinými místnostmi. Může nastat situace, že se 2 místnosti posouvají navzájem tam a zpět. Pokud se takto místnost vrátí několikrát na stejné místo (počet vrácení se porovnává s konstantou v třídě místnosti, `ADGRoom`), je jí vypnuta možnost detekovat či produkovat kolize a označí se jako *nepoužitá*.



(a) Delaunayova triangulace (b) Výsledné propojení

Obrázek 4.2: Na obrázku vlevo je příklad výsledku Delaunayovy triangulace (modré linky) mezi největšími místnostmi. Obrázek vpravo pak zobrazuje výsledek po nalezení minimální kostry (modré linky) a přidání několika hran z původního grafu zpět (bílé linky).

S rozprostřenými místnostmi je třeba vytvořit graf propojení. To začíná u Delaunayovy triangulace[1], která se spustí funkcí `CreateDelaunayGraph()`. Tato funkce vloží všechny

[1]Pro Delaunayovu triangulaci byla využita knihovna ze zdroje https://github.com/delfrrr/delaunator-cpp

počáteční souřadnice *hlavních* místností do knihovny `delaunator.hpp`[2] a její výsledek se převede na interní reprezentaci grafu pomocí struktur `FDGEdge`, `FDGVertex` a `FDGGraph`. Z tohoto grafu je třeba získat minimální kostru grafu, pro což byl zvolen a implementován Jarníkův algoritmus[3] ve třídě `MinimumSpanningTree`. Po dokončení běhu Jarníkova algoritmu je vrácen nový graf s minimální kostrou bez žádných jiných hran, což neodpovídá návrhu. Proto bude určitý počet propojení do grafu vrácen tak, že se náhodně seřadí hrany v původním grafu a z nich se vybere určitý poměr (zadán parametrem uživatele `DelaunayLinkageRatio`) a vloží se do grafu minimální kostry. Nyní tento graf obsahuje jak zaručený průchod všemi body (místnostmi), tak smyčky a možnost vstupu do místnosti z více chodeb.

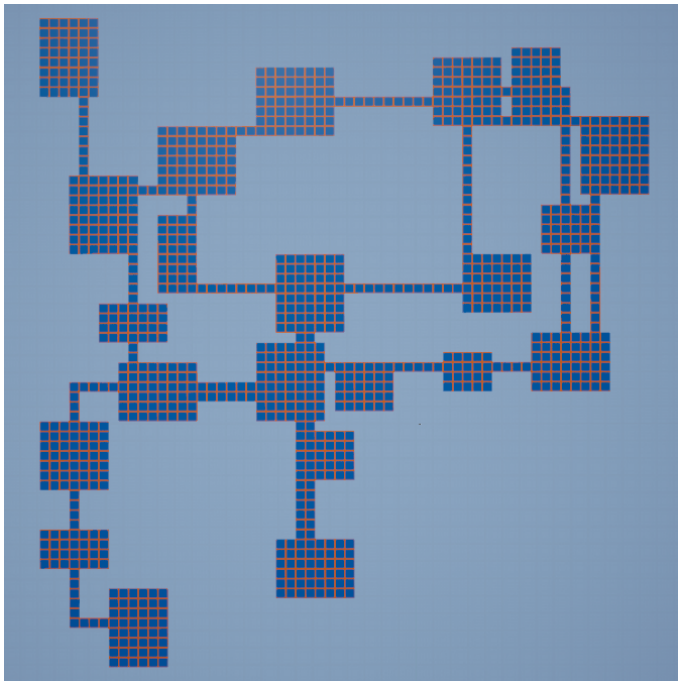
Nyní je na řadě vytvoření interní reprezentace mřížky. Nejprve si každý objekt místnosti zavolá funkci `BecomeTiles()`, která vymění původní `UStaticMeshComponent` za `UInstancedStaticMeshComponent`, která má tu výhodu, že umí efektivně renderovat mnoho instancí stejného modelu. Této komponentě se předá model podlahy a následně se celková plocha místnosti pokryje jednotlivými instancemi předaného modelu. V dalším kroku se funkcí `FillTileGrid()` vytvoří jednorozměrné pole objektů `DGCell`, které bude reprezentovat mřížku a buňky použité pro další generování. První krok ke spočítání velikosti této mřížky je zjištění rozdílů minimální a maximální složky X, respektive Y, kterých rozprostřené místnosti nabyly. Tyto 2 rozdíly se poté podělí velikostí podlahy místností a vyjde počet řádků r pro osu X a sloupců c pro osu Y. Pole buněk pak bude mít velikost $r \times c$. Buňky jsou na počátku inicializovány pouze s jejich pozicí, kterou v herním prostoru reprezentují a s jejich indexem v poli. Dále se pro každou místnost vypočítá, které buňky jsou ní v mřížce obsazeny. Těmto buňkám je přiřazen typ *CRoom*, index buňky ve vztahu k místnosti a ukazatel na tuto místnost. Tyto kroky jsou nezbytné pro správné stavění chodeb a generování příkladových místností.

S možností zjistit, co se v buňce nachází, lze začít stavět chodby. Vstupem pro jejich tvorbu je upravený graf minimální kostry, který vznikl před tvorbou mřížky. Pro každý uzel v grafu se nalezne místnost A se stejnou pozicí a všechny hrany, které tento uzel spojují. Pro každou hranu se vezme druhý uzel a nalezne se místnost B, která má stejnou pozici jako tento uzel. Nyní je vytvořen objekt chodby třídy `ADGCorridor`. Pro místnosti A a B se zjistí, zda se na jedné z os X a Y alespoň částečně překrývají. Pokud ano, objekt chodby zavolá funkci `Build()` s parametry počátečního a koncového indexu. Tato funkce prochází buňky mezi zmíněnými indexy a vkládá modely podlahy na jejich pozici. Pokud se vyskytne kolize, tedy buňka je již obsazená, chová se algoritmus dle návrhu této situace 3.2. Pokud se místnosti A a B nepřekrývají, volá se funkce `BuildLShape()` s parametry ukazatelů na obě místnosti. V této funkci se dle vzájemných pozic místností vypočítá počáteční, prostřední (*v případě rohové místnosti*), a koncový index, s kterými je potom volána 2× funkce `Build()`.

Po vložení chodeb je již výsledný dungeon průchozí a základní proces generování je dokončen (obrázek 4.3. Zbývá pouze odstranit místnosti označené jako *nepoužité*. Kromě odstranění samotného objektu místnosti je třeba odstranit ukazatel ze seznamu místností a inicializovat buňky, které původně místnost obsazovala, do počátečního stavu.

Před začátkem a těsným koncem generování jsou odstartovány události `OnBeforeGenerationStarts`, respektive `OnAfterGenerationEnds`, na které si uživatel může v rámci Blueprints nebo C++ navázat další funkcionalitu (například umístit hráče hned po dokončení generování). Z obou těchto událostí lze získat použitý seed.

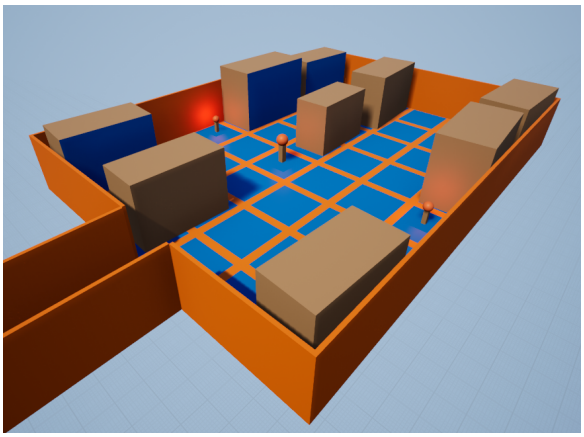
[2] zdroj: https://github.com/delfrrr/delaunator-cpp
[3] https://www2.karlin.mff.cuni.cz/



Obrázek 4.3: Výsledné rozložení herní úrovně po aplikaci mřížky a vložení chodeb.

4.4 Generování příkladových místností

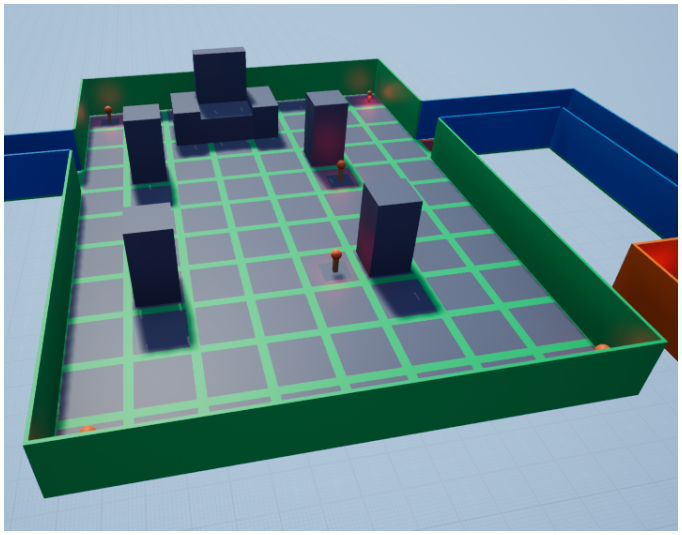
Z navržených místností byly implementovány typy knihovna a sál, skladiště z důvodu časové tísně není součástí pluginu. Oba objekty místností dědí z rodiče `DGRoomFiller`, která obsahuje funkci `Run()` a jiné, pomocné funkce. Funkce `Run()` přijímá jako parametr ukazatel na instanci třídy `DGGenerator`, ze které si načte potřebné parametry pro inicializaci. Je náhodně vybrána orientace místnosti mezi osami X nebo Y. Pokud se mají postavit zdi (lze ovlivnit parametry), je zavolána funkce `BuildWalls()`, která objektu místnosti vymění komponentu `UStaticMeshComponent` za `UInstancedStaticMeshComponent`, inicializuje ji a začne stavět zdi na hraničních buňkách místnosti. Po dostavění zdí je spuštěna metoda `Run()` i pro potomky – tedy třídy `DGLibraryFiller` a `DGHallFiller`:



Obrázek 4.4: Příklad vybavené místnosti typu knihovna.

V **knihovně** jsou do buněk místnosti vyznačeny průchody, které navazují na vstupy. Průchody jsou vždy vyznačeny (buňkám změněn stav na *průchod*) přes celou délku, případně šířku (záleží na orientaci a na pozici vchodu). Dále jsou vloženy další průchody (vždy je alespoň 1 průchod po směru osy X a Y, aby bylo možné místností volně projít) po obou směrech, přičemž jejich počet je ovlivněn parametrem a velikostí místnosti. Následně se do buněk, které zůstaly ve stavu *volný*, vkládají objekty knihovny. Tyto objekty je třeba správně otočit (podle orientace a sousedních buněk) s možnou náhodnou odchylkou a umístit na pozice buněk. Na zbylá volná místa jsou vloženy lampy. Ty budou umístěny částečně i do průchodů, pokud tím úplně nezablokují cestu.

Sál nejprve vkládá lampy do rohů místnosti. Pozice lampy může být náhodně vychýlena, pokud je nastaven příslušný parametr. Následně jsou vloženy sloupy. Ty se zarovnávají do místnosti podle nastavené mezery mezi sloupy tak, aby byl mezi zdí a prvním, respektive posledním, sloupem vždy co nejpodobnější odstup. Při vkládání sloupů se může vložit těsně kolem sloupu i lampa. To závisí na nastaveném počtu lamp na místnost. Nakonec se může (pokud je dostatek místa) vložit do místnosti objekt trůnu, který je vždy otočen směrem ke středu místnosti.



Obrázek 4.5: Příklad vybavené místnosti typu sál

Pro oba typy místností platí, že před vložením objektu je kontrolováno, zda je pro něj dostatek volných buněk a nezakrývá vchod či nekoliduje s jinými objekty. Buňky obsazené nějakým objektem jsou přepnuty do stavu *nábytek*.

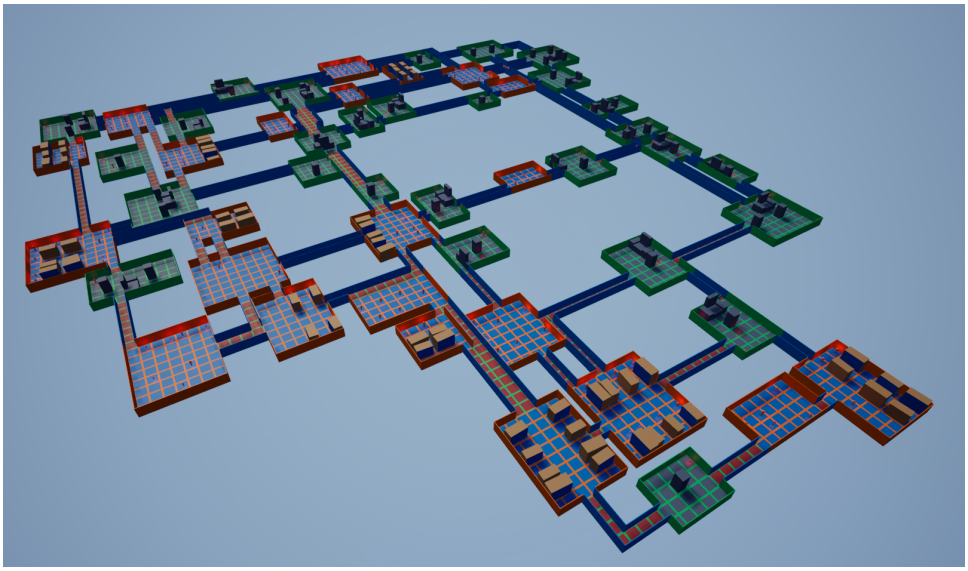
Kapitola 5

Testování

V této kapitole bude popsán způsob testování pluginu. To probíhalo se zaměřením na vlastnosti požadované v kapitole 3. K zajímavějším případům při testování bude připojen obrázek.

5.1 Rychlost generování

V rámci testování bylo zjištěno, že nejpomalejší část (pokud nejsou místnosti vybavovány) pluginu je proces rozmisťování místností. Test byl proveden vygenerováním 100 dungeonů (bez zdí a dalšího vybavení) o počtu místností 100 a velikostí místností 4 – 8 (šířka i výška). Bylo naměřeno, že z 12,2 sekund zabralo rozmisťování 3,82 sekund, což je přibližně 31,31 %. Snížení režie by se dalo dosáhnout například zvolením efektivnějšího algoritmu pro detekci kolizí místností a jejich posouvání (například aplikováním simulovaného žíhání[1]). Při zapnutém vybavování místností zabralo vybavování samotné 22,55 sekund (58,67 %) z celkových 38,43 sekund.



Obrázek 5.1: Příklad vygenerovaného dungeonu průměrné velikosti

[1] Z anglického Simulated Annealing, jeho stručný popis aplikace na oddělování obdélníků a srovnání s jinými metodami lze najít na https://mikekling.com/...

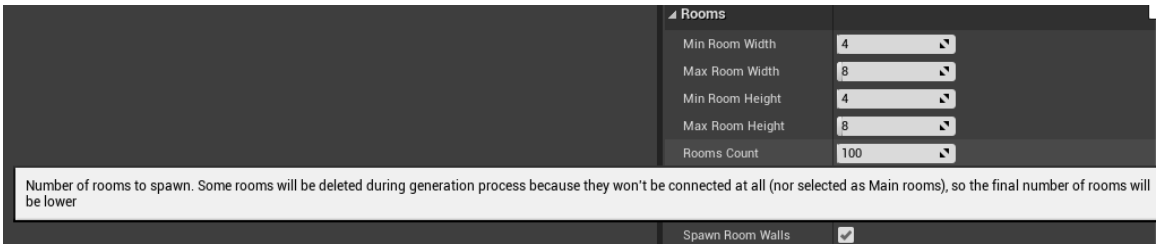
Pro zmíněnou konfiguraci však zabere vygenerování 1 dungeonu průměrně 0,13 sekundy (se stavbou zdí 0,14, s kompletním vybavením 0,39), což dostatečně splňuje nárok z kapitoly 4 na vytvoření dungeonů do nižších jednotek sekund. Možný výsledek pro takový dungeon lze vidět na obrázku 5.1.

Největší dopad na rychlost má samozřejmě nastavení parametrů uživatelem. Kromě velikosti a počtu místností má značný vliv parametr `Main Room Selection Ratio`. Ten udává, jaký musí mít místnost poměr obsahu vůči vypočítanému průměrnému obsahu všech místností, aby byla označena jako *hlavní*. Místnosti označené jako *hlavní* jsou totiž zahrnuty do minimální kostry propojených místností a čím více těchto místností, tím více chodeb je třeba postavit. A čím více chodeb, tím více *vedlejších* místností je v úrovni ponecháno (protože jimi prochází alespoň 1 chodba).

5.2 Vsupní parametry

Požadavek na **nastavitelnost** byl splněn zpřístupněním množství parametrů ovlivňující chování pluginu. Vstupním parametrům pro základní proces generování místností a chodeb byly v rámci uživatelského prostředí stanoveny doporučené hranice pro minimální a maximální hodnotu. Přestože by tyto hranice neměly být omezující pro zamýšlené a běžné použití pluginu, může si uživatel chtít tyto hranice upravit. K tomu lze využít systém Blueprints, případně jazyk C++. K parametrům patří i `Seed`, jehož používáním v procesu generování dungeonu se splňuje i požadavek na **opakovatelnost**. Je však zároveň se seedem nutné použít i stejnou konfiguraci.

Pro podrobnější vysvětlení každého parametru a jeho vlivu na fungování pluginu byl přidán popisek po najetí myši (obr. 5.2).



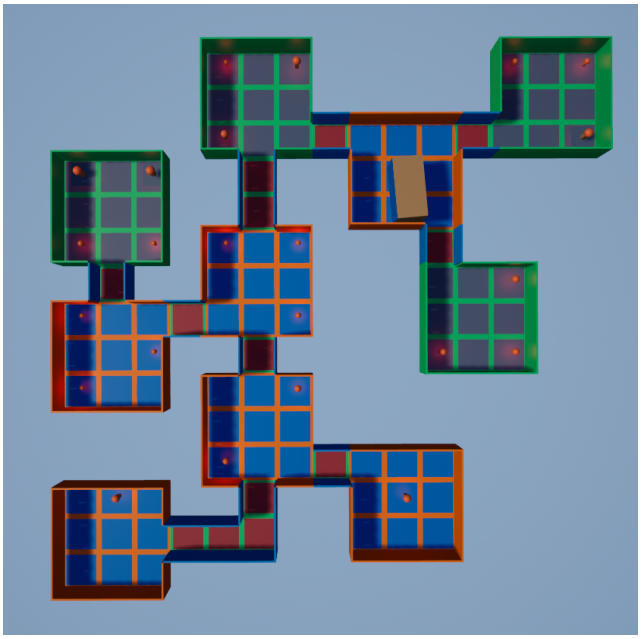
Obrázek 5.2: Příklad podrobného popisku po najetí myši na parametr `Rooms Count`.

Meze uspokojivého výsledku

Hodnoty parametrů jsou klíčové pro vygenerování takového dungeonu, který uživatel může použít do svého projektu. Algoritmus pluginu je navržen tak, aby podával co nejuspokojivější výsledky pro co největší rozsah hodnot parametrů. I přes to se však při určitých hodnotách parametrů začnou stávat výsledky méně zajímavé, obskurní, téměř nepoužitelné.

Při testování rozsahů parametrů v tomto projektu bylo zjištěno, že při nízkých hodnotách jsou podávány výsledky bez větších problémů, naopak by se dalo říct, že jsou poměrně zajímavé pro hry s potřebou menších map a bludišť (obrázek 5.3).

S navýšenými hodnotami parametrů je dungeon komplexnější, místnosti větší (maximální šířka a výška v řádu vyšších jednotek) a vzdálenější (záleží na parametru `Separation Speed Multiplier`). Dá se předpokládat, že při užití pluginu uživateli se v okolí těchto hodnot budou parametry generátoru pohybovat nejčastěji. Výsledek je totiž

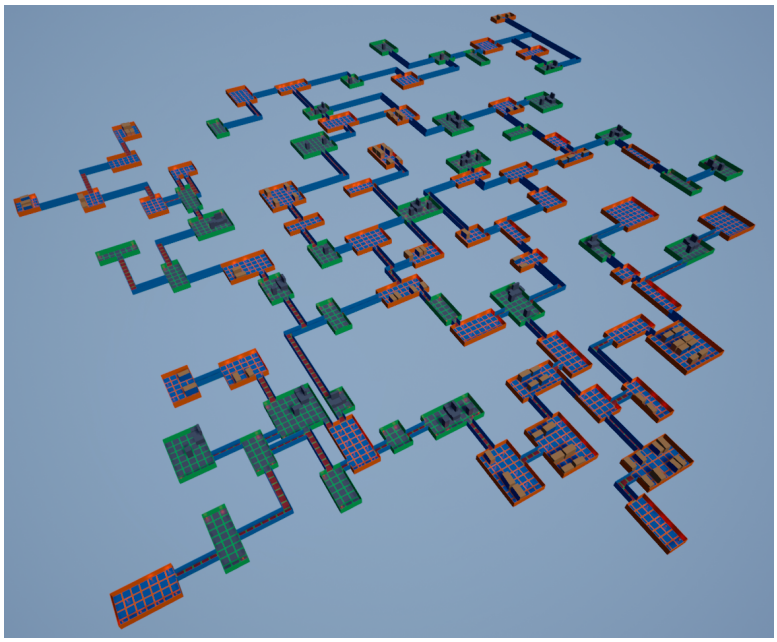


Obrázek 5.3: Výstup pro nejnižší nastavenou doporučenou hranici parametrů. Místnosti mají meze výšky i šířky na hodnotě 3, počet místností byl nastaven na 10 a jako propojení byla použita pouze minimální kostra (bez dalších vrácených propojení z Delaunayovy triangulace).

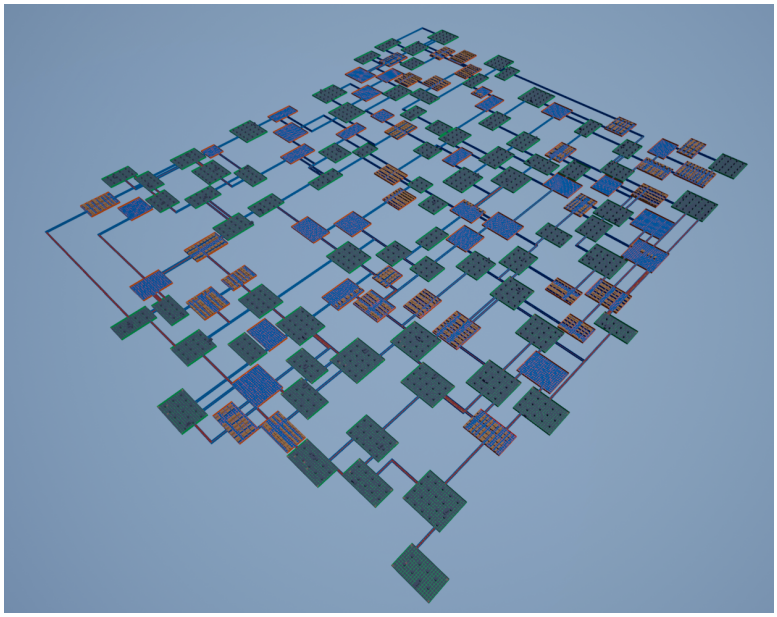
vhodný pro menší až střední dungeony (obrázek 5.4), je kompozičně zajímavý a chodby nejsou příliš dlouhé.

Při dalším navyšování parametrů se dojem z výsledných dungeonů začne mírně zhoršovat. Chodby se prodlužují, více vrácených propojení znamená více chodeb a při větším počtu místností může tato skutečnost znamenat příliš mnoho vedlejších místností. Rozlehlé místnosti dále nabuzují nudný dojem. Tyto dungeony se dají použít například pro rozsáhlá bludiště 5.5.

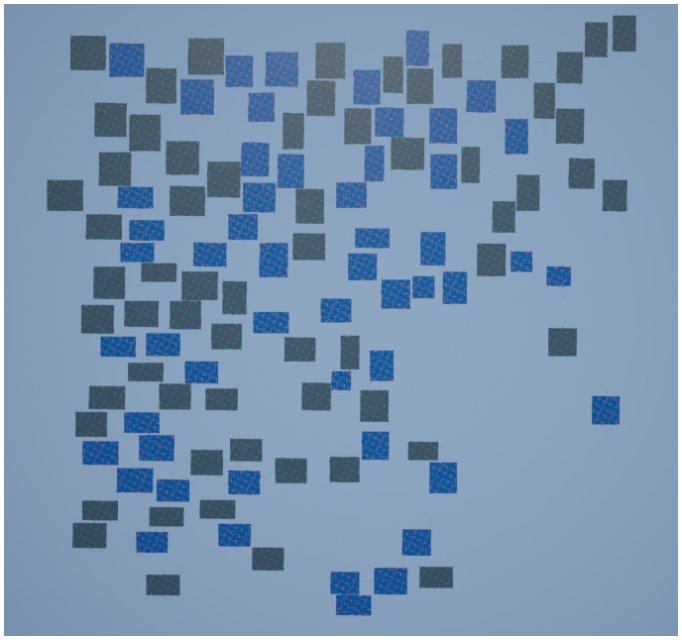
V případě dalšího navyšování se místnosti stávají příliš velkými, obecně se dá mluvit o dungeonu jen velmi abstraktně. Místnosti s těmito parametry začínají být dost velké na to, aby v mnoha hrách svým prostorem postačily na samostatnou herní úroveň (5.6).



Obrázek 5.4: Výstup pro navýšené hodnoty parametrů. Minimální výška a šířka je na hodnotě 3, maximální na 8. Z Delaunayovy triangulace je vráceno 3% propojení. Velikost kroku při rozdělování je 9, proto jsou místnosti mírně separovány.



Obrázek 5.5: Výstup s ještě vyššími hodnoty parametrů. Velikost místností začíná pro výšku i šířku na hodnotě 10 a 20. Počátečních místností je 200 a jako propojení byla použita pouze minimální kostra (bez dalších vrácených propojení z Delaunayovy triangulace).



Obrázek 5.6: Ukázka výsledku pro příliš vysoké hodnoty parametrů. Minimální výška a šířka je na hodnotě 50, maximální na 100, a počátečních místností bylo nastaveno 200. Chodby jsou v poměru k místnosti tak malé, že je při pohledu na celou úroveň téměř nelze vidět.

Kapitola 6

Závěr

Cílem této práce bylo nastudovat techniky procedurálního generování herních dungeonů, herní engine Unreal Engine, možnosti integrace pluginu a navržení a implementace pluginu. Tento plugin má za použití vstupních parametrů generovat herní struktury typu dungeon a výsledek vizualizovat v Unreal Enginu. Dále mělo být vytvořeno krátké video zobrazující výsledek.

Zmíněné cíle byly splněny. Z nastudovaných technik procedurálního generování dungeonů byl vybrán vhodný algoritmus a použit jako základ pro generování struktury místností a chodeb. Dále byl navrhnut a implementován plugin, který zmíněný algoritmus využívá pro generování struktury herní úrovně obsahující místnosti propojených chodbami, dungeonu. Plugin nabízí i možnost místnosti rychle vybavit objekty, což může uživatel využít například pro rychlé vytvoření testovacího prostředí pro svou hru. Výsledek generování je vizualizován v herním prostoru v rámci Unreal Enginu pomocí herních objektů a tento stav lze uložit a dále využít pro potřeby uživatele. Plugin také disponuje uživatelským prostředím a mnoha parametry, kterými lze ovlivnit nejen proces generování dugenonu samotného, ale i následné vybavování místností. Těmito parametry lze například efektivně měnit proces generování a vytvářet různě rozsáhlé a propojené dungeony. V případě potřeby si může uživatel uložit použitý seed a konfiguraci pro budoucí zopakování výsledku. Dále si může změnit i kterýkoli model, který se má při procesu použít.

Vytvořený plugin lze jednoduše nainstalovat bez nutnosti zásahu do projektů. Zde je třeba jen dbát na verzi enginu kvůli zpětné kompatibilitě – plugin byl tvořen ve verzi 4.26.

Práce mi přinesla značné rozšíření obzorů v oblasti procedurálního generování a vyvolala ve mě zvědavost a zájem o další studování a praktikování této metody. V průběhu práce jsem si také osvojil práci s Unreal Enginem a jeho fungováním s kódem psaným v C++. Výzvou byla například vizualizace výsledku za pomocí 3D objektů a navrhnutí systému pro tvorbu příkladových místností.

Protože je plugin tvořený pro obecné použití a nezaměřuje se velmi konkrétní typ dungeonu, je celá řada možných funkcí, kterými jej lze zdokonalit a které mohou být uživateli žádané. Jednou z nich může být možnost upravovat nejen velikost místností, ale i jejich tvar, případně možnost vytvoření více propojených podlaží. Dále by mohly být místnostem přidělovány role či význam, například za použití návrhového vzoru Lock-and-Key[1] a tím by uživatel dostal jasnější představu, jak s dungeonem dále naložit. Dalším příkladem pro užitečnou funkci je komplexnější vkládání vybavení místností, aby mohl uživatel definovat pravidla pro jejich umisťování a jejich význam v rámci herní úrovně.

[1] http://howtomakeanrpg.com/...

Literatura

[1] DAUM, B. 1 - Foundations. In: DAUM, B., ed. *Modeling Business Objects with XML Schema*. San Francisco: Morgan Kaufmann, 2003, s. 3–40. The Morgan Kaufmann Series in Software Engineering and Programming. DOI: https://doi.org/10.1016/B978-155860816-0/50003-3. ISBN 978-1-55860-816-0. Dostupné z: https://www.sciencedirect.com/science/article/pii/B9781558608160500033.

[2] HARTSOOK, K., ZOOK, A., DAS, S. a RIEDL, M. O. Toward supporting stories with procedurally generated game worlds. In: *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*. 2011, s. 297–304. DOI: 10.1109/CIG.2011.6032020.

[3] HUSÁKOVÁ, M. *Celulární automaty - Znalostní technologie III, materiál pro podporu studia* [https://lide.uhk.cz/fim/ucitel/fshusam2/lekarnicky/zt3/zt3_dokumenty/CelularniAutomaty.pdf]. Přistoupeno: 2022-5-5.

[4] JOHNSON, L., YANNAKAKIS, G. a TOGELIUS, J. Cellular automata for real-time generation of. Září 2010. DOI: 10.1145/1814256.1814266.

[5] KNUTZEN, J. *Generating Climbing Plants Using L-Systems*. Göteborg, Sweden, 2009. Diplomová práce. Chalmers University of Technology, University of Gothenburg. Dostupné z: https://www.cse.chalmers.se/~uffe/xjobb/climbingplants.pdf.

[6] LINDEN, R., LOPES, R. a BIDARRA, R. Procedural Generation of Dungeons. *Computational Intelligence and AI in Games, IEEE Transactions on*. Březen 2014, sv. 6, s. 78–89. DOI: 10.1109/TCIAIG.2013.2290371.

[7] PRUSINKIEWICZ, P. Graphical applications of L-systems. In: *Proceedings of graphics interface*. 1986, sv. 86, č. 86, s. 247–253.

[8] SARKAR, P. A Brief History of Cellular Automata. *ACM Comput. Surv.* New York, NY, USA: Association for Computing Machinery. mar 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.

[9] SHAKER, N., TOGELIUS, J. a NELSON, M. J. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016.

[10] TOGELIUS, J., KASTBJERG, E., SCHEDL, D. a YANNAKAKIS, G. N. What is Procedural Content Generation? Mario on the Borderline. In: *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. New York, NY, USA: Association for Computing Machinery, 2011. PCGames '11. DOI:

10.1145/2000919.2000922. ISBN 9781450308724. Dostupné z: https://doi.org/10.1145/2000919.2000922.

Příloha A

Obsah přiloženého média

- `plugin` – složka obsahující adresář DungeonGenerator, který lze vložit do projektu Unreal Enginu
- `text` – složka obshající text práce ve formátu PDF a adresář se zdrojovými soubory pro LaTeX
- `README.txt` – návod na instalaci a ovládání pluginu
- `demovideo.mp4` – krátké video demonstrující funkce pluginu