FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Generátor dungeonů
Student:	Illia Brylov
Vedoucí:	Ing. David Bernhauer
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Počítačová grafika
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

V případě herního průmyslu jdou proti sobě dva koncepty. Ručně vytvořené mapy jsou nádherné, ale jejich tvorba je časově náročná. Automaticky generované mapy tento problém nemají a mohou být případně upraveny ručně.

Proveďte rešerši alespoň 3 různých her implementujících generování map.
Popište fungování Wave Function Collapse (WFC) algoritmu pro generování map.
Diskutujte omezení WFC a popište případné techniky pro řešení těchto problémů.
Navrhněte, implementujte a otestujte prototyp založený na WFC pro generování map dungeonů, buď jako plugin nebo jako samostatnou aplikaci.

Elektronicky schválil/a Ing. Radek Richtr, Ph.D. dne 16. prosince 2021 v Praze.

Bakalářská práce

GENERÁTOR DUNGEONŮ

Illia Brylov

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. David Bernhauer
11. května 2022

Obsah

Seznam obrázků

Seznam výpisů kódu

Chtěl bych poděkovat především svému vedoucímu práce Ing. Davidu Bernhauerovi za pomoc během psaní, svému příteli Tomáši Hegerovi za psychologickou podporu a kontrolu českého pravopisu a nakonec své rodině za podporování během celého studia.

Prohlášení

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

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

V Praze dne 11. května 2022 .....

Abstrakt

Techniky procedurálního generování jsou hojně používané v herním průmyslu. Nejvíc se tyto techniky zužitkují při vytváření map. Tato práce uvádí čtenáře do problematiky procedurálního generování map ve videohrách a popisuje relativně nový algoritmus Wave Function Collapse, který takové generováni implementuje. Autorem je pak vytvořen plugin pro generování map dungeonů v 3D modelovacím softwaru Blender. Plugin je založen na výše zmíněném algoritmu WFC. Výstupy generování jsou následně otestované na podobnost očekávaným výsledkům a na soulad s principy fungování algoritmu WFC.

Klíčová slova procedurální generování map, Wave Function Collapse, generátor dungeonů, Blender

Abstract

Procedural generation techniques are widely used in the game industry. These techniques are utilized in the map creation. This thesis introduces the reader to procedural map generation in video games and describes a relatively new algorithm called Wave Function Collapse, which implements such generating. The author then creates a plugin for generating dungeon maps in the Blender 3D modeling software. The plugin is based on the above mentioned WFC algorithm. The generation outputs are then tested for similarity to the expected results and for compliance with the principles of the WFC algorithm.

Keywords procedural map generation, Wave Function Collapse, dungeon generator, Blender

Seznam zkratek

MD	Man Day
WFC	Wave Function Collapse
PCG	Procedural Content Generation
NPC	Non-Player Character
BFS	Breadth-First Search

Kapitola 1
Úvod

Herní průmysl se stal důležitou součástí života posledních generací. Hodin strávených u hraní nejen počítačových her je každým rokem víc a víc, počet hráčů roste a ceny e-sport turnajů překonávají dosud nečekané rekordy.[1][2]

Důsledkem tohoto jevu je zvýšení poptávky po nových hrách a zajímavém herním obsahu, u kterého se dá strávit v součtu desítky a někdy i stovky hodin. Vytváření těchto her mají na starosti *indie*, střední a *AAA studia*, která zaměstnávají tisíce zaměstnanců po celém světě. Vývoj jedné hry v závislosti na množství jejího obsahu a velikosti týmu může trvat od půl roku do několika let. Nejvíce času pro vývoj zabírají tzv. *open-world hry*. To jsou hry, ve kterých je hráč schopen se volně pohybovat po mapě bez větších omezení. Omezeními jsou často například princip, jakým je vytvořena herní lokace (má lineární průběh), nebo schopnosti herní postavy (její úroveň nebo potřebné dovednosti). Příkladem takových her jsou například hry sérií The Elders Scrolls nebo také The Assasins Creed a The Witcher.

Rozsah map a jednotlivých lokací v takových hrách často bývá velký. V případě map jde o řádově desítky až stovky kilometrů čtverečních. Některá města, jakožto herní lokace, mají často rozlohu několika km^2 a jsou postavena z obrovského množství budov a dalších struktur.[3]

Toto vyžaduje od herních vývojářů vytvořit enormní počet různého obsahu: příběh, příběhové úkoly, 3D modely, textury, implementace herních mechanik a logiky atd. Jinými slovy vytvořit něco, do čeho bude investováno tisíce *MD*[1] a obrovské množství peněz, aby se z nápadu stal reálný hratelný produkt. Proto se herní studia aktivně zabývají tím, jak zrychlit a zlevnit vývoj, jak ulehčit práci programátorům a umělcům, aby se místo řešení vedlejších problémů a úkolů mohli věnovat důležitějším věcem, například těm, které vyžadují opravdovou kreativitu a nejen repetitivní nebo manuální práci.

Jednou z metodik, která často přijde v úvahu a je hojně používaná, je *PCG — Procedural Content Generation*. Přesně touto metodikou se v této práci nejvíce zabývám.

1.1 Cíle práce

Má práce má několik cílů. Do cílů práce patří rešerše existujících způsobů využití metodiky *PCG* pro generování map na příkladě několika již vydaných her, detailnější průzkum algoritmu, který vznikl relativně nedávno, s názvem *WFC — Wave Function Collapse* a praktická ukázka jeho použití v generování map dungeonů.

Cílem rešerše, resp. teoretické části, je znázornit souvislosti ve volbě určitých typů algoritmů/způsobů *PCG* v kontextu zkoumaných her s výhodami těchto algoritmů a jejich přínosem

[1] MD — z angl. man day, je jednotka odhadování pracnosti pracovních úkolů. 1MD je roven 8 hodinám práce jedné osoby.

pro určité *use-casy* během vývoje, a také popis algoritmu *WFC*.

Cílem praktické části práce je vytvoření prototypu programu pro zadaný *use-case* — generování map dungeonů. Prototyp musí být založený na použití algoritmu *WFC*. Výsledná implementace bude otestována na podobnost očekávaným výstupům.

1.2 Struktura práce

Práce je rozdělena do pěti kapitol. Po úvodní části v kapitole 1 následuje kapitola 2 s rešerší různých her implementujících generování map. Následující kapitola 3 detailně popíše principy fungování *WFC* a jeho praktické ukázky, srovná výstupy s jemu podobnými technikami a prodiskutuje případná omezení v použití a způsoby jejich řešení. Po popisu fungování *WFC* bude následovat praktická část této práce v rámci kapitoly 4, ve které se implementuje prototyp programu pro generování map dungeonů. Prototyp ve svém jádru bude používat algoritmus WFC. Na konci čtvrté kapitoly bude implementace otestována empirickými testy na podobnosti očekávaným výstupům a také automatickými testy na absenci případů, které jsou v rozporu s principy generování pomocí algoritmu *WFC*. Práce a její výstupy budou shrnuty v poslední kapitole 5.

Kapitola 2

Generování map v existujících hrách

"The computer should be doing the hard work. That's what it's paid to do, after all." — Larry Wall

Tato kapitola si bere za cíl seznámit čtenáře s tím, co se rozumí pod pojmem *Procedural Content Generation* a proč je tato metodika aktivně využívána v herním průmyslu, jakým způsobem a jaké má výhody a nevýhody. K detailnějšímu pohledu slouží konkrétní příklady her, kde si tato metodika našla místo buď během vývoje, nebo v roli centrální myšlenky a mechaniky.

2.1 Procedural Content Generation

Videohry na začátku své existence byly vytvářeny hlavně ručně bez použití různých algoritmických technik. V rámci vývoje hry je zapotřebí vytvořit její grafický obsah — to, co na obrazovce bude hráč vidět a s čím bude po celou dobu hry interagovat. To jsou třeba zbraně, budovy, hratelné a nehratelné postavy, mapy, jiné struktury atd. Nejčastěji právě tvorba tohoto obsahu zabírala nejvíce času lidské práce ve srovnání s jinými aktivitami. V případě větších projektů mohl obsah práce v té době dosahovat neuvěřitelného měřítka, což vyžadovalo víc a víc lidí potřebných ve vývoji. Proto se začalo zamýšlet nad tím, jakým způsobem se dá využít výpočetní sílu počítačů pro zjednodušení a zmenšení obsahu potřebné práce. Z toho důvodu vznikla nová metodika s názvem — *Procedural Content Generation.*

„Procedural content generation (PCG) refers to the algorithmic generation of game content with limited or no human contribution. "Game content" is here understood widely asincluding e.g. levels, maps, quests, textures, characters, vegetation, rules, dynamics andstructures, but not the game engine itself nor the behaviour of NPCs.“ Z téhle definice ze článku [4] je vidět, že danou metodiku se dá použít pro vytvoření širokého spektra objektů, které jsou přítomné ve hrách.

„These techniques (PCG - pozn. autora) can be used to rapidly create lots of content that would be otherwise too expensive or nearly impossible to create (due to the cost or time constraints for example). They can be employed to provide replay value for games, add procedural details at run-time on in-game characters, create non-repeating realistic texture of a surface, or generate objects of nature such as vegetation or trees.“[5]

Použitím *PCG* se myslí hlavně změna způsobu vytvoření obsahu z manuálního (běžně 3D modelování objektů, focení nebo malování textur, modelování a zaplnění herní mapy a lokací) na plné nebo částečné procedurální s případnými manuálními úpravami.

Nejčastěji se k danému způsobu schyluje v případě, kdy je potřeba vytvořit velké množství podobného obsahu, který se dá popsat určitými pravidly. Vezmeme jako příklad nějakou budovu, kterou se dá poskládat z různých modulárních prvků. Takovými prvky můžou třeba být: zeď, zeď s oknem, zeď s dveřmi, rohová zeď, pilíře, střecha.

Po programovém určení, co je tedy budova a z čeho se skládá, se dá vytvořit spousta variací, které v důsledku sestaví knihovnu již připravených objektů. Poté už stačí takové objekty pouze vložit do herní scény, aby se splnil určený estetický cíl. Jak je vidět, tento způsob ušetřil obrovský kus práce v případě, kdy by byla potřeba objekty vytvářet ručně.

Dalším příkladem může být vytváření knihovny různých stromů. Na to se už úplně nedá aplikovat předchozí postup, vzhledem k tomu, že jde o vytvoření organické nepravidelné struktury. Pro takové případy existují jiné algoritmy založené na odlišných principech, pomocí kterých se dá takovou knihovnu stromů vygenerovat.

Není to ale všechno. Opačným případem je vytvoření časově velmi náročného unikátního objektu. Jde například o generování map.

Všechna použití *PCG* se dají rozdělit na dva případy:

- použití ve fázi vývoje hry,
- použití v samotné hře, jakožto herní mechanika.

Tyhle případy budou detailněji rozebrány v dalších částech kapitoly jak z obecného hlediska, tak i na příkladech existujících her. Příklady z her jsou zaměřeny hlavně na použití *PCG* v kontextu generování 2D a 3D map.

2.1.1 Fáze vývoje hry

Daný příklad použití *PCG* je zvláštní tím, že všechno, co je ve hře vygenerované, se generuje pouze ve fázi vývoje. To znamená, že se potřebné objekty vytvářejí jen během produkce hry. Následně se objekty z vygenerovaných knihoven umisťují do scén na místa určená vývojáři, kde už zůstávají navždy. Dalo by se říci, že v takovém případě jsou tvořené *statické scény*, které se už nemění v průběhu hry.

Další důležitou odlišností je, že takový obsah je generovaný mimo počítač hráče. To znamená, že hráč vůbec není omezen výkonem svého počítače díky tomu, že se na něm neprovádí často náročné a dlouhé výpočty.

2.1.2 Použití v samotné hře

Část herního obsahu se v takových hrách generuje v průběhu hraní a tím pádem, když zkusíme srovnat instance hry u několika různých hráčů, uvidíme, že nejsou stejné, často ani podobné. S tímhle se můžeme nejčastěji setkat v *rogue-like*[1] hrách, kde je herní lokace po každém spuštění vygenerována úplně jinak. Místnosti jsou jinak rozmístěné, vybavení (dekorace) místností se může lišit polohou nebo i složením atd.

Některé hry jsou založené na tom, že se obsah generuje dynamicky v závislosti na určitých podmínkách a kritériích. Často to je generace nehratelných postav (NPC), jako jsou třeba zvířata a to včetně jejich tvarů a charakteristik, nebo také generace celých planet a planetárních systémů. Můžou totiž být úplně unikátní.

Naráží se tady ale na to, že ne každý počítač zvládne takový obsah v reálném čase generovat. Proto herní studia provádějí své výzkumy v tomhle oboru a vytvářejí nové metodiky a algoritmy, které nejsou relativně náročné a produkují zajímavý obsah.

[1] Rogue-like je označení her podobných hře Rogue (1980) klíčovými aspekty, kterými jsou permanentní smrt (po smrti hráč začíná hru úplně od začátku) a procedurálně generované úrovně.

2.1.3 Výhody a nevýhody

U jakéhokoliv přístupu jsou různé výhody a nevýhody, některé jsou lepší pro jedny *use-casy*, druhé pro jiné. Následující sekce popisuje výhody a nevýhody použití PCG.

Výhody Jistou výhodou *PCG* je zjednodušení práce vývojářů her, vzhledem k tomu, že není potřeba všechno vytvářet samostatně. Je často jednodušší napsat skript, který daný obsah vygeneruje za vývojáře, než vytvářet danou samotnou věc ručně.

Další důležitou výhodou je, že se u her, které mají *PCG* zakomponované do sebe jako herní mechaniku, zvyšuje znovuhratelnost. Děje se to z toho důvodu, že se hráč po začátku nové hry ocitne v odlišném prostoru, než byl předtím, což u něj může vyvolat zájem a chuť daný prostor znovu zkoumat. Některé hry jsou schválně obtížné tím, že se v nich nedá naučit procházet nějakou určitou lokaci, protože pokaždé vypadá jinak.

Mapy vymyšlených světů jsou často generované, protože umělec nemá možnost zaručit, že mapa, kterou zrovna vytváří, je ta nejlepší možná. Taky nemá kapacitu na vymýšlení stovky variací. Ano, často jsou výstupy generátoru podivné, ale právě proto existuje další krok, ve kterém umělec zvolí nejlepší variantu a bude pokračovat v jejích úpravách a zlepšování.

Nevýhody Nevýhod je také několik. Někteří hráči si můžou stěžovat na používaní velkého množství „repetitivního umění“. To znamená, že hráči často potkají objekty, které si jsou velmi podobné, i když se něčím liší. Tohle může vyvolat pocity, že se scéna skládá ze stejných objektů a že do jejího vytvoření nebyl investován čas ani kreativita, aby byla zajímavější.

Nevýhodou je také často dlouhá etapa příprav, která v sobě obsahuje důkladné plánování, napsání nebo nalezení vhodného algoritmu pro generování, zvážení výkonnostních omezení na straně hráčů atd.

Je jednoduché říct, že věci tvořené ručně jsou většinou lepší než ty generované a to alespoň z důvodu, že algoritmus nemá umělecké cítění, které může mít živá osoba věnující se tvorbě dané věci.

2.2 Generování map

Mapou nebo herní mapou jsou nadále v textu chápány jakékoliv lokace nebo úrovně (levely), které jsou součástí hry. Lokace/level představuje otevřené nebo zavřené prostranství. Otevřené prostranství může být třeba město nebo terén vymyšlených světů ve 2D nebo ve 3D. Příkladem zavřených prostranství jsou jeskyně, bludiště, interiéry budov atd.

Generování map můžeme rozdělit na dva větší případy:

- generování samotného půdorysu,
- generování půdorysu a rozmístění dalších objektů na něm.

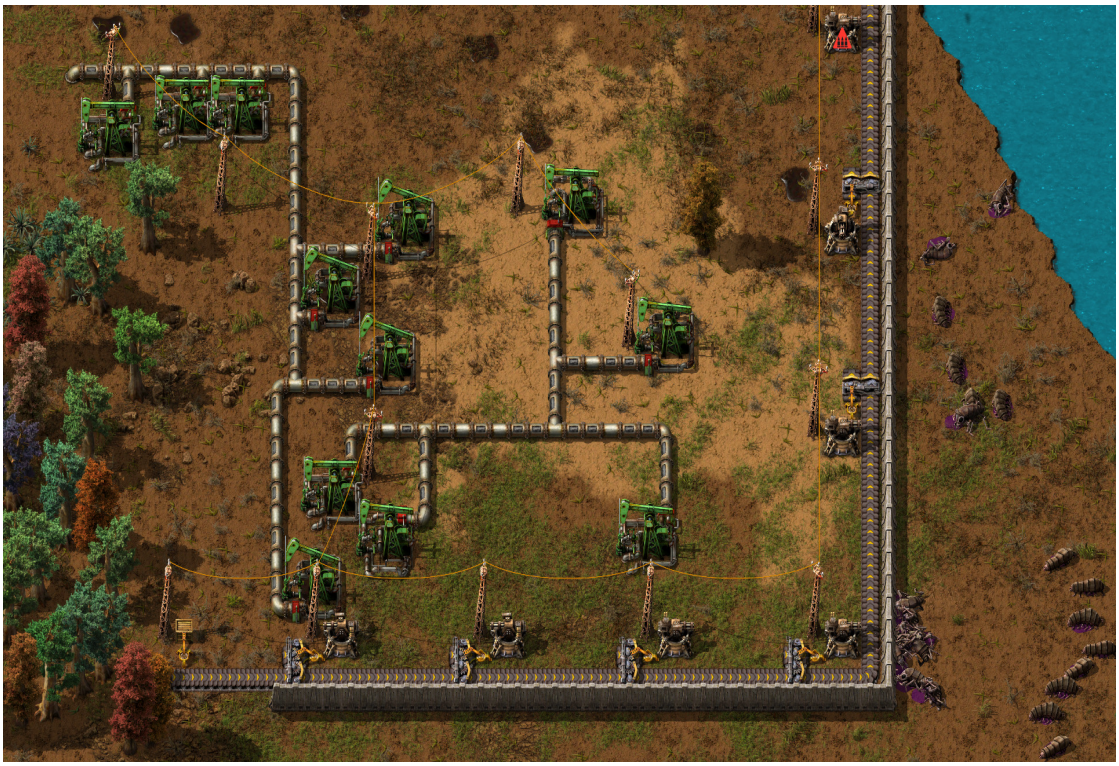
Pod generováním půdorysu se dá představit vytvoření mapy nějaké vymyšlené země. Taková mapa bude sestavena z různých regionů a struktur. Struktury jsou třeba moře, řeky, pobřeží, roviny, hory atd. Regiony přidávají specifické vlastnosti daným strukturám v závislosti na vygenerovaných parametrech. Hory můžou být zasněžené nebo s loukami, rovina může být pouští nebo džunglí.

Také můžeme pod generováním půdorysu rozumět vytvoření scén, které nejsou otevřeného typu. Bludiště, patro nějaké budovy, jeskyně nebo dungeon jsou dobré příklady takových půdorysů.

Druhý případ se liší tím, že kromě vytvoření samotného půdorysu je také chtěné vygenerované struktury něčím zaplnit. Na mapě vymyšleného světa se dají přidávat města, silnice atd. Patro budovy může vypadat nezajímavě bez nábytku a dekorativních prvků.

Tohle všechno můžeme ponechat počítači na vymyšlení, aby se na konci z výstupu generování dala vybrat ta varianta, které nejlépe odpovídá zadáním a cílům.

2.3 Generování 2D map



Obrázek 2.1 Hra Factorio využívající pohled ze shora, převzato z: `https://webcdn.factorio.com`

2D mapy se nejvíce vyskytují ve hrách s pohledem *top-down view* jak v ukázce ze hry Faktorio na obrázku 2.1. To znamená, že hráč vidí hrací plochu jako kdyby se nacházel hned nad ní a koukal ze shora.

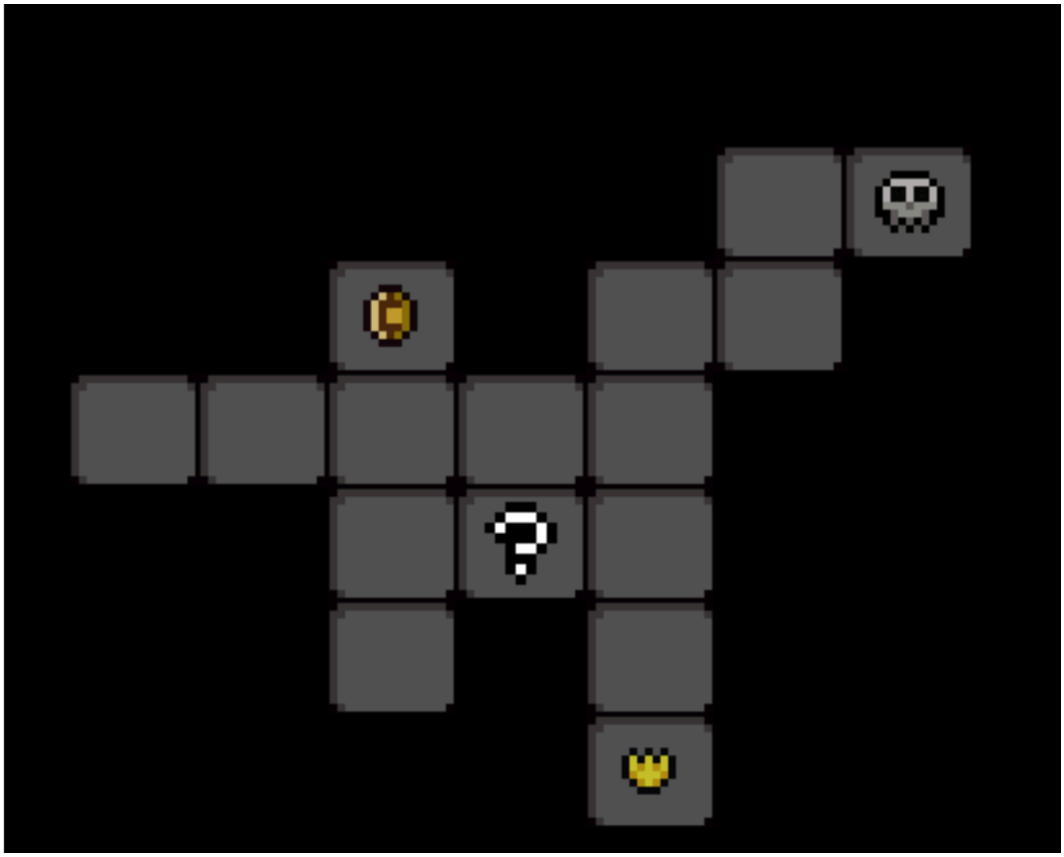
Nebo také s pohledem zboku (viz screenshot 2.2 ze hry Terraria), který se vyskytuje ve hrách typu *platformer*. Často se pod tímto pohledem dá představit také hrací plánek deskových her, který obsahuje nějakou mapu buď přírodní lokace, nebo půdorys nějakých místností a chodeb.

2.3.1 The Binding of Isaac

Binding Of Isaac (2011) a její *remake* Binding Of Isaac: Rebirth (2014) jsou populární *rogue-like* hry. Příběh her je volně inspirován starozákonním biblickým příběhem o Izákovi. Hlavní herní postava Isaac má za cíl dostat se ze sklepa svého domu na „finální boj“ s hlavním antagonistou hry — jeho matkou. Příběh hry má několik zakončení v závislosti na způsobu průchodu, který hráč zvolí.[6]

Jedno podlaží sklepa Isaacova domu se považuje za jednu lokaci. *Rogue-like* elementy hry spočívají právě v tom, že po smrti hlavní postavy začíná hráč úplně na začátku hry. Žádná z již navštívených lokací nezůstává stejná a pokaždé se daná lokace vygeneruje znova. Změní se polohy místností v lokacích, konfigurace každé místnosti, rozmístění obchodu a množství sběratelských surovin a předmětů.

Obrázek 2.2 Hra Terraria používající pohled zboku, převzato z `https://terraria.wiki.gg`



Obrázek 2.3 Příklad lokace ve hře Binding Of Isaac, převzato z [6]

Lokace je vždycky sestavena z nějakého množství místností spojených mezi sebou. Jde o jakýsi druh *dungeonu*. Příklad takové lokace je k nalezení na obrázku 2.3. Na obrázku jsou vidět vymezené šedé bloky ve tvaru obdélníků — místnosti dané lokace. Místnosti jsou propojené dveřmi. Každá místnost představuje herní plochu ohraničenou zdmi. V místnostech hráč bojuje proti různým nepřátelům a sbírá předměty za účelem zlepšení své postavy. Místnosti jsou rozdělené

do několika druhů: normální (většina z místností) a speciální. Speciální místnosti se pak dělí na další typy, nejdůležitějšími jsou třeba obchody, místnosti s odměnami nebo hlavním nepřítelem. Ukázka místnosti s hlavním nepřítelem je na obrázku 2.4.

Obrázek 2.4 Příklad místnosti s hlavním nepřítelem ve hře Binding Of Isaac: Rebirth, převzato z `https://multiplayer.net-cdn.it`

Generování lokace ve hře Binding Of Isaac (2011). Informace jsou převzaty z blogu [7], kde autor detailně popisuje proces generování lokace. Herní lokace je vždy generována na čtvercové síti velikosti 9×8 čtverců (X x Y). Buňky jsou očíslované ve tvaru „YX“, kde Y a X jsou kartézské souřadnice polohy buňky v síti. Y je v rozmezí [0, 7], X v rozmezí [1, 9]. Počet místností v dané lokaci se vypočte pomocí patřičného vzorce. Počet je závislý na tom, jak daleko hráč ve hře pokročil.

Generace ve hře začíná náhodným umístěním počáteční místnosti do sítě a přidáním buňky s číslem „35“ (Y-3, X-5) do fronty. Pak se pokračuje v iterování přes frontu. Pseudokód je k nahlédnutí ve výpisu 2.1.

Pokud během procházení frontou bude nalezena taková buňka, která nevytvořila kolem sebe žádnou další místnost, buňka se přidá do seznamu „slepých konců“. Seznam „slepých konců“ se použije později. Také se každé vytvořené místnosti přidělí její druh.

Pokud je potřeba vygenerovat lokaci, která obsahuje víc než 16 místností, počáteční místnost může být přidána do fronty vícekrát, aby se vynutil větší růst. Obecně by se dalo říct, že daný algoritmus patří do skupiny algoritmů *BFS — Breadth First Search* — vyhledávání do šířky. Algoritmus má ale ve hře své omezení, které spočívá v tom, že se místnost nevygeneruje v případě, kdy by měla víc než 2 sousedy. Dané omezení zaručuje, že místnosti budou sebou vytvářet větší chodby, které by se ale nikdy nepropojily mezi sebou a vytvořily by smyčku.

Na konci se vygenerovaný „půdorys“ lokace zkontroluje na splnění určitých požadavků. Požadavkem je vyžadovaný počet místností a to, že počáteční místnost není sousedem místnosti s hlavním nepřítelem.

Speciální místnosti. Poslední vygenerovaná místnost během iterování fronty je místnost s hlavním nepřítelem. Taková volba by měla ve většině zaručit to, že místnost s hlavním nepřítelem nebude sousedem počáteční místnosti. Po tom je rozmístěna tajemná místnost. Algo-

Výpis kódu 2.1 Pseudokód algoritmu generace lokace ve hře Binding Of Isaac

```
fronta bunek A;
pridat do A bunku "35";
dokud neni fronta A prazdna:
    bunka C := odstranena bunka z fronty A
    pro kazdeho souseda S bunky C ve smeru (nahoru, vpravo, dolu, vlevo):
        pokud S je obsazena mistnosti:
            pokracovat

        pokud S ma za souseda vic nez jednu bunku s mistnosti:
            pokracovat

        pokud je dostatek mistnosti:
            pokracovat

        s pravdepodobnosti 50 %:
            pokracovat

        oznacit S za bunku s mistnosti
        pridat S do fronty A
```

ritmus náhodně hledá takové buňky, které ještě nejsou místnostmi a aby byly sousedy s aspoň třemi dalšími místnostmi, ale ne sousedem ze seznamu „slepých konců“. Pokud nebude po prvních 300 pokusech taková buňka nalezena, požadavky se jemně zmírní. Pokud se to ani po 600 pokusech nepodaří, požadavky jsou zase zmírněny. To by mělo zaručit, že lokace bude vždy obsahovat jednu tajemnou místnost. Zbytek speciálních místností je rozmístěn na místa buněk ze seznamu „slepých konců“. Některé speciální místnosti jsou vygenerované vždy, zbytek má malou pravděpodobnost na vygenerování.[7]

Výše popsané generování map ve hře Binding Of Isaac je dalším příkladem použití *PCG* v samotné hře. Tohle použití si bere za cíl zvětšení znovuhratelnosti a zároveň obtížnosti hry, vzhledem k tomu, že se hráč skoro nemá možnost naučit určité lokace procházet.

2.4 Generování 3D map

2.4.1 Dwarf Fortress

Dwarf Fortress (2006) je hra od vývojářů Tarna Adamse a Zacha Adamse, kteří na hře v době psaní této práce pracují dodnes. Obsahuje dva herní módy: fortress mode a adventurer mode. Ve fortress módu hráč kontroluje skupinu trpaslíku a má za cíl postavit a zlepšovat pevnost, přežít a bránit se proti útokům nepřátelského okolí. V adventurer módu hráč cestuje světem a bojuje s příšerami.[8]

Hra se nedostala do široké veřejnosti kvůli své vysoké obtížnosti. Hra se však stala docela známou díky jejímu hernímu světu, který je zcela procedurálně tvořený.

Hráč si na začátku hry vygeneruje svět tím, že mu nastaví určité parametry. Takovými parametry, jak je vidět v 2.5, jsou třeba délka simulované historie daného světa, počet civilizací, množství surovin atd. Po úpravách parametrů se spouští simulace, výsledkem které bude mapa, na které hráč začne svou novou hru. Mapy můžou mít různou velikost (celkově 5 různých velikostí) a simulace proto můžou trvat relativně dlouhou dobu.

Nejdříve se vygeneruje samotná mapa a její terén. Do toho patří rozmístění extrémů jako jsou třeba zóny s nejnižší a nejvyšší teplotou, zóny různé vlhkosti, nadmořské výšky a četnosti srážek. Pak se rozmístění extrémů zkontroluje na splnění určitých omezení. V případě splnění pokračuje

Obrázek 2.5 Menu nastavení parametrů světa ve hře Dwarf Fortress, převzato z `https://dwarffortresswiki.org`

proces generování dál do fáze počítání odvozených parametrů, v případě neúspěchu začíná proces znova. Odvozeným parametrem je třeba vegetace, která je závislá na nadmořské výšce, teplotách a vlhkosti. Po tomto kroku se spouští proces erodování mapy, ve kterém se propočítávají koryta řek, snížení výšky hor v důsledku větrné eroze, vyschnutí některých vodních ploch…

Ve chvíli, kdy je mapa terénu hotová, následuje simulace historie světa, během které se vytváří, vyvíjejí a zanikají celé civilizace. To ponechává na mapě světa zbytky bývalých měst, budov, silnic a dalších struktur, které mají svůj význam v samotné hře.

Na obrázku 2.6 je vidět příklad použití *PCG* ve hře Dwarf Fortress jako jedné z její herních mechanik, kde výpočetní problém a rychlost jeho řešení je ponecháno na hráčovi. I když se na první pohled zdá být mapa ve 2D, ve skutečnosti je ve 3D. Kromě povrchu země se totiž generuje i podzemí.

2.4.2 Marvel's Spider-Man

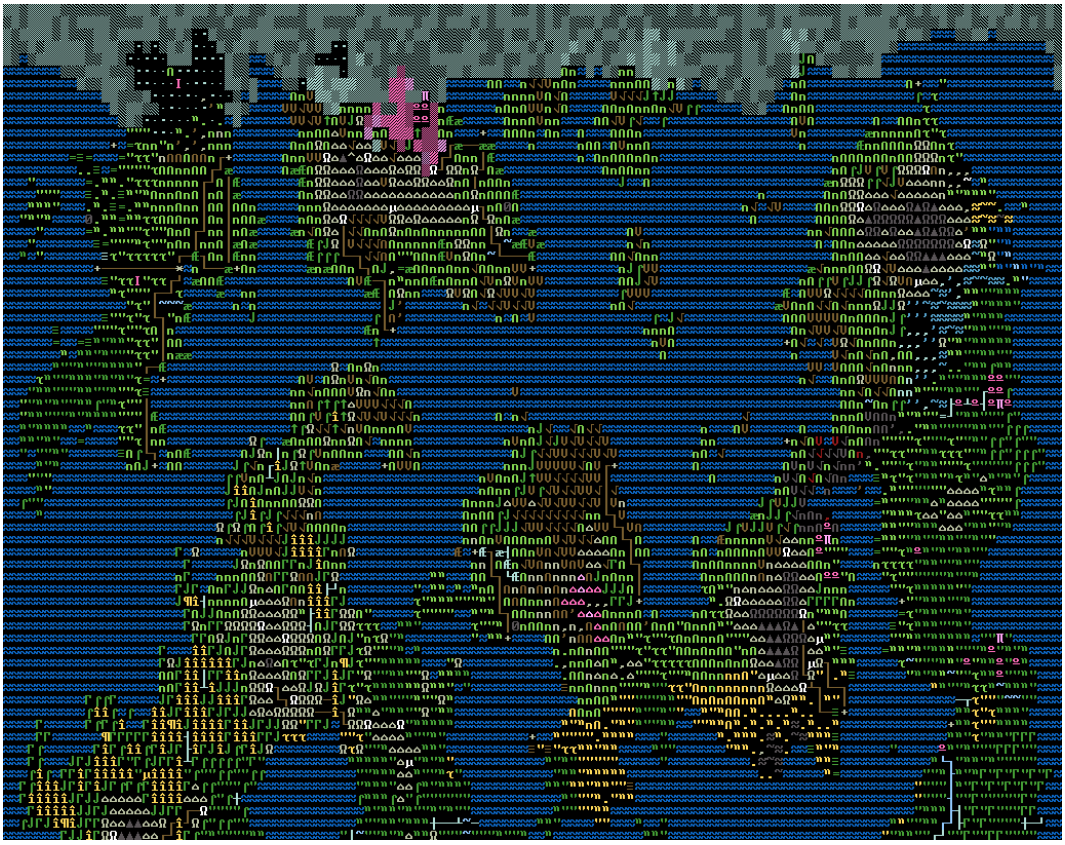
Marvel's Spider-Man (2018) a pokračování Marvel's Spider-Man: Miles Morales (2021) jsou hry od amerického herního studia Insomniac Games. Hráč těchto her se ocitne v roli známé postavy z komiksu Marvel — Spider-Mana. Hry se odehrávají v americkém městě New-York na ostrově Manhattan.

Jedna z nejvíc charakteristických mechanik obou her je schopnost hlavní postavy bez omezení cestovat celým Manhattanem pomocí pavučin. Takové cestování se velmi podobá houpání se na liánách v džunglích, v případě obou her v městských džunglích. Nechybí ani lezení po zdích nebo také běhání a skákání po střechách.

Přítomnost takových herních mechanik vyžadovalo od vývojářů umožnit přístup hráčům k místům, která nejsou běžně přístupná — zdi domů a jejich střechy. Ve hrách je obvykle všechno, co je mimo blízký dosah hráče, maximálně zjednodušené kvůli zmenšení obsahu práce vývojářů a optimalizaci samotné hry. Nebyl to ale případ těchto dvou her. V první a druhé části měly být zdi domů až do nejvyšší výšky a střechy domů propracované do detailu.

Herní lokace — Manhattan — má ve hře rozlohu 6 km krát 9 km a je postavena z tisíce budov, jak uvádí v [9]. Je snadné si představit obsah práce, který měli vykonat vývojáři a umělci z Insomniac Games. Po ruce ale měli hodně populární a hojně využívaný software od společnosti SideFX — Houdini.

„Houdini is an advanced procedural modeling, animation, effects, simulation, rendering, and



Obrázek 2.6 Vygenerovaná mapa střední velikosti ve hře Dwarf Fortress, převzato z `https://dwarffortresswiki.org`

compositing package. Houdini's power is based on procedural workflows. Working in Houdini involves creating networks of nodes connected together that describe the steps to accomplish a task. For example, a node that creates a box might be wired into a node that extrudes sides of the box, then a node that subdivides the polygons, then a node that edits the point positions."[10]

Houdini se v době psaní této práce považuje za velmi pokročilý a výkonný software. Díky svým vlastnostem se stal oblíbeným ve filmovém průmyslu, vývoji her, motion designu a virtuální realitě. V dalších sekcích se rozebere, jak přesně si Houdini našel své místo ve vývoji Marvel's Spider-Man (2018) a Marvel's Spider-Man: Miles Morales (2021)

Marvel's Spider-Man (2018). Před Insomniac Games povstal nelehký úkol postavit laicky řečeno „dvojče" reálného města. Celkově si procedurální možnosti Houdini našly použití v těchto dílčích částech: vytvoření půdorysu města, vytvoření a rozmístění budov, nastavení dopravy, rozmístění míst zločinů, rozmístění dekorativních prvků atd.

Nejdříve game designeři ručně vytvořili herní mapu, která vypadala jako něco, s čím se člověk v reálném životě běžně setkává, pokud jde o mapu města. Je na ní vidět síť ulic s jejich názvy, rozdělení na čtvrtě, parky a polohy unikátních objektů. Taková mapa byla hlavním podkladem pro přípravu digitální definice města v Houdini a game-enginu studia Insomniac Games, která se v dalších etapách vývoje použila v procedurálním generování obsahu.[11]

Vygenerovaný půdorys se skládá ze dvou částí: 3D modelu vozovek/chodníků a samotné 3D plochy půdy různých druhů. Síť ulic z graficky zpracované mapy byla konvertována do křivek ve 2D prostoru. Křivky obsahovaly údaje o názvech ulic, velikosti, typu jejich vozovek a další užitečné informace pro generaci výsledných 3D modelů. Křivky ulic mají různou barvu v závislosti

na tom, o který typ ulice jde. Typy jsou třeba aleje, třídy, běžné ulice a průjezdy mezi domy. Po převedení ulic do křivek v digitální definici města byly do ní přidány křivky vymezující zóny, na jejichž místě by se půda měla generovat odlišným způsobem. Šlo třeba o parky, pěší zóny nebo náměstí. Kromě vytvoření křivek bylo potřeba v Houdini vytvořit modulární 3D modely vozovek, chodníků, obrubníků atd., které by se sebou zaměňovaly rozmístěné křivky.[11] Výsledný půdorys města je k nahlédnutí také na obrázku 2.10.

Dalším krokem bylo vytvoření budov. Na vygenerovaném půdorysu měli umělci za úkol postavit *grey-box* celého města. Nešlo naštěstí o hotové budovy, jenom o jejich hrubou hmotu. *Grey-box* se stal předběžnou 3D podobou města, kde byly jasně vidět výškové omezení budov a jejich velikostí. V průběhu práce nad *grey-boxem* města byla také vytvořena v různých stylech knihovna modulárních prvků pro budovy. Využil se podobný princip, jaký byl zmíněn v úvodu 2.1. Zbytek těžké práce byl ponechán na Houdini. Šedé krabičky hrubé hmoty se změnily do podoby reálných budov pomocí scriptu v Houdini, jak je ukázáno na obrázku 2.10. Umělcům jenom zbylo provést drobné úpravy. Taková generace ale nebyla spuštěna jenom jednou. Vždycky zůstávala možnost zásahu do původní hrubé hmoty za účelem změny tvaru budovy. Taky existovala kontrola nad volbou stylu jednotlivých budov. Jedná se o takový pracovní postup, který umožňuje rychlé změny v prototypu, což obvykle vůbec není možné bez použití procedurálních technik. Při použití klasických postupů mohou takové změny zabrat desítky hodin vývojového času.[11]

Jedny z posledních se do mapy přidávaly různé dekorativní elementy. Je to vidět na příkladě odpadkových košů, pytlů a dalšího smetí na obrázku 2.10. Správnému rozmístěni takových elementů napomáhala existující digitální definice celého města, která v sobě obsahovala všechna „metadata“. Kromě smetí se třeba jednalo o autobusové zastávky, ventilace na střechách budov, stromy a lavičky v parcích.[11]

Výsledná herní mapa města New-York na ostrově Manhattan ve hře Marvel's Spider-Man (2018) je na obrázku 2.7.

Obrázek 2.7 Herní mapa města New-York na ostrově Manhattan ve hře Marvel's Spider-Man (2018), převzato z [11]

Marvel's Spider-Man: Miles Morales (2021). Pokračování příběhu o Spider-Manovi už nemělo před sebou tak velkou výzvu, jakou je vytvoření celého města ve 3D. Neměli ale vývojáři

úplně vystaráno. Roční období ve městě se muselo změnit z podzimního na zimní (viz screenshot ze hry 2.8). Na první pohled to nezní jak něco, co může vyvolat větší potíže. Jenomže při zamyšlení, co všechno taková změna období obnáší, se dá přijít na to, že práce nakonec není málo. Například je nutné tisíce dříve vytvořených baráků zasněžit nebo také změnit vzhled vozovky na zimní, přidat na ni sníh a sněhové závěje s ohledem na to, že jsou v reálném světě závěje v městských ulicích rozmístěné nerovnoměrně (záleží na intenzitě dopravy a míře využití chodníku). Vánoční dekorace na budovách a v ulicích také nemůže chybět. Podobný problém ve své bakalářské práci [12] zkoumal Bc. Jan Tislický, kde řešením bylo přimíchávání textury sněhu k textuře objektů.

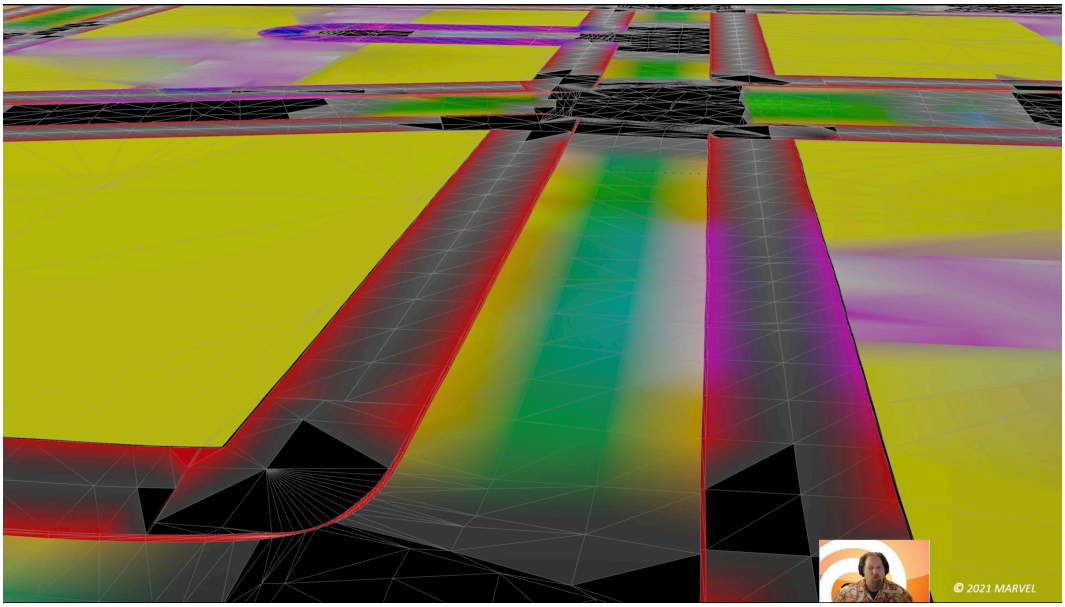
Obrázek 2.8 Zimní nálada ve hře Marvel's Spider-Man: Miles Morales(2021), převzato z [9]

I tentokrát si Houdini našel využití společně s digitální definicí města. Již vygenerované modely ulic měly projít důležitou úpravou, která spočívala v obarvení vrcholů daných modelů ulic barvami. Obarvení probíhalo na základě určitých pravidel. Během obarvování určitého vrcholu v modelu ulice se brala v potaz jeho vzdálenost od fasády nejbližší budovy nebo od vozovky, velikost vozovky, zda se nejedná o křižovatku nebo chodník nebo zda se vrchol nenachází na místě určeném pro parkování aut. Právě v téhle fázi se nejvíce hodila „metadata" o ulicích. V případě rozmístění objektů do ulic (třeba sněhových závějí) se kontrolovala absence kolizí s jinými objekty již přítomnými v daném místě.[9] Po skončení obarvování vypadaly modely ulic tak jak je zobrazeno na obrázku 2.9.

Barvy vrcholů byly následně použity i pro správné míchání různých materiálů. Zelená barva odpovídala jednomu materiálu, červená jinému. V případě smíšených barev se různé materiály ploužily s různou silou v závislosti na poměru barevných složek.

Údaje o budovách a jejich hrubých hmotách pomohly správně pověsit na zdi a mezi baráky světélka nebo rozmístit další vánoční dekorace.

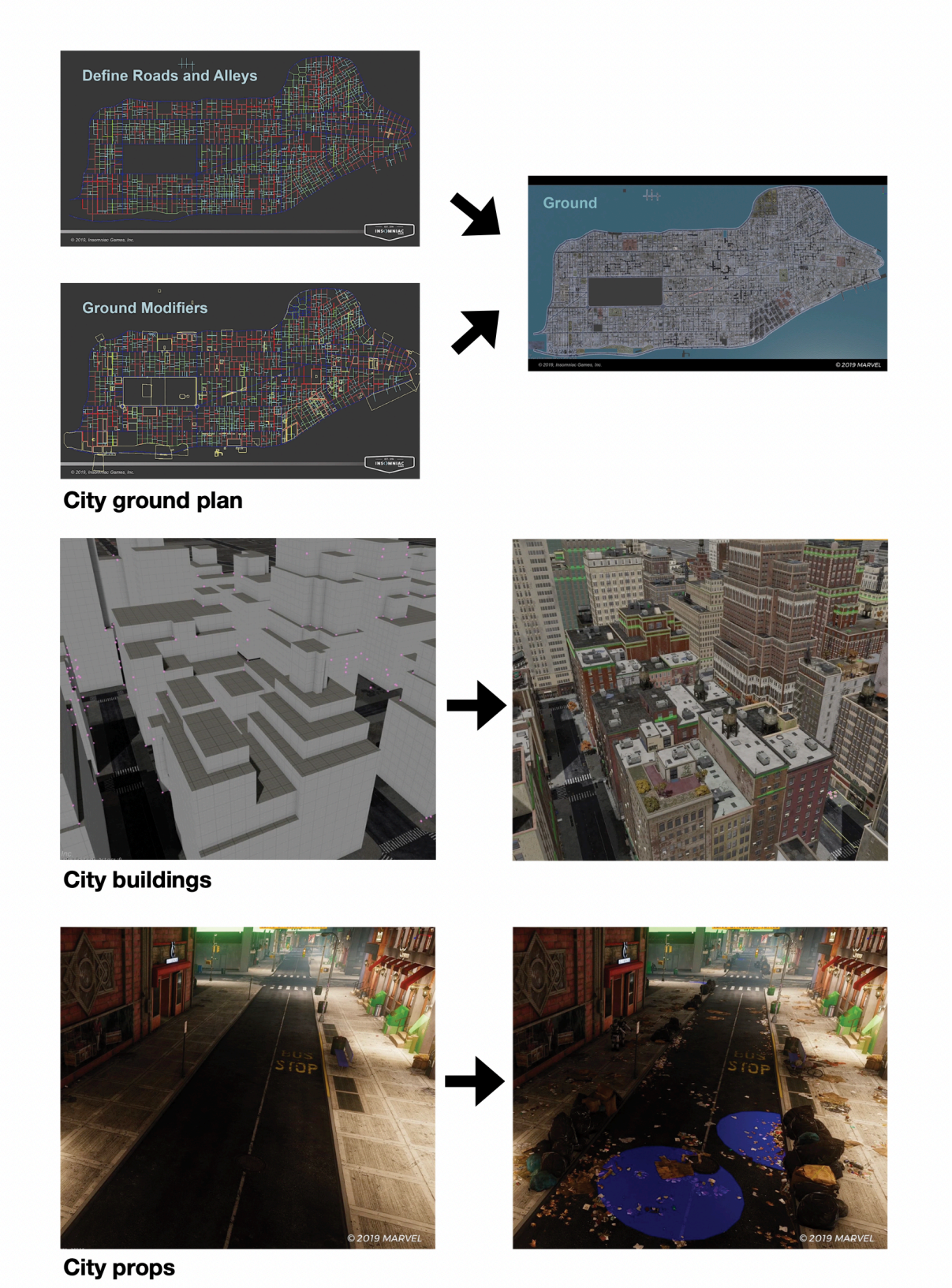
Výše popsané použití *PCG* se tentokrát týká prvního případu, kde se *PCG* používá ve fázi vývoje a ne v samotné hře jakožto jedna z herních mechanik. Vývojáři si usnadnili práci tím, že provedli důkladné přípravy, které ale nespočívaly v repetitivní a manuální práci. Dané přípravy napomohly využít jiné pracovní postupy. Takové postupy dovolují rychlé změny v pozdějších etapách vývoje a vynechávají potřebu ve stovkách hodin lidské práce. Také si vývojáři nechali široce otevřené dveře pro další růst a zlepšení jejich produktu.



Obrázek 2.9 Obarvené vrcholy geometrie ulic ve hře Marvel's Spider-Man: Miles Morales (2021), převzato z [9]

2.5 Shrnutí

Jak z příkladů uvedených v sekcích výše vyplývá, procedurální generování map se používá zejména k zrychlení vývoje, zredukování manuální práce během vývoje a díky tomu i ušetření velkého množství lidského úsilí a peněz. Toto generování samozřejmě není bezchybné a i přes tuto automatizaci je nutný lidský zásah ke zdokonalení výsledku a jeho věrohodnosti. To však je ve srovnání s množstvím ušetřené práce téměř zanedbatelné úsilí. Navíc se procedurální generování map dá aplikovat na řadu různých případů, generování 2D/3D map, k simulaci změny světa apod. Je tedy jasné, že v dnešní době se používá v řadě známých a moderních projektů a v budoucnu bude pravděpodobně využíván ještě více než dnes.



Obrázek 2.10 Použití Houdini ve hře Marvel's Spider-Man (2018), převzato z [11]

Kapitola 3
Wave Function Collapse

Wave Function Collapse je pojem z kvantové mechaniky. Tento pojem popisuje jev, ve kterém částice během pozorování přechází ze superpozice do stavu určeného. Jinými slovy z nějaké množiny všech svých možných stavů se částice ocitne právě v jednom z nich. Takovému okamžiku se říká kolaps.[13]

Algoritmus procedurálního generování, který je popisován dál, nemá s kvantovou mechanikou nic společného, ale ve svém základu používá princip kolapsu stavů. Právě proto dostal stejný název.

„WFC is a constraint based procedural algorithm based in the idea that a quantum particle can exist in a multi-state when unobserved. As soon as the particle is observed, the possibilities disappear and the wave function collapses.“[14] Z definice algoritmu je přímo vidět souvislost s pojmem kvantové mechaniky. Algoritmus byl poprvé zveřejněn *indie* vývojářem Maximem Guminym v GitHub repozitáři. Algoritmus byl určen pro vytvoření variací 2D *pixel-based* obrázku z předlohy, která byla rozdělena do malých kousků. Z takových kousků se vytvářely nové obrázky, které byly podobné předloze.[15]

Po zveřejnění se algoritmus rychle adoptoval pro účely *PCG*. Nezávislí vývojáři a experimentátoři vytvořili modifikace algoritmu pro generování map ve 2D. Po takových modifikacích bylo jasné, že se *WFC* vypořádá i s vyššími dimenzemi. Proto se daný algoritmus začal používat i pro generování 3D map.[16]

3.1 Popis fungování

Tato sekce popisuje dvě fáze běhu algoritmu WFC. Jedna má na starosti přípravu sítě a druhá představuje samotný proces generování.

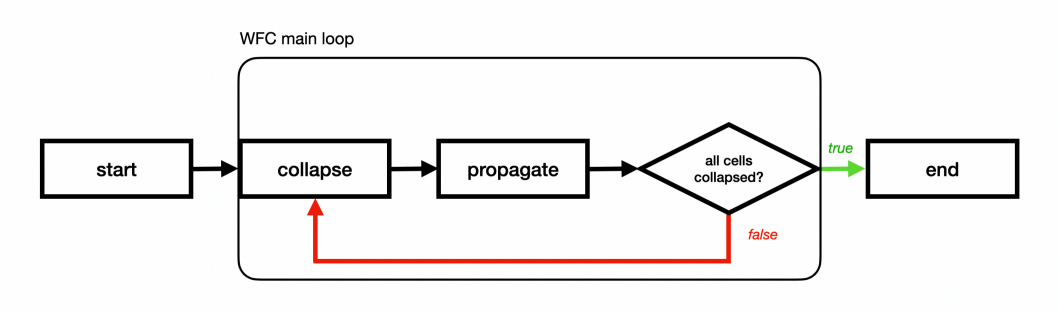
3.1.1 Inicializace

Běh algoritmu začíná počáteční fází, ve které se nainicializuje *grid* (síť). Síť je sestavena z buněk. Síť může být založena na buňkách různých tvarů, nejčastěji se ale používá čtvercová síť, kde buňkou je čtverec. Každá buňka představuje *place-holder* neboli místo, kam se může umístit výsledný objekt.

Objektem se rozumí datová struktura, která v sobě obsahuje informace o svém stavu a také o svých povolených sousedech, které představují jiné objekty. Seznam takových objektů je vstupem daného algoritmu. Způsoby vytvoření seznamu objektů jsou různé a budou popsané dále.

Tato práce se zaměří právě na jednu část stavu objektu — jeho vzhled. Po inicializaci je celá síť v neprozkoumaném (unobserved) stavu, což znamená, že každá její buňka je v superpozici,

v níž jsou povoleny všechny možné vzhledy (na místo buňky se může umístit jakýkoliv objekt ze seznamu všech objektů).



Obrázek 3.1 Hlavní smyčka algoritmu WFC

3.1.2 Hlavní smyčka

Hlavní smyčka – `main loop` – algoritmu *WFC* na diagramu 3.1 běží, dokud není celá síť prozkoumána. To znamená, že všechny buňky v síti mají určený svůj finální vzhled, neboli mají do sebe umístěný přesně jeden objekt ze seznamu všech objektů. Smyčka obsahuje dva kroky: collapse a propagation.

Collapse. V daném kroku musí algoritmus zvolit takovou buňku, která má nejmenší entropii. Entropií se zde myslí míra neurčitosti. Čím víc možných stavů má buňka, tím větší je její entropie. Po inicializaci je entropie celé sítě nejvyšší možná, vzhledem k tomu, že každá buňka sítě může umístit do sebe jakýkoliv objekt. V případě, že buněk s nejmenší entropií je několik, vybere se náhodně libovolná z nich, nebo se ve výběru použije zadaná heuristika. Pro výpočet entropie buňky se používají různé vzorce. Můžeme entropii buňky vypočítat třeba takto [17]:

$$E = -\sum_{i=1}^{n} P_i log P_i$$

kde n je počet povolených objektů na místě dané buňky a P_i je pravděpodobnost výskytu daného objektu v očekávaném výsledku (očekávaný výsledek sebou představuje pravděpodobnostní rozdělení).

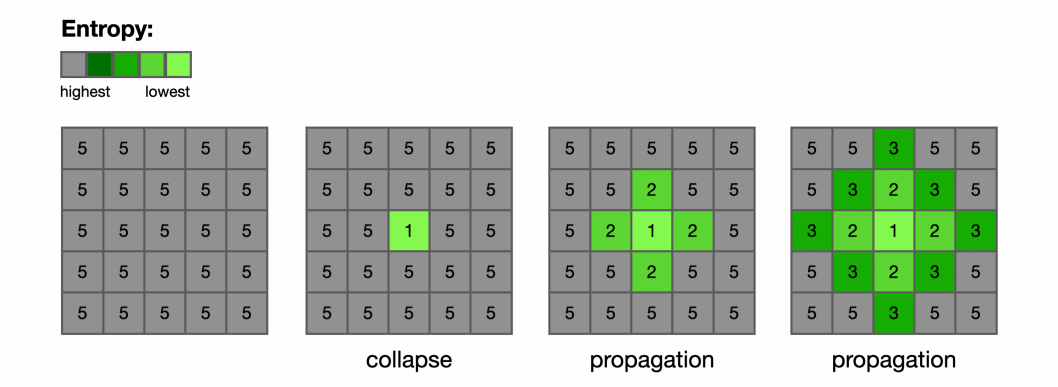
Volbou určená buňka bude prozkoumána a následně proběhne její „kolaps“. Kolapsem buňky je chápána volba objektu, který se na její místo umísti. Takovou volbu je možné provést znovu několika způsoby. Takovými způsoby jsou náhodný výběr, výběr na základě pravděpodobností nebo na základě vah. Použití vah vždycky závisí na daném *use-casu* a na očekávaném výsledku.

Na konci kroku se na místě buňky objeví objekt zvolený během kolapsu.

Propagation. Po kolapsu buňky se zbytek sítě musí dozvědět o tom, co se stalo, a odpovídajícím způsobem zareagovat. Probíhá propagace změny. Proces propagace je znázorněn na obrázku 3.2. Číslo každé buňky znamená počet možných objektů, které se na její místo dají umístit.

Po kolapsu určité buňky se taková změna nejdříve zpropaguje jejím přímým sousedům. Propagace může mít za následek snížení jejich entropií. Takové snížení je důsledkem existence seznamu povolených sousedů, který si drží každý objekt. Po umístění určitého objektu nemůže být definitivně v sousedních buňkách objekt, který není v seznamu povolených sousedů. Proto všechny takové nepovolené objekty můžeme rovnou vynechat. Na přímých sousedech propagace ale nekončí. Po odebrání některých možností u přímých sousedů změnou můžou být zasažené i jejich

sousední buňky. Všechny buňky zasažené takovou změnou se přidají do fronty za účelem pokračování propagace. Takový proces propagace končí ve chvíli, když nezbude ve frontě žádná buňka, která si zkrátila seznam možných objektů.



Obrázek 3.2 Etapa propagace změn algoritmu WFC

3.2 Vytvoření seznamu povolených sousedů

Existuje několik způsobů, jak pro každý objekt vytvořit seznam povolených přímých sousedů. Za přímé sousedy nějaké buňky se považují takové buňky, které mají s ní v síti sdílenou hranu. V případě čtvercové sítě jsou takovými sousedy buňky nahoře, dole, vpravo a vlevo.

Původní způsob od Maxima Guminyho je v krátkosti založený na zjišťování povolených sousedů na základě celé předlohy, která se rozdělí na menší části. Procházením předlohy se kouká na rozmístění pixelů vůči sobě, takovým rozmístěním se následně řekne, že jsou validní. Tímhle se vytvoří seznam povolených sousedu. [15]

Manuální zápis povolených sousedů. Jedná se o nejjednodušší, ale zároveň o nejméně časově efektivní způsob v případě velkého počtu objektů.

V souboru zvlášť, nebo přímo v kódu napíše uživatel natvrdo pro každý objekt seznam jeho povolených přímých sousedů 3.1. Následně před inicializací *WFC* se takovými seznamy proiteruje a u objektů budou nastaveny jejich povolení sousedé.

Sockety. Sockety se nejčastěji používají v případě, kdy se pracuje s objekty v síti, která je sestavena z buněk s jasně definovanými protistranami (např. top — bottom, right — left). Právě takový případ bude popsán.

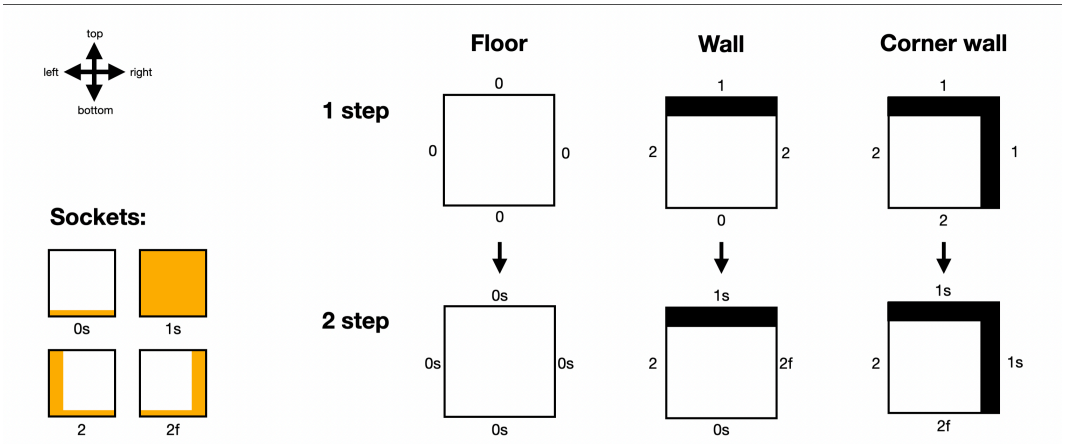
Sockety jsou založené na symetrických vlastnostech jednotlivých stran objektů. Opět ale musí uživatel manuálně nebo pomocí skriptu vytvořit seznam. Seznam tentokrát nebude obsahovat povolené sousedy pro každou stranu, ale pouze název socketu, který jednotlivá strana má. Ve srovnání s předchozím případem se taková úloha zdá být jednodušší pro velký počet objektů.

Sockety se vytváří na základě několika jednoduchých pravidel. Rozebere se to na příkladě objektů, ze kterých se dá poskládat místnost s pohledem shora, viz obrázek 3.3. V příkladu 3.3 musí pro generování místností každý objekt existovat navíc ve 4 variacích podle své rotace. Ve čtvercové síti jde o rotaci po 90° (0°, 90°, 180°, 270°). Pro zjednodušení popisu kroků se použijí jenom objekty s rotací 0°.

V prvním kroku se musí projít všemi 4 stranami každého objektu a podívat se na to, jak vypadá její pohled zboku. Právě tomuto pohledu se bude dále říkat socket. Číslování začne například od nuly. Pokud je socket pozorován poprvé, označí se dalším číslem. V případě, že

Výpis kódu 3.1 Příklad manuálního zápisu seznamu povolených sousedů

```
A, B, C: objects
valid_neighbours = [
    A: {
        top: [B, C]
        bottom: [A, B]
        right: [A, B]
        left: [B]
    },
    B: {
        top: [A, B, C]
        bottom: [A, B, C]
        right: [A, B, C]
        left: [A, B, C]
    },
    C: {
        top: [B, C]
        bottom: [A, B, C]
        right: [B]
        left: [B]
    }
]
```



Obrázek 3.3 Přidělení socketů objektům

pozorovaný socket vypadá jak vertikálně odzrcadlený socket, který již byl očíslován, přidělí se mu existující číslo odzrcadlené verze.

Ve druhém kroku budou vzaty v úvahu symetrické vlastnosti socketu. Pro dvě kopie socketu, které se dají bez úprav rozmístit hned naproti sobě, aby na sebe správně navazovaly, přidáme k číslu takového socketu suffix `s` — symmetrical. To bude znamenat, že socket je plně symetrický. Socket, který pro stejnou úlohu bude vyžadovat vertikální zrcadlení své kopie, ponechá svoje číslo bez suffixu. Jeho zrcadlená kopie získá suffix `f` – flipped. Vytvořený seznam bude mít takovou podobu 3.2.

Seznam se sockety bude vstupem pro funkci, která ho zpracuje do podoby 3.1. Funkce projde každý objekt a srovná ho se všemi ostatními objekty včetně sebe. Řekne se, že pro objekt A a objekt B je B validní přímý soused pro A na straně S, pokud je splněna jedna z podmínek:

Výpis kódu 3.2 Příklad manuálního zápisu seznamu povolených sousedů

```
floor, wall, corner_floor: objects
sockets = [
    floor: {
        top: 0s
        bottom: 0s
        right: 0s
        left: 0s
    },
    wall: {
        top: 1s
        bottom: 0s
        right: 2
        left: 2f
    },
    corner_floor: {
        top: 1s
        bottom: 2f
        right: 1s
        left: 2
    }
]
```

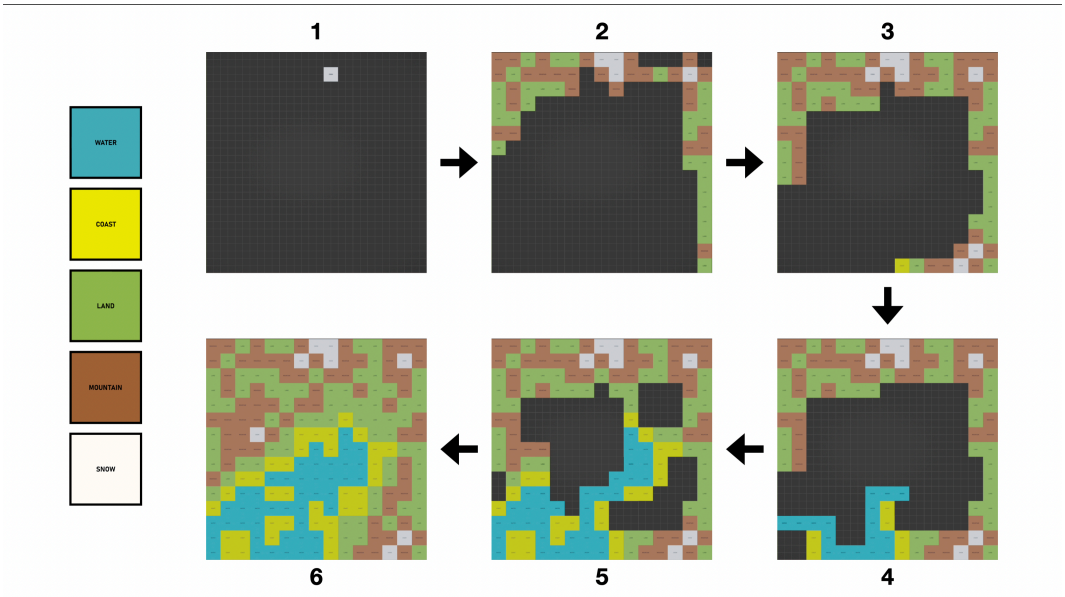
- socket objektu A na straně S je se suffixem `s` a zároveň socket objektu B na protistraně S je se suffixem `s` a má stejné číslo,
- socket objektu A na straně S je bez suffixu a zároveň socket objektu B na protistraně S je se suffixem `f` a má stejné číslo,
- socket objektu A na straně S je se suffixem `f` a zároveň socket objektu B na protistraně S je bez suffixu a má stejné číslo.

3.3 Generování map

2D. První ukázka použití algoritmu WFC je případ generování map v rovině. V případě dané práce to je čtvercová síť, která má dvě osy — X a Y. Potom, co je vytvořen seznam objektů a samotné objekty vědí o svých validních sousedech, stačí jen spustit hlavní smyčku WFC a čekat na vygenerování výsledku. Takový proces se ukáže na příkladě, kde je zadáním vygenerovat *pixel-based* mapu terénu. Mapa se bude moct sestavit z 5 různých „políček“: voda, pobřeží, zem, hora a sníh. Pravidla sousedství by mohla vypadat takto: voda může sousedit s pobřežím, pobřeží se zemí, zem s horou a hora se sněhem, a naopak. Navíc implicitně každé políčko může být sousedem samo se sebou.

Průběh algoritmu by v síti 15x15 buněk mohl vypadat jako na obrázku 3.4. Nejdříve se nahoře objevilo políčko sněhu, pak kolem něj další políčka sněhů a hory. Volba algoritmem takových políček je jasná — kolem sněhu se může objevit jenom další sníh nebo hory. V dalších krocích se kolem hor začala objevovat zem, ale stále se neobjevil žádný písek. Změnilo se to až na obrázku 3.4. Písek se stal na mapě místem „krystalizace“ vodní plochy. Generace se dokončila na místě prostředních buněk, kde se objevil zbytek pobřeží. Výsledná mapa terénu připomíná mořskou zátoku.

Na obrázku je vidět, ze WFC striktně dodrželo pravidla sousedství políček. Není žádné políčko sněhu vedle políčka vody atd. Takový výsledek se zaručí fází propagaci změn, právě během které se takové možnosti vyloučí.



Obrázek 3.4 Generace *pixel-based* mapy terénu pomocí WFC

3D. Případ generování mapy ve 3D se bude málo čím lišit od případu s menším počtem dimenzí. Hlavní rozdíly jsou v tom, že síť místo dvou os obsahuje 3, a seznam validních sousedů se pro každý objekt rozšířil o 2 další položky: validní sousedé seshora a zespoda.

Pro vytvoření seznamu validních sousedů pomocí socketů taky přibudou dvě další položky: horní a dolní socket. Tyto sockety ale mají jiný způsob zápisu. Na začátku socketu se přidá prefix `v` – version a na konci suffix `_r`, kde r bude číslo z množiny $0, 1, 2, 3$ podle toho, jak je původní objekt otočený. Fakticky r se dá spočítat ze vzorce $r = object\_rotation/90$. Vzhledem k tomu, že název takových socketů dostal prefix, samotné číslování může běžet zvlášť od číslování 4 zbylých socketů (sockety `v0_0` a `0` jsou různé). V okamžik zpracování seznamu socketů bude platit, že objekt je validní soused jiného objektu seshora, resp. zespoda, pokud si jsou jejich protější sockety rovny a mají prefix `v`.

3.4 Příklady použití v existujících hrách

Bad North (2018). První známé použití algoritmu WFC bylo zaznamenáno ve hře Bad North od švédského herního studia *Plausible Concept*, která byla vydána v roce 2018, což je relativně krátká doba po prvním zveřejnění algoritmu. Jedná se o strategii v reálném čase, ve které hráč musí bránit království od útoků Vikingů. Kolo hry se odehrává na ostrově, kde musí hráč zachránit co nejvíce civilistů, kteří žijí ve svých domovech. Ostrovy jsou generované procedurálně a právě v této generaci je použito WFC. Generátor používá sadu modulárních objektů ve 3D, ze kterých se dají poskládat různé variace ostrovní půdy. Generace probíhá ve 3 osách kartézského souřadnicového systému.[18][19]

Townscaper (2020). Jedná se o indie hru od švédského vývojáře Oskara Stålberga. Zajímavým detailem je to, že Oskar Stålberg je jedním ze zakladatelů jiý zmíněného studia Plausible Concept a podílel se na vývoji hry Bad North. Zkušenosti získané v předchozím projektu si zjevně našly použití i v tomto projektu.

Townscaper nemá definovaný konec a jedná se spíš o relaxační hru. Hráč začíná na prázdné mapě někde uprostřed oceánu a má možnost postavit město z krychliček několika barev. Žádnou

další schopnost než postavit krychličku, odstranit krychličku a zvolit barvu krychličky hráč nemá. Všechno za něj udělá samotná hra.[20]

Spojením jednotlivých krychliček vznikají budovy. Občas spojení krychliček může způsobit větší změny. Nejlepším příkladem takové změny je okamžik, kdy hráč postaví krychličku tak, aby vznikl vnitroblok několika budov. Úvodní povrch mezi budovami je dlážděný a není tam žádná zeleň. Ve chvíli, kdy se z těchto budov stane vnitroblok, dláždění uvnitř se promění na zelený park nebo zahrady.[20]

Ve hře rozmístění takových dekorací a celkovou změnu prostředí řídí WFC. Po každém umístění, nebo odstranění krychličky algoritmus zkontroluje splnění všech omezení a zpropaguje akci hráče. Důsledkem takové propagace může být změna vzhledu již vygenerovaných struktur.[20]

1. Bad North (2018)

2. Townscaper (2020)

Obrázek 3.5 Záběry ze hry Bad North (2018) a Townscaper (2022), převzato z `https://www.badnorth.com` a `https://www.townscapergame.com`

3.5 Omezení WFC a způsoby jejich řešení

Konflikty. Občas může za běhu algoritmu WFC během propagace změn nastat stav, kdy nějaké buňce nezbude ani jeden objekt, který by se mohl na její místo umístit. Stane se to z důvodu, že přestane existovat taková možnost, která by dodržela pravidla sousedství. Takovému stavu se řekne *konflikt* nebo *kontradikce*. Kontradikce má dva způsoby řešení a to: začít generaci úplně od začátku, nebo přidat backtracking. Je jasné, že začínat od začátku je často neefektivní (obzvlášť pokud je hotová skoro celá mapa), proto by byl lepší volbou *backtracking*.

Backtracking je založený na ukládaní rozhodnutí a změn, které nastaly v průběhu hlavní smyčky algoritmu. V případě konfliktu se musí algoritmus vrátit do okamžiku posledního kolapsu buňky a to včetně znegovaní změn provedených během propagace. V kolapsu se tentokrát zvolí jiný objekt a provede se typická propagace. V případě, že všechny možnosti způsobují konflikt (včetně případů, když buňka měla jenom jednu možnost), musí se vrátit až do okamžiku před předchozího kolapsu a musí se provést nový kolaps. Takové vrácení se nazpět v historii může klidně dostat běh až na úplný začátek — na kolaps první buňky v síti.

Kontrola nad výstupy. Z popisu algoritmu je zřejmé, že jeho uživatel nemá jak ovlivnit výsledky generování bez svého zásahu. Všechno je ponecháno na pravděpodobnosti a vahách jednotlivých objektů. Co když ale uživatel vyloženě chce, aby nějaké místo na mapě vypadalo určitým způsobem, nebo aby se určitá místnost v dungeonu nacházela v horním levém rohu mapy? Jsou naštěstí určité techniky, které můžou zaručit větší kontrolu nad výsledkem.

První technika přidá kontrolu tím, že se předběžně nadefinuje očekávaný půdorys a zbytek generování (interiéry místností a chodby) bude ponechán na algoritmu. Na příkladě mapy s místnostmi a chodbami v čtvercové síti stačí v inicializační fázi manuálně natvrdo umístit do buněk objekty zdí nebo pouze ukázat, že se v označených buňkách musí nacházet objekt typu zeď. Volba

určité instance zdi bude ponechaná na algoritmu WFC. Tím se zaručí, že vygenerovaný výsledek má předem určený půdorys.

Další technika spočívá v tom, že na začátku do vyznačených buněk budou natvrdo rozmístěné určité objekty, které ve fázi generování vynutí kolem sebe „krystalizaci“ určitých typů objektů. Taková krystalizace byla ukázána u generování 2D terénu na obrázku 3.4. Kolem sněhu se objevoval sníh a hory a od prvního políčka pobřeží začala generace vodní plochy. Dané specifikum fungování *WFC* si může najít různé využití. Na příkladě dungeonů se v určitých místech mapy dá vynutit generace speciálních místností, třeba těch s hlavním nepřítelem. Ještě lepší výsledky by se daly očekávat od kombinování dvou zmíněných technik.

Časová složitost. Jednou z hlavních nevýhod algoritmu je jeho časová složitost. V případě velkých map může běh trvat relativně dlouho. Navíc tomu nepomáhají možné stavy kontradikce a potřeby je řešit. Časová složitost algoritmu je jednoduše odvoditelná: pro mapu velikosti n políček bude minimálně n kolapsů, navíc po každém kolapsu proběhne propagace změn, která v nejhorším případě ovlivní n políček. Tohle implikuje časovou složitost $O(n^2)$. Se zohledněním případů kontradikce se složitost může zvětšit na $O(n^3)$.

Existují způsoby zlepšující časovou složitost algoritmu. Většina z nich je založena na ranní predikci stavu kontradikce, která zastaví pokračování ve větvi, která v budoucnosti určitě skončí v kontradikci. Dalším způsobem je vylepšení designu objektů pro generaci nebo pokoušení se o celkové zmenšení granularity knihovny objektů. Čím jsou větší objekty, tím je menší počet políček potřebný pro generování a tím pádem menší n.

Kapitola 4
Generátor dungeonů

Na začátku této kapitoly se zadefinuje význam slova *dungeon*. V literatuře a na internetu existuje spousta interpretací tohoto slova.

Cambridgeský slovník [21] definuje *dungeon* následně: *„an underground prison, especially in a castle“*. Podle webu [22] je *dungeonem* také mapa videohry, která sebou představuje uzavřené nepřátelské prostředí, ve kterém hráč potkává nepřátele. Nejčastěji jsou takové lokace součástí hradů nebo jeskyň.

Je vidět, že společným prvkem obou definic je hlavně uzavřené prostranství a nepřátelské prostředí (žalář z první definice se jinak než nepřátelským prostředím v obecném významu nazvat nedá). V této práci se tedy omezí význam slova dungeon na uzavřené prostranství, které je tvořeno sadou místností a chodeb propojených mezi sebou.

4.1 Způsob a prostředí implementace

Na začátku se sepíše seznam funkčních požadavků generátoru.

Funkčními požadavky generátoru dungeonů jsou:

1. generování uzavřeného prostranství, které sebou představuje sadu místností a chodeb v rámci jednoho podlaží,
2. načtení seznamu uživatelem nadefinovaných objektů,
3. zobrazování procesu generování a vygenerovaných výsledků v reálném čase,
4. exportování výsledků do souborů běžných formátů.

Výše sepsané funkční požadavky splní rozšíření pro nějaký 3D modelovací software nebo herní engine. Softwarem může být třeba Blender, 3dsMax, Maya nebo Houdini a nejznámějšími otevřenými herními enginy jsou Unreal Engine a Unity. Takové možnosti jsou lepší z hlediska 4. funkčního požadavku, protože už mají vestavěné moduly exportu výsledků nebo jsou již dokonce konečnou destinací exportovaných výsledků.

Za účelem implementace volí autor prostředí modelovacího softwaru Blender. Blender je bezplatný open-source software vydávaný pod licencí GPL pro tvorbu 3D obsahu, který je udržován organizací Blender Foundation. Umožňuje práci v jakékoliv části produkční grafické pipeliny: modelování, rigging, animace, simulace, renderování, compositing, motion tracking a také střih videí a vývoj videoher.[23] Kromě výše zmíněných vlastností Blender také umožňuje rozšiřovat svou funkčnost pomocí add-onu, které jsou napsané v jazyce Python. Pro takové účely Blender zveřejnil Blender API.

Důvody volby prostředí Blender pro implementaci generátoru dungeonů:

- industry standard ve vývoji videoher,
- bezplatný,
- Blender API v jazyce Python,
- možnost vytvořit objekty pro vstup generátoru,
- možnost exportu výsledků pro další použití.

Samostatná aplikace splní dané funkční požadavky. Přináší ale se sebou potřebu implementovat spoustu logiky a obslužných procedur, aby se vůbec spustila. Navíc tato možnost není multiplatformní. Rozšíření pro existující software takové problémy nemá, protože se jedná o již hotovou aplikaci, která je navíc vyvíjena pro několik různých platforem. Z těchto důvodů byla varianta samostatné aplikace vyloučena.

Je třeba také zmínit, že pro uživatele je jednodušší nacházet se v prostředí 3D modelovacího softwaru z toho důvodu, že uživatelem nadefinované objekty pro vstup generátoru můžou být vytvořené ve stejném místě, kde se i použijí.

Rozšířením funkčnosti Blenderu bude tedy plugin v podobě skriptu. Varianta pluginu v podobě add-on modulu nepřichází v úvahu kvůli tomu, že uživatel musí dostat transparentní API, se kterým bude moct interagovat a měnit ho. Klasické grafické rozhraní zásuvného modulu takovou svobodu nabídnout nemůže, navíc by takové rozhraní mohlo být dost neintuitivní.

Implementovaný nástroj by se používal jakožto nástroj ve fázi vývoje nějaké hry a ne jako její nedílná součást v roli mechaniky. Výstupy generování generátoru se dají použít pro rychlejší vytvoření levelů dungeonů, které by pak následně podstoupily kosmetickým úpravám.

4.2 Návrh

Skript je reprezentován sadou souborů, ze kterých by ale uživatel mohl měnit jenom dva. Jeden je vstupní bod skriptu, kde se definují důležité proměnné jako třeba velikost mapy nebo název kolekce, kde se nacházejí připravené modely objektů, ze kterých se mapa bude generovat. Druhý soubor obsahuje seznam objektů, ve kterém se nacházejí určité údaje o objektech jako třeba jejich validní sousedé, sockety nebo váha, resp. pravděpodobnost. Kromě seznamu objektů se v tomto souboru nachází i funkce, které takový seznam zpracují, vytvoří prototypy objektů a následně spustí samotný běh generátoru.

Prototyp objektu je datová struktura, která v sobě obsahuje všechny informace potřebné k vytvoření instancí prototypu. Takovými informacemi jsou data o geometrii objektu a jeho natočení. Kromě toho jsou v těchto prototypech také uložené seznamy povolených sousedů (jako v sekci 3.2) nebo socketů (jako v sekci 3.2).

Instance prototypu je hotový objekt na mapě, který byl vytvořen z informací uložených v prototypu. Prototypy pomáhají zefektivnit práci s generátorem tím, že vyloučí potřebu mít ve scéně připravené všechny variace objektu.

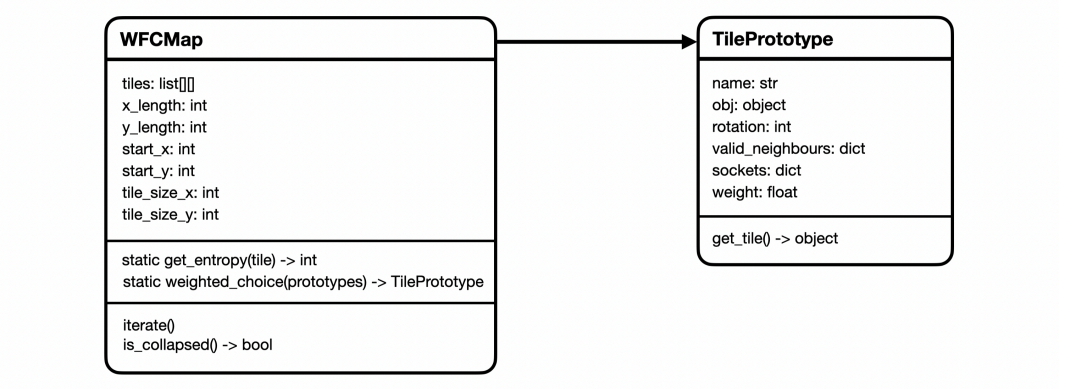
Neměnnou součástí skriptu jsou dvě třídy: `WFCMap` a `TilePrototype`.

TilePrototype. Jedná se o třídu, která reprezentuje prototyp objektu. Ukládá v sobě název objektu `name`, referenci na objekt `obj`, který definuje geometrii objektu, dále otočení objektu `rotation`, seznam validních sousedů `valid_neighbours`, socketů `sockets` a jeho váhu `weight`, resp. pravděpodobnost. Obsahuje jedinou metodu, která má na starosti vytvoření reálných objektů, které se budou moct zobrazit ve scéně. Dá se říct, že jde o továrnu objektů. Po volání vrátí metoda správně pootočený duplikát objektu, na který ukazuje atribut `obj`. Prototyp v sobě totiž obsahuje pouze referenci na geometrii, která je pootočená o 0 stupňů za účelem šetření paměti a pro lepší uživatelskou přívětivost.

WFCMap. Tato třída reprezentuje generovanou mapu, která je založená na 2D čtvercové síti. Má na starosti uložení aktuálního stavu mapy a samotný proces generování. Aktuální stav mapy je uložen v 2D poli `tiles`. Na začátku generování jsou v každém políčku pole uložené všechny možné prototypy objektů (princip superpozice). Velikost mapy v ose X a Y je zadána proměnnými `x_length` a `y_length`. `start_x` a `start_y` určují levý dolní roh mapy a velikost jednotlivých buněk v osách X a Y uchovávají `tile_size_x` a `tile_size_y`.

Před spuštěním generování má uživatel možnost změnit chování generátoru v části způsobu počítání entropie buňky, nebo způsobu volby výsledného objektu v okamžik kolapsu. Tyhle dvě situace jsou konfigurovatelné pomocí nastavení příslušných statických metod třídy: `get_entropy` a `weighted_choice`.

Po vytvoření instance třídy `WFCMap` stačí spustit hlavní smyčku algoritmu WFC viz obrazek 3.1. Metoda `iterate` provede kolaps a propagaci změn podle pravidel fungování algoritmu popsaných v kapitole 3. Metoda `iterate` uvnitř sebe volá privátní metody třídy, které zajišťují jednotlivé kroky algoritmu. Následně metoda `is_collapsed` zjistí, zda se podařilo všechny buňky kolapsovat právě do jednoho stavu.



Obrázek 4.1 Diagram tříd generátoru dungeonů

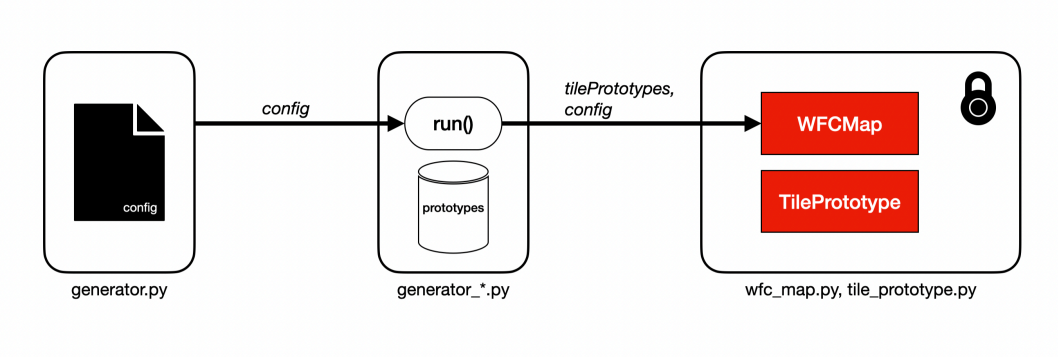
Vstupním bodem skriptu v této implementaci je soubor `generator.py` (viz diagram 4.2), ve kterém má uživatel možnost konfigurovat některé parametry skriptu. Tyto parametry jsou umístěné do slovníku `map_info`. Dalším důležitým místem je řádek s importem samotného modulu generátoru, ve kterém musí být zadefinována funkce `run`, která má jeden vstupní parametr `map_info`.

Funkce `run` provede přípravu prototypů objektů ze seznamu objektů a následně spustí samotnou generaci mapy. Implementace generátoru, resp. funkce `run`, může být upravena uživatelem pro případ generování pomocí jiné sady objektů. V kontextu dané práce se například jedná o ukázku 3.4, ve které byl použit autorem napsáný skript, ale s upravenou funkcí `run` a jinou sadou objektů.

Všechny části skriptu mají možnost používat pomocné struktury a funkce, které jsou nadefinované v modulu `utils`.

4.3 Vytvoření modulárních prvků

Nejdříve je potřeba nadefinovat, z jakých objektů bude mapa dungeonu generována. V rámci práce bylo vyzkoušeno generování map dungeonů, které jsou sestavené buď z 3D objektů, nebo z 2D obrázků.



Obrázek 4.2 Struktura skriptu generátoru dungeonů

4.3.1 2D obrázky

Generování mapy dungeonu pomocí obrázků může připomenout skládání podobné mapy ze sad, které jsou často k nalezení v deskových hrách o dungeonech. Právě taková sada je ukázána na obrázku 4.3. Jsou v ní přítomna různá čtvercová políčka, která jsou napojitelná na sebe, když jsou správně otočená. Políčka se dají rozdělit na 3 druhy. Jsou tam chodby, místnosti a ostatní. Taková sada bude využita v implementaci generátoru pro generování 2D plánků dungeonů.

Pravidla sousedství se vygenerují z přidělených socketů. Sockety se vygenerují podobně jako v příkladu z popisu fungování WFC v sekci 3.2. Pro větší kontrolu nad výstupy se druhy políček zapíšou jako `enum`, který se použije pro přidělení pravděpodobností jednotlivým políčkům. Ke 3 druhům se přidá ještě jeden druh — prázdné políčko, které zaručí generování lepších výsledků. Není vhodné, aby každé políčko v síti obsahovalo nějakou strukturu, nevypadalo by to realisticky.

V souboru `generate_wfc_land_tiles.py` byla definována funkce `run`, která připravuje vše potřebné pro vytvoření instance třídy `WFCMap`. Po vytvoření objektu mapy funkce spouští smyčku, která doběhne v okamžik, když bude vygenerována celá mapa. V případě konfliktu je mapa očištěna a proces generování začíná od začátku. Statická metoda `entropy_func` byla nastavena tak, že entropie buňky bude spočítána jako počet možných prototypů, které se dají umístit na místě buňky. Druhá statická metoda `choice_func` bude volit finální prototyp na místě buňky na základě pravděpodobnosti jednotlivých políček. Použije se náhodný výběr z pravděpodobnostního rozdělení, které je zadáno pravděpodobnostmi prototypu.

V konfiguračním slovníku `map_info` je na pozici klíče `PROBABILITIES` nadefinované pravděpodobnostní rozdělení políček v generovaném výsledku.

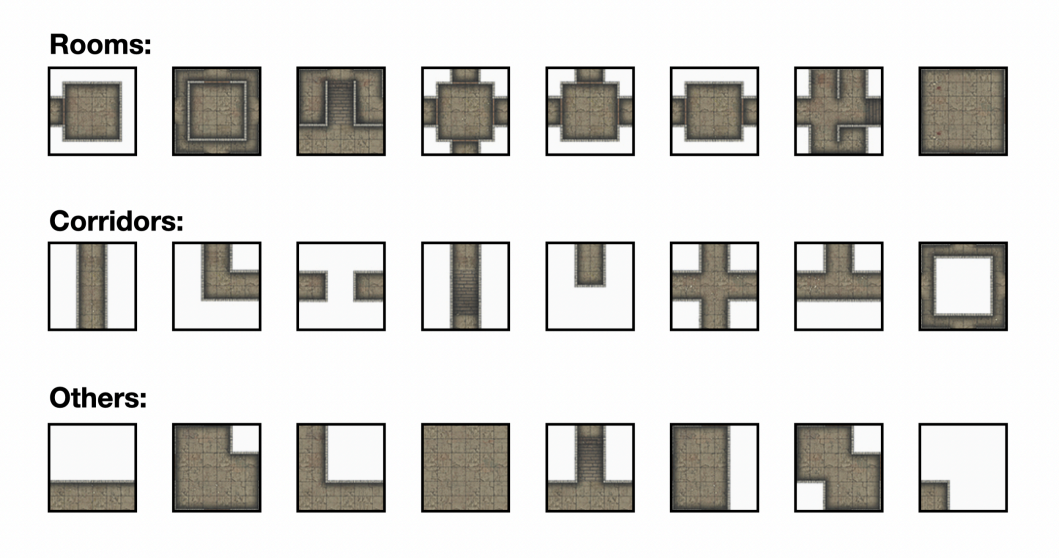
Na obrázku 4.4 jsou vidět výsledky generování map velikosti 8×8 a 15×15 s odlišnými nastaveními konfiguračních proměnných. Takovými proměnnými byly pravděpodobnosti umístění chodeb, místností, prázdných a ostatních políček.

4.3.2 3D objekty

Jako reference byl zvolen dungeon na obrázku 4.5. Na obrázku jsou dobře odlišitelné modulární prvky, navíc mapa je založena na pravidelné čtvercové síti.

V Blenderu se vymodelovalo 22 různých modulárních prvků, které mají v základu čtverec velikosti $2m \times 2m$ a jsou vysoké $5m$. Jednalo se o prvky různých druhů jako je podlaha, zeď, průsečík zdí, dveře, schody a další (viz 4.6). Navíc byl k sadě přidán prázdný prvek a to z důvodu, aby bylo umožněno generování prázdných buněk uprostřed mapy.

Následně byl vytvořen slovník prototypů 4.1, kde klíčem je název objektu v Blenderu a hodnotou další slovník, kde se nachází pravidla sousedství, které je nadefinované pomocí socketů (viz obrazek druhů socketů 4.7), a typ objektu. Typ objektu je `enum`, který říká, zda se jedná



Obrázek 4.3 Výsledky generování dungeonů z 2D obrázků

o zeď, podlahu, nebo třeba průsečík. Údaj o typu byl přidán za účelem zajištění kontroly nad generováním pomocí vah, resp. pravděpodobností, umístění jednotlivých typů.

Obdobně jako v předchozím případě 4.3.1 v souboru `generate_wfc_dungeon.py` byla definována funkce `run`, která připravuje vše potřebné pro vytvoření instance třídy `WFCMap`. Statické metody třídy `WFCMap` jsou nastavené stejným způsobem.

Důležitým zásahem do stavu mapy je umístění zdí na krajích mapy před začátkem generování za účelem toho, aby výsledek sebou představoval uzavřené prostranství. Tento zásah popisuje funkce `prepare_edges` v `generate_wfc_dungeon.py`, kde jsou do krajních políček mapy rozmístěné zdi, nebo průsečík ve tvaru písmena T. Počet takových průsečíků má vliv na členitost mapy. Čím víc průsečíku bude, tím víc zdí se může v prostředních částech mapy objevit. Stupeň členitosti je vystavena ven jako jedna z proměnných konfigurace generátoru ve slovníku `map_info` – `SEGMENTATION`. Proměnná musí být v rozsahu 0–1.

Na obrázku 4.8 jsou vidět výsledky generování map velikosti 15×15 a 30×30 s odlišnými nastaveními konfiguračních proměnných. Takovými proměnnými byly členitost, pravděpodobnost umístění zdí a podlahy.

4.4 Zhodnocení výsledků

Výsledky se dají ohodnotit ze dvou stránek — technické a estetické. Z technického hlediska se implementace zdá být povedená. Skript je relativně snadno konfigurovatelný a umožňuje použití různých sad objektů. Konfigurace také dovoluje uživateli samostatně definovat druhy objektů a nastavit jejich pravděpodobnostní rozdělení v očekávaném výsledku.

Z estetické stránky nejsou výsledky tak jednoznačné. V případě generování 2D plánků dungeonů byla sada políček dost rozmanitá a dost dobře navržená pro to, aby se daly generovat zajímavé a použitelné výsledky. Na ukázkách generování 4.4 je vidět, že některé výsledky vypadají přirozeně a po drobných úpravách by se daly použít i v produkci. Místy je vidět, že některé chodby a místnosti jsou odpojené od hlavní struktury, což znamená, že tato políčka nejsou dosažitelná.

3D varianta dungeonu má ale některé zásadní problémy. Stačí výsledky generování na obrázku 4.4 srovnat s referencí 4.5. Po srovnání se dá říct, že vygenerované výsledky jsou víc nepravidelné a ve většině nevypadají přirozeně. Bude to zdůvodněno tím, že v nich nejsou odlišitelné

Výpis kódu 4.1 Slovník prototypů 3D objektu

```
...,
'Wall_ending': {
    'sockets': {
        'pX': '3s',
        'nX': '0s',
        'pY': '0s',
        'nY': '0s'
    },
    'type': Type.FLOOR
},
'Wall_straight': {
    'sockets': {
        'pX': '3s',
        'nX': '3s',
        'pY': '0s',
        'nY': '0s'
    },
    'type': Type.WALL
},
...,
```

místnosti a chodby. Připomíná to spíše bludiště. Na druhou stranu, bludiště taky může zastávat roli dungeonu.

Další věc je opět nedosažitelnost některých míst mapy. Vygenerované dungeony nejsou celkově tak vizuálně bohaté jak reference. Je možné, že manuální úpravou povedenějších výstupů a přidáním dekorativních prvků se dá dospět k očekávané vizuální kvalitě.

Výše popsané problémy můžou být důsledkem nedostatečně dobrého návrhu sady modulárních prvků. WFC ve své osnově nic neví o širším okolí a kontextu nějaké buňky v síti v okamžik kolapsu. Ví pouze její povolené stavy. Bez větších zásahů do způsobu volby stavů v okamžik kolapsu se nedá dospět k výsledkům, které by v širokém kontextu připomínaly něco z reálného světa.

Existuje ale jedno řešení, které nebude vyžadovat zavedení pokročilejších heuristik ve volbě stavu objektu v okamžik kolapsu. Takovým řešením je zmenšení granularity sady pro generování. Čím větší je objekt v sadě, tím větší kontrolu má uživatel nad estetickou stránkou výstupu. V případě 3D objektů na obrázku 4.6 byla granularita dost vysoká (na úrovni jednotlivých kousků jedné zdi), naopak 2D plánek se sestavuje z políček na obrázku 4.3, která už jsou sama o sobě nějak zajímavá. Jsou to již sestavené místnosti a chodby.

Právě granularita je důvodem toho, proč výsledky 2D varianty vypadají více přirozeně. Potřebovaly totiž více lidské práce v etapě návrhu sady políček. Problém granularity modulárních knihoven objektů se často řeší i v jiných oborech herního vývoje, které ne vždy souvisí s proceduralním generováním obsahu. Jedná se o pokusy najít tu správnou cestu k vytvoření knihovny objektů takovým způsobem, aby výsledky jejich skládání nevypadaly repetitivně, byly vizuálně bohaté a vypadaly přirozeně.

4.4.1 Testování skriptu

Skript byl otestován dvěma druhy testů a využíval sadu políček pro generování *pixely-based* mapy terénu z ukázky 3.4. Skript pro testování je trošku upravenou verzí generátoru ze souboru `generate_wfc_land_tiles.py`, do které byly přidány funkce pro zaznamenávání výsledků jednotlivých běhů algoritmu a také pro výpis výsledků testování. Do konfiguračního slovníku

`map_info` byl přidán klíč `ROUNDS`, který říká, kolikrát by se měla generace spustit. Během testování se tedy generování spouštělo několikrát a pro různé velikosti map.

První druh testu prošel na konci každého generování všemi buňkami mapy a zkontroloval, zda jsou ve výsledku splněna pravidla sousedství, která byla uložena do objektu `TilePrototype`.

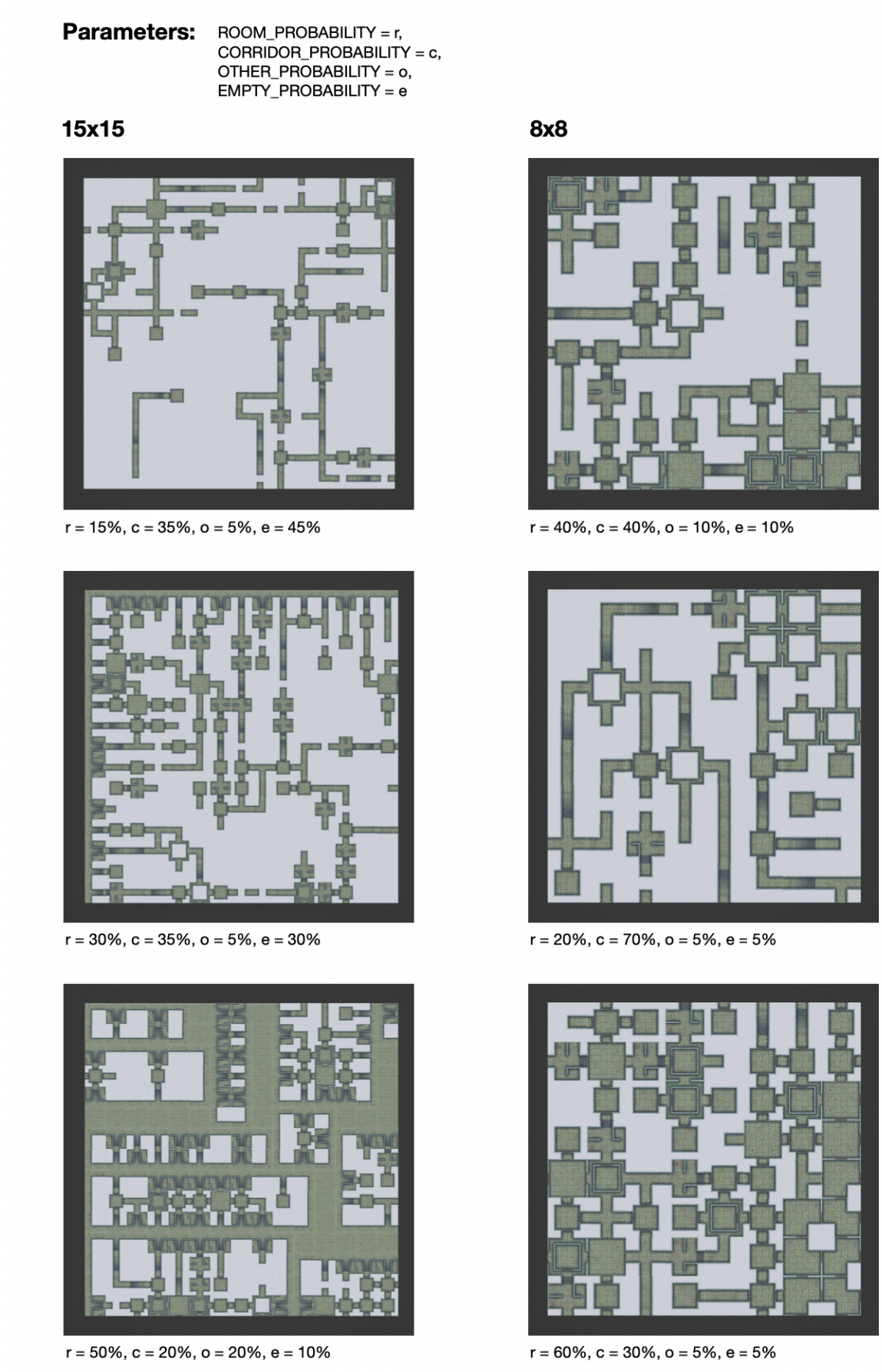
Druhý druh testu kontroloval, zda splňují výsledky generování očekávané pravděpodobnostní rozdělení druhů objektů, které se zadává v konfiguraci skriptu. Test sečte všechny výskyty jednotlivých druhů objektů ve všech bězích testování. Tyhle součty se použijí pro zprůměrování reálně naměřeného rozdělení. Průměr naměřených rozdělení by se měl lišit od očekávaného maximálně o nějakou určenou odchylku.

Na obrázku 4.9 jsou vidět výsledky testování generování map terénu pro různé konfigurace. Spuštění se lišily počtem rund generování, pravděpodobnostním rozdělením druhů políček a velikostmi map.

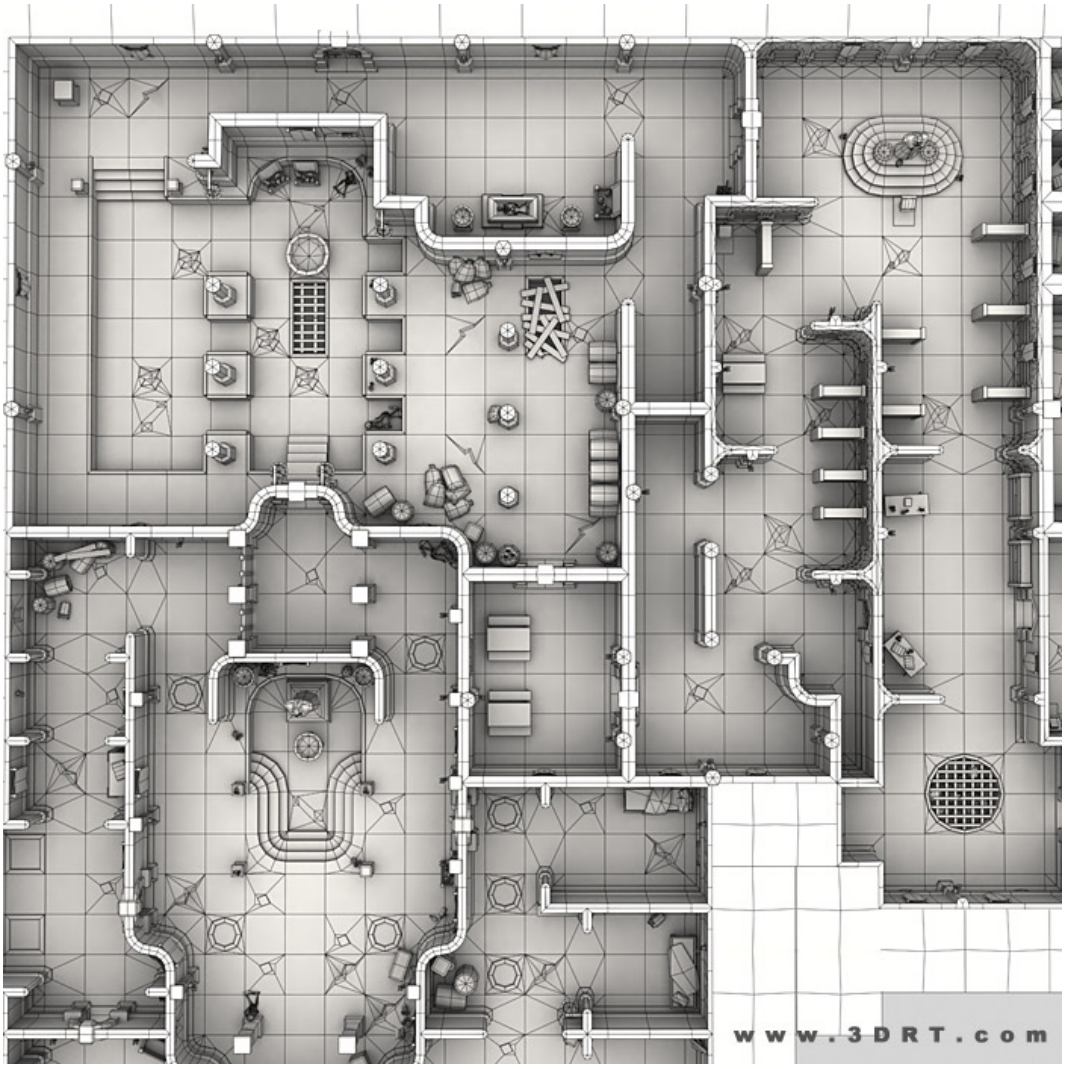
4.4.2 Splnění očekávání

Celkově se výsledky generování dostaly do rámce pragmatického očekávání. Očekávání bylo omezené tím, že se jedná o algoritmus, který nemá za cíl zkoumat širší okolí buněk a nic neví o tom, jak by mohly vypadat struktury zajímavých nebo přirozených tvarů. Vytvoření takových struktur je ponecháno na pravděpodobnosti a náhodě.

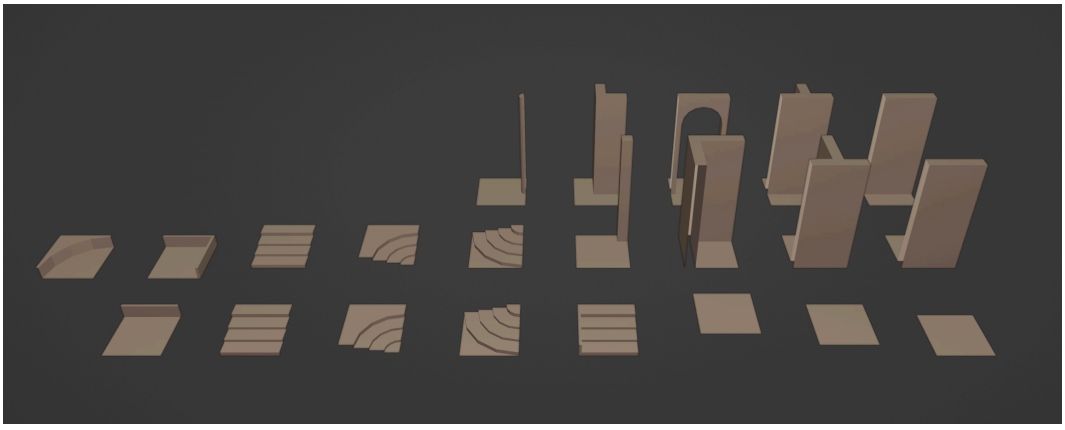
Po dost velkém zásahu do osnovy algoritmu s ponecháním jeho hlavních aspektů by se daly vytvořit generátory, které budou ve většině generovat dobré výsledky. Zase by to ale bylo za cenu snížení univerzálnosti takové implementace. Je těžké navrhnout nástroj, který bude fungovat stejně dobře pro každý případ použití.



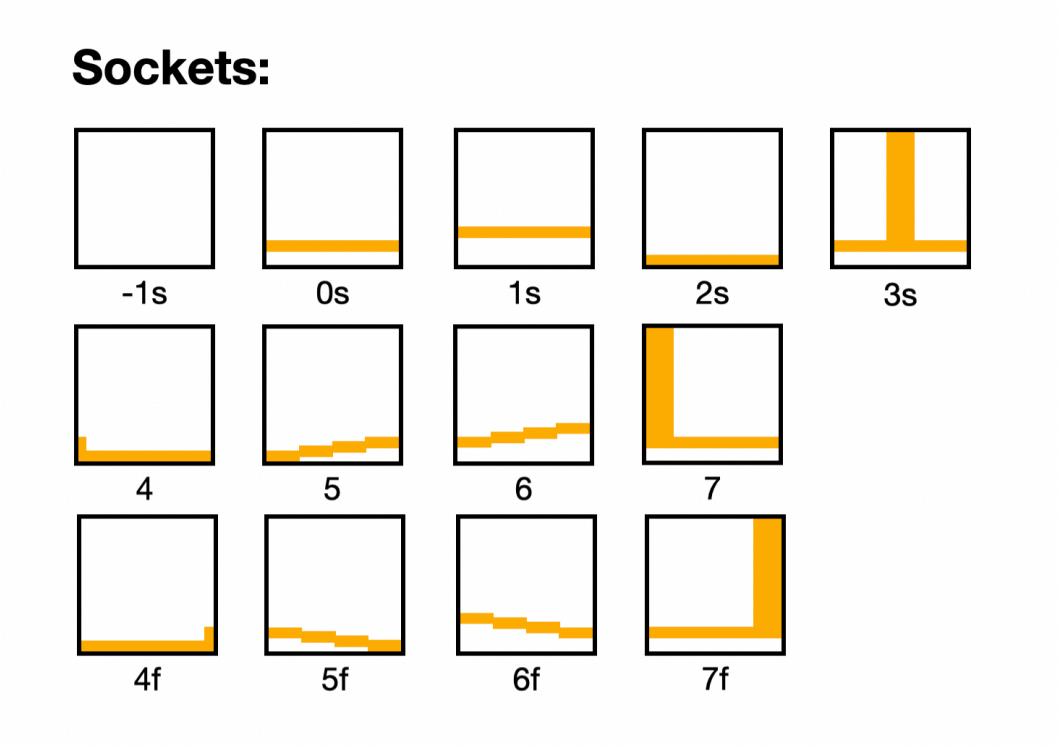
Obrázek 4.4 Sada políček pro skládaní 2D plánku dungeonu



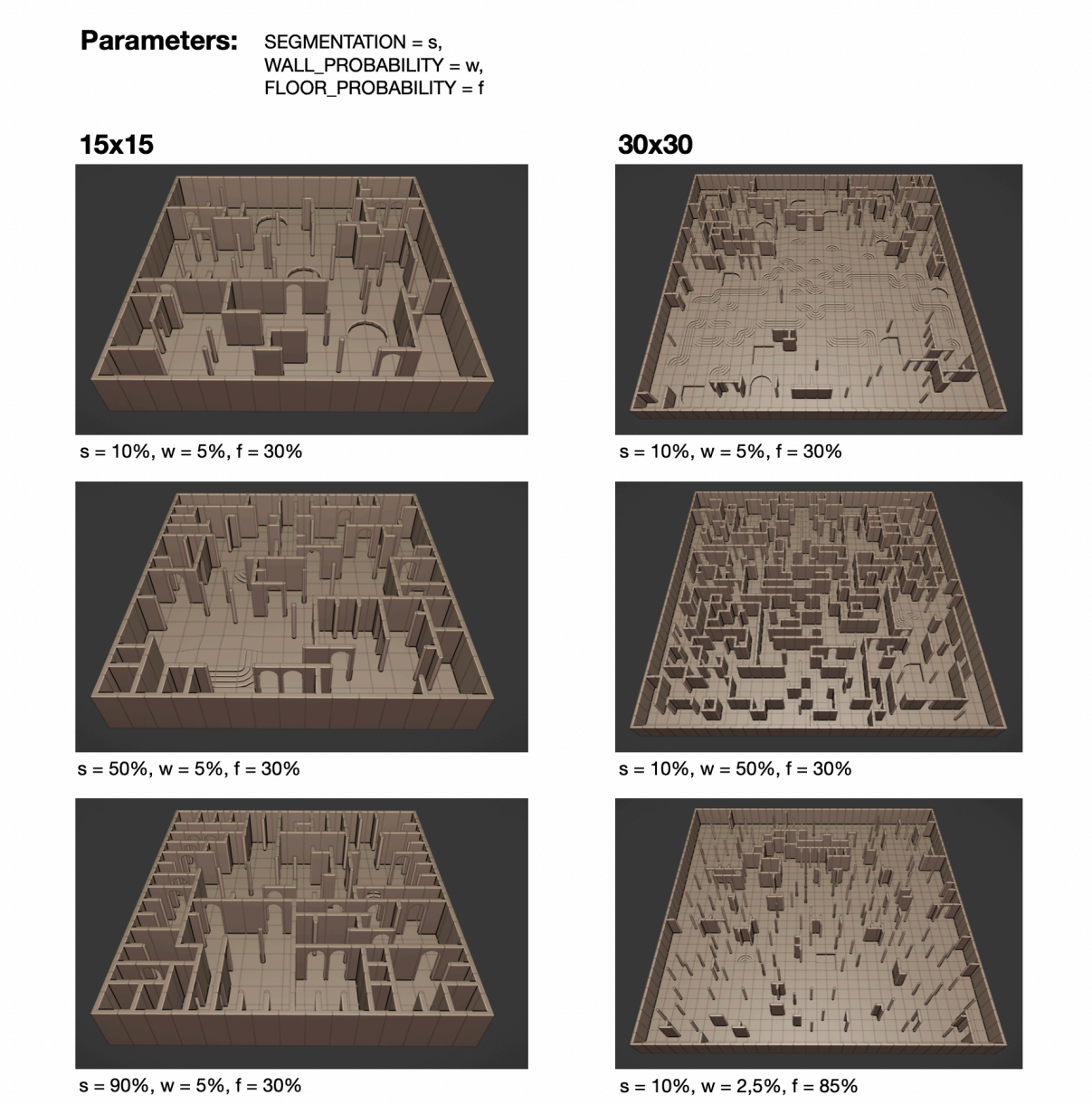
Obrázek 4.5 Referenční dungeon pro generátor pomocí 3D objektů



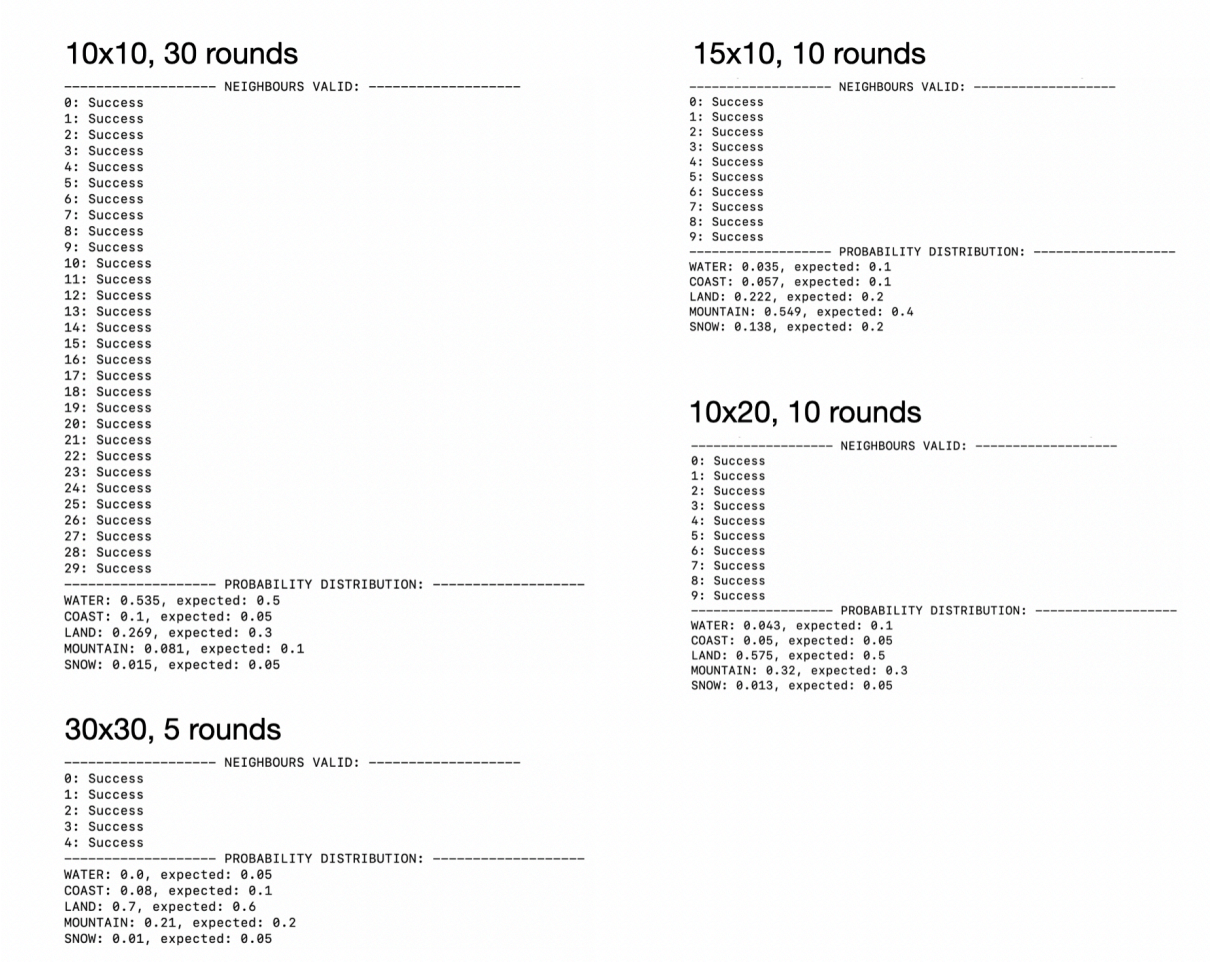
Obrázek 4.6 Modulární prvky dungeonu vytvořené v Blenderu



Obrázek 4.7 Seznam socketů 3D objektů



Obrázek 4.8 Výsledky generování dungeonů z 3D objektů



Obrázek 4.9 Výsledky testováni

Kapitola 5

Závěr

5.1 Shrnutí

Tato práce uvedla čtenáře do problematiky procedurálního generování obsahu pro videohry. Seznámila ho s důvody použití procedurálního generování a také s jeho výhodami a nevýhodami. Pro detailnější popis bylo zvoleno procedurální generování map.

Byly rozebrány příklady několika vydaných her, ve kterých si techniky procedurálního generování map našly místo buď v procesu vývoje, nebo jako jedna z klíčových herních mechanik. Bylo popsáno jak generování 2D, tak i generování 3D map v daných hrách. Kapitola 2 také srovnala jednotlivé přístupy v kontextu konkrétních her a jejich potřeb.

Jako jeden ze způsobů procedurálního generování map byl popsán algoritmus *Wave Function Collapse*. Popis obsahuje vysvětlení principu jeho fungování pro generování map ve 2D a v 3D a také příklady reálných her, ve kterých byl algoritmus použit. Implementace prototypu poukázala na to, že *WFC* má svá určitá omezení. Některá z nich, jako třeba případy kontradikce, jsou řešitelná zavedením backtrackingu. Jiná omezení řešení nemají kvůli klíčovým principům, na kterých je postaven *WFC*. Jde například o rychle rostoucí časovou složitost se zvětšením velikosti generovaných map.

Pro praktickou ukázku použití algoritmu bylo zvoleno prostředí modelovacího softwaru Blender. Pro tento nástroj byl napsán plugin ve formě skriptu, pomocí kterého byly vygenerované mapy dungeonů v rámci jednoho podlaží. Skript byl navrhnut pro univerzální použití. To je zajištěné ponecháním možnosti uživateli samostatně definovat moduly, ze kterých se mapa skládá, což dovoluje generovat nejen mapy dungeonů, ale i jiné lokace.

Skript byl úspěšně otestován testy, ve kterých se zkontrolovalo, zda jsou výsledky generování v souladu s pravidly *WFC*. Také bylo provedeno srovnání pravděpodobnostních rozdělení objektů vygenerovaných výsledků s očekávaným rozdělením, které nastavuje uživatel.

Implementace prototypu poukázala na to, že výsledky generování jsou silně závislé na kvalitě návrhu knihovny modulárních objektů a na její granularitě. Je to důsledkem toho, jak ve své podstatě algoritmus WFC funguje.

Vygenerované výsledky 3D dungeonu můžou v případě potřeby podstoupit dalším kosmetickým úpravám, ve kterých se do mapy přidají dekorativní prvky. Po úpravách jen stačí model exportovat ve zvoleném formátu a naimportovat ho do herního enginu pro použití ve vyvíjené videohře.

5.2 Budoucí rozšíření

Prvním rozšířením funkčnosti skriptu, které nás může ihned napadnout, je přidání možnosti generovat mapy ve třech dimenzích namísto dvou. To by dovolilo vytvářet lokace, které se budou skládat z několika pater, nebo také scény krajiny či města.

Dalším zlepšením může být automatizace přidělení socketů pro objekty. Místo manuálního procházení seznamu všech objektů a přidělení pro každou jeho stranu odpovídajícího socketu bude úkol ponechán na zpracování skriptem. Takové zlepšení jistě usnadní práci uživateli, kterému bude poté potřeba už jen navrhnout objekty a spustit skript bez dalších příprav.

Pro efektivnější fungování skriptu by bylo dobré také implementovat backtracking pro případy konfliktu a taky jejich ranní detekci. Taková úprava může razantně zvýšit rychlost generování pro případy velkých map.

Rozšířením, které už nesouvisí přímo s algoritmem WFC, může být automatické přidání různých dekorativních prvků pro zlepšení estetických vlastností vygenerovaných map. Z vizuálního hlediska by se také hodilo vytvoření propracovanější knihovny modulárních objektů, která by měla dobrou granularitu.

Příloha A

Návod k použití

A.1 Příprava prostředí

Pro použití skriptu je potřeba mít na svém stroji nainstalovaný Blender[1] `https://www.blender.org/download/`.

Také je potřeba přemístit na svůj stroj obsah složky `src` přiloženého média.

A.2 Spuštění

Ve složce `src` se nachází složky `scripts`, `examples` a `tests`. V `examples` je na výběr ke spuštění několik Blender souborů.

Obsah `.blend` souborů:

- `2d.blend`: generování plánku dungeonu pomocí sady 2D obrázků,
- `3d.blend`: generování mapy dungeonu pomocí navržené sady 3D objektů,
- `land.blend`: generování mapy terénu pomocí sady 5 různých políček,
- `test.blend`: testováni implementace.

Po otevření `.blend` souboru stačí přejít do layoutu `Scripting`. V něm se už nachází uložená varianta skriptu `generator.py`. Jediné, co zbývá nastavit, je proměnná `packages_path`, která je v horní částí skriptu. Proměnná musí být nastavena na absolutní cestu do přemístěné složky `src/scripts`. V případě systému Windows je potřeba escapovat zpětná lomítka. Tato proměnná se použije proto, aby prostředí Blenderu vědělo, kde hledat další soubory, ze kterých je skript sestaven.

Následně zbývá jenom zmáčknout tlačítko `Run Script` a pozorovat proces generování.

Pro spuštění generovaní od začátku je potřeba vymazat ze scény předchozí výsledek.

Mezi každým spuštěním je možné měnit parametry ve slovníku `map_info`. Jsou tak třeba nastavitelné pravděpodobnosti jednotlivých druhů objektů. Součet nastavených pravděpodobností musí být 1.

[1] Testováno na verzi 3.0

A.3 Úprava skriptu

Implementace umožňuje napsání vlastních generátorů pro jiné sady objektů. Při použití jiné sady objektů je potřeba nejdříve takovou sadu vytvořit nebo najít. Objekty v sadě musí mít čtvercový tvar a být schopné na sebe navazovat.

Ve složce `skripts` se nachází složka `templates`, kde jsou k nalezení dvě šablony — jedna pro vstupní bod skriptu a druhá pro vlastní implementaci generátoru.

Po jakékoliv úpravě souborů s implementací (ne úpravách proměnných) je potřeba restartovat Blender, jinak se změny v kódu neprojeví.

A.3.1 Úprava sady

Sada se musí umístit do scény v Blenderu do kolekce `Source`. Budou to klasické objekty Blenderu se stejnou velikostí. Počátek každého objektu se musí nacházet v jeho centru.

A.3.2 Vlastní implementace souborů generate_*.py

Jako příklad může být vzat jakýkoliv existující soubor s názvem `generate_*.py` a může být použita nabídnutá šablona. V této šabloně je potřeba definovat 4 funkce — `create_prototypes`, `set_valid_neighbours`, `entropy_func`, `choice_func`. Kromě funkcí je třeba vyplnit slovník `raw` a případně dodefinovat třídu `Type`.

A.3.3 Úprava souboru generator.py

V šabloně pro `generator.py` se musí nastavit řádek importu, aby byla naimportována správná implementace. Kromě toho se musí aktualizovat slovník `map_info` podle potřeb.

A.4 Spuštění testů

Testy se dají spustit a nastavit v souboru test.blend. Skript je spustitelný obdobně, jako bylo popsáno na začátku návodu. Parametry testování se dají nastavit přes proměnné slovníku `map_info`. `ROUNDS` nastavuje počet spuštění generování, než se zobrazí výsledky.

Pro zobrazení výsledků testování musí být otevřena příkazová řádka (Toggle System Console). V případě, že taková možnost není, je také možné spustit Blender přes příkazovou řádku. V takovém případě se výpisy testů budou objevovat v ní.

Bibliografie

1. KAŠÍK, PETR. *Videohry hraje téměř každý druhý Čech. Během pandemie se zdvojnásobil počet hráčů nad 35 let* [online]. Praha, CZECHSIGHT s.r.o. [Cit. 2022-04-24]. Dostupné z: `https://www.czechsight.cz/videohry-hraje-temer-kazdy-druhy-cech-behem-pandemie-se-zdvojnasobil-pocet-hracu-nad-35-let/`.
2. *Esport je na vzestupu. Víte ale, co to je?* [Online]. Seznam.cz a.s., 1996 [cit. 2022-04-24]. Dostupné z: `https://www.sport.cz/clanek/komercni-clanky-esport-je-na-vzestupu-vite-ale-co-to-je-2455063`.
3. *37 Biggest Open-World Video Games by Map Size* [online]. Howchoo, LLC [cit. 2022-04-24]. Dostupné z: `https://howchoo.com/gaming/biggest-video-game-maps`.
4. TOGELIUS, Julian; CHAMPANDARD, Alex J.; LANZI, Pier Luca; MATEAS, Michael; PAIVA, Ana; PREUSS, Mike; STANLEY, Kenneth O. Procedural Content Generation: Goals, Challenges and Actionable Steps. In: LUCAS, Simon M.; MATEAS, Michael; PREUSS, Mike; SPRONCK, Pieter; TOGELIUS, Julian (ed.). *Artificial and Computational Intelligence in Games*. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, sv. 6, s. 61–75. Dagstuhl Follow-Ups. ISBN 978-3-939897-62-0. ISSN 1868-8977. Dostupné z DOI: `10.4230/DFU.Vol6.12191.61`.
5. KAHOUN, Martin. *Realtime library for procedural generation and rendering of terrains*. Prague, 2013. MASTER THESIS. Charles University in Prague.
6. *Title Sequence* [online]. State of Delaware, U.S.A.: Fandom [cit. 2022-04-24]. Dostupné z: `https://bindingofisaacrebirth.fandom.com/wiki/Title_Sequence`.
7. *Dungeon Generation in Binding of Isaac* [online]. 2008 [cit. 2022-04-24]. Dostupné z: `https://www.boristhebxrave.com/2020/09/12/dungeon-generation-in-binding-of-isaac`.
8. *The Dwarf Fortress*. 2002. Dostupné také z: `https://bay12games.com/dwarves/index.html`.
9. HALPERIN, Xray. Marvel's Spider-Man: Miles Morales Procedural Tools for PlayStation 5 Content Authoring. *SIGGRAPH Asia 2021 Technical Communications*. 2021-12-14, s. 1–4. ISBN 9781450390736. Dostupné z DOI: `10.1145/3478512.3488594`.
10. *Introduction to Houdini* [online]. Toronto Ontario, Canada: SideFX, 1987 [cit. 2022-04-24]. Dostupné z: `https://www.sidefx.com/docs/houdini/basics/intro.html`.
11. SANTIAGO, David. *Marvel's Spider-Man, meet Houdini* [online]. Toronto Ontario, Canada: SideFX, 1987 [cit. 2022-04-24]. Dostupné z: `https://www.sidefx.com/community/gdc-2019-presentations/`.
12. TISLICKÝ, Jan. *VĚNNÁ MĚSTA ČESKÝCH KRÁLOVEN II. - ÚPRAVA TEXTUR*. Praha, 2020.

13. TUMULKA, Roderich. On spontaneous wave function collapse and quantum field theory. *PROCEEDINGS OF THE ROYAL SOCIETY A*. 2006. ISSN 1364-5021. Dostupné z DOI: https://doi.org/10.1098/rspa.2005.1636.

14. MØLLER, Tobias Nordvig; BILLESKOV, Jonas; PALAMAS, George. Expanding Wave Function Collapse with Growing Grids for Procedural Map Generation. In: *International Conference on the Foundations of Digital Games* [online]. New York, NY, USA: ACM, 2020-09-15, s. 1–4 [cit. 2022-04-24]. ISBN 9781450388078. Dostupné z DOI: 10.1145/3402942.3402987.

15. KIM, Hwanhee; LEE, Seongtaek; LEE, Hyundong; HAHN, Teasung; KANG, Shinjin. Automatic Generation of Game Content using a Graph-based Wave Function Collapse Algorithm. In: *2019 IEEE Conference on Games (CoG)* [online]. IEEE, 2019, s. 1–4 [cit. 2022-04-24]. ISBN 978-1-7281-1884-0. Dostupné z DOI: 10.1109/CIG.2019.8848019.

16. SANDHU, Arunpreet; CHEN, Zeyuan; MCCOY, Joshua. Enhancing wave function collapse with design-level constraints. In: *Proceedings of the 14th International Conference on the Foundations of Digital Games* [online]. New York, NY, USA: ACM, 2019-08-26, s. 1–9 [cit. 2022-04-24]. ISBN 9781450372176. Dostupné z DOI: 10.1145/3337722.3337752.

17. NUNES, Nuno J.; MA, Lizhuang; WANG, Meili; CORREIA, Nuno; PAN, Zhigeng. *Entertainment Computing – ICEC 2020* [online]. Cham: Springer International Publishing, 2020 [cit. 2022-04-24]. ISBN 978-3-030-65735-2. Dostupné z DOI: 10.1007/978-3-030-65736-9.

18. *Bad North: Press Kit*. Plausible Concept, [b.r.]. Dostupné také z: https://www.badnorth.com/press-kit.

19. STÅLBERG, Oskar. *Developing The Bad North Look*. Konsoll, 2012. Dostupné také z: https://konsoll.org/archive/2018.

20. STÅLBERG, Oskar. *The Story of Townscaper*. Konsoll, 2012. Dostupné také z: https://konsoll.org/archive/2021.

21. *Cambridge Dictionary*. Cambridge University Press, 1999. Dostupné také z: https://dictionary.cambridge.org/dictionary/english/dungeon.

22. *Cyber Definitions*. [B.r.]. Dostupné také z: https://www.cyberdefinitions.com/definitions/DUNGEON.html.

23. *Blender: About*. The Blender Foundation, 2002. Dostupné také z: https://www.blender.org/about/.

Obsah přiloženého média

```
readme.txt ..... stručný popis obsahu média
src
    scripts ..... zdrojové kódy implementace
    examples ..... blend soubory ukázek
text ..... text práce
    thesis.pdf ..... text práce ve formátu PDF
```