FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Parametricky vygenerovaný 3D model hradu
Student:	Petra Svíčková
Vedoucí:	Ing. Radek Richtr, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2020/21

Pokyny pro vypracování

Cílem práce je pomocí sady pravidel vytvořit parametrizovatelný a randomizovaný průřez 3D modelem hradu, jež bude vhodný například jako prostředí pro jednoduchou hru. Součástí práce je i vytvoření podkladů (modely místností, nábytku atp.) o vhodné detailnosti.
1) Proveďte krátkou rešerši problematiky procedurálního generování prostředí. Zaměřte se na obdobné projekty a použití.
2) Proveďte krátkou rešerši zvoleného typového prostředí (hrady, zámky atp.).
3) Navrhněte sadu pravidel zvolené (model) vhodné pro vytváření prostředí hradu na úrovni jednotlivých místností.
4) Vytvořte prototyp parametrizovatelného pluginu do Blenderu, jež dle navržené sady pravidel bude schopen vygenerovat průřez 3D modelem hradu. Samotný model pak bude vhodně editovatelný (např. přidávání nových místností, zvolení jen podmnožiny pravidel atp.).
5) Na hradu demonstrujte a diskutujte vlastnosti zvoleného modelu v závislosti na bodu 2.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 28. března 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Parametricky vygenerovaný 3D model hradu

Petra Svíčková

Katedra softwarového inženýrství
Vedoucí práce: Ing. Radek Richtr, Ph.D.

15. května 2019

Poděkování

Na tomto místě bych ráda poděkovala vedoucímu své práce Ing. Radku Richtrovi, PhD. za jeho vedení, pomoc a podporu po celou dobu psaní této bakalářské práce. Dále bych chtěla poděkovat panu Ing. Radomíru Poláchovi za velmi podnětnou konzultaci v začátku tvorby. Mé díky patří taktéž Ing. Elišce Šestákové za psychickou podporu a konzultace ohledně psaní odborné práce. A nemohu opomenout též své drahé – maminku, sestru a přítele – kterým děkuji za podporu nejen v posledních měsících, ale po celou dobu studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o etické 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. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2019 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Svíčková, Petra. *Parametricky vygenerovaný 3D model hradu.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Procedurální generování obsahu je stále se rozvíjející oblast, která již teď nabízí množství způsobů, jak tvořit obsah do her i filmů. Tato práce se zabývá představením základních metod generování obsahu obecně, stejně jako tvorbou generátoru náhodného rozmístění místností v hradu jako pluginu do programu Blender. Zvolenou metodou je programování s omezujícími podmínkami (ASP). Součástí práce je též návrh možných pravidel pro umístění místností.

Klíčová slova procedurální generování, programování omezujícími podmínkami, množina dotazů, hrady

Abstract

Procedural Content Generation is constantly developing field that already offers a number of ways to create content for games and movies. This thesis deals with the introduction of basic methods of content generation, as well as the development of a random room arrangement generator in the castle as a plugin for Blender software. The selected method is Answer set programming (ASP). Part of this thesis is also a draft of rules for generating room locations.

Keywords procedural generation, constraint logic programming, answer set, castles

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Herní průmysl je stále mohutnější a oblíbenější oblastí, což dokazuje nejen počet nově vydaných her, ale i vznikající obory na vysokých školách a internetové kurzy – obojí hojně navštěvované. Hry se stávají stále komplexnějšími, graficky kvalitnějšími, ale taktéž je jejich vývoj díky rozvíjejícím se technologiím a programům pomáhajícím s tvorbou snazší a i pro jednoho člověka či menší tým je již možné vytvořit něco, co dříve bylo doménou jen obřích týmů.

Jednou z oblíbených možností částečného usnadnění práce je i procedurální generování, které umožňuje tvorbu rozsáhlých světů s mnoha možnostmi, ať už v podobě různých prostředí, zbraní, úkolů, nebo třeba i hudby a vegetace, avšak bez nutnosti každý jednotlivý kus vytvářet ručně. Jde tedy o úsporu času i financí nejen pro velká studia, ale i pro nezávislé vývojáře (tvořící bez finanční podpory vydavatele). Oblíbenost tohoto přístupu k tvorbě her lze vypozorovat v mnoha statistikách – např. distribuční platforma *Steam* nabízí okolo 350 her s označením „procedural generation“ (aneb „s procedurální tvorbou prostředí“), z toho 100 jich bylo vydáno jen v roce 2018 a první hra s tímto označením je z roku 2008. (Nutno podotknout, že štítky jsou přidávány uživateli, nemusí tedy jít o skutečná čísla a lze předpokládat, že jich je více, neboť procedurální generování mohlo být využito jen při vývoji hry a hráč toto nemusí na výsledku poznat.)

Jde však o přístup, který ač je často vychvalován, s sebou nese taktéž rizika. Ta velká studia mnohdy nechtějí pokoušet a zůstávají u ověřených nástrojů pro generování. Nové možnosti jsou tedy objevovány hlavně nezávislými vývojáři a nadšenci. Na distribuční platformě *Steam* lze mezi celkovými 350 hrami se štítkem „procedural generation“ (aneb „s procedurální tvorbou prostředí“) najít pouze 14 titulů, které nemají štítek „nezávislé“ a jde tedy pravděpodobně o hry závislých studií.

Ač jsou hry nezávislých vývojářů často spojovány s nízkou kvalitou, nemusí být nutně vždy nedokonalým začátkem nebo podivným experimentem, který ihned zapadne mezi tituly velkých herních studií. Ukázkovým příkladem nechť

je velmi oblíbená hra jednoho člověka *Minecraft (2011)*, která je založena na nekonečném procedurálně generovaném terénu. Další nezávislé hry, které procedurálně generují některou část obsahu, jsou např. *Dead Cells (2018)* (v době psaní této práce nejprodávanější hra s označením „procedural generation“ na platformě *Steam*) a ve spojitosti s procedurálním generováním velmi oblíbená hra *Spelunky (2008)*.

Cíle práce

Cílem teoretické části této práce je stručně uvést do vývoje a podoby hradů a seznámit se základními principy procedurálního generování obsahu a jeho metodami, nasměrovat případné další zájemce o tuto oblast v dalším studiu a zanalyzovat současná řešení použitá ve vybraných projektech.

Praktická část si klade za cíl návrh pravidel vhodných pro rozložení místností hradu a vytvoření prototypu parametrizovatelného a náhodného generátoru hradu jako pluginu do modelovacího programu *Blender*. Prototyp umožní uživateli vygenerovat jednoduchý hrad, případně dodat vlastní modely místností a nadefinovat pravidla pro každou z nich. Součástí práce budou i vlastnoručně vytvořené modely místností.

Struktura práce

Text této práce bude dále členěn následovně: v kapitole 1 budou představeny různé pohledy na procedurální generování, zavedeny pojmy a popsána obecná problematika tohoto přístupu k tvorbě. Dále budou stručně zmíněny některé z metod a vybrané projekty využívající procedurální generování. Kapitola 2 seznámí se stručnou historií vývoje hradů a zámků a popíše jejich vnitřní uspořádání pro pozdější analýzu. Kapitola 3 pak představí existující řešení a zanalyzuje metody vhodné pro generátor interiéru budov, popíše vybranou metodu a shrne typy místností stavěné ve hradech. V kapitole 4 budou definovány místnosti a pravidla vhodná pro generátor interiéru hradu. V kapitole 5 bude představena konkrétní implementace zvolené metody a prototyp generátoru. Kapitola 6 připraví materiály k testování generátoru.

Kapitola 1

Procedurální generování obsahu

„Procedurální generování je mocný nástroj a skvělý způsob,
jak zničit vlastní herní design.“
— z knihy *Procedural Generation in Game Design, 2017* [1]

Tato kapitola začíná sekcí 1.1, která se věnuje tématu procedurálního generování obsahu obecně, seznamuje s klasifikací dřívějších prací a zmiňuje výhody a nevýhody tohoto přístupu. Je následována sekcí 1.2 představující základní metody procedurálního generování a sekcí 1.3 s ukázkou některých her a programů využívajících tento přístup tvorby obsahu.

1.1 Úvod do problematiky

Stejně jako mnoho jiných pojmů v počítačové grafice či v oblasti návrhu her nemá ani pojem „procedurální generování“ jednu jedinou správnou definici, se kterou by souhlasili všichni. Příkladem mohou být následující definice Doulla[1] [2] (viz definice 1.1) a Togelia[2] [3, kap. 1] (viz definice 1.2):

Definice 1.1. *„Procedurální generování obsahu je programové generování herního obsahu pomocí náhodného, či pseudonáhodného procesu, jehož výsledkem je nepředvídatelný rozsah možných herních prostředí.“*

Definice 1.2. *„Procedurální generování obsahu je algoritmická tvorba obsahu (např. herního) s omezeným nebo nepřímým vlivem uživatele.“*

[1] Andrew Doull, vývojář her *UnAngband* a *UnBrogue* a deskových her, zakladatel blogu *Ascii Dreams: A roguelike developer's diary* a stránky *Procedural Content Generation Wiki.*

[2] Julian Togelius, vědec zabývající se umělou inteligencí a počítačovými hrami, publikující články od roku 2004 v množství blížícím se dvěma stům. Viz `http://julian.togelius.com/`

Zatímco první definice určuje náhodnost jako povinnou součást tohoto přístupu, druhá tuto podmínku nezavádí, ani pojem nevztahuje jen k hernímu obsahu. Nicméně obě souhlasí v tom, že procedurální generování obsahu je činnost počítače, ze které vzniknou výsledky, jež je možné využít ve hrách.

Ani v označení produktu generátoru není terminologie zcela daná. Togelius [3] výsledek pojmenovává jako „artefakt“, v diskuzi [4] je označován zkratkou PGC (*procedurally-generated content*) a Grinblat v [1, kap. 19] označuje výsledek modulárního přístupu (skládání výsledku z menších, předem předpřipravených částí) jako „gestalt“.

Při studiu nejen odborné literatury je možné se setkat s několika snadno zaměnitelnými pojmy, které se však v některých zdrojích zcela oddělují.

Procedurální generování (PG) je dle [5] proces, při kterém je procedurálními metodami tvořen obsah do hry, který však nijak zásadně neovlivňuje samotnou hratelnost. Např. textury, hudba, animace atd.

Procedurální generování obsahu (PCG) je naopak dle [5] generování obsahu, který přímo ovlivní hratelnost, tedy např. generování levelů (jejichž poskládání ovlivní samotný průběh hrou), příběh, statistiky u zbraní, jejich síla atd. Někdy lze narazit i na zkratku PCG-G (Procedural Content Generation for Games) [6] označující stejnou oblast.

Zatímco Doull [2] se hlásí k přísnému oddělení obou pojmů, Togelius [3] a Mateas [7] je sjednocují do jednoho. Označují PCG za techniku herního designu, který vytváří obsah pomocí zautomatizovaného procesu, a nerozlišují mezi tím, zda vytvořené artefakty mají vliv na hratelnost, či ne. Zatímco pro jednu hru může být hudba pouze doplněním atmosféry, pro jinou může jít o důležitý prvek, na kterém je hra postavena.

Wiggill[3] v diskuzi o terminologii [4] striktně rozlišuje, zda hra sama za svého běhu obsah tvoří, nebo vybírá výsledky vytvořené dříve, ve fázi vývoje hry (např. z nějaké databáze). V prvním případě ji zařazuje mezi hry s PCG přístupem, ve druhém nikoliv. Podobné rozdělení je i v [3, kap. 1], kde je za PCG považován pouze software sám výsledky generující.

Procedurální generování může být využito různě, od náhodného výběru zbraně přes zvuk ovlivněný akcí hráče až po odlišné úrovně. Jako známější příklady použití PCG uvádí [8] generování dungeonů v *Rogue (1980)*, map v herní sérii *Civilization*, zbraní v *Borderlands 2 (2012)* a vegetace pomocí programu *SpeedTree (2002)*. Ani rozdělení oblastí, které lze generovat, však není jednotné a lze najít několik různých pohledů na rozdělení herního obsahu.

[3] Nick Wiggill, přispěvatel známý jako Engineer, herní vývojář na volné noze s několikaletou praxí, zaměřený na procedurální generování světa, a mentor pro nové vývojáře.

Hendrikx v článku [6] rozděluje generovatelný obsah na šest hlavních částí, kdy každá část je poskládána z částí předchozích:

- základní prvky – základní kameny hry, tedy textury, zvuk, vegetace, budovy, chování objektů ve scéně, živly...
- prostor – vnitřní, vnější, vodní části jako řeky a jezera,
- systémy – ekosystém (obsahující evoluci, potravní řetězce apod.), silniční síť, městské prostředí a chování entit, tzv. NPC (Nonplayable character – postava obývající herní svět, ale neovládaná hráčem – např. obchodníci, obyvatelé města apod.),
- scénáře – hlavolamy, obrázkové scénáře (tzv. *storyboard*), příběh, úrovně,
- design hry – pravidla a cíle, např. generátor šachů [9],
- odvozený obsah – tabule výsledků hráčů, novinky a zpravodajské vysílání ve hře založené na aktivitách hráče.

Doull v [2] představuje jiný systém rozdělení způsobů, jak se může PCG objevit ve hrách:

- náhodné generování úrovní za běhu programu,
- design obsahu úrovně,
- dynamická generace světa,
- tvorba herních entit,
- obsah vedený uživatelem,
- dynamické systémy,
- procedurální rébusy a generace příběhu.

Klíčovou vlastností kteréhokoliv generovaného obsahu je samozřejmě být vhodný ať už z pohledu hratelnosti, nebo požadovaného stylu – level tedy musí být možno dokončit, zbraň použít, hru vyhrát.

Výhody procedurálně generovaného obsahu

Výhody PCG lze shrnout do následujících bodů (některé nemusí souhlasit s definicemi výše, nicméně jsou zmiňované různými zdroji).

- Znovuhratelnost [10, 11] – je nejčastěji zmiňovanou výhodou PCG.
- Komprese [12] – snížení paměťové náročnosti (viz hra *Elite (1984)*).

- Rozmanitost [12] – roste touha po větších a složitějších prostředích, ale zůstává příliš vysoká cena ručně tvořeného obsahu. Mnoho herních AAA titulů[4] využívá program *SpeedTree (2002)* pro zaplnění světa různorodou vegetací bez nutnosti modelovat každý odlišný strom či rostlinu.
- Nekonečné generování [12] – hra může za běhu generovat nekonečný svět, příběh či terén.
- Obsah upravitelný parametry [10] – snadná úprava obtížnosti hry či vzhledu artefaktu na základě parametrů dodaných generátoru.
- Základ pro další lidskou tvorbu [12] – algoritmy mohou vytvořit pouhý základ, který bude následně ručně upraven člověkem, a tedy usnadnit práci při opakujících se činnostech, aby se designér mohl soustředit na zajímavější části.

Náhodnost

Zdroje se značně odlišují v názoru, zda by PCG mělo být náhodné, či nikoliv (viz definice 1.1 a 1.2). S přikloněním k té obecnější od Togelia, která nepřikazuje náhodnost jako součást generátoru, lze generátory rozdělit na dvě kategorie:

- stochastické[5],
- deterministické[6].

Stochastický přístup ke generování je podle J. Bycera[7] [13] evolucí obyčejného náhodného generování, které staví pouze na náhodném výběru z omezené množiny artefaktů (např. věcí z truhly) a jehož možné výsledky je uživatel schopný – za určitý čas – vidět všechny. Procedurální generování vidí jako mocnější nástroj, kterému není dána pouze množina výsledných artefaktů, ze kterých vybírat, ale algoritmus, který sám takové artefakty dokáže vytvořit (a mnohdy ve větším množství).

Stochastické generátory tedy při stejných parametrech vyprodukují odlišné artefakty. Dle článku [8] by však neměly být za PCG považovány generátory, které kolekci herních elementů rozmístí po ploše bez jakékoliv podrobnější struktury či omezení, protože takový obsah by mohl být zcela nehratelný, a tvrdí, že proto převážná většina existujících generátorů obsahuje v určité míře opatření zajišťující hratelnost a definující pravidla (např. že v bludišti musí existovat alespoň jedna cesta ven). Snižují množství výsledků pomocí

[4] neformální pojem pro hry s velkorozpočtovou produkcí

[5] stochasticita – vlastnost postrádat jakýkoliv předvídatelný řád

[6] determinismus – při konkrétním vstupu proběhne vždy stejná sekvence stavů a vznikne stejný výsledek; vlastnost procesu, jehož každý stav je určen předcházejícím

[7] Josh Bycer – více než 7 let výzkumu a tvorby pro game design.

různých omezení, ale se zachováním těchto omezení může být výsledek náhodný. Takto implementuje PCG většina známých generátorů [3].

Definice 1.1 obsahuje podmínku náhodného či pseudonáhodného generování. Podle [3] PCG však prvek náhody nutně nemusí obsahovat. V takovém případě je připuštěna i možnost tzv. deterministických generátorů.

Deterministické generátory vždy při stejných vstupních parametrech vygenerují stejný výsledek. Takový přístup je tedy vhodný zejména pro kompresi dat [12], kdy se složitá data vygenerují, až když jsou třeba, a není tedy nutné uchovávat stovky a tisíce map, zbraní, textur, příběhů, úkolů, skladeb, postav a podobně. Ukázkovými příklady jsou *Elite (1984)* a *.kkrieger (2004)*, kde v prvním případě hra obsahovala mnoho solárních systémů a v druhém byla její celková velikost pouhých 96 kB. O první jmenované bude více pojednáno v části 1.3.

Nevýhody procedurálně generovaného obsahu

PCG může vypadat jako ideální přístup k tvorbě obsahu, nicméně ani zde nechybí rizika při jeho použití. Ač nabízí mnoho, Bycer [13] zmiňuje i oblasti, ve kterých (zatím) PCG nevyniká a které mohou být důvodem, proč tento přístup nepoužít:

Limitované základy – čím složitější možnosti má hráč při interakci s aplikací, tím složitější je tvorba algoritmu, který by dokázal na vše reagovat. Právě z tohoto důvodu hry často generují spíše prostředí a předměty než herní mechaniky. Příkladem mohou být tahové strategie s generovanými mapami – hráč se již během jedné hry naučí postup, ví „jak na to“ a nemusí své chování výrazně měnit.

Nemožnost zpracovat unikátní herní mechaniky – čím jsou herní mechaniky zajímavější a složitější, tím náročnější až nemožné je vygenerovat úroveň, která využije jejich potenciál. Příkladem jsou např. hry *Portal*[8] *(2007)* a *Portal 2 (2011)*.

Masová produkce vs. ruční výroba – obsah procedurálně generovaný je mnohdy poznat (rozpoznatelné vzory, prázdná místa, chyby). Může být taktéž považován za nudný, opakující se a nezajímavý oproti ručně tvořenému obsahu. Bycer zmiňuje sérii her *Dark Souls (2011–2015)* jako zástupce impozantního ručního designu. Každá oblast je pečlivě navržena, každý detail promyšlen a scény dechberoucí. Takového výsledku je těžké dosáhnout procedurálním generátorem, nicméně Bycer netvrdí, že by neexistovaly opravdu dobře navržené procedurálně generované hry. Uvádí např. *XCOM 2 (2016)* a *Invisible, Inc. (2015)*.

[8] *Portal* je logická hra z prvního pohledu, kde hráč prochází úrovněmi za pomoci dvou portálů, navzájem propojených, které může umístit skoro kdekoliv (a např. využít rychlost pádu do jednoho k přeskočení propasti vyletěním z druhého).

Kromě těchto oblastí, nad kterými je třeba se před využitím PCG zamyslet, existují další rizika, která je třeba mít na paměti a vhodným způsobem je řešit. Dále v textu budou popsána tyto rizika:

- chyby v artefaktu [3, 8],
- složitost algoritmu [2],
- možné neetické výsledky [1, kap. 5].

Chyby v artefaktu
Náhodný algoritmus může snadno vést k chybám v designu. Artefakt ze zcela náhodného generátoru může být nehratelný [8] – bludiště bez cesty mezi startovací pozicí a východem, nevyřešitelné rébusy, nesmyslný příběh, prostor, ze kterého se nedá dostat ven, data, která rozbijí hru. Řešením je artefakt otestovat, zda je vhodný (splňuje alespoň minimální požadavky pro to, aby byl správný), nebo se už během jeho tvorby ujistit, že za žádnou cenu nebude vytvořen špatně.

Dále zdroj [3] rozlišuje, zda si výsledný projekt (např. hra) může takové výsledky dovolit. Pro bludiště je nepřípustné, aby některé nešly projít. Naopak např. ve hrách, kde je možné špatnou zbraň s nevhodnými statistikami vyměnit za jinou, vyhnout se budově bez vchodu apod. je určitá míra těchto „chybných“ artefaktů přípustná. Je však na zvážení tvůrců, v jaké míře. Pro realistickou hru je i nesprávně postavený dům vadou, pro jinou hru s menšími nároky na realističnost může být takový dům i příjemnou zvláštností prostředí.

Složitost algoritmu

Metody, kterým bude věnována další podkapitola, jsou jinak náročné a hodí se k různým cílům. Složitost jednotlivých metod bohužel často závisí na reprezentaci dat, proto volba konkrétní z nich není jistotou a záleží na algoritmech k tomu použitých.

Při návrhu je nutno při volbě způsobu generování zvážit i to, kdy bude generování probíhat – zda za běhu programu (*runtime*) nebo během vývoje hry (*offline*) [2, 3].

V případě generování obsahu *offline*, během vývoje, nemusí být časově náročný algoritmus překážkou, protože je na to zpravidla více času. Vážnější situace nastává, pokud takové generování má probíhat za běhu, před každým vykreslením nové úrovně, nebo přímo v čase hry např. na okrajích obrazovky při pohybu terénem. V takovém případě by měl být algoritmus rychlý, neboť by jinak nepříjemně ovlivnil herní zážitek, a to ať už je zvolen kterýkoliv přístup zmíněný v následující části o metodách.

Neetické výsledky

Zatímco ohromné množství obsahu, které je generátor schopen vytvořit, může být považováno za velkou výhodu, Cook ve své kapitole v knize [1, kap. 5], kde rozebírá etický důsledek procedurálních generátorů, považuje právě tuto schopnost generátorů za riskantní. Při vědomém definování pravidel (např. velikost dungeonu, která ovlivní výslednou složitost či dobu hraní) můžou být nevědomky definována další pravidla, která vyústí v chyby nebo výsledek, který není vhodný.

Jako příklad uvádí generování náhrobků se jménem a rokem úmrtí pro hřbitov ve hře. Generátor náhodně vybere ze seznamu jmen a dodá náhodný rok. Je přidána možnost pohřbených milenců, tedy rok a dvě jména. V takovém případě je důležité i to, jak budou data prezentována. Jeden ze způsobů je vytvořit dva seznamy jmen, na jednom jména ženská, na druhém mužská. Druhým způsobem je jeden seznam, který se může zdát jednodušší pro implementaci a s rychlejším přístupem k výběru. Ať je zvolen kterýkoliv ze způsobů, ovlivní následnou informaci, kterou výsledný hřbitov sdělí o sexualitě v daném herním světě. V případě dvou seznamů se nikdy neobjeví dvě ženská, ani dvě mužská jména, a tím je řečeno, že v daném světě existují pouze heterosexuální vztahy. V případě pouze jednoho seznamu to tak není.

Nejde o to, zda je tato informace ze společenského hlediska vhodná, či ne (pokud nemá být hra vydána i v zemi, kde daná myšlenka není přípustná a její existence by mohla způsobit zákaz hry v konkrétní zemi), ale o to, aby si tvůrce generátoru byl vědom informace, kterou předává, a že i taková drobnost může mít vliv, ať už podvědomý, nebo zcela znatelný, na reprezentaci herního světa a na samotné hráče, např. děti.

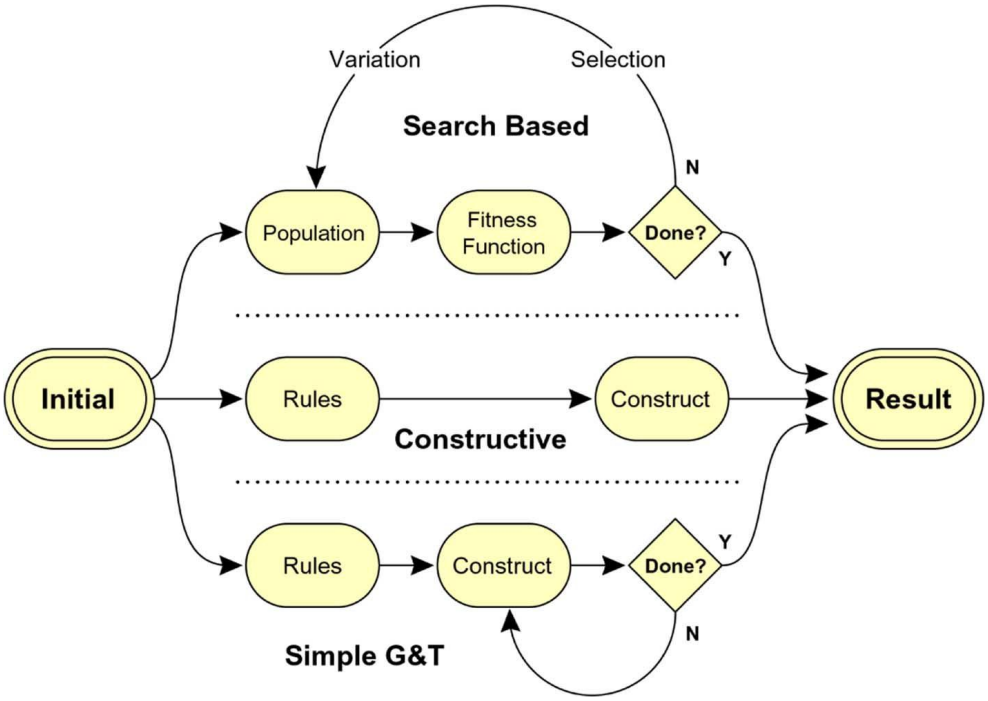
1.2 Metody procedurálního generování

Ač je procedurální generování stále oblíbenější a mluví se o něm čím dál více i v souvislosti s hrami, množství materiálů pro tuto oblast je třeba hledat mimo herní obor, např. v oborech počítačová grafika, zpracování obrazu, umělá inteligence, lingvistika atd. [6]

Metod pro generování obsahu lze nalézt mnoho, od jednoduchých v podobě zcela náhodného generování čísel, přes simulace fyzikálních jevů až po složitější s pomocí umělé inteligence a neuronových sítí.

Průzkum [6] z roku 2013 definuje šest skupin metod procedurálního generování herního obsahu. Jsou jimi:

- pseudonáhodné generování čísel – např. *Perlinův šum*,
- generativní gramatiky – např. *L-systémy, dělící gramatiky, tvarové gramatiky*,
- filtrování obrazu – např. *binární morfologie a konvoluční filtry*,



Obrázek 1.1: Tři přístupy k procedurálnímu generování obsahu: (odshora) na bázi vyhledávání, konstrukční a jednoduchý generuj-a-testuj. Ne všechny vyhledávající algoritmy vhodné pro PCG udržují populaci kandidátů, ale většina známých ano. [12]

- prostorové algoritmy – např. *dlaždicování a vrstvení, mřížkové dělení, fraktály, Voronoi diagramy*,
- modelování a simulace komplexních systémů – např. *celulární automat, tenzorové pole, simulace agenta a další*,
- umělá inteligence – např. *genetické algoritmy, umělá neuronová síť a plánování*.

Togelius v [3] z roku 2016 představuje metody podle použití, ale taktéž je dělí na dvě kategorie podle průběhu generování.

Generuj-a-testuj algoritmy vygenerují artefakt a otestují, zda splňuje podmínky a zda může být použit. Ty, které podmínky nesplní, jsou zahozeny. V případě algoritmů založených na vyhledávání (tzv. *search-based*, speciálního případu generuj-a-testuj) je využita i tzv. *fitness funkce*, která výsledek ohodnotí a cílem dalšího generování je vytvořit artefakt s vyšší hodnotou. Togelius do tohoto přístupu zařazuje:

- programování množinou dotazů *answer set programming*,
- evoluční algoritmy.

Konstrukční algoritmy jsou takové algoritmy, které proběhnou jednou a na konci svého běhu rovnou vytvoří konečný a vhodný artefakt, aniž by kontrolovaly, zda splňuje podmínky. Není tedy nutné přegenerovat výsledek. Zdroj [3] zmiňuje první tři, zdroj [14] poslední tři:

- dělení prostoru,
- generativní gramatika,
- celulární automat,
- Perlinův šum,
- diamond-square algoritmus.

Následující text se bude držet Togeliova rozdělení v knize [3], tedy na generuj-a-testuj a konstrukční metody.

1.2.1 Generuj-a-testuj přístup

T. Adams, autor hry *Dwarf Fortress (2006)*[9], v diskuzi [15] uvedl, že tento přístup je často volen ve fázi vývoje hry, kdy není nutné získat výsledek rychle a opakované generování a testování není tolik na škodu jako při běhu hry.

A. Smith ve stejné diskuzi rozdělil tento přístup na dva typy podle testovací funkce. V jednom případě je výsledek pouze buď přijat, nebo odmítnut. V tom druhém je výsledek ohodnocen tzv. *fitness funkcí* (známou například z evolučních algoritmů) přičemž cílem další generace je vytvořit lepší výsledek (s vyšší hodnotou). První typ dovoluje vygenerovat nejdříve celý prostor a až poté otestovat jednotlivé výsledky. Druhý již ze své podstaty toto nedovoluje, protože ke generování potřebuje znát ohodnocení předchozího výsledku.

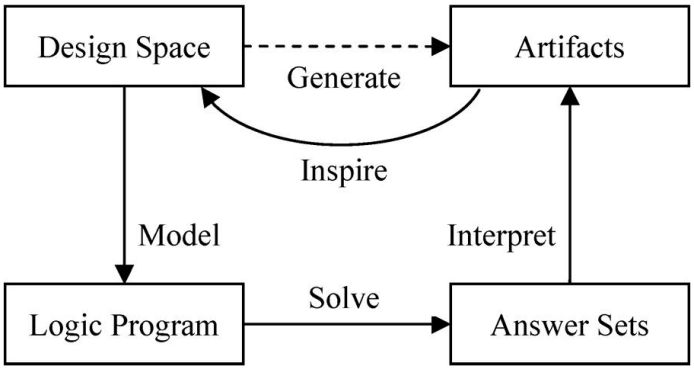
Speciálním případem generuj-a-testuj metody je tzv. *search-based* přístup. Klíčovou vlastností je mapování genotypu na fenotyp. Toto mapování je funkce, která genotyp (chromozom, struktura dat uvnitř vyhledávacího algoritmu, reprezentace artefaktu) převádí na fenotyp (konečný artefakt). Pro to, aby bylo vyhledávání efektivní, je třeba, aby mapování zachovávalo chování, kdy malé změny v genotypu produkují proporčně malé změny ve fenotypu. [3]

Evoluční algoritmy

Tyto algoritmy jsou inspirované biologickou evolucí a darwinismem – přirozeným výběrem, přežitím nejsilnějších, reprodukcí, genetickou mutací. Využívají rekombinaci, mutaci a výběr, aby nalezly optimální řešení problému.

Každé řešení ohodnotí tzv. *fitness funkcí* podle toho, jak si výsledek vedl v porovnání s chtěným cílem. Tyto algoritmy vytvoří populaci výsledků, těch nejhorších se zbaví a ve vývoji postupují ty výsledky s lepším hodnocením. Řešení tedy „evolvuje“, lepší se, až dojde k uspokojivému výsledku.

[9] Tato hra je považovaná za ukázkový příklad projektu využívajícího PCG. [5]



Obrázek 1.2: Postup při tvorbě artefaktů metodou ASP je nejdříve definice výsledného prostoru pomocí logického programu a dále zavolání řešícího programu nezávislého na doméně, který nalezne vhodné výsledky, jenž zároveň definují původní navržený prostor. [7]

Answer set programming (ASP)

Answer set programming[10] je dle [7] označení programovacího paradigma, stejně jako například objektově-orientované nebo funkcionální programování. Jde o formu deklarativního programování[11] založenou na logickém programování. Dovoluje definovat generativní prostor v abstraktních pojmech a následně přidat omezení pro výstup. Podle [16] je ASP zaměřeno na řešení složitých vyhledávacích problémů, převážně NP-těžkých.

Jazyk, který se v tomto paradigmatu používá pro psaní programů, je např. AnsProlog. Výsledný program je dále zpracováván řešícím programem (tzv. *solverem*). Více o tomto jazyku a zvoleném řešícím programu v kapitole 4.

Syntaxí je velmi podobný logickému jazyku Prolog. Základní možnosti jsou stejné – programy obsahují fakta a pravidla:

```
s.              % fakt
p :- q.         % pravidlo
q :- not r.     % pravidlo
```

Obsahuje však dvě konstrukce navíc. Těmi jsou:

choice rules – výběrová pravidla

```
% pokud platí p, může se ve výsledku
% objevit kterákoliv podmnožina {a,b}
{a,b} :- p.
```

[10] Překládáno někdy jako „programování množinou dotazů“. V této práci však bude použita rozšířenější anglická verze pojmu a příslušná zkratka ASP.

[11] Program dostane instrukce, *co* se má udělat, ne *jak* se to má udělat.

integrity contraints – jde o pravidlo, které je tvořeno pouze tělem. Důležitou vlastností není to, že jen zakáže zobrazení výsledků, které toto omezení nesplňují, ale některé řešící programy skutečně zabrání samotnému vygenerování těchto odpovědí. [7, 16]

```
% pokud nastane tato situace,
% artefakt se neobjeví ve výsledcích
:- a, not b.
```

1.2.2 Konstrukční algoritmy

Tyto algoritmy na konci běhu vrátí jeden správný[12] výstup. Jsou vhodné pro generování za běhu programu.

V diskuzi[13] z roku 2009 [15] jsou rozebírány oba přístupy – generuj-a-testuj i konstrukční[14] algoritmy. Togelius označuje jako konstrukční algoritmy generování podle pravidel, Adams dělení prostoru, Perlinův šum, diamond-square, náhodnou procházku a modulární generaci. Smith v diskuzi označuje konstrukční algoritmy jako generuj-a-testuj algoritmy, avšak bez *backtrackingu*.

Pro mnoho typů obsahu je náročné vytvořit konstrukční algoritmus, jehož výsledkem bude vždy správný výsledek. Je možné zvolit i nedokonalé konstrukční algoritmy, které ne vždy vytvoří správný artefakt, ale taková situace by neměla hru rozbít či znemožnit hráči další postup. Používají se tedy mj. pro volitelný obsah, tedy např. pro generování předmětů, které hráč může zahodit. [3]

Z konstrukčních metod budou stručně popsány tyto:

- dělení prostoru,
- agentem řízený průchod (též označovaný jako *digger*),
- celulární automat,
- generativní gramatika.

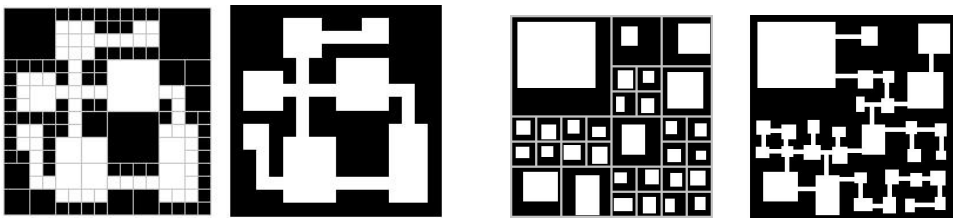
Dělení prostoru

Tato metoda je důležitá pro počítačovou grafiku obecně. Výsledkem metody je tzv. *space partitioning tree*, který se využívá v běžných problémech počítačové grafiky, např. v efektivním sledování paprsku, detekci kolizí atd. Smyslem je reprezentovat existující elementy, jako např. polygony nebo pixely. Podle knihy [3] lze však tento přístup využít nejen pro reprezentaci dat, ale i pro

[12] v ideálním případě

[13] Diskuze se účastnili Togelius, Adams (autor hry *Dwarf Fortress*), Doull, Smith, Nelson a další.

[14] V originále *constructive algorithm*. Z důvodu, že tyto algoritmy konstruují výsledek byl zvolen překlad „kostrukční“.



(a) (b)

Obrázek 1.3: (a) *Dungeon* vytvořený s pomocí quadtree, každá buňka je buď celá součástí místnosti (bílá), nebo prázdného prostoru (černá). (b) *Dungeon* vytvořený s pomocí quadtree, ale s každým kvadrantem obsahujícím jednu celou místnost (umístěno náhodně) i prázdný prostor zároveň – chodby jsou přidány až po dělícím procesu. [3, kap. 3]

samotné generování. Jako vhodnou aplikaci určuje tvorbu místností či jiných oddělených prostor.

Velmi populární metodou je dle stejného zdroje tzv. binární dělení prostoru (BSP), tato metoda prostor rekurzivně dělí na dvě části a výsledkem je binární strom (*BSP tree*). Dalším možným způsobem dělení je např. datová struktura *quadtree* (viz ukázka *dungeonu* vytvořeného tímto způsobem na obrázku 1.3), při jejímž vytváření se 2D prostor dělí na čtyři části (a také *octtree* dělící 3D prostor na osm částí, obecněji pak tzv. k-d strom operující v k-dimenzionálním prostoru).

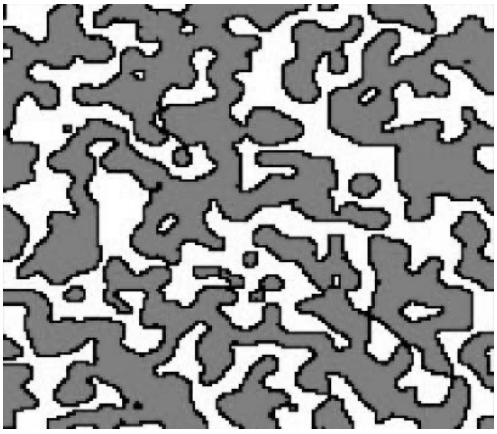
Zdroj taktéž označuje tuto metodu za „makro přístup“ – tedy že algoritmus ví o celém prostoru a pracuje na základě této znalosti. Výsledek generovaný pomocí BSP stromu bude vypadat organizovaně a zaručuje, že se žádné dvě místnosti nebudou překrývat.

Agentem řízený průchod

Togelius v [3, kap. 3] uvádí dva typy agentů. Jednodušší variantou je stochastický, „slepý“ agent, který začne v některém bodu prostoru, pohybuje se v náhodném směru a „kope“ chodby. Může mít dané pravděpodobnosti, že změní směr a že umístí místnost náhodné velikosti (obě pravděpodobnosti mohou být za určitých podmínek zvyšovány i vynulovány).

Jako druhý typ určuje agenta se schopností vidět dopředu. I tento začíná na náhodném místě, avšak než vloží místnost, ujistí se, že nedojde k prolnutí s existujícími místnostmi. Pokud všechny možné místnosti způsobí prolnutí, agent vybere náhodný směr a vzdálenost, po kterou bude „kopat“ chodbu, aby k prolnutí s jinou místností nebo chodbou nedošlo. Agent zastaví, pokud již nejde již nic přidat.

Tato metoda na rozdíl od metody dělení prostoru využívá tzv. „mikro přístup“ – agent procházející prostorem se tedy chová jako slepý a pravděpodobněji vytvoří organický až chaotický výsledek. Zdroj zmiňuje nepředvídatelnost a riziko vzniku nehratelných či nezajímavých map.



Obrázek 1.4: Příklad mapy vygenerované celulárním automatem. Bílá barva představuje skálu, černá okraje skály a tmavě šedá volný prostor. [17]

Celulární automat

Celulární automaty jsou využívány v počítačové vědě, fyzice i v některých odvětvích biologie jako model výpočtu, růstu, vývoje a dalšího. Celulární automat sestává z n-dimenzionální mřížky, množiny stavů a množiny pravidel přechodu. Nejčastější automaty jsou jednodimenzionální (vektory), nebo dvoudimenzionální (matice). Každá buňka může nabývat právě jednoho stavu z dané množiny. Nejjednodušším příkladem jsou automaty se dvěma možnými stavy – zapnuto a vypnuto. Následná podoba mřížky je dána aktuálním stavem a stavy okolních buněk. Okolí se určuje buď jako Mooreovo okolí(osm sousedů okolo buňky), nebo von Neumannovo okolí (čtyři sousedi – nahoře, dole, vlevo a vpravo od buňky). Všechny možné konfigurace okolí se rovnají počtu možných stavů na počet buněk v okolí.

Jedním z nejznámějších celulárních automatů je tzv. Hra života. Jde o dvourozměrný automat se dvěma stavy, který připomíná vývoj společenství živých organizmů.

Článek [17] popisuje systém generování nekonečného jeskynního *dungeonu* za použití celulárních automatů. Žádné úložné medium není schopno uchovat skutečně nekonečný prostor, proto musí být obsah generován za běhu. Hra představená v článku se neposouvá za běhu, ale hráč prochází jednotlivými lokacemi (jedna obrazovka) – je tedy čas vygenerovat novou lokaci mimo obrazovku, když hráč opustí lokaci původní.

Obecně u této metody není zaručeno, že se budou východy shodovat s vchody dalších jeskynních lokací, proto je vytvořen tunel mezi nejbližšími sousedními vchody, pokud neexistuje propojená cesta mezi jednotlivými lokacemi. Taktéž podle stejného zdroje u této metody není možné vytvořit mapu se specifickými požadavky v podobě počtu místností s jistotou propojení. [17]

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

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

Obrázek 1.5: Ukázka gramatiky pro tvorbu *dungeonu*, převzato z [19], graficky upraveno a přeloženo. Slova začínající velkým písmenem jsou v tomto případě neterminály (později budou přepsány) a slova napsaná kurzivou označují terminály (konečné symboly).

Generativní gramatiky

Generativní gramatiky byly původně navrženy pro formální popis struktur přirozeného jazyka [3]. Tyto struktury jsou modelovány pomocí množiny pravidel, které popisují způsob, jak jsou větší celky tvořeny z menších částí. Nazývají se generativní z důvodu, že popisují lingvistické struktury způsobem, který zároveň dává návod, jak je generovat. Se znalostí těchto struktur lze vygenerovat jakoukoliv větu, která bude správná.

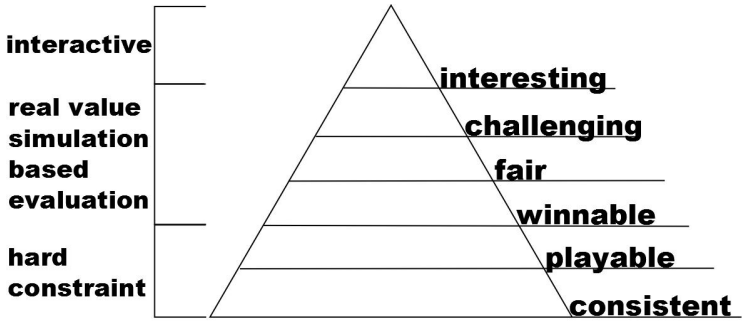
Definice 1.3 (Gramatika)**.** *Gramatika je čtveřice* $G = (N, \Sigma, P, S)$*, kde*

- N *je konečná množina neterminálních symbolů,*
- $\sum$ *je konečná množina terminálních symbolů,*
- P *je množina (přepisovacích) pravidel. Je to konečná podmnožina množiny* $(N \cup \Sigma)^*.N.(N \cup \Sigma)^* \times (N \cup \Sigma)^*$*, element* (α, β) *z* P *zapíšeme jako* $\alpha \to \beta$ *a nazveme pravidlem,*
- S *je počáteční symbol gramatiky.*

Přednáška [18] kromě definice 1.3 udává též jako příklad jednoduchou gramatiku generující řetězce $00, 010, 0110, \ldots$ s následujícími pravidly:

$G = (\{A, S\}, \{0, 1\}, P, S)$, kde P:

- $S \to 0A$
- $A \to 1A \mid 0.$



Obrázek 1.6: Pyramida žádoucích vlastností a jak mohou být rozumně vyhodnoceny podle [14]. Vpravo odspodu vlastnosti artefaktu (např. úroveň hry): jednotnost, hratelnost, lze ho vyhrát, spravedlivost, výzva, zajímavost. Vlevo odspodu způsoby zajištění těchto vlastností: tvrdé omezení, ohodnocení na základě simulace, interaktivně.

Podobný přístup jako u tvorby nul a jedniček lze aplikovat i na tvorbu *dungeonů* či schémat úrovní, jak lze vidět na obrázku 1.5. Lze také využít jedné z mnoha jiných typů gramatik, všech však založených na stejném principu přepisovacích pravidel. Např.

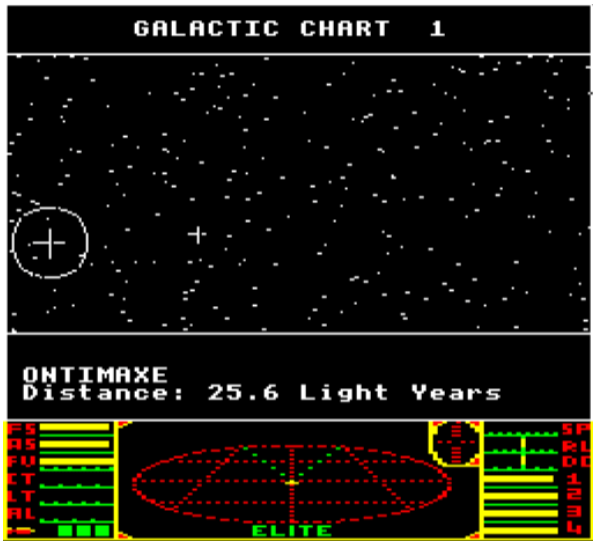
- grafové gramatiky – přepisovací pravidla jsou tvořena vrcholy a hranami,
- tvarové gramatiky (*shape grammars*) – generují geometrický útvar,
- dělící gramatiky (*split grammars*) – generují geometrický útvar dělením větších celků.

1.2.3 Hybridní metody

Každý algoritmus má své klady a zápory a nemusí stačit na konkrétní problém. Proto může být více než chtěné mít k dispozici způsob, jak metody zkombinovat a jejich nevýhody nahradit výhodami jiných.

Obecný systém při tomto návrhu ukazuje obrázek 1.6, kde jsou jednotlivé vlastnosti her, jako např. hratelnost a spravedlivost, rozděleny podle toho, jak je každá část složitá k definování – např. hratelná hra lze snadno definovat konkrétními podmínkami (musí existovat východ z bludiště, musí existovat klíč k zamčeným dveřím v místě, kam se hráč dokáže dostat a podobně), zatímco poutavost hry je těžké určit a často je to stále záležitost lidského názoru. Ta nejjednodušší omezení tedy lze řešit už na úrovni kódu a ty složitější potřebují např. simulaci průchodem a ohodnocení na základě nějakých parametrů. Poutavost či zajímavost výsledku je však něco, co je stále věcí osobního názoru a nelze to nijak obecně popsat.

Článek [14] se zaměřuje právě na takové hybridní metody generování. Uváděným typem hybridní metody je „kompozice“, kde jedna metoda existuje uvnitř jiné. Jako příklad uvádí evoluční-ASP generátor *dungeonů*.



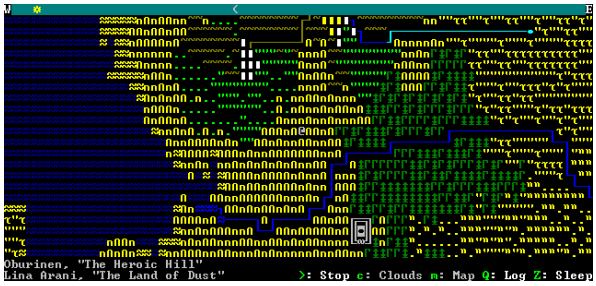
Obrázek 1.7: Ukázka ze hry *Elite (1984)*. Jedna z mnoha galaxií, které je možné v *Elite* prozkoumat. [20]

Implementace kompozice z článku výše je založena na definování číselných parametrů pro generování *dungeonu* jako např. výška a šířka mapy, minimální počet kroků od startu po cíl, počet pastí, počet pomocných předmětů atd. Tato čísla jsou během mapování převedena do předem napsaného souboru pro ASP *solver*, kde nahradí definované proměnné. ASP *solver* najde řešení splňující tyto podmínky (žádné až mnoho) a ty jsou dále poslány do vyhodnocovací funkce, která určí rozdíl mezi mírou výzvy a potřebné dovednosti každého *dungeonu*. Dva agenti projdou úrovní a jejich výsledky jsou doplněny do vzorce. (Jeden z agentů jde po přímé cestě ze startu do cíle, přes nepřátele i pasti, druhý se díky heuristice vyhýbá nepřátelům, pokud to jde, a snaží se posbírat nejbližší pomocné předměty).

Obecně je autory článku doporučeno využít některý konstrukční algoritmus, jakým je např. mapování genotypu na fenotyp, uvnitř algoritmu založeného na vyhledávání. Není však vyloučeno, že lze kompozici vytvořit i jiným způsobem. Výhodou tohoto přístupu je, že „vnitřní“ algoritmus (v případě výše je to ASP *solver*) zaručí správnost „nízkoúrovňových“ požadavků, zatímco ten „vnější“ ohodnotí složitější aspekty (jako spravedlivost, složitost apod., které nejdou snadno popsat).

1.3 Reálné využití procedurálního generování

V této části budou pro ukázku rozmanitosti a různých přístupů představeny některé významnější či zajímavé hry a programy, které PCG využívají.



Obrázek 1.8: Ukázka ze hry *Dwarf Fortress (2006)* [22]

Elite (1984)

Elite je hybrid dvou žánrů: vesmírného obchodování a leteckého simulátoru. [20] Taktéž je ukázkovým příkladem deterministického procedurálního generátoru obsahu. Kvůli nedostatku paměti tehdejších počítačů bylo nemožné mít uložené kompletní informace o všech galaxiích (jména, souřadnice, velikosti hvězd i planet, ceny surovin, četnost pirátů atd.). Proto byl vytvořen algoritmus, který všechny tyto informace deterministicky vygeneroval ze zdroje (autory pojmenován jako „*a seed*“ [21]) a celý vesmír tak byl jednoduše uložen jako seznam těchto zdrojů. PCG tedy bylo v tomto případě využito jako metoda komprese.

Přestože způsob procedurálního generování obsahu – založený na Fibonacciho číslech – umožňoval až 2^{42} různých galaxií, vydavatel Acornsoft přikázal radikálně snížit množství. Došli k osmi galaxiím, s každou obsahující 256 planet, což považovali za množství, které jim v té době lidé uvěří a zároveň nebudou odrazeni krátkou dobou zábavy. [21]

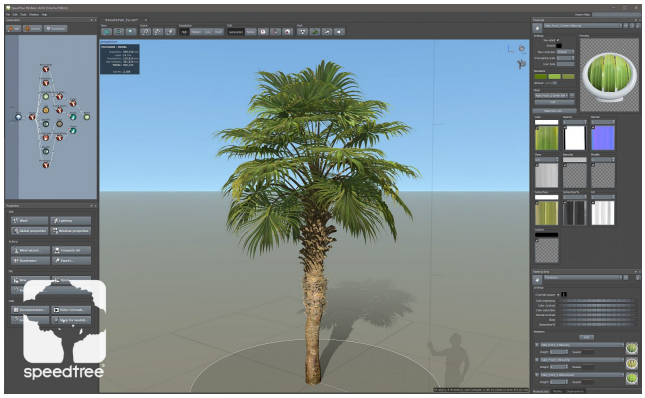
Dwarf Fortress (2006)

Původní název: *Slaves to Armok: God of Blood Chapter II: Dwarf Fortress*. Podle [5] jde o ukázkový příklad PCG vytvořený T. Adamsem a jeho bratrem Z. Adamsem. Jde o textový simulátor fantasy světa (dnes dostupný již s grafikou), který je považován za velmi složitý systém [23].

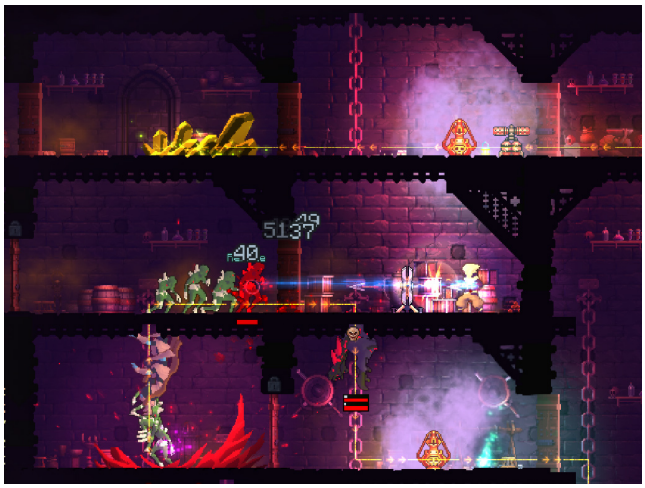
Svět této hry je kompletně generovaný počítačem, včetně počasí, jedinečných historických událostí, postav, terénu, mnoha měst atd. Celá historie je zaznamenávána a hráč se tedy kdykoliv může vrátit do starých míst a za postavami. [22]

SpeedTree (2001)

SpeedTree je sada nástrojů pro 3D modelování, která kombinuje procedurální a ruční modelování. Je dostupný pro *Unreal Engine*, *Unity* a *Lumberyard*. Dovoluje uživateli snadno vytvořit vegetaci a procedurálně vygenerovat mnoho



Obrázek 1.9: Program *SpeedTree* [24]



Obrázek 1.10: Hra *Dead Cells (2018)* [25]

lehce odlišných verzí daného výtvoru. Používá k tomu typ formální gramatiky, tzv. Lindenmayerův systém (L-systém).

Je používán herními studii v mnoha AAA titulech (např. *Zaklínač 3: Divoký hon (2015)*) i v indie hrách a filmech (*Avatar (2009)*). V roce 2015 získal ceny *Sci-Tech Academy Award*, *Develop Award* a *Engineering Emmy*. [24]

Dead Cells (2018)

Tato hra je zařazována do žánru *roguelike*, kde jsou úrovně generované náhodně a po smrti se hráč přesouvá zpět na začátek s více zkušenostmi.

S. Bénard, hlavní designer hry, zmiňuje jako vzor pro řešení PCG hru *Spelunky (2008)*. Dalším uváděným vzorem byla např. hra *Binding of Isaac (2011)*. Původně měla být hra typu *tower defence*. Bylo využito přístupu kombinace procedurálního generování s ručně vytvořenými levely. [25]

Postup při tvorbě levelu hry probíhá následovně:

1. Umístění fixních elementů, vytvoření jakéhosi rámce v podobě hlavního rozvržení typů levelů, jak jsou spojené, kde budou umístěny klíče pro odemčení dalšího obsahu dostupného při dalším hraní atd. – tato část je tvořena ručně

2. Ruční vytvoření tzv. dlaždic – variace místností podle určitého nastavení, jako v případě hry *Spelunky (2008)*:

   - specifické rozložení platforem pro konkrétní účel doplněné o parametry jako počet vstupů a východů, účel místnosti (místnost určená pro boj bude jiná než místnost s pokladem nebo obchodníkem)
   - platformy patří do určitého biomu – každá úroveň má vlastní specifickou podobu (např. biom stok je velmi stísněný, omezující schopnost skoků atd.)

3. Vytvoření konceptu v podobě grafu pro každý biom, postupně se vytvoří:

   a) vstup a východ
   b) přidání speciálních místností (s pokladem, obchodníkem atd.)
   c) mezi to dodány místnosti pro boj a průzkum

   Funguje jako množina instrukcí pro algoritmus generování, určuje délku úrovně, jak daleko od vstupu je nejbližší východ atd. Každý biom má vlastní graf. (Např. biom hradeb je přímočařejší než stoky.)

4. Fáze samotného procedurálního generování. Pro každý vrchol grafu algoritmus zkusí náhodnou místnost z těch, které v daném biomu může vygenerovat, a otestuje, zda řešení vyhovuje instrukcím grafu (lokace a počet vstupů, typ místnosti – obchodník, pokladnice, určeno pro boj atd.). Pokud podmínky nesplní, algoritmus zkusí jinou místnost, dokud nenajde vhodnou.

5. Přidání nepřátel. Jejich počet ovlivňuje celkové množství „soubojových“ dlaždic v levelu. Každý nepřítel má vlastní omezení (někteří jsou více nebezpeční, jiní mohou být vygenerováni pouze jedenkrát na dlaždici či level, další nemůžou být s ostatními na stejné platformě).

6. Generování zlata, cel, kořisti. [25]

Kapitola 2

Hrady

Tato kapitola v sekci 2.1 stručně uvádí do vývoje hradů a v sekci 2.4 do vývoje zámků, popisuje jejich vzhled v sekci 2.2 a rozložení místností a budov v sekci 2.3 a v sekci 2.5 zmiňuje též využití hradů ve fantastice.

2.1 O hradech obecně

Středověk je označením období mezi koncem antiky a počátkem novověku. Přesné rozmezí není jednoznačně určeno, ale běžné ohraničení je od 6. do konce 15. století.

Období do 11. století je označováno za raný středověk. V této době byla stavěna hradiště – předchůdci hradů a měst. Šlo o velkou plochu fungující jako útočiště pro okolní obyvatelstvo, která se však těžko bránila. Od 10. století se odděluje sídlo správce a předznamenává se vznik hradů.

Hrady byly v našich podmínkách obvykle umísťovány na skalním ostrohu nebo jiném těžko dobyvatelném místě. Nejstarší typ kamenného hradu vznikl z jednoduchých strážních opevnění tvořenými jedinou obrannou věží a šlo o hrad s masivní obytnou věží a jednoduchým opevněním kolem nevelkého nádvoří. Později byly přistavovány další budovy a hrady se rozrůstaly. Rozvoj je datován od 1. poloviny 13. století až do 2. poloviny 15. století. V 16. století význam rychle upadá s vývojem palných zbraní. Některé hrady byly přestavěny na zámky, jiné zanikly a dnes jsou zříceninami. Mnoho z nich bylo poté rekonstruováno v 19. století do „původní“ podoby (domnělé). [26]

Hrad jako takový je *Slovníkem spisovného jazyka českého* definován jako: „(v starověku a středověku) opevněné sídlo panovníků nebo šlechticů“. Dále lze najít podobnou definici: „polyfunkční opevněné feudální[15] sídlo vystavěné v období 12. až 16. století.“

[15] feudalismus – systém lenních vztahů typický pro vrcholný středověk

Jde tedy o opevněnou stavbu z období vrcholného středověku s několika funkcemi, např.

- rezidenční – život na hradě měl být pro majitele pohodlný,
- hospodářskou – správa panství,
- vojenskou – obranný účel, hlídání důležitých cest, ochrana pána,
- reprezentativní – ukázka moci majitele.

2.2 Vzhled hradu

Hlavním znakem většiny hradů byla věž. V historii měla různý tvar i účel, obvykle jsou však rozlišovány na:

- *donjon* – věž obytná, často čtverhranného půdorysu, větší okna, mnoho prostoru,
- *bergfrit* – věž útočištná, poskytující obyvatelům hradu poslední útočiště v případě útoku a umožňující obranu. V Čechách nejčastěji okrouhlá. Byla přístupná v úrovni prvního patra po můstku, přízemí bylo bez oken i dveří a nedalo se do něj dostat jinak než podlahou v prvním patře. Prostor mohl být využit jako hladomorna nebo sklad. [26, 27]

Hradů existovalo mnoho typů – lišily se způsobem a rozsahem opevnění, dispozicí zástavby, umístěním v terénu apod. Kastelologové[16] rozlišují hrady na několik typů:

- hrad přechodného typu – stavby tohoto typu mají prvky raně středověkých hradišť i vrcholně středověkých hradů (např. *Tachov*),
- hrad s obvodovou zástavbou – rozměrný, vícedílný typ královského hradu, obvykle více věží a později i s *donjonem* (např. *Křivoklát*),
- hrad bergfritového typu – vzor v německých hradech, jednoduchá stavba zejména šlechticů s utočištnou věží (např. *Kokořín*),
- hrad kastelového typu – vzor ve francouzských hradech, měl pravidelný půdorys (čtvercový či obdélníkový), více věží a obvodové hradby (např. *Týřov*),
- hrad s plášťovou zdí – bezvěžový typ hradu, hlavní obranou je silná zeď s ochozem (např. *Košumberk*). [27]

[16] Kastelologie – věda zabývající se studiem středověkých hradů.

2.3 Rozložení místností

Hlavní částí hradu byl palác. Z obranných důvodů mělo přízemí pouze úzká okna a bylo využíváno převážně jako skladiště potravin, zbraní a všeho, co bylo dále potřeba. Obytný prostor se nacházel až v prvním patře. Hlavním prostorem prvního patra (ne však obytným) byl velký sál a dále obytné prostory, zpočátku jen nemnoho, později vznikaly pavlače pro přístup zvenčí, neboť místnosti nebyly průchozí. Ve vyšších patrech se často nacházela kaple (z důvodu jejích velkých oken nebyla kaple běžně stavěna v nižších patrech – porušovala by obranyschopnost hradu – ale pokud byla, osvětlovala ji pouze okna do vnitřního nádvoří. Pokud majitel hradu neměl dostatek financí na okázalou kapli, mohl jí být malý vikýř v obytných prostorách prvního patra. Mnoho menších hradů kapli ani nemá. Mezi patry se přesouvalo převážně po točitých schodištích. Hrad byl vytápěn černou kuchyní – oheň hořel od rána do noci a často i nepřetržitě. Spaliny stoupaly vzhůru systémem průduchů v hradních obvodních stěnách či komínem. [26]

Hlavní palác tedy byl, co se týče typů místností, značně omezen, což je dáno i snahou o kompaktnost a obranyschopnost. Palác obývala pouze rodina, pár komorných, kuchařka, nejbližší stráž a nezbytná obsluha.

Toto je však jen jeden z mnoha případů, jak mohl být hrad postaven. Hrady měly zpočátku účel zcela obranný, později se však objevovaly i hrady s primární funkcí obytnou – kde tedy obranné funkce hradu byly potlačeny až na nezbytné minimum. Záleželo taktéž na tvaru hradu a prostoru a tím bylo mnohdy určeno rozložení místností. Např. hrad *Forna* ve Španělsku je čtyřhranný s nádvořím uprostřed a v jeho přízemí byly stáje, kuchyň, jídelna i velký sál a v patře obytné místnosti.

2.4 Zámky

Hrady jsou záležitostí vrcholného a pozdního středověku, tedy doby od 12. do 15. století, doby neustálého boje o území i menší majetek, zatímco zámky se začaly stavět až později.

Jeden z hlavních rozdílů je ve funkci stavby – hrady měly primárně obrannou funkci , sloužily jako strážní místa a šlechtická opevnění v době válek. V době vzniku zámků již hrady pozvolna ztrácely vzhledem k vynálezu střelných zbraní své obranné schopnosti (stísněné prostory hradů tedy ztrácely na výhodě a hradby proti stále silnějším dělům mnoho nezmohly). Války utichaly a nepanovalo neustálé nebezpečí, šlechta tedy začala mít větší nároky na pohodlí a vzhled svých sídel. Zámky tedy plnily hlavně funkci obytnou a reprezentativní.

Ač by zámky mohly být vhodnějším zdrojem pro tuto práci, co se týče rozložení a množství místností, byl jako inspirace zvolen jejich předchůdce. Prostory i nábytek zámků byly mnohem složitější a zdobenější a pro mode-

lování náročnější než strohé prostory hradů. Dále jsou hrady se svými důmyslnými obrannými opatřeními a tím, že jsou s oblibou využívány pro reálie fantastických světů, zajímavějším zdrojem. Z toho důvodu byly zvoleny hrady, sice s méně možnostmi rozložení v základu než u zámků, nicméně s mnohem více možnostmi s využitím bezbřehé fantazie.

2.5 Hrady ve fantastice

Hrady a obecně středověké reálie jsou oblíbenou inspirací pro kulisy fantastického žánru, ať už v knihách, filmech, či hrách.

Zkoumáním podoby hradů vzniklých ne rukama skutečných stavitelů dob minulých, ale fantazií současných spisovatelů a umělců bylo zjištěno, že pro potřeby této práce jde o bohatší zdroj inspirace. Nejen, že v takovém hradu lze mnoho místností, které ve skutečnosti nebývaly u sebe, natož pod jednou střechou, vygenerovat vedle sebe, ale k základním místnostem se přidá ještě mnoho a mnoho jiných s více či méně pravidly.

Dobrými zdroji se staly diskuze stavitelů pro hru *Minecraft (2011)* a taktéž příručky pro hráče *Dungeons and Dragons*, např. [28], ve kterých lze najít nejen typy místností, ale i jejich velikost a jakési podmínky pro postavení (v případě jídelny je třeba kuchyň, ve které je třeba mít služebnictvo, pro které je zase třeba místností pro jejich ubytování). Autoři těchto příruček kladou vysoké nároky na promyšlený svět do drobností, ač jde často o nereálné objekty, a informace nalezené v mnoha edicích mohou být velice užitečné.

V příručkách lze např. najít i shluky místností tvořící souvislou komponentu, např. pro hostinec je třeba taverny, místností pro hosty, kuchyně, část pro služebnictvo, sklad. Tyto informace lze využít pro návrh pravidel, kde vzájemné pozice místností mají nějaký pevně daný vztah či omezení. Kromě základních a zcela logických místností lze v této příručce najít i tzv. „úžasnou architekturu“, např. Warding Bell (obří zvon, který aktivuje ochranné kouzlo), nebo Summoning Stone (kámen, který se aktivuje, když se někdo k němu přiblíží, a povolá stvoření, které zaútočí na narušitele). [28]

Kapitola 3

Analýza

Tato kapitola v sekci 3.1 rozděluje metody podle typů generovaných artefaktů. Metody vhodné pro generování budov jsou rozebrány v samostatné sekci 3.2. V sekci 3.3 jsou představena existující řešení pro procedurální tvorbu budov a sekce 3.4 uzavírá kapitolu seznamem typů místností hradu.

3.1 Metody podle zaměření

Kromě rozdělení metod podle přístupu k tvorbě artefaktu, dělí kniha [3] metody též podle toho, na který obsah se běžně používají a hodí. Např. při tvorbě mapy, kde je třeba nejen terén, ale i stromy a rostliny, případně domy či celé vesnice, doporučeno rozdělit artefakt na menší části (tedy negenerovat rovnou vše najednou, ale postupně, nejdříve terén, poté ho zaplnit vegetací a vesnicemi, které – pokud mají být také procedurálně tvořené – mohou být generovány stranou a poté dodány do terénu).

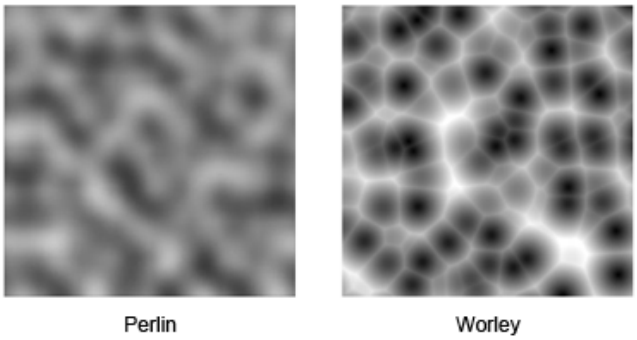
Je totiž možné, že pro konkrétní artefakt je těžké najít vhodnou metodu generování, a to z důvodu, že jde o odlišné přístupy (jak se může ukázat po rozdělení problému na části, např. případ terénu s vegetací a městy). Není však vyloučeno, že nelze přijít s novým použitím či metodou.

Terén a mapy

Pro generování terénu je používáno následující:

- šumy – Perlinův šum, Simplex noise, Worleyho šum[17] a další,
- diamond-square algoritmus,
- gramatiky (grafové, generativní [19]),
- ASP [30].

[17] také označovaný jako celulární šum (*cellular noise*) [29]



Obrázek 3.1: Porovnání Perlinova a Worleyho šumu. [29]

Šumy a *diamond-square* algoritmus jsou používány pro generování tzv. výškových map, kde každý bod terénu má své číslo představující výšku. Je vhodný pro generování realistických terénů (realistických ve smyslu postupné změny terénu, ne ve způsobu grafického zobrazení – např. hra *Minecraft (2011)* založená na kostkách využívá pro generaci map právě 3D Perlinův šum [31].

Vegetace

Pro generování vegetace jsou používány gramatiky, konkrétně L-systémy, který používá taktéž nástroj pro tvorbu vegetace *SpeedTree (2001)* (o nástroji viz sekce 1.3 předchozí kapitoly).

Dungeony

Dungeony jsou často podzemní komplexy mnoha místností a chodeb, vytesané např. do skály. Ať už jsou místnosti propojené chodbami zasazené do volného prostoru, nebo neexistuje žádn0 okolí a *dungeon* tedy představuje komplex jen místností a chodeb, lze dělení prostoru využít. V prvním případě tak, jak ukazuje obrázek 1.3, v tom druhém tak, že informace o rozdělení prostoru bude symbolizována zdí.

Bludiště

Tato oblast je velice oblíbená, hojně probíraná a lze pro ni nalézt mnoho materiálů.[18] V odborných pracích lze často vidět, že bludiště a jiné logické rébusy jsou řešeny pomocí ASP, např. v [7, 32]. Vzhledem k původu ASP v logickém programování je volba logického rébusu pro seznámení se s metodou pochopitelná.

[18] Zájemci nechť jsou odkázáni na stránku s mnoha příklady algoritmů nejen pro tvorbu bludišť, ale i pro jejich řešení. Viz `http://www.astrolog.org/labyrnth/algrithm.htm`

Ostatní obsah

Dalším obsahem, který může být generován a kterým se podrobněji zabývají jiné práce, je např. generování zvuku, příběhu či herních pravidel [3, kap. 6] (metoda generování herních pravidel byla použita např. ve hře *Keep Talking and Nobody Explodes* [1, kap. 12]).

3.2 Metody vhodné pro generátor budovy

V řešerši bylo představeno mnoho metod, ne všechny se však hodí pro tvorbu generátoru budovy (konkrétně v případě této práce generátoru hradu). Metody tvořící náhodné organické artefakty (jako např. celulární automat, Perlinův šum apod.) vhodné nejsou. Jde převážně o metody využívané při generování realistického terénu. Tvorbu hradu a jeho místností je třeba korigovat a omezit více, než tyto metody dovolují.

V případě generování budov velmi záleží na části, která je generována. Z existujících řešení nebylo zjištěno, které metody používají, pouze že je procedurálně generována samotná geometrie budov a nejsou tedy použity deformace na již předpřipravené budovy.

Metody vhodné pro generování budov:

- gramatiky (tvarové a dělící gramatiky, L-systémy [33]),
- dělení prostoru,
- ASP.

Tato práce je zaměřena na generaci interiéru, rozdělení prostoru do místností a určení pravidel pro jejich umístění. Problém generování budov lze připodobnit k tvorbě *dungeonů* či bludišť, kde je třeba organizovanost a kde metody umožňují definovat pravidla pro umístění místností, povinný minimální obsah atd.

Takové možnosti by mohly umožnit gramatiky, ASP a pro připravení prostoru pro místnosti je používána metoda dělení prostoru. Není vyloučeno, že by šlo využít některé z jiných metod, ale jejich převod na generování hradu by nemusel být snadný a pro potřeby tohoto generátoru vybrané metody postačují.

V následujících tabulkách (3.1, 3.2 a 3.3) budou jednotlivé vybrané metody rozebrány podrobněji a zváženy výhody a nevýhody.

Gramatiky	
Pro	Proti
• definice prostoru pomocí pravidel	• tvorba pravidel může být náročnější
• konstrukční – je vytvořen správný artefakt již po jednom běhu	• náročnější taktéž následná parametrizace – nutno generovat kód

Tabulka 3.1: Porovnání možností generování pomocí gramatik

Dělení prostoru	
Pro	Proti
• lze dělit prostor na různě velké místnosti	• nijak neurčí logiku místností

Tabulka 3.2: Porovnání možností generování pomocí dělení prostoru

ASP	
Pro	Proti
• definice omezení	• nutnost instalace ASP *solveru*
• výsledek nalezen díky externímu řešícímu programu	• náročnější na parametrizaci – nutno generovat kód

Tabulka 3.3: Porovnání možností generování pomocí ASP

Obrázek 3.2: Ukázka výsledku generátoru budov (T. Ibele) [35]

3.3 Existující řešení

3.3.1 Generátory budov

Generování budov je důležitou součástí při tvorbě městských scén a tvůrci potřebují mnohdy zaplnit ohromné množství prostoru. Tvořit každou budovu zvlášť by bylo časově i finančně náročné, proto se využívají nástroje pomáhající s touto činností. V následujícím textu budou představeny některé z generátorů budov a stručně popsány.

S2ENGINE HD – Building generator (2006)

S2 ENGINE HD je kompletní sada nástrojů pro tvorbu a spouštění her zaměřená hlavně na herní nadšence a nezávislé vývojáře, vytvořená nezávislou firmou Profenix Studio SRLS.

Building generator je plugin do *S2 ENGINE HD* editoru generující realistické 3D modely budov. Generátor je zcela procedurální, tedy neupravuje externí modely – všechnu geometrii tvoří sám. Tvorba vnitřku budovy zatím není podporována, ale tato funkcionalita je plánována. [34]

Building Generator (2009)

Skript procedurálního generátoru budov pro *3d Studio Max* napsaný T. Ibelem. Procedurálně generuje kompletní geometrii, nevyužívá externích modelů. Dovoluje pouze jednoduchou tvorbu interiéru – místnosti, dveře, světla. Podle informace na jeho stránkách již není tento generátor dále vyvíjen [35].

BuildR 2 (2009)

Plugin napsaný J. Stockerem pro program *Unity*. Dokáže generovat i interiér, ale pouze v jednoduché rovině – tedy dokáže rozdělit prostor do místností, propojit je průchozími dveřmi a umožňuje přidat textury. [36]

3.3.2 Generátory půdorysu

Někdy není třeba generovat vnější vzhled budovy, ať už z důvodu, že generátor pouze vyplní již existující prostor, nebo nebude vnější vzhled nikdy viděn, a v té chvíli přicházejí na řadu generátory půdorysu, které rozdělí prostor na místnosti a případně je vyplní obsahem. Generátor tedy pracuje v rámci jednoho patra a řeší 2D rozložení, velikosti místností, vzdálenosti, počet.

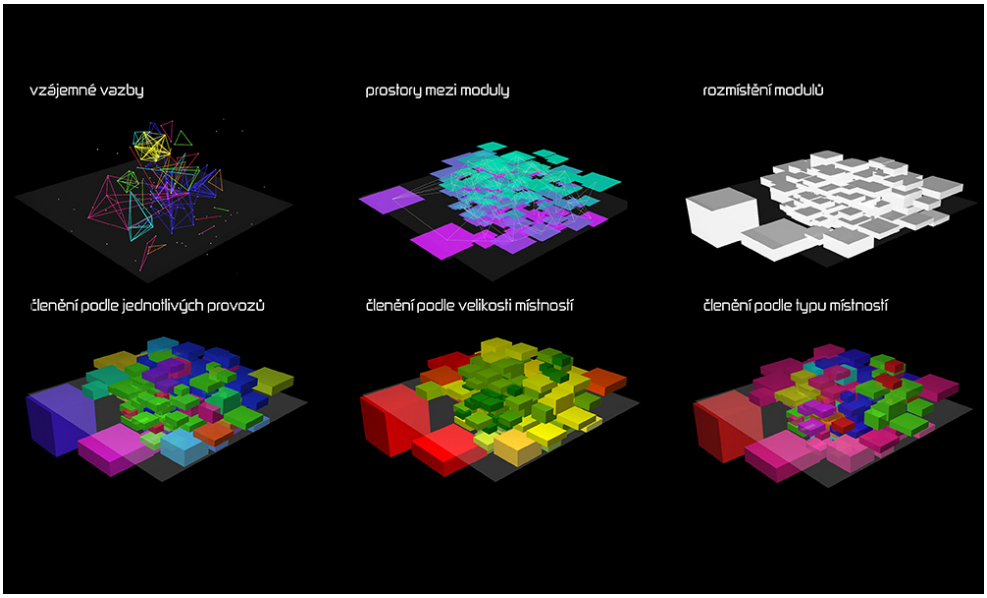
Většina existujících nástrojů pro generaci půdorysu využívá metodu dělení prostoru či metodu, kde jsou místnosti náhodně rozmístěny a poté až propojeny chodbami. Z nalezených řešení však pouze dvě berou v potaz i logiku samotného rozložení, jako např. vzdálenosti jednotlivých typů místností.

Floor Plan Generator (2016)

Projekt *Floor Plan Generator* od Martina Žatečky pro Studio Florián (FLOW) generuje na základě několika pravidel kompletní návrh stavby. Rozmístění místností řeší na základě druhu a funkce, což je možné definovat ve vstupních údajích generátoru. Nerozmisťuje však místnosti do pater, jak je běžné, ale podle tzv. „raumplanu“, což je pojem v architektuře popisující prostor členěný do místností, které se prolínají v různých výškových úrovních, kde prakticky neexistují chodby a místnosti jsou propojeny pomocí schodů a ramp. [37]

Magnetizing Floor Plan Generator (2019)

Tento generátor vychází z myšlenky, že všechny místnosti budovy jsou dostupné z jiných místností a tvoří tak propojenou strukturu. Každá místnost je rozšířena o chodbu po jedné, dvou nebo čtyř ze svých stěn. Místnosti jsou poté rozmístěny tak, aby vždy byly vlastní chodbou připojeny k hlavní struktuře chodeb, a navíc aby sousedily s požadovanými jinými místnostmi. Proces běží, dokud je možné místnost umístit. Pokud ne, proces začne znovu další iteraci. Projekt využívá „kvazi-evoluční“ algoritmy pro ohodnocení výsledků, zpravidla na základě počtu umístěných místností, nebo zastavěné plochy. [38]



Obrázek 3.3: *Floor Plan Generator (2016)* od Martina Žatečky [37]

Obrázek 3.4: *Magnetizing Floor Plan Generator (2019)* [38]

3.4 Místnosti hradu

Z rešerše hradů (převážně ze zdrojů [27, 39]) byly identifikovány tyto místnosti:

Skladiště bylo používáno na skladování zbraní, potravin a všeho, čeho bylo třeba. Jde o místnost, která vyplňovala prostor hlavně v přízemí, ale mohla být prakticky kdekoliv. Byla potřeba pro vše – sklad potravin pro kuchyni, sklad zbraní a zbrojí pro vojáky, sklad jídla, sena a vybavení pro koně ve stájích a mnoha dalšího.

Stáje zpravidla přímo v hlavním paláci nebývaly, nicméně pokud ano, zápach zvířat bylo lepší držet dál od obytných místností. Taktéž byly umisťovány do přízemí.

Velký sál (též rytířský sál či síň) byl hlavní místností, kde se přijímaly návštěvy, jednalo se a oslavovalo. Vyskytoval se zpravidla v prvním patře, výjimečně ve vyšších patrech, jako např. na hradě v Říčanech.

Kaple byl prostor určený k bohoslužbám. Nacházela se zpravidla ve vyšších patrech a byla nejnákladnější místností pevnosti. Její okna nesměla narušovat obranu hradu, proto byla umístěna ve vyšších patrech, nebo byla její okna na obranné straně velmi malá, tzv. střílnová. Mohlo jít o velký prostor s vysokým stropem a zdobenými okny umístěný ať už v hlavním paláci, nebo mimo, ale i skromný výklenek v soukromé komnatě s malým oltářem. Nebyla nezbytnou součástí hradu.

Komnata byla soukromá obytná místnost s postelí, truhlami na předměty a oblečení. Postel byla často oddělena těžkým závěsem a na zdích visely tapiserie kvůli ochraně před chladem.

Černá kuchyně byla malou místností a sloužila pouze k vaření. Vařilo se v ní pro obyvatele hradního paláce, ne pro obyvatele celého hradu. Jídlo se připravovalo v místnosti jiné, např. na hradě Pernštejn. Byla vázána na místo, pro které vařila, neboť bylo třeba přenášet jídlo na co nejmenší vzdálenost.

Jídelna byla místem, kde se scházelo k jídlu. Jídelny byly rozdělené na ty pro panstvo a pro služebnictvo.

Prevét byl často výklenek ve zdi hradu s kamennou sedačkou a otvorem, kterým splašky putovaly do hradního příkopu či kamkoliv tato díra vedla.

Schodiště bylo zpravidla točité a směrem vzhůru se stáčelo doprava. Šlo o výhodu obránců, neboť nepřítel postupující zdola měl kvůli zbrani v pravé ruce omezený prostor pro pohyb, na rozdíl od obránců, kterým takto zatočené schodiště umožňovalo používat zbraně pohodlněji.

Vězení taktéž označováno jako „hladomorna“. Mohlo jít o sklepení přístupné pouze otvorem v zemi, kterým se vězni spouštěli dolů, jako např. na hradě Křivoklát.

Mlýn byl součástí hospodářského vybavení hradu a velikost i vybavení mlýnu záleželo na nárocích majitele hradu a podmínkách dané oblasti.

Sýpka se objevuje stojící samostatně až v 15. století. Do té doby bylo obilí skladováno v jiných prostorách hradu, nejspíše ve skladu.

Pivovar se taktéž objevil až v pozdějších stoletích, do té doby mohlo být pivo připravováno v běžné kuchyni. S příchodem pivovarů se výroba přesunula mimo palác a vznikly k tomu přidružené prostory jako sladovna, spilka (v ní probíhá kvašení piva) a ležácké sklepy.

Kapitola 4

Návrh

V této kapitole je navržen generátor středověkého hradu pomocí zodpovězení základních otázek: *co* bude generováno a *jak* a *čím* bude výsledku dosaženo. V sekci 4.1 jsou představeny vybrané místnosti zvolené jako výchozí pro použití v prototypu generátoru a v sekci 4.3 je seznam jejich pravidel. Sekce 4.4 představí praktickou aplikaci metody zvolené v analýze (metoda ASP) pomocí jazyku AnsProlog a v sekci 4.5 je navržen samotný plugin a jeho struktura.

4.1 Vybrané místnosti

Generátor umožní uživateli nastavit pravidla rozmístění vlastních modelů, ale jeho výchozí chování bude určeno bez uživatele díky dodaným modelům a výchozí logice představené v dalších sekcích. V následujícím seznamu jsou představeny místnosti, které by měly stačit pro základní podobu středověkého hradu inspirovaného skutečným rozložením, a jejich role v generátoru.

Chodba může být použita jako „vyplňovací“ místnost, tedy místnost, která může být kdekoliv, nemá žádné omezení na počet, ani závislost na jiných blocích.

Skladiště nebude mít omezení na počet, ale bude moct být pouze v prvních třech patrech.

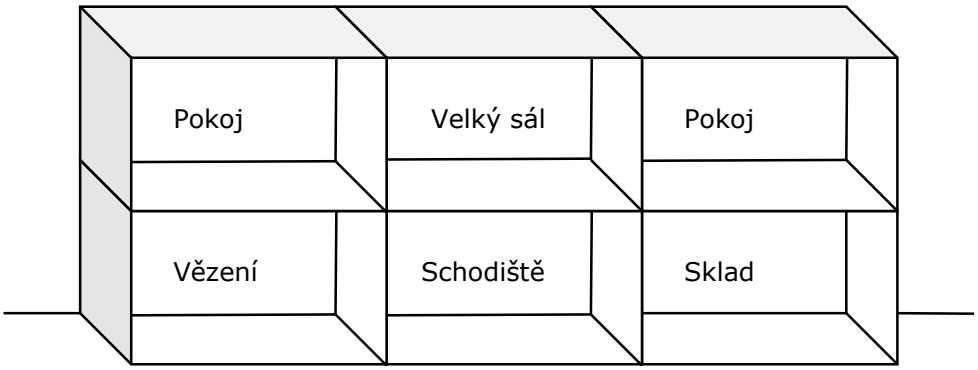
Velký sál bude důležitou místností a bude třeba, aby byla v každém vygenerovaném hradu právě jednou.

Kaple bude využita jako dvoupatrová místnost.

Komnata bude obytnou místností ve vyšších patrech.

Kuchyně ve spojení s jídelnou budou tvořit dvojici místností, které se musí objevit na patře v omezené vzdálenosti od sebe.[19]

[19] Neboť není výhodné, aby se služebnictvo uběhalo při nošení jídla na stůl.



Obrázek 4.1: Nákres podoby hradu s příkladem místností.

Jídelna ve spojení s kuchyní bude tvořit dvojici místností, které se musí objevit na patře v omezené vzdálenosti.

Točité schodiště bude procházet celou budovou (až na poslední patro) a bude muset být v jedné linii. Tedy musí existovat pro přechod do dalšího patra a musí být všechny nad sebou. Nesmí být v nejvyšším patře.

Vězení bude představovat místnost, která musí být od kuchyně vzdálena určitý počet místností.[20]

4.2 Podoba hradu

Ač bude hrad tvořen v trojrozměrném prostoru skládáním taktéž trojrozměrných místností, metoda bude pracovat s 2D polem. Zvolenou metodou, tedy ASP, by bylo pravděpodobně možné tvořit i ve 3D prostoru – hlavní výzvou by však bylo určení způsobu, jakým prostor reprezentovat, a poté sepsání samotných pravidel tento prostor zohledňujících. Dalším úskalím by mohla být složitost a množství všech možností s jakou by se řešící program potýkal, pokud by byl prostor takto zvětšen.

Hrad tedy bude generován do šířky a výšky, ne do hloubky – od spodního patra výše s každým patrem složeným z řady místností. Pro jednodušší implementaci bude každá místnost stejně široká i vysoká s výjimkou kaple vysoké dvojnásobek běžné výšky místnosti. Na obrázku 4.1 lze vidět nákres popsaného vzhledu výsledného hradu.

Kromě samotné generace systému místností, která vytvoří pouze blok m krát n, kde m, n jsou přirozená čísla, bude třeba, aby generátor dotvořil zbytek scény specifické pro hrad – hradby na vrcholu bloku, věž, okolí atd.

[20] Majitelé hradu, kteří ho nechali vystavět, nejsou tak krutí, aby nechali hladové vězně čelit všem vůním linoucím se z kuchyně.

4.3 Pravidla

Pravidla budou pro pozdější implementaci rozdělena do dvou kategorií podle toho, zda potřebují pro své provedení více místností, nebo ne. Takové rozdělení poté bude zváženo při návrhu jednotlivých komponent v uživatelském rozhraní. Jednotlivé místnosti a jejich omezení na počet a umístění v patrech je popsáno v tabulce 4.2.

- Pravidla zahrnující více než jednu místnost:
  - minimální/maximální vzdálenost mezi konkrétními místnostmi
  - skupiny místností
- Pravidla neovlivňující ostatní místnosti:
  - minimální/maximální počet dané místnosti
  - povolená/zakázaná patra pro danou místnost
  - typ místnosti
  - dvoupatrová místnost
  - konkrétní umístění místnosti
  - široké/úzké místnosti

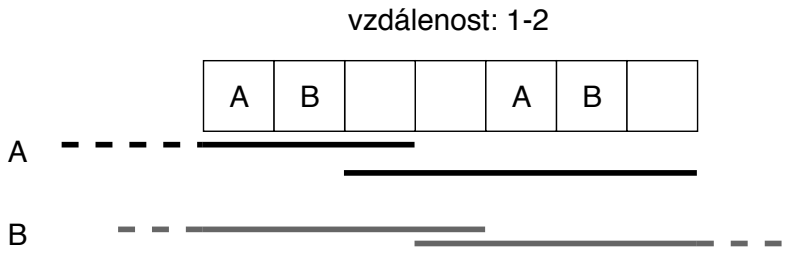
Minimální/maximální počet

Množství každého modelu místnosti bude moct být omezeno jak shora, tak zdola. Minimální počty jednotlivých místností nesmí překročit rozměry hradu, maximální nesmí být menší.

Minimální/maximální vzdálenost

Toto pravidlo lze v případě více instancí stejných typů místností v patře brát různě. Pokud je dvěma typům místností (např. typy A a B) určena minimální a maximální vzdálenost od sebe, může být např. požadováno, aby toto omezení splňovaly pouze dvojice místností, tedy pokud A a B mají od sebe být určitou vzdálenost, může existovat jiná dvojice, která toto také splňuje v rámci své dvojice (grafické znázornění této možnosti je na obrázku 4.2). Toto pravidlo tedy lze chápat tak, že např. místnost B nesmí být od místnosti A v menší než minimální vzdálenosti a zároveň se musí nacházet v té maximální (další instance místnosti B se od A může nacházet ve vzdálenosti větší než maximum, ale za podmínky, že k ní lze přiřadit jinou instanci místnosti typu A.

V případě tohoto generátoru bude zavedeno striktnější omezení, tedy nebude povolena žádná místnost typu B vzdálenější než maximum od místnosti typu A. Příklad na obrázku 4.2 tedy nebude správný.



Obrázek 4.2: Zakázaná možnost výsledku pro pravidlo vzdálenosti. Vzdálenost je určena odečtením indexů místností, tedy místnosti položené přímo vedle sebe mají vzdálenost 1. Čáry pod polem znázorňují okolí místnosti položené uprostřed (černá barva pro místnost typu A, šedá pro místnost typu B.

První místnost	**Min.**	**Max.**	**Druhá místnost**
Kuchyně	1	4	Jídelna
Kuchyně	3	–	Vězení

Tabulka 4.1: Návrh vzdáleností.

Povolená/zakázaná patra

Každá místnost bude moct mít omezení pro objevení se na konkrétním patře. Např. v případě stájí je více než vhodné je umístit pouze do nejnižšího patra, pokud vůbec. V případě soukromých komnat panovníka naopak může být žádané, aby byly ve vyšším patře než v prvním, nebo např. v tom nejvyšším.

Dvoupatrové nebo široké místnosti

Některé místnosti mohou být dvoupatrové, jako např. kaple, která měla často vysoké stropy, nebo široké přes dvě běžné místnosti, například pro rytířský sál. S takovou místností bude zacházeno jako se dvěma přímo vedle sebe (v případě širokých místností), nebo nad sebou (v případě vysokých), s tím, že jedna část bude obsahovat větší model a v místě druhé části bude prázdný prostor.

Typ místnosti

Místnosti by se hodilo roztřídit do různých skupin. Mohou se tak oddělit hospodářské objekty od těch obyvatelných, kdy např. kuchyně musí být vzdálena od hospodářských objektů alespoň tři bloky.

Skupiny místností

Některé místnosti mohou společně tvořit skupinu. Např. v případě jídelny je dobré mít poblíž kuchyň, která však potřebuje často zásoby ze skladu a pro jejíž chod je třeba služebnictva, které je zase nutno někde ubytovat atd.

Název místnosti	Počet min.	Počet max.	Povolená patra
Vstup	1	1	přesné souřadnice (1,1)
Kuchyně	1	2	1, 2
Jídelna	1	3	1, 2
Sklad	2	–	<4
Chodba	0	–	všechna
Velký sál	1	1	2, 3
Komnata	1	polovina	>2
Kaple	0	1	>1
Vězení	0	1	1
Schodiště	záleží		všechna kromě posledního

Tabulka 4.2: Návrh pravidel místností. „Polovina“ u počtu komnat znamená, že maximum je polovina celkového počtu místností v hradu. Prvním patrem je myšleno přízemí, první generované patro.

4.4 Využití jazyka AnsProlog a programu Clingo

Hlavní metoda zvolená pro realizaci je generování pomocí ASP. Zástupcem tohoto přístupu je programovací jazyk AnsProlog a jeho řešícím programem (tzv. *solver*) je např. *Clingo* [40].

Tento jazyk umožňuje definovat výsledný prostor, který je požadován, pomocí čtyř možností zmíněných v rešeršní sekci 1.2.1. Rozhraní samotných řešících programů dále může poskytovat více možností, jak s kódem pracovat. *Clingo* např. umožňuje spustit kód s následujícími přepínači, které zajistí náhodný artefakt (za předpokladu, že je z čeho vybírat):

```
clingo filename.lp --rand-freq=1 --seed=`echo $RANDOM`
```

Clingo taktéž umožňuje definovat proměnné nastavitelné zvenčí při volání programu uživatelem. V logickém programu je taková proměnná zapsána následovně:

```
#const variableName = defaultValue.
```

Při volání programu z terminálu lze proměnnou přepsat jinou hodnotou. Pokud toto není uděláno, nastaví se jí výchozí hodnota zapsaná v kódu.

```
clingo filename.lp --const variableName=newValue
```

Takto je možné snadno změnit např. šířku a výšku. V případě omezené množiny místností by bylo možné takto zadefinovat i samotné místnosti a jejich pravidla, takže např. příprava kódu pro pozdější změnu množství jedné místnosti (jejíž číselný identifikátor může být taktéž změněn, např. pro změnu modelu místnosti při vykreslování) by vypadalo takto:

```
#const room = 1
#const roomMin = 1.
#const roomMax = 5.

:- not roomMin {sprite(T, room)} roomMax.
```

Změna množství takto vygenerované místnosti i s jejím číselným identifikátorem by poté byla zařízena následovně (přepínač `--const` lze nahradit kratší verzí `-c`, tato možnost je v následujícím příkladu pro ušetření místa použita):

```
clingo filename.lp -c room=3 -c roomMin=2 roomMax=5
```

Tento přístup však není vhodný, neboť plugin bude umožňovat přidat libovolný počet místností. Nelze tedy předem určit, kolik proměnných bude třeba. Z toho důvodu bude zvoleno generování konkrétních pravidel až na základě uživatelského vstupu. Každé nastavení bude mít vliv na konkrétní pravidlo (množství, vzdálenosti atd.). Např. při zadání minimálního a maximálního počtu dané místnosti se vygeneruje následující pravidlo pro omezení počtu, které znamená, že pokud nebude vygenerováno určité množství z intervalu <min, max>, bude takový výsledek zahozen. Za `[max]` a `[min]` budou dosazeny vstupy od uživatele a za `[roomId]` identifikátor konkrétní místnosti.

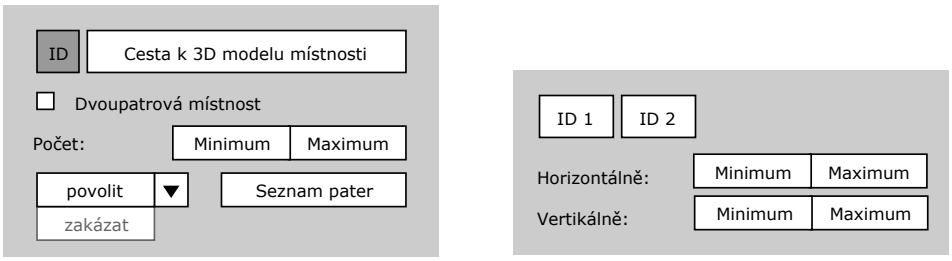
```
:- not [min] {sprite(T, [roomId])} [max].
```

Vstup uživatele tedy vyprodukuje speciální kód, který bude poté poslán do ASP *solveru* a vyhodnocen a plugin poté výsledek zobrazí v prostředí *Blender*.

4.5 Návrh pluginu

Vzhledem k povaze zvolené metody i programu *Blender*, pro který je generátor tvořen, je návrhová část značně omezena. Jazyk AnsProlog specifický pro zvolenou metodu, tedy ASP, potřebuje řešící program. Byl zvolen často používaný program jménem *Clingo*, který je možné používat jako konzolovou aplikaci, nebo je též dostupný jako modul pro Python aplikace.

Blender ve svém rozhraní pro tvorbu rozšíření nabízí třídy pro snadné definování uživatelského rozhraní, které se drží celkového vzhledu programu. Součástí návrhu tedy nebude design tlačítek a textových polí, ale rozložení jednotlivých prvků.



(a) (b)

Obrázek 4.3: Návrhy položek v uživatelském rozhraní. (a) Položka definující místnost. (b) Položka definující vzdálenost mezi dvěma místnostmi.

Návrh celkového procesu

Pluginy pro *Blender* musí být psány v jazyce Python. V době psaní tohoto textu je nejnovější stabilní verze *Blender 2.79* a pro něj jazyk Python 3.5. V tomto jazyce bude napsána hlavní část pluginu, která nastaví uživatelské rozhraní, přijme vstupy od uživatele a vygeneruje soubor v jazyce AnsProlog, který nechá vyhodnotit externím řešícím programem *Clingo*, jehož výsledky poté rozebere na jednotlivé místnosti se souřadnicemi a vykreslí je ve 3D prostoru.

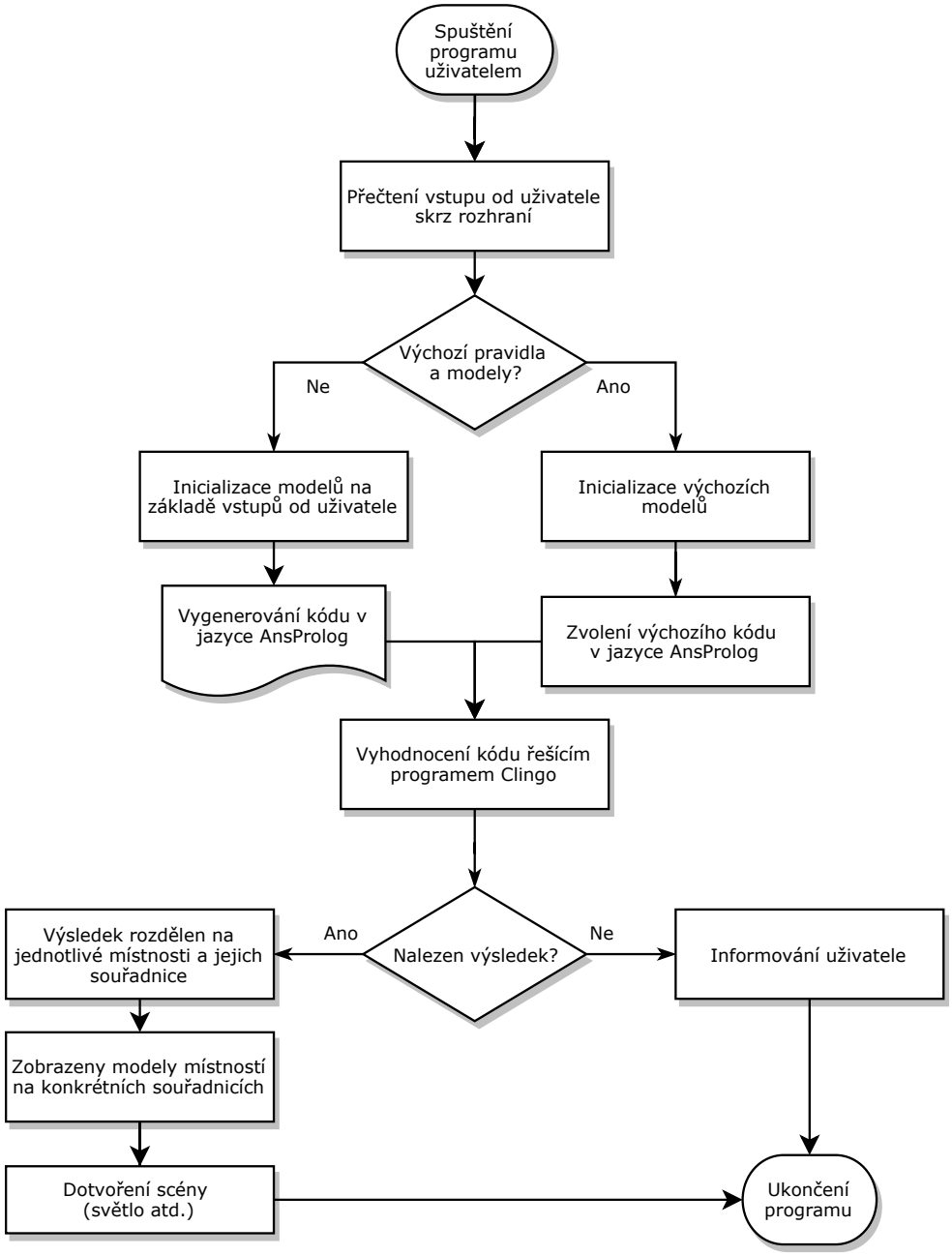
Návrh uživatelského rozhraní

Uživatelské rozhraní bude řešeno jako okno (v *Blenderu* nazývané „panel“) v části označované jako *T-bar*[21]. Tyto panely jsou v základu úzké a umístěné po okrajích obrazovky, navíc jsou často používány s potřebou kontrolovat nastavení na samotném modelu. Z těchto důvodů bude panel navrhován spíše do výšky, po vzoru panelů programu, na které jsou uživatelé zvyklí.

Nejdříve bude uživatel moct zadat informace k samotnému hradu obecně, tzn. jeho šířku a výšku. Dále si bude moct zvolit, zda bude chtít vygenerovat hrad podle výchozího chování, nebo podle vlastních pravidel.

Obrázek 4.3 zobrazuje návrh položky pro vlastní pravidla jedné místnosti a položky pro pravidlo vzdálenosti. První z nich obsahuje definování modelu, možnost nakládání s místností jako s dvoupatrovou, omezení množství a povolení/zakázání konkrétních pater. Na druhém jsou zobrazena dvě pole pro různé identifikátory místností, na které se bude pravidlo vzdálenosti vztahovat, a horizontální i vertikální omezení vzdáleností.

[21] *T-bar* – možnosti zobrazené/skryté po zmáčknutí klávesy *T*.



Obrázek 4.4: Diagram procesu generování

Kapitola 5

Realizace

Tato kapitola v sekci 5.1 představuje implementaci zvolené metody, sekce 5.2 popisuje zrealizované rozhraní pluginu a v sekci 5.3 jsou ukázány vytvořené modely místností a výsledek generátoru.

5.1 Implementace zvolené metody

Plugin je napsán v jazyce Python 3.5 (verze Pythonu určená pro *Blender 2.79*), který je využit pro práci s uživatelským rozhraním, pro převod uživatelem definovaných parametrů, generování souboru pro externí řešící program a pro zobrazování jeho výsledků.

Samotná tvorba textové reprezentace hradu je řešena metodou ASP. K tomu je třeba soubor v jazyce AnsProlog a k němu řešící program *Clingo*, který je v průběhu běhu pluginu žádán o výsledky. Ze všech možných je náhodně vybrán pouze jeden (na základě náhodného čísla předaného řešícímu programu) a následně je zobrazen pluginem přímo v prostředí programu *Blender*.

5.2 Uživatelské rozhraní

Vzhled polí, tlačítek a jiných částí je dán nabízenými funkcemi *Blenderu* pro tvorbu uživatelského rozhraní, nicméně tyto funkce nabízí též možnosti pro vlastní rozložení jednotlivých prvků do sloupců a řádků, což umožňuje přehlednější rozmístění prvků na ploše.

Součástí uživatelského rozhraní generátoru hradu je určení rozměrů (počet místností na výšku a šířku), možnost zvolení výchozího chování generátoru, možnost přidání vlastních místností a vzdáleností a tlačítko pro spuštění generace. Oproti návrhu chybí možnost označení místnosti jako dvoupatrové a určení vertikální vzdálenosti mezi místnostmi.

Obrázek 5.1: Položka v seznamu místností

Obrázek 5.2: Položka v seznamu vzdáleností

Seznam místností

Uživatel může přidat libovolný počet vlastních modelů místností. K tomu slouží jednoduché tlačítko pro přidání modelu a jeho pravidel do kolekce. Položka v seznamu místností obsahuje:

- informaci o ID (index v kolekci),
- cestu k vlastnímu modelu – pokud cesta není zadána, je použita prázdná místnost,
- minimální a maximální počet místností daného modelu v hradu – horní hranice není omezena,
- povolená/zakázaná patra – uživatel může sám zvolit, zda chce zadat povolená, nebo zakázaná patra (např. z důvodu kratšího výpisu jednoho či druhého) a v dalším poli je očekáván seznam jednotlivých pater oddělených čárkou.

Seznam vzdáleností

Uživatel může přidat pravidla pro vzdálenosti mezi jednotlivými typy místností. K tomu taktéž slouží tlačítko pro přidání nového pravidla. Položka v seznamu vzdáleností obsahuje:

- informaci o ID dvou různých místností,
- minimální a maximální vzdálenost mezi jednotlivými typy místností.

Varování, chyby

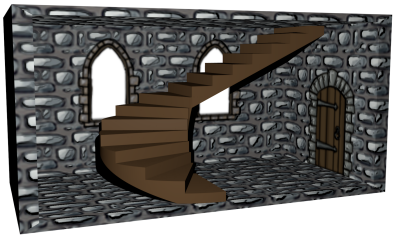
Plugin pracuje i s varovným hlášením a doporučením v případě chyb či jiných problémů. Konkrétně detekuje případy, kdy:

- řešící program nenalezl žádný výsledek → v takovém případě doporučuje změnu nastavení,
- součet minimálního počtu všech typů místností je větší než celkový počet místností hradu → upozorní, že je problém v tomto součtu,
- součet maximálního počtu všech typů místností je menší než celkový počet místností hradu → upozorní, že je problém v tomto součtu
- nesprávný řetězec psaný uživatelem v místě seznamu místností:
  - řetězec není ve správném formátu → generování není umožněno, uživatel je upozorněn na nesprávný vstup,
  - v řetězci je mezi čísly jedno či více písmen → písmena jsou ignorována.

5.3 Modely místností

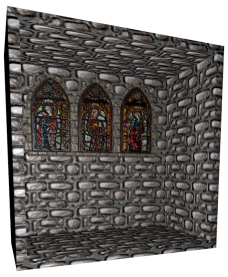
Součástí pluginu jsou i modely základních místností, které generátor využívá pro tvorbu hradu. Veškeré textury jsou ručně vytvořené pro potřeby tohoto prototypu (s výjimkou fotografie barevného skla pro texturu kaple [41]) a modely obsahuji několik vlastních i volně dostupných modelů (jmenovitě trůn [42] a schodiště [43]). Cílem fáze modelování nebylo vytvořit realistické ani kompletně vybavené místnosti, ale modely od sebe snadno rozlišitelné pro snadné pozorování chování generátoru.

Aktuální prototyp generátoru potřebuje pro správné rozmístění místnosti tvořené pouze jedním objektem. Je tedy třeba nejdříve vše sloužit do jednoho.



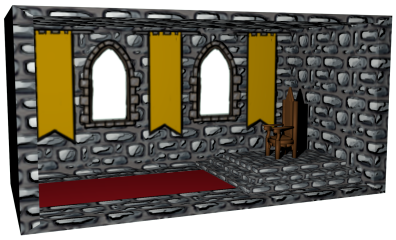
Schodiště

- počet je odvozen od výšky hradu
- v každém patře kromě posledního



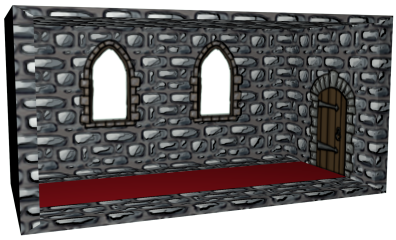
Kaple

- pouze jedna
- od druhého patra výše



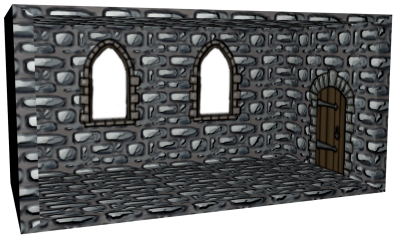
Velký sál

- právě jednou
- ve druhém, nebo třetím patře



Chodba

- neomezený počet
- může být kdekoliv

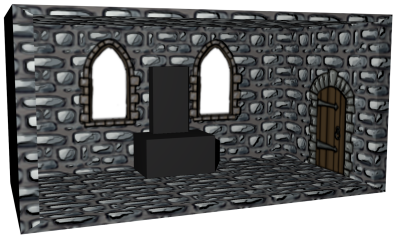


Vstupní místnost

- právě jednou
- v levém dolním rohu hradu

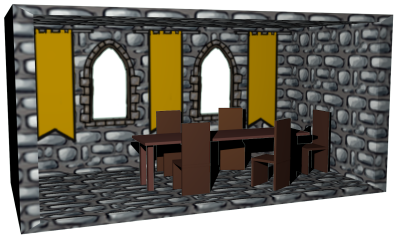
Komnata

- až polovina místností hradu
- od třetího patra výše



Kuchyně

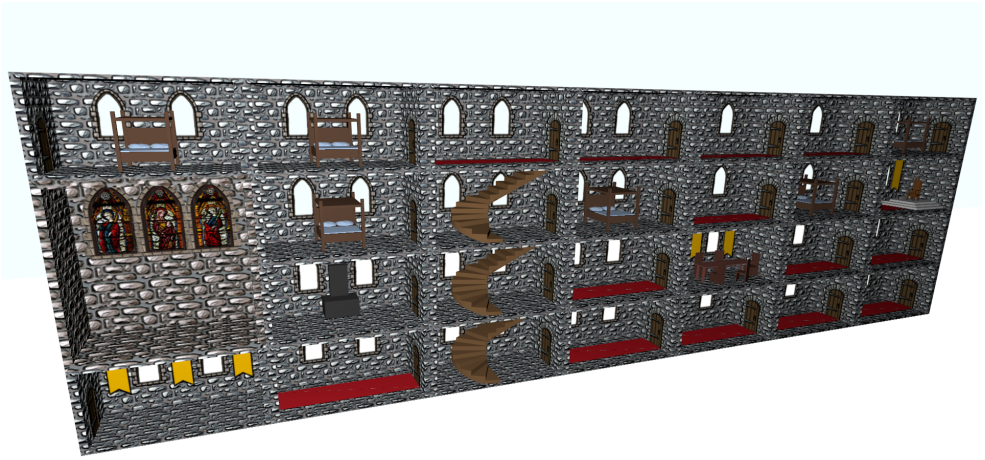
- jedna až dvě
- pouze v nejnižších dvou patrech



Jídelna

- jedna až tři
- pouze v nejnižších dvou patrech

Tabulka 5.1: Modely a pravidla jednotlivých místností



Obrázek 5.3: Jeden z výsledků generátoru.

5.4 Ukázka výsledku generátoru

Na obrázku 5.3 je zobrazen jeden z výsledků prototypu generátoru ve výchozím módu, tedy s výchozími modely a dle pravidel představených v předchozích částech tohoto textu (schodiště nad sebou, kuchyně a jídelna od sebe vzdáleny maximálně 4 bloky, kaple ve vyšším než prvním patře, vstupní místnost v levém dolním rohu atd.).

Prototyp v této fázi nedokresluje vzhled hradu (cimbuří, věž, bránu…), pouze staví budovu na plochu představující zem a za ni dá plochu s texturou pozadí. Doplnění např. o věž může být vyřešeno přidáním pravidla do generovaného souboru s možností nastavitelné výšky a bude s ní nakládáno jako s další místností, stejně jako s hradbami, kde jednotlivé bloky mohou být postaveny na poslední patro.

Kapitola 6

Testování

V této kapitole jsou představeny materiály k testování pluginu – persona 6.1, scénáře 6.2 a dotazník 6.3. Fáze samotného testování proběhne až po odevzdání této bakalářské práce. Měla by proběhnout na akci pro středoškolské studenty a mimo ni s pomocí několika vysokoškolských studentů. Vzhledem k věku testovaných budou otázky i zadání úkolů psána ve formě tykání.

6.1 Persona

Vývojář

Člověk, který potřebuje vytvořit budovu s rozmístěním místností podle vlastních pravidel a s vlastními modely, neboť potřebuje dodržet design konkrétního projektu. Např. chce tvořit hru, ve které potřebuje budovu mj. s pokojem a místností s chemikáliemi. Tyto dvě místnosti musí být v dostatečné vzdálenosti od sebe, aby hráč přespávající do dalšího dne během noci nezemřel na otravu, než najde způsob, jak se chemikálie zbavit. Zároveň má však hra nabízet výhodu znovuhratelnosti a tedy pokaždé jinou budovu.

Tato pravidla si tedy vývojář může nastavit sám podle potřeby a nemusí řešit, jak tato pravidla konkrétně zapsat v daném jazyce a jak podle nich výsledek vygenerovat.

6.2 Scénáře

Scénáře se skládají z informací pro testera s popisem úkolu a případnými daty k zadání a z konkrétního postupu pro moderátora. Výchozí situací je již aktivovaný plugin v prostředí programu *Blender* – předpokládá se, že uživatel používající tento generátor již ví, jak pluginy do prostředí instalovat a aktivovat, případně je naveden uživatelskou příručkou.

jméno souboru	min-max počet	patra povolena/zakázána	seznam pater
`corridor.obj`	1-3	zakázána	1
`storeroom.obj`	*neomezeně*	povolena	1,2,4
`stairs.obj`	3-10	*nezáleží*	–

Tabulka 6.1: Data k testovacímu scénáři č. 2

Scénář 1: Spustit výchozí generátor

Úkolem testovaného je vygenerovat hrad pomocí výchozího nastavení generátoru bez zasahování do nastavení. Dalším úkolem ihned následujícím je změnit rozměry vygenerovaného hradu a znovu spustit generování.

Konkrétní postup:

1. Zakliknout políčko „Default rules“.
2. Kliknout na tlačítko „Generate Castle“.
3. Nastavit jiné rozměry v prvním řádku (položky „width“ a „height“).
4. Znovu kliknout na tlačítko „Generate Castle“.

Scénář 2: Nahrát vlastní místnosti a nastavit pravidla

Úkolem testovaného je přidat tři vlastní modely místností do hradu o šířce i výšce čtyř polí a nastavit jim pravidla dle tabulky 6.1.

Konkrétní postup:

1. Pokud není splněno, nastavit rozměry hradu 4x4.
2. Zrušit zakliknutí políčka „Default generation“.
3. Po každou položku v tabulce 6.1:
   a) Kliknout na tlačítko „Add room“.
   b) Vložit cestu k souboru do horního pole položky v kolekci.
   c) Nastavit minimální a maximální počet.
   d) Povolit/zakázat patra v rozklikávacím poli podle tabulky.
   e) Zapsat seznam pater.
4. Kliknout na tlačítko „Generate Castle“.

Chyba, která může nastat:

- špatný formát seznamu pater → nutno zkontrolovat, zda je formát ve tvaru *číslo,číslo,...* – plugin by na toto měl sám upozornit

6.3 Dotazník

Vzhledem k povaze některých úkolů je vhodné zjistit o testovaných několik informací. Tyto informace mohou být předány ústně moderátorovi po testování, nebo pomocí psaného dotazníku.

- Máš zkušenosti s programem *Blender* a jeho pluginy?
- Máš zkušenosti s jiným modelovacím programem? Pokud ano, s kterým?
- Byl plugin srozumitelný? Pokud ne, co konkrétně nebylo?

Závěr

Cílem této práce bylo seznámit se základními principy procedurálního generování a s vývojem hradů, navrhnout sadu pravidel a vytvořit generátor hradu jako plugin do modelovacího programu *Blender*, parametrizovatelný uživatelem a generující náhodné rozmístění místností.

V rešerši bylo kromě vývoje hradů představeno několik přístupů a metod ke generování obsahu, vybrané hry využívající procedurální generování a podobné existující projekty zabývající se rozložením místností či obecněji stavbou budov. Metody byly taktéž rozděleny podle oblasti využití a pro další zájemce o procedurální generování tedy může jít o jednoduchý rozcestník představující možnosti, kudy se vydat v dalším průzkumu. Některé metody byly zanalyzovány a pro pozdější implementaci byla vybrána jedna z nich, méně častá, nicméně pro generování interiéru taktéž vhodná, jak bylo dokázáno v prototypu.

Dále byly zjištěny základní místnosti používané v hradech a na jejich základě byla navržena pravidla a vytvořeny modely. Z modelů a se znalostí pravidel nakonec byl vytvořen plugin pro *Blender* využívající metodu ASP a náhodně generující rozložení místností hradu s možností přidat vlastní modely a pravidla. Výchozí chování tohoto generátoru tvoří hrad splňující omezení nalezená při rešerši hradů.

Testování z časových důvodů neproběhlo, nicméně bude uskutečněno po odevzdání práce.

Rozšíření do budoucna

Generátor je uzpůsoben, aby mohl být snadno přeměněn např. v generátor domů či fantasy budov (změna výchozích místností na modernější podobu či místnosti s magickými prvky a přidání dodatečných úprav scény, např. střechu, okolní prostředí atd.).

Navrženou sadu pravidel i myšlenku jejich řešení pomocí ASP lze využít jako základní kámen pro tvorbu komplexních budov s interiérem, kde může být

tímto přístupem řešena fáze rozložení typů místností v prostoru. Výsledek této fáze pak může být použit jako informace určující postup dalších algoritmů, např. pro vkládání konkrétních modelů, či dělení prostoru a další procedurální generování samotných místností založených na konkrétním typu (v ložnici postel a skříň, v jídelně lednice, stůl a židle...).

Dalším rozšířením této práce může být kompletní dokončení prototypu a tedy změna v hotový projekt přidáním více pravidel a lepším přizpůsobením různým uživatelským vstupům, jako např. jiný rozměr základního modelu místnosti, přidání více modelů pro jeden typ místnosti a jejich náhodný výběr, určení vzdálenosti i do výšky atd.

Dalším krokem by taktéž mohlo být generování hradů i do hloubky, tedy skutečně v trojrozměrném prostoru, nalezený vhodného způsobu reprezentace dat a tvorba ASP pravidel rozšířených o další prostor.

Bibliografie

1. SHORT, Tanya X. a Tarn Adams. *Procedural Generation in Game Design*. CRC Press, 2017. ISBN 9781351642910.
2. DOULL, Andrew. The Death of the Level Designer. *ASCII Dreams: A roguelike developer's diary* [online]. 2008 [cit. 6. květ. 2019]. Dostupné z: `http://njema.weebly.com/uploads/6/3/4/5/6345478/the_death_of_the_level_designer.pdf`.
3. SHAKER, Noor; TOGELIUS, Julian; NELSON, Mark J. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research* [online]. Springer, 2016 [cit. 15. květ. 2019]. Dostupné z: `http://pcgbook.com/`.
4. Why is this Procedural Content Generation while that other isn't? In: *Game Development Stack Exchange* [online]. 2018 [cit. 20. břez. 2019]. Dostupné z: `https://gamedev.stackexchange.com/questions/154583/why-is-this-procedural-content-generation-while-that-other-isnt`.
5. Procedural Content Generation Wiki [online]. 2019 [cit. 20. dub. 2019]. Dostupné z: `http://pcg.wikidot.com/`.
6. HENDRIKX, Mark; MEIJER, Sebastiaan; VAN DER VELDEN, Joeri; IOSUP, Alexandru. Procedural Content Generation for Games: A Survey. *ACM Trans. Multimedia Comput. Commun. Appl.* 2013, roč. 9, č. 1, s. 1:1–1:22. ISSN 1551-6857. Dostupné z DOI: `10.1145/2422956.2422957`.
7. M. MATEAS, Smith A. M. a. Answer Set Programming for Procedural Content Generation: A Design Space Approach. *IEEE Transactions on Computational Intelligence and AI in Games*. 2011, roč. 3, č. 3, s. 187–200. ISSN 1943-068X. Dostupné z DOI: `10.1109/TCIAIG.2011.2158545`.

8. TOGELIUS, Julian; KASTBJERG, Emil; SCHEDL, David C.; YANNAKAKIS, Georgios N. What is Procedural Content Generation?: Mario on the Borderline. In: *Proceedings of the 2Nd International Workshop on Procedural Content Generation in Games*. Bordeaux, France: ACM, 2011, 3:1–3:6. PCGames '11. ISBN 978-1-4503-0872-4. Dostupné z DOI: 10.1145/2000919.2000922.

9. PELL, Barney. METAGAME in SymmetricChess-Like Games [online]. 1992 [cit. 13. květ. 2019]. Dostupné z: http://www.kmjn.org/pdf/UCAM-CL-TR-277.pdf.

10. EVANS, Martin. *What Is Procedural Generation?* [online]. 2012 [cit. 19. břez. 2019]. Dostupné z: http://martindevans.me/heist-game/2012/11/18/What-Is-Procedural-Generation/.

11. MOTION TWIN. Let's talk about procedural generation – VLOG 2 – Mars 2017. In: *Youtube* [online]. 2017 [cit. 10. dub. 2019]. Dostupné z: https://www.youtube.com/watch?v=tyMrRW-Li_I&feature=youtu.be. Oficiální kanál francouzského nezávislého studia Motion Twin.

12. TOGELIUS, Julian; YANNAKAKIS, Georgios N.; STANLEY, Kenneth O.; BROWNE, Cameron. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*. 2011, roč. 3, č. 3, s. 172–186.

13. BYCER, Josh. *3 Failings of Procedurally Generated Game Design* [online]. 2019 [cit. 10. dub. 2019]. Dostupné z: http://game-wisdom.com/critical/procedural-game-design.

14. TOGELIUS, Julian; JUSTINUSSEN, Tróndur; HARTZEN, Anders. Compositional Procedural Content Generation. In: *Proceedings of the The Third Workshop on Procedural Content Generation in Games*. Raleigh, NC, USA: ACM, 2012, 16:1–16:4. PCG'12. ISBN 978-1-4503-1447-3. Dostupné z DOI: 10.1145/2538528.2538541.

15. AL., Togelius et. *Why is this Procedural Content Generation while that other isn't?* [online]. 2009 [cit. 11. dub. 2019]. Dostupné z: https://groups.google.com/d/topic/proceduralcontent/TUVfuus1zKw/discussion groups.google.com.

16. LIFSCHITZ, Vladimir. What is Answer Set Programming? In: *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*. Chicago, Illinois: AAAI Press, 2008, s. 1594–1597. AAAI'08. ISBN 978-1-57735-368-3. Dostupné také z: http://dl.acm.org/citation.cfm?id=1620270.1620340.

17. JOHNSON Lawrence, Georgios N. Yannakakis a Julian Togelius. Cellular Automata for Real-time Generation of Infinite Cave Levels. In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. Monterey, California: ACM, 2010, 10:1–10:4. PCGames '10. ISBN 978-1-4503-0023-0. Dostupné z DOI: 10.1145/1814256.1814266.

18. HOLUB, Jan. *Automaty a gramatiky: Základní pojmy* [online]. 2018 [cit. 11. květ. 2019]. Dostupné z: https://courses.fit.cvut.cz/BI-AAG/lectures/bi-aag-01-zakladni_pojmy.pdf [Soubor přístupný po přihlášení do sítě ČVUT].

20. BARTON, Matt a Bill Loguidice. *The History of Elite: Space, the Endless Frontier* [online] [cit. 19. dub. 2019]. Dostupné z: http://www.gamasutra.com/view/feature/3983/the_history_of_elite_space_the_.php.

21. SPUFFORD, Francis. *The Guardian*. Masters of their universe [online]. 2003 [cit. 19. dub. 2019]. Dostupné z: https://www.theguardian.com/books/2003/oct/18/features.weekend.

22. *Dwarf Fortress*. Features [online] [cit. 19. dub. 2019]. Dostupné z: http://www.bay12games.com/dwarves/features.html.

23. FENLON, Wes. *Great moments in PC gaming: Accomplishing literally anything in Dwarf Fortress* [online]. 2019 [cit. 19. dub. 2019]. Dostupné z: https://www.pcgamer.com/great-moments-in-pc-gaming-accomplishing-literally-anything-in-dwarf-fortress//.

24. *SpeedTree* [online]. Interactive Data Visualization, Inc., 2017 [cit. 28. břez. 2019]. Dostupné z: https://store.speedtree.com/.

25. BENARD, Sebastien. *Building the Level Design of a procedurally generated Metroidvania: a hybrid approach* [online] [cit. 19. dub. 2019]. Dostupné z: http://www.gamasutra.com/blogs/SebastienBENARD/20170329/294642/Building_the_Level_Design_of_a_procedurally_generated_Metroidvania_a_hybrid_approach.php.

26. ŠEFCŮ, Ondřej. *Architektura: lexikon architektonických prvků a stavebního řemesla*. Praha: Grada, 2013. ISBN 978-80-247-3120-9.

27. *O hradech* [online]. David Mikoláš, 2019 [cit. 28. břez. 2019]. Dostupné z: http://www.ohradech.eu/.

28. FORBECK, Matt a David Noonan. *Stronghold Builder's Guidebook*. Wizards of the Coast, 2002. ISBN 0786926554. Dostupné také z: https://digitalniknihovna.mlp.cz/mlp/view/uuid:2fce5290-3a33-11e2-af6c-0030487be43a?page=uuid:20002540-3a3e-11e2-ad63-0030487be43a.

29. CEPERO, Miguel. *Hello Worley* [online]. 2011 [cit. 19. dub. 2019]. Dostupné z: http://procworld.blogspot.com/2011/05/hello-worley.html.

30. SMITH, Adam M. *A Map Generation Speedrun with Answer Set Programming* [online]. 2011 [cit. 28. břez. 2019]. Dostupné z: https://eis-blog.soe.ucsc.edu/2011/10/map-generation-speedrun/.

31. *The World of Notch* [online]. Markus „Notch" Persson, 2011 [cit. 28. břez. 2019]. Dostupné z: https://notch.tumblr.com/post/3746989361/terrain-generation-part-1.

32. SMITH, Adam. Answer Set Programming in Proofdoku. 2017. Dostupné také z: https://aaai.org/ocs/index.php/AIIDE/AIIDE17/paper/view/15885.

33. PARISH, Yoav I. H. a Pascal Müller. Procedural Modeling of Cities. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. New York: ACM, 2001, s. 301–308. SIGGRAPH '01. ISBN 1-58113-374-X. Dostupné z DOI: 10.1145/383259.383292.

34. PROFENIX STUDIO SRLS. *Building Generator* [software]. 2017 [cit. 13. květ. 2019]. Dostupné z: https://www.s2powered.com/copiasito/.

35. IBELE, Tyson. *Building Generator* [software]. 2009 [cit. 13. květ. 2019]. Dostupné z: http://tysonibele.com/building-generator/.

36. STOCKER, Jasper. *BuildR 2: Procedural Building Generator* [software]. 2017 [cit. 14. květ. 2019]. Dostupné z: https://assetstore.unity.com/packages/tools/modeling/buildr-2-procedural-building-generator-82220.

37. ŽATEČKA, Martin. *Floor Plan Generator* [software]. 2016 [cit. 14. květ. 2019]. Dostupné z: http://www.studioflorian.com/projekty/317-martin-zatecka-floor-plan-generator.

38. GAVRILOV, Egor. *Magnetizing Floor Plan Generator* [software]. 2019 [cit. 13. květ. 2019]. Dostupné z: https://toolbox.decodingspaces.net/magnetizing-floor-plan-generator/.

39. *Popis a dějiny obce a hradu Křivoklátu*. Křivoklát: Nitsch, 1895. Dostupné také z: https://digitalniknihovna.mlp.cz/mlp/view/uuid:2fce5290-3a33-11e2-af6c-0030487be43a?page=uuid:20002540-3a3e-11e2-ad63-0030487be43a.

40. *Clingo, version 5.3.0* [software]. University of Potsdam, 2018 [cit. 13. květ. 2019]. Dostupné z: https://potassco.org/.

41. VASSIL. *Wikipedia: Medieval stained glass*. Detail of a medieval window at Troyes Cathedral, France (14th century) [fotografie]. 2008 [cit. 19. dub. 2019]. Dostupné z: %7Bhttps://en.wikipedia.org/wiki/Medieval_stained_glass#/media/File:Vitrail_Cath%C3%A9drale_Troyes_150208_01.jpg%7D.

42. GODOFTHEFOOLARCANA. *TurboSquid*. Simple throne [3D model]. 2012 [cit. 13. květ. 2019]. Dostupné z: `https://www.turbosquid.com/FullPreview/Index.cfm/ID/702871` [ID modelu: 702871].

43. VLADIMIR. *Archive3D*. Stair 3D Model [3D model] [cit. 14. květ. 2019]. Dostupné z: `https://archive3d.net/?a=download&id=a9572931`.

Příloha **A**

Seznam použitých zkratek

AAA neformální pojem pro hry s velkorozpočtovou produkcí

ASP (Answer Set Programming) programování množinou dotazů

BSP (Binary Space Partitioning) binární dělení prostoru

NPC (Nonplayable Character) postava obývající herní svět, ale neovládaná hráčem

PCG (Procedural Content Generation) procedurální generování obsahu

PCG-G (Procedural Content Generation for Games) procedurální generování obsahu do her

PG (Procedural Generation) procedurální generování

PGC (Procedurally-generated Content) procedurálně generovaný obsah

Příloha **B**

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu CD
- `addon` ..... adresář s archivem připraveným pro instalaci do *Blenderu*
- `src` ..... zdrojové kódy
  - `impl` ..... zdrojové kódy implementace
    - `models` ..... modely a textury
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `images` ..... složka s obrázky
- `text` ..... složka obsahující soubory k práci
  - `BP-Svickova-Petra-2019.pdf` ..... práce ve formátu PDF