Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Bakalářská práce

Procedurální generování pro hru "Bludiště"

Ondřej Štembera

Vedoucí práce: Michal Lukáč

26. května 2016

Poděkování

Rád bych poděkoval Ing. Michal Lukáč za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce. Dále bych chtěl poděkovat rodině a přátelům za jejich podporu v průběhu studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) 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, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

Prague dne 26. května 2016 .....

České vysoké učení technické v Praze
Fakulta elektrotechnická

Odkaz na tuto práci

Štembera, Ondřej. *Procedurální generování pro hru "Bludiště"*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická, 2016.

Abstrakt

Práce se zaměřuje na porovnání metod procedurálního generování úrovní ve hrách za účelem zjištění a porovnání metod k získání přehledu nad rozdílem metod a jejich možným výběrem v závislosti na jejich limitech a výhodách. Nejdříve budou zmíněny základní informace o procedurálním generování obsahu (PCG). Dále metody pro vytváření úrovní do her (převážně pro tvorbu dungeonů). A následně vyberu tři z nejlépe aplikovatelných metod pro naši definici hry a uvedu z jakého důvodu by měly být použity a porovnám je s ostatními. Tyto tři vybrané metody pak budou použity v experimentu, který bude zjišťovat rozdíly mezi metodami z pohledu hráčů (účastníků experimentu).

Abstract

The work focuses on the comparison of methods for procedural generation of levels in games in order to determine and compare the methods to obtain an overview of the differences of methods and their possible selection depending on their limits and advantages. First, we will cover basic information about the procedural content generation (PCG). Continue with methods used for creating levels in games (mostly for creating dungeons). And then select the three most suitable methods for our definition of game and admit why they should be used and compare them with each other. That methods will be used

in an experiment which will try to find out differences between methods from the perspective of the players (participants of the experiment).

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Procedurální generování obsahu (PCG)

Z počátku byly hry vytvářeny tak, že všechen obsah do her byl tvořen ručně, což vedlo k velké časové náročnosti, která je směřována do aktivit, které většinou nejsou obtížné. Například umísťování stromů po jednom do prostoru nebo jejich ruční vytváření nezávislé na ostatních stromech, zároveň tato aktivita není nijak výrazně kreativní, tím pádem zbytečně odpoutává pozornost designera od důležitých věcí a ubíjí jeho kreativitu. Se zlepšením technologií a možností grafiky se také zvýšila náročnost vytváření, a tím i nutnost nových možností jak rychleji vytvořit obsah pro hry. To řeší procedurálním generování (Procedural Content Generation), které se zaměřuje na vytváření obsahu procedurálním způsobem s použitím náhodnosti. Například, pokud by byly stromy umísťovány pomocí PCG, poté by nebyla úplná kontrola nad místem umístění stromu ani nad stromem jako takovým, ale zároveň by došlo k ušetření spousty času. Dále by byla možnost upravit parametry generování, udělat změny a úpravy velmi rychle. Kdežto v manuálním případě by se musela většina práce znovu opakovat.

V procedurálním generování úrovní jsou hlavními výhodami zkrácení času, znovu-hratelnost, ušetření peněz, rozmanitost, adaptace prostoru k hráči. Znovu-hratelnost je dosažená tím, že procedurální generování obsahuje pseudo-náhodnost a pomocí ní dochází k tomu, že hráč nepřichází o zážitek z opakování stejné úrovně tím, že by znal celý průchod nebo postup k cíli dané úrovně. Možnou nevýhodou a také důvodem, proč není někdy využíváno procedurální generovaní úrovní je částečná ztráta kontroly nad vytvořeným prostorem. Navíc designer nemusí z počátku vědět, jak si představuje, že prostor bude vypadat a může být pro něj těžké určit parametry specifikující tento prostor a jeho generování.

Motivace a Cíl práce

Z důvodu nárůstu potřeby procedurálního generování úrovní. Kvůli zvýšení časové náročnosti vytvářených prostorů byla vytvořena spousta metod zabírající se specifickými typy prostorů a úrovní. Nedošlo ale k zjištění možností jejich použití a porovnávání. Není jednoduché vybrat správně nebo s jistotou určit možnosti dané metody. Navíc může mít designer zkreslený pohled na věc a názor hráče je ve finále důležitější.

V této práci se budeme zaměřovat na metody pro generování úrovně pro určitý případ a jejich porovnání z pohledu hráčů. Dojde k popisu dostupných metod. Vybrání tří metod nejlépe splňující náš případ hry a jejich následné implementování. Navrhnutí experimentu zjišťujícího možnosti porovnání těchto tří metod a jeho vyhodnocení.

Typ hry a prostoru je v rychlosti zmíněn níže. Kompletní definice prostorů a hry pro kterou se bude v našem případě vybírat metoda generování je kompletně uvedena v GDD (Game Design Dokumentu) v sekci A.

Definice hry

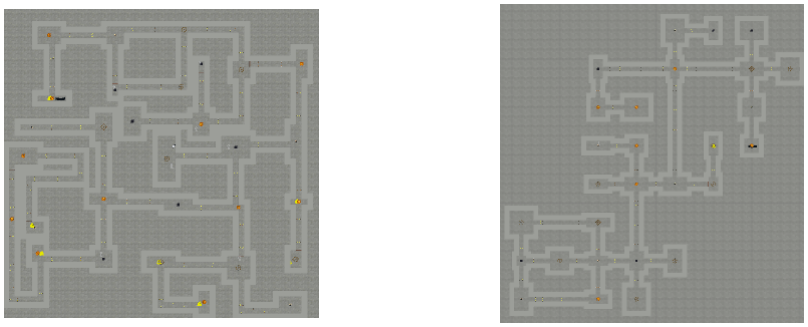
Hra bude probíhat v 3D prostoru obsahujícím obdélníkové místností s jejími propojeními a bude hrána s pohledu hráče (first person view). Místnosti můžou mít pouze na každé straně jednu cestu, tím pádem pouze čtyři celkově. Cílem je nalézt portál a tím z něj utéct. Vygenerovaný prostor by tomu měl v závislosti na nastavené obtížnosti hráči co nejvíce znepříjemnit. Je to vlastně atypické bludiště, kde hráč bloudí místnostmi nežli mezi zdmi a chodbami. Na cestě z bludiště se hráč snaží nalézt co nejvíce peněz a zároveň musí hledat klíče k zamčeným dveřím, aby se mohl posunout dál.

Hra bude probíhat na mobilním telefonu a jednou z podmínek je, aby hry byly vygenerované přímo na zařízení a to v čase, který je hráč schopen akceptovat (maximálně 5 vteřin nejlépe 1 - 2 vteřiny). Vytvořený prostor by měl obsahovat boční cesty (rozcestí) a v některých případech také cyklus pro zvýšení obtížnosti. Algoritmus musí umožňovat možnost přidání peněz, klíčů a zamčených dveří. Dále by měl 3 typy obtížností. Nejlépe nezaložených pouze na změně velikosti vygenerovaného prostoru, ale i na jiných parametrech. Dále hráč bude mít přehled o času, počtu peněz a klíčů, které již sebral. A jeho skóre se odvíjí od času, který strávil v labyrintu a počtu peněz, které posbíral. Protože sám z vlastní zkušenosti vím, že nejlepší je názorný příklad, tak je na obrázku 0.1 a 5.7 vidět, jak bude vypadat pohled hráče a příklady prostorů.

Struktura dokumentu

V práci nejdříve uvedu metody pro procedurálního generování úrovní, jejich výhody a nevýhody 1. Zhodnotím a vyberu 3 metody nejlépe splňující pod-

Obrázek 0.1: Pohled hráče



Obrázek 0.2: Příklady prostorů

mínky našeho zadání hry 2. Navrhnu experiment, který se nejdříve bude snažit možnosti k porovnání těchto metod pomocí kvalitativní studie. Ze zjištění navrhnu kvantitativní experiment porovnávající metody již podle specifických charakteristik 3 . Dále popíši, jak jsem implementoval jednotlivé metody a technický návrh hry 4. Pak vyhodnotím výsledky obou experimentů 5 a zhodnotím, jak si která metoda vedla 5.2.3.

Kapitola 1

Metody PCG

Nejdříve budou zmíněny metody, které jsou k dispozici, jejich výhody a zápory. Následně bude provedeno porovnání a bližší popis vybraných metod. Níže jsou zmíněny všechny metody, které budeme brát v úvahu.

Tyto metody jsem získal nastudováním literatury zadané vedoucím práce.

1.1 Všechny metody

- Rozdělování prostoru [Togelius et al., 2015] 1.2
- Metoda založená na vytváření prostoru agentem [Togelius et al., 2015] 1.3
- Cellular automata [Togelius et al., 2015] 1.4
- Gramatikou řízené generování 1.5
  - Prostorová gramatika [Dormans, 2010] 1.5.1
  - Podle Designing Procedurally generated levels [Linden et al., 2013] 1.5.2
- Genetické algoritmy 1.6
  - Towards Supporting Stories with Procedurally Generated Game Worlds [Hartsook et al., 2011] 1.6.1
  - Evolving Dungeon Crawler Levels With Relative Placement [Valtchanov and Brown, 2012b] 1.6.2
- Metoda používaná v Game Level Layout from Design Specification [Ma et al., 2014] 1.7
- Constraint-Based [Togelius et al., 2015] 1.8

- Pokročilé metody pro generování platformových her [Togelius et al., 2015] 1.9
  - rytmem řízení generování 1.9.1
  - Occupancy regulated extension (ORC) 1.9.2
- Procedural natural systemes for game level design [Huijser et al., 2010] 1.9.3
- A Survey on Procedural Modelling for VirtualWorlds [Smelik et al., 2014] 1.9.4

1.2 Rozdělování prostoru (Space partitioning for dungeon generation)

Rozdělování prostoru funguje na způsobu postupného dělení plochy (převážně 2D), následného skládání a propojování vzniklých podprostorů. Plocha se nezmenšuje ani nezvětšuje, pořád je celá obsažená v podmnožinách vytvořených rekurzivním dělením. Tyto podprostory můžeme ukládat do stromu (space partitioning tree), kde kořenem je celá plocha a listy obsahují nejmenší rozdělené kusy. Vstupem této metody může být výška, šířka prostoru a velikost nejmenší možné plochy nebo obrázek (image). V případě prvního vstupu se prostor rekurzivně dělí, dokud není daný kus menší než nejmenší možná plocha uvedená vstupem algoritmu. V případě vstupu obrázku(image) se nemění, pouze to kdy bude ukončeno dělení, a to v případě vstupu obrázku dochází ve chvíli, kdy rozdělený kus obsahuje pouze jednu barvu. Dává nám to určitou kontrolu nad velikostí části úrovně. Pro vytvoření prostoru z rozděleného prostoru je více možných metod. Budou uveden dvě uvedené v článku [Togelius et al., 2015]. První metoda označuje rozdělené kusy jako místnosti (1) a zbytek (0) a přitom hlídá, aby byl celý vzniklý prostor propojen. Druhá metoda vytváří místnosti tak, že vezme každý list a vytvoří v něm náhodně velkou místnost. Tyto místnosti poté propojuje průchodem od listů v stromu ke kořenu a propojením místností, které mají stejného rodiče. Stačí aby bylo propojení kompletní, pokud se totiž bude dělit na čtyři části a ne pouze na dvě (binárně), pak není nutné propojovat všechny, ale pouze sousedící nebo tak jak se uváží za vhodné.

Metoda ze základu vytváří odbočky ve dvou případech. Když aspoň na jedné straně propojení dvou částí je obsaženo více místnosti. Pak může být propojení vedeno z místnosti na cestu v tom případě vždy vzniká odbočka a nebo z místnosti na jinou místnost v tom případě může vzniknou odbočka a to vždy, když se nepropojuje s boční místností nebo pouze někdy když se propojuje s boční místností(Může vzniknout klikatá cesta).

1.3 Metoda založená na vytváření prostoru agentem (Agent-based dungeon growing)

Základem algoritmu je agent obsahující množinu možných aktivit, jenž může provádět a také definované pořadí jak tyto aktivity na sebe můžou navazovat. Nejdříve je agent náhodně umístěn do prostoru a v závislosti na aktuálním stavu se vzhledem k nastaveným pravděpodobnostem rozhoduje, kterým směrem se vydá nebo jestli vytvoří místnost. Algoritmus probíhá v krocích a v každém kroku se podle předchozího výsledku pravděpoděbnosti nastaví ty budoucí. Například bude vetší pravděpodobnost, že agent umístí místnost, protože ji v minulém kroku neumístil. Opakováním zmíněných akcí se vytvoří výsledný prostor. Algoritmus se zastaví ve chvíli, když agent nemůže provést žádnou z akcí. Tento způsob vytváření prostoru umožňuje vytvářet neuspořádané propojené prostory. Pokud chceme zaručit, aby se místnosti nepřekrývaly, pak se musí dopředu kontrolovat, zdali je možné nějakou místnost umístit. Jedním příkladem takového algoritmu může být umístění agenta na náhodnou pozici. Kde pokud agent bude moct umístit nějakou místnost, pak ji umístí. Pokud ne, pak zvolí náhodný směr a pokračuje v něm dokud nenastane šance zatočení nebo dokud nebude možné umístit další místnost. Pokud se agent dostane do situace, kde se nemůže vydat žádnou cestou, a zároveň nemůže vytvořit místnost, pak zanikne a algoritmus skončí.

Pro příklad, jenž je metoda vybírána, potřebujeme určité možnosti nastavení obtížnosti - to lze vyřešit možností nastavení velikosti plochy, na které se agent může pohybovat a také počtem agentů použitých pro vygenerování jedné úrovně. Určitou nejistotou algoritmu může, být že pokud agent nekontroluje více možností cest může nastat, že vytvoří úroveň o malé velikosti. Například, pokud bude vybrán náhodně pouze jeden směr výstupu z vytvořené místnosti, pak může nastat k rychlému konci při umístění agenta k jedné straně prostoru. Naopak při velké kontrole může vznikat dlouhá klikatá chodba z důvodu nemožnosti agenta zahrabat se na místo, ze kterého už neví kudy kam. Vytvoření této chodby může na první pohled vypadat žádané, ale pokud má být bludiště komplikované, pak už nezbývá místo pro další agenty k vytvoření odboček a cyklů. Tomuto problému může být zabráněno parametrem určujícím maximální počet místností, kteří jednotliví agenti můžou vytvořit.

1.4 Cellular Automata

Cellular automata pracuje většinou s dvou dimenzionální sítí. Každý bod v síti může nabývat různých stavů. Na začátku je každý bod náhodně nastaven do určitého stavu, například zeď nebo místnost. Pak probíhá určitý počet iterací algoritmu, při kterém se každý bod přenastavuje do dalšího stavu podle svého aktuálního stavu a jeho okolí (sousedů). Metodu určují parametry - velikost sítě, možné stavy, podmínky pro přechody stavů a jací sousedi jsou bráni v

potaz. Například přechod stavu můžou ovlivňovat sousedi na diagonále nebo také nemusí. Metoda nezaručuje, že bude existovat průchod celým prostorem, můžou se vytvořit místnosti, ke kterým se nedá dostat a proto se tento algoritmus většinou používá pouze pro vytváření místností nebo částí úrovně a ty jsou pak dále propojovány.

Algoritmus generuje přirozeně vyhlížející prostory. Převážně je využíván pro vytváření jeskynních chodeb a prostorů nebo pro hry, ve kterých je umožněno hráči prokopat si cestu zdmi/kameny a zajistit možnost úspěšného ukončení úrovně. Metoda umožňuje rozšiřování vytvořeného prostoru za běhu a hry a tím vytvořit hráči představu nekonečného prostoru.

Pokud by tento algoritmus byl porovnán s určenými požadavky bludiště, pak již z počátku nesplňuje tvar vytvořených místnosti a vždy možnost úspěšného ukončení úrovně. Další možnou nevýhodou může být obtížné nastavení a nízká kontrola nad vygenerovaným obsahem, neumožňuje nám definovat obtížnost jinak, než velikostí prostoru. Metoda nám také nijak neulehčuje práci s klíči a zamčenými dveřmi.

1.4.1 Parametry metody

- zaplněnost plochy kameny v procentech
- počet cellular automata generování (přidávání hodnoty polím)
- práh (threshold) definující, zda je pole cesta nebo kámen
- počet sousedících polí (normálně se používá osm, ale nemusí tomu tak být)

1.5 Gramatikou řízené generování (Grammar-based dungeon generation)

Tento přístup generování se liší od ostatních již uvedených přístupů tím, že se nezaměřuje pouze na vytváření herního prostoru, ale zaměřuje se hlavně na aktivity hráče, jenž mají být vykonány k ukončení úrovně. a podle kterých je následně vytvořen odpovídající prostor. Příkladem jednoduché úrovně může být „najít klíč“-> „otevřít dveře klíčem“ -> „najít portál“. Gramatikou řízené generování jednoduše řečeno popisuje hráčovy budoucí aktivity, jenž musí vykonat a pak podle nich vytváří prostor splňující tyto podmínky. Tím nám dává velkou kontrolu nad generováním, ale to může být ve výsledku zároveň i problémem, pokud gramatika nebude správně nastavená. Výsledek nemusí odpovídat očekávání.

Gramatika používaná pro generování úrovní je stejná jako jazyková gramatika akorát pracuje s aktivitami hráče. Obsahuje všechny možné aktivity hráče a pravidla, jak a co může být nahrazeno nebo přidáno, stejně jako v

jazykové gramatice, ale v tomto případě se týká aktivit hráče. Na rozdíl od normální gramatiky, která pracuje s textem. Pracuje gramatika pro generování úrovní se směrovým acyklickým grafem, kde každá aktivita je bod a vztahy mezi nimi jsou určeny směrovými hranami. Druhů hran může být více, některé hrany můžou určovat, že aktivity takto spojené musí nastat na stejném místě („odemknout truhlu“ -> „vybrat poklad“). Pravidla v gramatice se dělí na dva typy aktivity (terminal, unterminal). První uvedený typ (terminal) je aktivita, která už nelze změnit, to znamená že k ní nepatří žádná podmínka používající tuto aktivity jako vstup. Narozdíl od toho pro unterminal musí existovat aspoň jedna podmínka používající aktivitu jako vstup, ale může jich být i více. Z toho vyplývá, že generování končí v tom případě, když už graf obsahuje pouze terminální aktivity a generování začíná jedním neterminálním symbolem.

Prostor se může přizpůsobovat v průběhu hry hráčovým požadavkům a usměrňovat vytváření jeho aktivit. Například pokud se hráč vyhýbá nepřátelům a nebojuje s nimi, pak se může omezit vytváření místností s nepřáteli. Takto reagovat je umožněno, pokud jsou v grafu aktivit ponechány neterminální aktivity, na které se budou aplikovat transformační pravidla, až když se k nim hráč přiblíží. Metoda umožňuje jednoduché změny funkčnosti algoritmu díky přenastavení gramatiky a možnosti přidání dalších možných aktivit. Pokud by bylo třeba v metodě umožnit vytváření jiných obtížností, pak stačí mít pro každou obtížnost jiná pravidla transformace aktivit.

Graf aktivit musí být ještě převeden do reálného prostoru. Zde budou uvedeny dvě možnosti. První možnost používá Prostorovou gramatiku (Space grammar) viz [Dormans, 2010]. Druhá převádí graf aktivit pomocí podmínek rozmístění mezi jednotlivých uzly a následného převedení grafu na 2D mřížku [Linden et al., 2013].

1.5.1 Prostorová gramatika (Space/Shape grammar)

Tento typ gramatiky se používá pro převedení grafu aktivit na reálný prostor. Funguje na stejném způsobu jako gramatika pro vytváření aktivit, akorát už pracuje s prostorem. Aby bylo možné vygenerovat prostor z grafu aktivit, musí být každá terminální aktivita vstupem aspoň jednoho transformační pravidla v prostorové gramatice. Postupně se aplikují všechna pravidla na graf aktivit, a tím se vytvoří finální prostor úrovně.

1.5.2 Podle Roland van der Linden

Kompletní informace jsou uvedené v [Linden et al., 2013]. Stručný přehled kroků je uveden níže.

1. Ve vytvořeném grafu nahradí aktivity probíhající na stejném místě pouze jedním bodem. A dále definuje body jako místnosti a hrany cestami mezi nimi.

2. Pokud graf není planární, pak je upraven, aby splňoval podmínky planarity.

3. Protože graf může mít více cest než čtyři a algoritmus vyžaduje maximálně na každé straně čtvercové místnosti maximálně jednu cestu, musí být jejich počet zredukován.

4. Následně algoritmus spočítá pro každou místnost její odpovídající Y souřadnici podle podmínek vzájemných vztahů mezi uzly. Například pokud z bodu A vychází cesta na server do B, pak B a všechno následující musí být výše než A a vše následující po A. Souřadnice X vychází z průchodu grafem, a už nemusí být vypočítány. Podmínky určené Y souřadnicemi zaručují, že se nic vzájemně nepřekryje.

5. Následuje algoritmus, který zjistí zda lze zkrátit cesty, které vznikly určováním Y souřadnic.

6. Převedení prostoru na 2D mřížku. Kde se definuje velikost jedné buňky mřížky a velikosti místností obsažených v nich.

1.6 Genetické algoritmy (Genetic algorithms)

Postup nalezení řešení problému je založen na evolučních procesech z biologie. Nejdříve je náhodně vytvořena prvotní populace, ze které jsou vybrání jedinci nejlépe řešící daný problém, a ti jsou nalezeni pomocí fitness funkce, jenž je měřítkem úspěšnosti určitého jedince. Následně v populaci probíhá křížení jedinců a mutace. Tento proces opakovaného výběru, následných mutací a křížení se opakuje, dokud se nedosáhne dostačující kvality řešení nebo pokud neuplyne maximální možná doba průběhu.

Algoritmus umožňuje spoustu možností jeho nastavení. Například nastavovat úroveň tak, aby mezi startovní a koncovou pozicí hráče byla co nejdelší možná cesta nebo zvýšení důležitosti cest se rozpojit a následně se sejít. Další možností je přidání informací o hráči do dat použitých k vytváření prostoru, a tím přizpůsobit herní prostředí potřebám hráče. Tím, že se genetické algoritmy snaží najít nejlepší možné řešení podle určitého nastavení fitness funkce, špatné nastavení této funkce může vést k výsledkům, které nesplňují očekávání.

1.6.1 Towards Supporting Stories with Procedurally Generated Game Worlds

Jedním příkladem takové to metody je „Towards Supporting Stories with Procedurally Generated Game Worlds“ uvedná v [Hartsook et al., 2011]. Metoda se zaměřuje na vytváření světa z definovaného příběhu (mise) pomocí aktivit a míst asociujících s nimi. Metoda používá dva typy míst „mosty“ a „ostrovy“.

Ostrovy jsou místa, kde hráč plní nějakou aktivitu nebo úkol. Mosty jsou místa, která tyto prostory propojují a dávají hráči určitou volnost jít i jinam než-li jenom po hlavní cestě úkolu.

Převedení mise na model probíhá jako optimalizační proces vyvažující dvě podmínky výsledné mapy, a to realistické propojení částí a preference hráče. Genetický algoritmus pracuje ze stromem popisujícím prostor vycházející z příběhu. Kde každý uzel nese informaci o jednom místě. Pro strom je počítána fitness funkce, jenž říká jak moc strom splňuje preference hráče a přirozenost propojení, aby byly propojeny ty části, které se k sobě tématicky hodí (například les a pole).

Algoritmus začne z počáteční množinou náhodně vytvořených stromů, kde mezi každé dva ostrovy je umístěn náhodně jeden most jakéhokoliv typu. Poté se nad stromy provádí mutace a křížení. Pomocí kontroly se zjišťuje jak splňují nově vytvořené nebo upravené stromy podmínky a pokračuje se v dalším křížení a mutování. Poté je strom namapován na 2D mřížku (grid). Pokud to ale není možné, je strom zahozen a začne celý proces od znova. Po úspěšném vytvoření sítě jsou následně aplikovány dekorativní prvky a vytvořena grafická stránka hry.

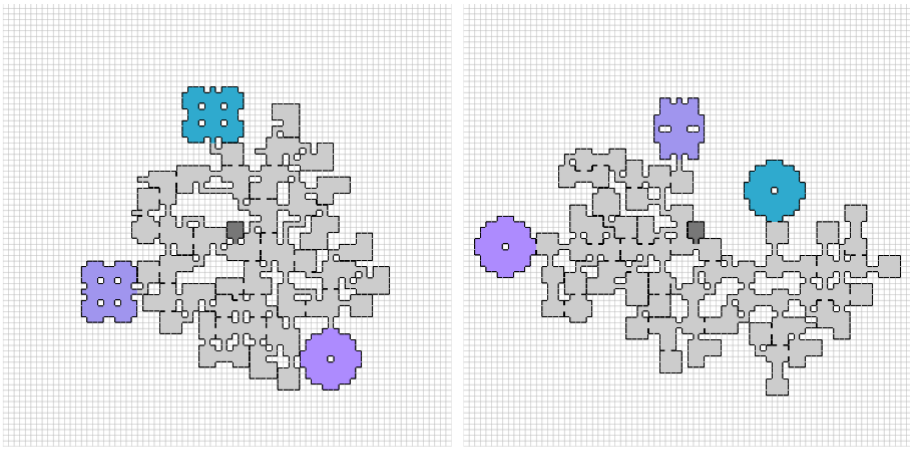
1.6.2 Evolving Dungeon Crawler Levels With Relative Placement

Dalším příkladem je [Valtchanov and Brown, 2012b],která používá stromovou strukturu reprezentující části úrovní, jako genetickou reprezentaci. Uzly ve stromu představují místnosti a hrany uzlu představují spojení do dalších místností. V průběhu vytváření jsou jednotlivé části/podgrafy otestovány, zda je možné vytvořit prostor definován tímto grafem. Pokud to není možné jsou odstraněny části, které to neumožňují.

Fitness funkce funkce je nastavena pro vytváření prostorů s co nejmenšími vzdálenostmi mezi uzly/místnostmi viz 1.1. Metoda umožňuje určit vzdálenost mezi místnosti obsahující akci(na obrázku barevně).

Algoritmus nám umožňuje určovat vzdálenost mezi hráčem a portálem. To by mohlo být využito, jako jedna z možností nastavení obtížnost vygenerovaného prostoru. Zároveň prostor odpovídá zadaní a vede k na průchod komplikovanému prostoru.

Stejně jako všechny genetické algoritmy přesahuje časová náročnost specifikovanou v našem případě a není schopna být provedena na mobilním telefonu v čase, který je vyžadován. Detailněji v sekci 2.



(a) (b)

Obrázek 1.1: Příklad prostoru pro Evolving Dungeon Crawler Levels z [Valtchanov and Brown, 2012b]

1.7 Metoda používaná v Game Level Layout from Design Specification

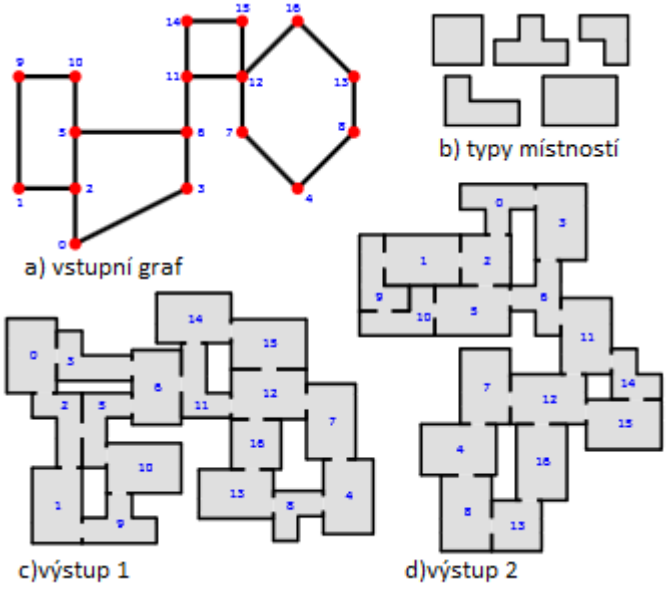
Tato metoda uvedená v [Ma et al., 2014] se zaměřuje na vytváření prostoru z grafu. Tento graf se skládá z bodů (místností) a z propojení (hrany) mezi nimi. Dalším vstupním parametrem jsou předvytvořené místnosti (mnohoúhelníky), které daný algoritmus používá. Metoda nijak neřeší vytvoření grafu a graf musí být vytvořen designerem. Výstupem metody jsou 2D náčrty prostoru. Na začátku je graf rozdělen na menší části(řetězy), a ty jsou nejdříve jednotlivě vyřešeny. Následně inkrementálně spojovány, protože pokud by se postupovalo postupem rozděl a panuj (divide-and-conquare), pak by následné spojení větších kusů nebylo jednodušší, než vyřešení celkového problému bez dělení.

Spojování probíhá tak, že se zjistí pomocí Konfiguračních prostorů (Configuration Spaces), kam všude se dá umístit místnost nebo řetěz, tak aby splňoval podmínky propojení. (Configuration Spaces je algoritmus, který hledá množinu všech možných umístění objektu prostoru). Algoritmus v průběhu vytváří více náčrtů, aby zajistil, že bude vytvořen aspoň jeden, protože někdy může nastat situace při které není možné propojit místnosti kvůli místnostem obklopujícím prostor propojení.

Místnosti jsou připojeny tak, že se vezme místnost k připojení (umožněný pohyb) a ta ke které se připojuje (zakázán pohyb) a zjistí se pro ně konfigurační prostor, kde správné propojení je takové, které vytváří spojení místností, ale zároveň nevzniká jejich překrytí. Pokud se místnost propojuje s více místnostmi, zjistí se možnosti umístění pro všechny místnosti nezávisle a pak se

provede průnik mezi nimi. Proces se ukončí po předložení všech řetězů s nalezeným dostatečným počtem kompletních náčrtů vycházejících z celého grafu. A nebo je ukončen, když není možné spočítat další rozdílné náčrty.

Výhodou metody je možnost použití jakýchkoliv mnohoúhelníkových místností, což není u většiny metod možné díky složitosti řešit spojení všech místností v závislosti na grafu a metody převážně vytváří obdélníkové místnosti. Přesto ale musí množina před-vytvořených místností splňovat určité podmínky - a to počet stran, pokud by totiž množina místností obsahovala pouze trojúhelníkové místnosti a graf by obsahoval bod s šesti hranami, pak by nebylo možné takto definovaný graf realizovat.



Obrázek 1.2: Příklad vygenerované úrovně z [Ma et al., 2014]

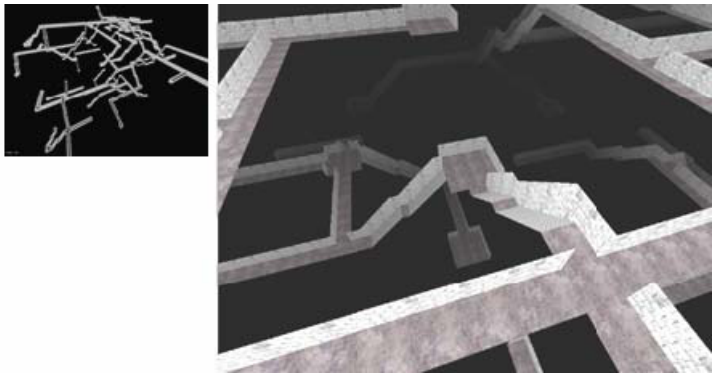
1.7.1 Možnosti pro naši definici prostoru

Základním nedostatkem pro případ definovaného bludiště je chybějící informace o generování grafu pro tento algoritmus. Díky tomu, že by v této práci mělo dojít k porovnání metod, neměly by metody být výrazně pozměněny a jejich podstata by měla být zachována. Kdybychom přidali metodu generování tohoto grafu, většina výsledků by se odvíjela právě od něj. Zároveň časová náročnost vytváření jedné úrovně sice vychází průměrně na velmi dobrá čísla, ale to při generování více prostorů v závislosti na jednom grafu, kde jsou v průměru časy velmi dobré. Pro představu zde zmíním jeden z vygenerovaných prostorů a jeho časovou náročnost. Na 1.2 je vidět velmi malá úroveň, kde při vygenerování 10 úrovní, bylo získáno jedno řešení v průměru za 1.1 vteřinu, ale první byl získán až po 4.9 vteřině. Na další příklady je možno se podívat

přímo do [Ma et al., 2014]. Nerozhodujeme se pouze na tomto případě, ale časová složitost této metody je velmi nestabilní, a proto jsem se rozhodl neriskovat, že bude možné získat výsledek pro všechny naše obtížnosti v reálném čase a zároveň na mobilním telefonu.

1.8 Constraint-Based (algoritmus pro vytváření 3D prostorů)

Algoritmus vytváří neorientovaný graf v 3D prostoru pomocí podmínek a zvolené topologie (například strom, kruh, hvězda). Podgrafy můžou být vytvořeny z rozdílných typů topologie grafu. Podmínkami může být například minimální a maximální vzdálenost mezi uzly. Následně jsou aplikovány předvytvořené části prostorů a cest k vytvoření finálního prostoru. Poté jsou přidány objekty do vytvořených místností.



Obrázek 1.3: Constraint-Based příklad výsledného prostoru z [van der Linden et al., 2014]

Podle zjištění z dostupných dokumentů odkazující na tuto metodu, bylo zjištěno, že byla pojmenována, až v dokumentu [van der Linden et al., 2014] a není známá a široce definována. Proto bychom se měli zaměřit spíše na jiné metody. Zároveň není výrazně definována, tím pádem by byl algoritmus ve finále velmi závislý na naší implementaci a porovnávání by nemohlo být výrazně objektivní. Navíc metoda řeší generování 3D prostoru, takže i tímto směrem bychom museli metodu upravit a komplikovalo by to celý algoritmus.

1.9 Pokročilé metody pro generování platformových her (Advanced platform generation method)

Procedurální generování pro vytváření platformových her je stavěno na jiných základech nebo používají již zmíněné algoritmy pouze aplikované na platfor-

mové hry a tím pádem je pouze zmíníme, ale nebudu je brát v úvahu.

1.9.1 Rytmem řízené generování (rhythm-base platform generation)

Nejdříve se generují malé kousky úrovně nazývané „rhythm groups“ pomocí grammar-based algoritmu. Pro rhythm goups jsou nejdříve vytvořeny akce hráče, a poté jsou převedeny do odpovídajíciho prostoru. Následně se vytváří levely spojováním těchto skupin a je vybrána ta nejlepší úroveň (podle specifických podmínek pro výběr). Lze použít i pro vytváření dungeonů, kde by se rozdělovalo do „dungeon-groups“ s rozdílnými hratelnostními (gameplay) charakteristikami.

1.9.2 Occupancy regulated extension(ORC)

Tato metoda na vytváření platformových her je založena na propojování předem vytvořených kusů (chunks) úrovně.

1.9.2.1 Postup algoritmu

1. Je zvolena náhodně možná pozice hráče pro umístění kusu
2. Vybere kus z před-vytvořených kusů, který je možný použít
3. Vybraný kus je propojen z již vytvořenou části úrovně
4. Pokud je volná možná pozice pro umístění hráče pak znovu (1)

1.9.3 Procedural Natural Systems for Game Level Design

Tento článek se zabývá procedurálním generováním přírodních prostorů a rozděluje generování přírodních prostorů na dvě nezávislé části vytvářející celek a to generování tvaru (shape) a generování vzhledu (footprint), které dokáže dostatečně popsat přírodní prostory. Tento způsob aplikuje autor [Huijser et al., 2010] na vytváření řeky s okolním prostředím.

Metoda se zaměřuje pouze na vytváření přírodních ploch, tím pádem není využitelná v případě bludiště, které je tvořeno místnostmi a průchody.

1.9.4 A Survey on Procedural Modeling for VirtualWorlds

Článek [Smelik et al., 2014] se zabývá metodami pro vytváření přírodního terénu, vegetace, vodních útvarů, cest, měst, budov a jejich interiéru.

Metody pro generování těchto prostorů a míst vytváří části mapy a nevytváří kompletní úrovně, jenž by mohly být použity v definovaném experimentu.

Kapitola 2

Výběr metod

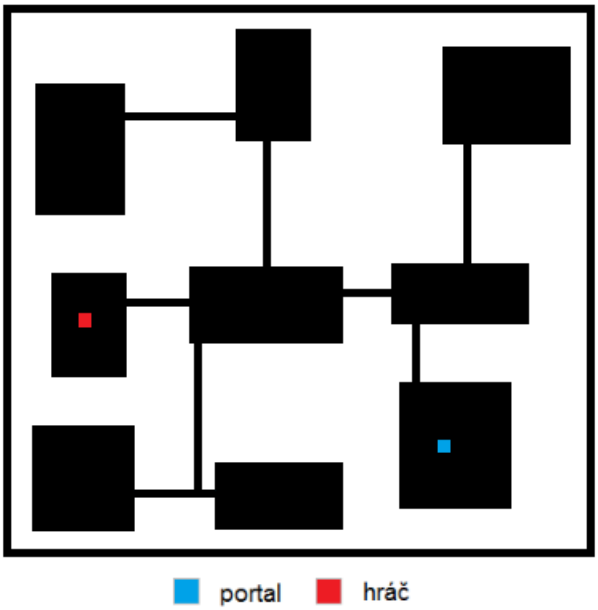
Nejdříve odeberu metody, které se vůbec netýkají generování úrovní a nejsou použitelné pro náš případ. Metody na generování řek, stromů a přírodních prostorů můžeme vynechat, protože nic tohoto typu nevytváříme a zároveň vytváří pouze kusy a ne celé úrovně. Dále můžeme vynechat algoritmy vytvářející interiery, budovy a města. Pro zbylé metody uvedu níže jejich výhody, kvality a možnosti a pak je následně porovnám a vyberu tři metody.

2.1 Rozdělování prostoru

Metoda ze základu vytváří odbočky ve dvou případech. Když aspoň na jedné straně propojení dvou částí je obsaženo více místnosti. Pak může být propojení vedeno z místnosti na cestu v tom případě vždy vzniká odbočka a nebo z místnosti na jinou místnost v tom případě může vzniknou odbočka a to vždy, když se nepropojuje s boční místností nebo pouze někdy když se propojuje s boční místností(Může vzniknout klikatá cesta). Viz 2.1 prostor odpovídá našemu zadání a zároveň hráč může zabloudit. K odbočkám tedy dochází ve spoustě možnostech, ale není možné zajistit určité množství odboček nebo komplikovanost prostoru. Jelikož jsme chtěli, aby prostor obsahoval, kružnice pak by jsme potřebovali pouze přidat možnost vytvořit více cest při spojování částí.

Možnou výhodou, kterou tato metoda umožňuje je jednoduchost při vytváření klíčů a uzamčených dveří, protože portál bude umístěn na opačné straně vytvořeného stromu nežli hráč, pak budou klíče umístěny v závislosti na (zamčených) dveřích umístěny náhodně do místnosti v podstromu obsahujícím hráče, aby bylo zajištěno, že bude umožněno hráči získat všechny klíče a ukončit úroveň. Obtížnost úrovně může být určena velikostí úrovně a zároveň nastavením spojovacího algoritmu, kde by každý algoritmus používal jiný způsob spojovaní pro každou obtížnost.

U algoritmu dělení prostoru bude hráč umístěn v úplně levém/pravém listu a portál na opačné straně. Zároveň bude muset být zvýšen počet cest z



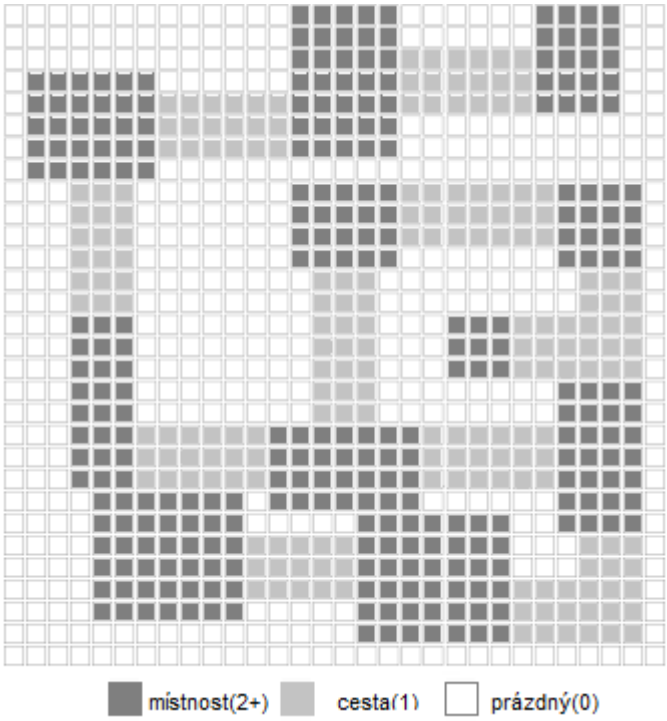
Obrázek 2.1: Příklad vygenerovaného prostoru pomocí dělení prostoru

důvodu jednoduchosti úrovně a nemožnosti vytvořit cyklické cesty, které jsou vyžadovány. Přidání zamčených dveří bude záviset na počtu propojení mezi dvěma podstromy, pokud bude pouze jedno pak podle náhodné šance můžou být přidány zamčené dveře. Klíče k zamčeným dveřím pak budou umístěny na náhodné místo v levém/pravém podstromu v závislosti na které straně byl umístěn hráč, aby bylo zajištěno, že je hráč schopen daný klíč sebrat.

2.2 Metoda založená na vytváření prostoru agentem

V případě generování pomocí agenta by byly definovány agentovi možnosti vybrat cestu ze čtyř možných cest, možnost umístit klíč a pokud již existuje vytvořený klíč pak možnost umístit zamčené dveře s tím to klíčem. Zároveň bude mít agent možnost vytvořit propojení s existující místností a tím vytvořit cyklus. Použitím většího množství agentů dojde ke zkomplikování úrovně a umožnění vzniku cyklů a bočních cest ve výsledné mapě. Obtížnosti u tohoto algoritmu budou definovány jak velikostí vzniklého prostoru, tak počtem agentů spolupracujících na vytvoření úrovně. Možný příklad výsledku viz 2.2

Pro příklad, který je metoda vybírána, potřebuje určité možnosti nastavení obtížnosti - to lze vyřešit možností nastavení velikosti plochy, na které se agent může pohybovat,počtem agentů použitých pro vygenerování jedné úrovně a šancí vytvoření cyklu. Určitou nejistotou algoritmu může být že pokud agent nekontroluje více možností cest může nastat, že vytvoří úroveň

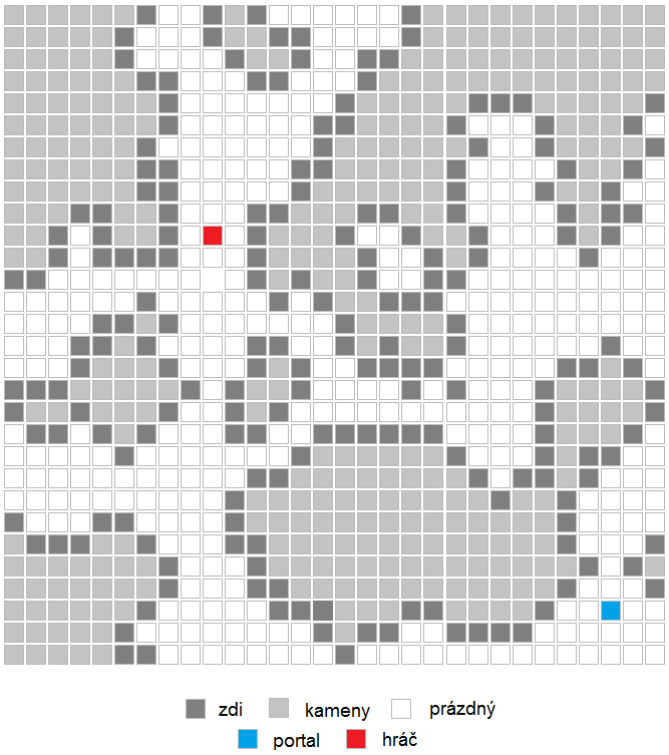


Obrázek 2.2: Příklad vygenerovaného prostoru pomocí agenta

o malé velikosti. Například pokud bude vybrán náhodně pouze jeden směr výstupu z vytvořené místnosti, pak může nastat k rychlému konci při umístění agenta k jedné straně prostoru. Naopak při velké kontrole může vznikat dlouhá klikatá chodba z důvodu nemožnosti agenta zahrabat se na místo ze kterého už neví kudy kam. Vytvoření této chodby může na první pohled vypadat žádané, ale pokud má být bludiště komplikované, pak už nezbývá místo pro další agenty k vytvoření odboček a cyklů. Tomuto problému může, být zabráněno parametrem určujícím maximální počet místností, který jednotlivý agenti můžou vytvořit.

2.3 Cellular Automata

Cellular Automata vytváří přírodně vyhlížející prostory a nelze pomocí ní vytvářet místnosti a cesty mezi nimi. Zároveň není možné zajistit možnost výhry úrovně viz 2.3. Ani pokud by jsme umísťovali portál a hráče do stejného sektoru, nejsme schopni nastavit obtížnost hry a ovlivnit velikost prostoru ve kterém se bude hráč pohybovat. Metoda nám také nijak neulehčuje práci z klíči a zamčenými dveřmi.



Obrázek 2.3: Problém možnosti vytvoření prostoru pomocí Cellular Automata

2.4 Gramatikou řízené generování

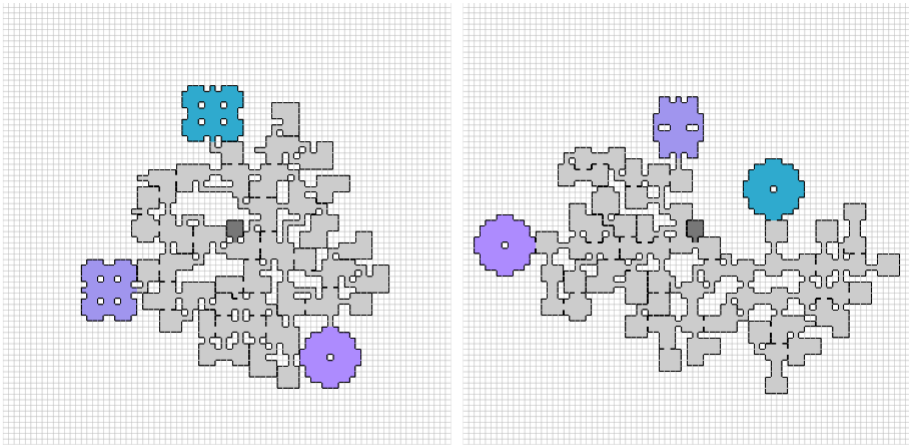
Kde gramatikou řízené generování nám umožňuje vytvořit úroveň pomocí definování gramatiky obsahující aktivity: klíč, dveře, start, portál, halda peněz, bloudění a jejich možná pravidla transformace, kde by byla určena jiná pravidla pro každou úroveň obtížnosti. Tento typ nám umožňuje vytvořit cyklické cesty a rozcestí, pouze závisí na použití pravidel pro transformace aktivit. Jediným rizikem může být použití špatné gramatiky.

2.5 Genetické algoritmy

Stejně jako všechny genetické algoritmy přesahuje časová náročnost specifikovanou v našem případě a není schopna být provedena na mobilním telefonu v čase, který je vyžadován.

Pro metodu uvedenou v [Valtchanov and Brown, 2012a], která z genetických algoritmů splňovala nejlépe podmínky námi definovaného prostoru viz 2.5, byl ale uveden průměrný čas pro získání úrovně 30 vteřin na stolním počítači o rychlosti 2.4 GHz. Tím pádem není možné tuto metodu použit.

Obrázek 2.4: Příklad vytvořeného prostoru pomocí gramatikou řízeného generování z [Linden et al., 2013]



Obrázek 2.5: Příklad prostoru pro Evolving Dungeon Crawler Levels z [Valtchanov and Brown, 2012b]

2.6 Metoda z dokumentu Game Level Layout from Design Specification

Základním nedostatkem metody [Ma et al., 2014] pro případ definovaného bludiště je chybějící informace o generování grafu pro tento algoritmus. Díky tomu, že by v této práci mělo dojít k porovnání metod. Neměly by metody být výrazně pozměněny a jejich podstata by měla být zachována. Kdybychom přidali metodu generování tohoto grafu většina výsledků, by se vyvíjela právě od něj. Zároveň časová náročnost vytváření jedné úrovně sice vychází průměrně na velmi dobrá čísla, ale to při generování více prostorů v závislosti na jednom

grafu.

K představě zde zmíním jeden z vygenerovaných prostorů a jeho časovou náročnost. Na 1.2 je vidět velmi malá úroveň, kde při vygenerování 10 úrovní, byl získáno jedno řešení v průměru za 1.1 vteřinu, ale první byl získán až po 4.9 vteřině. Na další příklady se můžete podívat přímo do [Ma et al., 2014]. Nerozhoduji se pouze na tomto případě, ale časová složitost této metody je velmi nestabilní a proto jsem se rozhodl neriskovat, že bude možné získat výsledek pro všechny naše obtížnosti v reálném čase a zároveň na mobilním telefonu.

2.7 Constraint-Based

Podle zjištění z dostupných dokumentů odkazující na tuto metodu, bylo zjištěno, že byla pojmenována, až v dokumentu [van der Linden et al., 2014] a není známá a široce definována. Proto by jsme se měli zaměřit spíše na jiné metody. Zároveň není výrazně definována, tím pádem by byl algoritmus ve finále velmi závislý na naší implementaci a porovnávání by nemohlo být výrazně objektivní. Dále metoda řeší generování 3D prostoru, takže i tím to směrem by jsme museli metodu upravit, což by komplikovalo celý algoritmus.

2.8 Porovnání a výběr 3 metod

S uvedených metod nejdříve odeberu metody, které nejsou použitelné z důvodu časové náročnosti nebo neschopnosti splnit zadání, a následně vyberu algoritmy nejlépe splňující podmínky k použití.

Když vezmu v úvahu genetické algoritmy, tak ty vedou k dobrým výsledkům, ale bohužel nemůžou být použity z důvodu velké časové náročnosti a nejsou schopny vygenerovat výsledek ve vyžadovaném čase.

Cellular Automata vytváří spíše přírodně vyhlížející prostory a nelze pomocí ní vytvořit místnosti a zajistit vždy možnost výhry úrovně, tak ji musíme taky vyřadit.

Další neodpovídající metodou je Constraint-Based, který vytváří prostor o více podlažích, což není žádáno. Bylo by obtížné tuto metodu upravit, aby jsme ji mohli použít. Zároveň není přesně definovaná a implementace by převážně byla na nás, což nechceme.

Metoda z dokumentu [Ma et al., 2014] používá jako vstup graf a není nijak uvedeno, jak se tento vstupní graf dá náhodně vygenerovat. Byla by tu například možnost vytvořit graf pomocí generativní gramatiky. Jelikož, ale není schopný vytvořit, podle zjištěných informací, úrovně dost rychle, pak by nám to nepomohlo. Tím pádem, bychom ji použili pouze v případě, pokud by jsme neměli dost metod.

Po této eliminaci nám zbyly tři algoritmy: dělení prostoru, generování pomocí agenta a gramatikou řízené generování.

Kde gramatikou řízené generování nám umožňuje vytvořit úroveň pomocí definování gramatiky obsahující aktivity: klíč, dveře, start, portál, halda peněz a bloudění a jejich možná pravidla transformace, by byla určena jiná pravidla pro každou úroveň obtížnosti. Tento typ nám umožňuje vytvořit cyklické cesty a rozcestí, pouze závisí na použití pravidel pro transformace aktivit.

V případě generování pomocí agenta by byly definovány možnosti vybrat cestu ze čtyř možných cest, možnost umístit klíč, a pokud již existuje vytvořený klíč pak možnost umístit zamčené dveře s tím to klíčem. Použitím většího množství agentů dojde ke zkomplikování úrovně a umožnění vzniku cyklů a bočních cest ve výsledné mapě. Obtížnosti u tohoto algoritmu budou definovány jak velikostí vzniklého prostoru, tak počtem agentů spolupracujících na vytvoření úrovně.

U algoritmu dělení prostoru bude hráč umístěn v úplně levém/pravém listu a portál na opačné straně. Zároveň bude muset být zvýšen počet cest z důvodu jednoduchosti úrovně a nemožnosti vytvořit cyklické cesty, které jsou vyžadovány. Přidání zamčených dveří bude záviset na počtu propojení mezi dvěma podstromy. Pokud bude pouze jedno, pak podle náhodné šance můžou být přidány zamčené dveře. Klíče k zamčeným dveřím pak budou umístěny na náhodné místo v levém/pravém podstromu v závislosti, na které straně byl umístěn hráč, aby bylo zajištěno, že je hráč schopen daný klíč sebrat.

Z těchto zmíněných informací je vidět, že bude možné nalézt tři metody odpovídající zadání a to generování dělením prostoru, pomocí agenta a gramatikou řízené generování.

Kapitola 3

Návrh experimentu

Z důvodu nemožnosti určit, co hráči upřednostňují ve hře, a tím pádem zjistit výhody algoritmů vzhledem k názorům hráčů, byl experiment rozdělen na dvě části, kde účelem prvního bylo zjistit možnosti porovnání metod (jak na co hráči reagují a co berou jako výhody a nevýhody). A pak následuje druhá část experimentu, která vychází z výsledků první části a bude obsahovat dotazník po dohrání jednotlivé úrovně. V obou případech experimentů se budou odesílat data o vygenerované úrovni a o úspěchu hráče ve sbírání peněz.

3.1 Kvalitativní studie

Experiment se sestával z předtestovacího dotazníku, hraní pod dohledem experimentátora a závěrečného pohovoru. Z hraní jsme sbírali data o dvou úrovních stejné obtížnosti a jejich následném porovnání účastníkem.

V této kvalitativní studii se zaměřujeme na menší skupinu lidí s cílem nalézt co nejvíce možných charakteristik k porovnání algoritmu a zároveň jejich opakovaného výskytu. Z důvodu časové náročnosti a nutnosti naší přítomnosti u všech algoritmů budou provedeny experimenty pouze pro jednoduchou a střední úroveň. Předpokládaná velikost cílové skupiny, které by jsme chtěli dosáhnout, se bude pohybovat přibližně kolem 15 účastníků.

Dalším cílem prvního experimentu bude odchytat drobné chyby, aby nezasahovali do kvantitativního experimentu a nemohli ovlivnit jeho výsledky.

3.1.1 Předtestovací dotazník

Počáteční část se skládá se zjištění vztahu účastníka experimentu k hrám, aby experiment nebyl proveden na skupině lidí, kterých se dané téma netýká. Zároveň informace od zkušenějších hráčů by měli mít větší informativní hodnotu. Díky tomu byl účastník tázán na jeho herní aktivitu (počítač/mobil).

- Jak často hrajete hry na počítači?

- Jak často hrajete hry na mobilu ?

3.1.2 Průběh hraní

Účastník absolvuje dvě úrovně o stejné obtížnosti bez toho, že by věděl o použití jiných algoritmů pro jejich generování. Může se stát, že hráč bude porovnávat dvě úrovně vygenerované stejným algoritmem, tím zjišťujeme co přijde hráči lepší a zábavnější i vzhledem k stejnému typu algoritmu. V průběhu hry účastník popisuje, jak a co mu přišlo kladem a záporem vygenerovaného prostoru.

- Popište co vám přijde kladem/záporem prostoru?
- Popište co vám se vám líbí/nelíbí, přijde zajímavé/nezajímavé?

3.1.3 Porovnání úrovní

Po absolvování obou úrovní, bude hráč dotázán aby popsal možné rozdíly, kterých si všiml, jak v obtížnosti tak v zábavnosti předložených úrovní. A vybral tu lepší a řekl proč mu přišla lepší.

- Jaká úroveň vám přišla lepší a proč?

3.1.4 Odesílaná data

Po dohrání hry se odesílají informace o vygenerované úrovni a úspěšnosti hráče při sbírání peněz. Při odesílání se dodá informace o identifikačním číslu aktuálního experimentu pro obě úrovně. Příklad odeslaných dat viz tabulka 3.1

Tabulka 3.1: Příklad odeslaných dat

seed	typ	id	obtížnost	čas	sebráno	celkem	vytvořeno
-1625198836	2	47	1	101	21	49	2016-04-10
-969207550	0	47	1	365	19	24	2016-04-10
934324044	1	46	0	105	40	40	2016-04-10
-306099622	0	46	0	189	2	2	2016-04-10

3.1.5 Příklad získaných informací

Průběh experimentu bude zaznamenáván, jako audio nahrávka a následně přepsán. Příklad výsledku experimentu níže.

3.1.5.1 Herní zkušenosti hráče

- počítač 1h/denně
- mobil skoro vůbec

3.1.5.2 První úroveň

- není tu moc orientačních bodů, jak je to prostředí všude stejný
- sakra nemám klíč, teď se budu muset vracet celou cestu zpátky
- jsem sebral klíč a pořád tam mám 0 klíčů
- mám klíč, ale nejdou otevřít dveře. Asi jsem měl doteď štěstí, že vždy jsem sebral klíč a hned jsem ho mohl použít.

3.1.5.3 Druhá úroveň

- hele já jsem sebral klíč a otevřel jsem dvě dveře (dveře bez zámku)
- už jsem to dohrál. To bylo rychlý no.
- našel jsem klíč sebral peníze, pak prošel dvěma dveřmi a byl tam portál. Akorát jsem asi nesebral všechny peníze, i když jsem žádný jiný neviděl, takže nevím.

3.1.5.4 Porovnání úrovní

- první byla o dost horší a musel jsem se hodně vracet a nebylo to tak jednoduchý.
- druhá úroveň tam jsem jenom přišel sebral klíč vybral jsem si mezi dvěma dveřmi. Vybral jsem si zrovna ty správný a už jsem jenom prošel a byl konec.
- druhý málo rozhodování praktický jsem šel rovně a byl jsem tam
- u první jsem se musel často vracet pro správný klíč, to mi nepřišlo zrovna špatný, že to ten hráč nemá tak lehký.
- u prvního bylo dobrý, že jsem se objevil a hned byli podemnou peníze, kterých jsem si díky tomu málem nevšiml, kdyby jsem se pořádně nerozhlídl, tak bych je přešel.

3.2 Kvantitativní Experiment

Druhý experiment provádí účastník samostatně, pouze s herními instrukcemi. Hráč po každé úrovni vyplňuje dotazník obsahující zjištěné charakteristiky z kvalitativní studie. Kde každá charakteristika bude mít hodnocení jedna až pět. Stejně jako v první algoritmu se budou odesílat data o vygenerované úrovni a úspěch hráče ve sbírání peněz, ale navíc budou přidány informace o hodnocení charakteristik. Hráč bude moci hrát kolikrát bude chtít, pokud by přesáhl určité množství her než určené (10 nebo 10%), pak se následující hry nebudou počítat do statistik experimentu. To se kontroluje pomocí identifikačního čísla zařízení, které je přiřazeno při instalaci aplikace a posílá se do databáze s ostatními daty.

3.2.1 Odesílaná data

Když hráč dohraje úroveň jsou odeslána první data o dokončení úrovně. Obsahující stejné informace, jako prvním experimentu. Následně hráč vyplní dotazník a odešle vše i s charakteristikami. Pokud by hráč neměl přístup k internetu má možnost zrušit odeslání. Kvůli tomu také odesíláme data ‚již při dohrání úrovně, abychom mohli zjistit jestli hráči mají tendenci nevyplňovat dotazníky. Jelikož se odesílají stejná data jako u předchozího experimentu, pak v tabulce 3.2 budou zobrazeny pouze informace obsažené navíc.

Tabulka 3.2: Tabulka odesilaných dat(bez výsledných charakteristik)

Název charakteristiky	A	B	C	D	E
Hodnota (1-5)	2	5	4	3	2

Kapitola 4

Implementace

V této sekci nejdříve popíšu základní strukturu a technický návrh hry. Následně uvedu, jak byli implementovány jednotlivé vybrané metody.

1. Metoda založená na vytváření prostoru agentem
2. Gramatikou řízené generování (Grammar-based dungeon generation)
3. Rozdělování prostoru (Space partitioning)

4.1 Struktura a technický návrh

Vytváření 3D prostor s 2D definic, které jsou výstupem všech tří metod probíhá pomocí třídy LevelGeneration. Ta se stará o vytváření kompletní úrovně. Vstupem metody jsou všechny parametry určující pouze co se má zobrazit z dat, které byli předány v definici a jaké dekorace mají být použity. Zároveň dostává jako parametr metodu generování.

4.1.1 Generování geometrie

Všechny metody vytvářejí cesty a místnosti podle určitého rozhraní, jenž se využívá pro generování finální geometrie. To je předáno jednotlivým továrnám, které vše vytvoří. Každá továrna používá třídu vytvářející zdi, a ty pak spojuje do finální podoby dané místnosti nebo cesty. Pro přidávání dekorací a herních předmětů do místnosti je možné použít dekorátory.

K vytváření zdí se používá objekt „MeshFactory“, který vytváří fyzickou podobu zdi. Zdi jsou přímo vytvořené pomocí tvorby jednotlivých stran pomocí trojúhelníku, UV, normál, finálních bodů a následného přidání materiálu.

4.1.1.0.1 Dekorátory pro místnosti

- TorchDecorator (přidá pochodně rozprostřené po zdech místnosti)

- RoomObjectDecorator (umístí herní předměty do místnosti)
- ChandelierDecorator (umístí na strop jeden z pěti lustrů)

4.1.1.0.2 Dekorátory pro cesty

- TorchHallwayDecorator (vytváří rozprostřené po zdech cesty).
- DoorDecorator (umísťuje dveře na konce místností) přidává dveře dekorační, které je možné otevřít bez klíče. Nebylo použito v experimentu.

4.2 Metoda založená na vytváření prostoru agentem (Agent-based dungeon growing)

Když jsem vytvářel implementaci podle agenta, tak jsem došel ve finále ke dvěma řešením, které fungují ve finále velmi podobně, zde zmíním tu co jsem vybral a použil. Na konci popíšu rozdíly, které byly v původní implementaci. Implementace obsahuje obě řešení, ale k experimentům byla použita pouze tato.

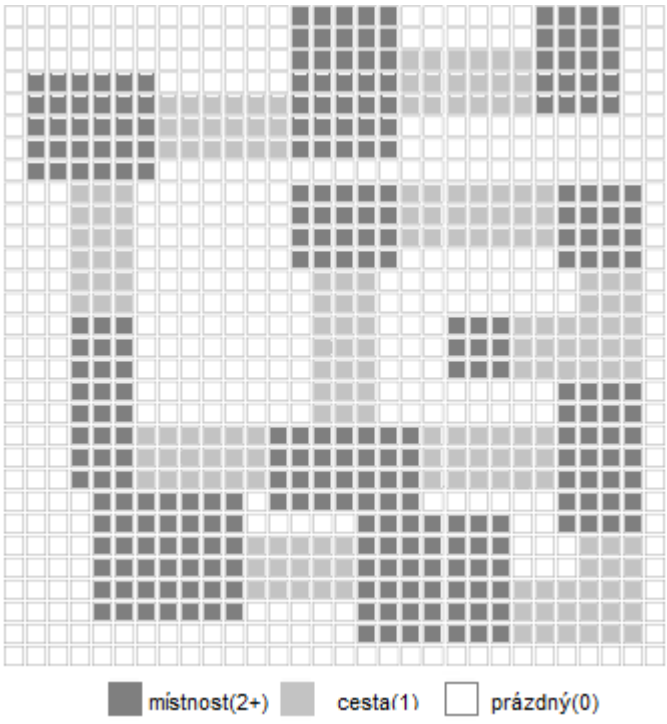
Moje řešení vytváření prostoru pomocí agenta funguje tak, že mám určité množství agentů k použití, kde každý agent může vytvořit další agenty. Agenti se postupně paralelně střídají po jedné místnosti. K tomu používají metodu next(newAgent,agentsLeft), která vrací zda agent přežil aktuální krok viz 3 řádek. Agenti vytvářejí prostor, dokud všichni agenti neumřou (dostanou se do situace, kde nemůžou pokračovat a vytvořit místnost).

Listing 4.1: Průběh agentů

```
while(agents.Count>0){
        actualAgent=agents.Dequeue();
        if(actualAgent.next(out newAgents,agentsLeft))
                agents.Enqueue(actualAgent);
        if(newAgents!=null){
                for(int i=0;i<newAgents.Count;++i){
                        agents.Enqueue(newAgents[i]);
                        --agentsLeft;
                }
        }
}
```

Každý agent má určenou svoji maximální délku v počtu místností, jenž může vytvořit, tak aby se nestalo, že pár agentů zaplní celý prostor. Zároveň agenti mají šanci vytvořit cyklus, tím že narazí na místnost ke které se můžou připojit, a tak vytvořit kruhovou cestu. Jelikož algoritmus obsahuje klíče a zámky, cesta vytvářející kruh by neměla zjednodušovat obtížnost. A proto se kontroluje kolik zámků stojí v cestě skrze cestu, do které vede tato zkratka a jaký z nich je nejsilnějším zámkem (nejblíže k tomu to místu). V případě,

že se splní nastavené parametry a to maximální počet zámků blokující cestu, pak se vytvoří propojení. Pokud byl nalezen aspoň jeden zámek, pak bude umístěn na místo vstupu do dané místnosti ten nejsilnější(umístěný nejblíže k dané místnosti). Agent, jenž vytvoří kruhové propojení umírá a jeho aktivita v tu chvíli končí.



Obrázek 4.1: Vytvořený prostor v 2D poli - cesta = 1, místnost = 2 + pořadí, prázdné = 0

4.2.1 Umístění předmětů

Agent může umísťovat peníze, klíče a zámky dělá to v závislosti na nastavené šanci a pro zámky také v závislosti, zda byli již nějaké klíče umístěny. Tím, že umísťujeme klíče dříve než zámky a vše umisťujeme pouze při vytváření místnosti máme zaručené, že bude vždy možné najít klíč a odemknout zamčené dveře. Díky tomu může jakýkoliv agent umístit zámek ke klíči umístěným jiným agentem a mít jistotu, že bude možné dokončit úroveň.

4.2.2 Rozhodování pohybu agenta

Každý jednotlivý agent má možnost vytvořit místnost vždy, když je kolem něj dost volného prostoru. Agent se dále vždy rozhlédne, jestli může odejít z místnosti/cesty je-li kam. Vybírá náhodně jeden směr a zkouší umístit místnost.

Není-li schopen umístit místnost, pak se snaží pokračovat náhodným směrem z cesty, kterou právě vytvořil. Ve chvíli kdy agent není schopen najít další cestu ani vytvořit místnost, tak se vrací o krok zpět, až dorazí do poslední vytvořené místnosti, kde se může rozhodnout pro zbylé směry, jestliže mu to parametr maximálního počtu rozhodnutí umožňuje.

4.2.3 Přidání agenta

Pokud agent vytvoří místnost a zbude mu volná cesta, tak může v závislosti na šanci vytvořit dalšího agenta a ten začíná na tomto místě. Tento agent dostává náhodnou délku v závislosti na parametrech min/max délky odbočky.

Algoritmus tedy začíná umístěním prvního agenta, ten má výchozí směr neboli žádný směr(NoDirection), také má nastavenou maximální možnou délku, které může dosáhnout. Vždy, když se agent přemístí do nové místnosti nebo se posune určitým směrem, tak si nastaví směr a podle něj se následně rozhoduje. První agent je hlavním agentem a jeho začátek určuje umístění startovní pozice hráče a poslední místo, kde bude umístěn portál. Agent vytváří místnosti a vytváří i rozcestí, dokud se nezaplní celý prostor nebo počet použitých agentů dosáhne počtu určeného v parametru.

4.2.4 Původní implementace

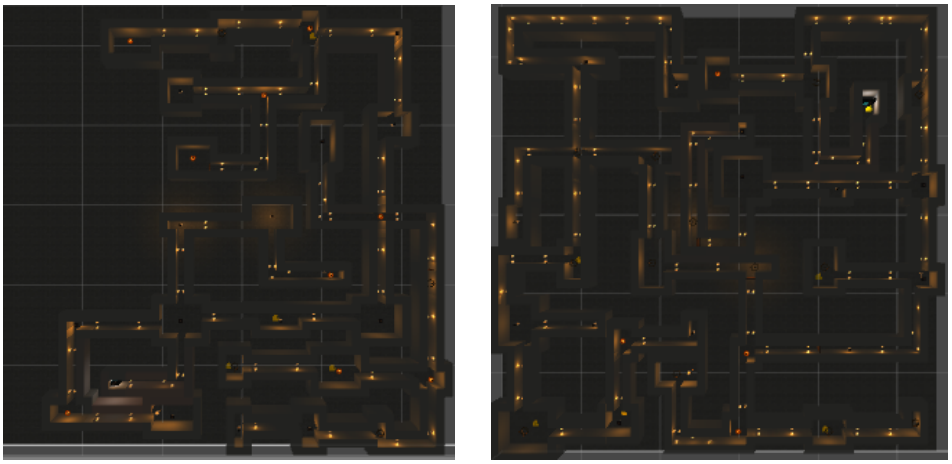
Původní průběh agenta byl velmi podobný, ale nebyl tak stabilní, jako ten aktuální. Všichni agenti začínali ze stejné startovní pozice. Měli možnost se pohybovat již vzniklými místnostmi a šanci pohybovat se směrem již do vytvořené místnosti. Posunovali se vždy, když neměli jinou možnost (nebyla odbočka) nebo nastala šance posunout se dál. Jakmile našli místo k odbočení nebo přesáhli maximální počet posunutí místnostmi, tak pokračovali stejně jako v předešlém popisu, pouze vždy skončil aktuální agent než mohl začít další.

4.2.5 Nastavení obtížností

Největší efekt na obtížnost vytvořené úrovně má velikost prostoru a počet agentů. Počet agentů je přímoúměrný počtu odboček, pokud je ještě možné nějaké umístit (nemůže jich být víc). Dalším efektem je chytrost agenta. Když má agent možnost rozhodování na maximum, pak je jen malá šance, že umře a vytváří dlouhý průchod a zároveň pak zabraňuje ostatním agentům vytvářet odboček kvůli prostoru který zabral. V tom mu zabraňuje parametr určující maximální počet místností, které může agent vytvořit. Všechny parametry a nastavení metod je zobrazeno v 4.1

Tabulka 4.1: Parametry a nastavení pro agenta

název	datový typ	easy	medium	hard
velikost prostoru	(šířka,výška)	(40,40)	(55,55)	(80,80)
počet agentů	int	10	15	20
počet místností h. agent	int	7	12	18
interval délky pro odbočku	(min,max)	(1,5)	(1,8)	(1,10)
šance gold	int	20	20	20
šance klíč	int	0	15	0
šance dveře	int	0	15	0
šance propojení	int	20	15	20
max počet zámků pro propojení	int	0	2	2



Obrázek 4.2: Příklady výsledků metody - agent

4.2.6 Výsledky

4.3 Gramatikou řízené generování (Grammar-based dungeon generation)

Zde bylo možné vybrat z více možností implementace, které jsem mohl zvolit nebo přizpůsobit viz dokumenty [Dormans, 2010] a [Linden et al., 2013], kde byli tyto metody vysvětleny a použity jinými způsoby.

Nakonec jsem se přiklonil implementovat velmi podobným způsobem, který je uveden v [Linden et al., 2013], protože vygenerovaný prostor odpovídá podobným měřítkům a specifikaci vygenerovaného prostoru, který jsem si definoval. V tomto řešení autor generuje podle postupu zmíněném níže.

4.3.1 Původní postup generování

1. Směrový graf aktivit
2. Sjednocuje aktivity probíhající na stejném místě
3. Transformaci grafu na planární
4. Snížení počtu hran na čtyři na místnost
5. Převedení prostoru na 2D Grid
6. Následné opravení dlouhých cest vzniklých v (5)

4.3.2 Naše definice a průběh

V našem případě neprobíhá více akcí na jednom místě. Autor [Linden et al., 2013] používal například pro popis případu „poražení nepřítele a sebrání odměny“, která z něj vypadla po jeho smrti. Pak tuto část můžeme eliminovat (není pro nás důležitá).

Dále jsem se rozhodl při generování získávat možné transformace z gramatiky podle počtu cest, které vstupují/vystupují z/do uzlu, který transformujeme. Jelikož chceme v každém případě vygenerovat 4-stupňový graf a takto je pořád realizace prostoru oddělená od definice grafu. Navíc finální prostor, který nastavím pomocí gramatiky bude zachován stejný a nebude se měnit ani rozšiřovat. To nastávalo při (3) a (4), kde byli přidávány další uzle.

K převedení prostoru na 2D mřížku(Grid) jsem se inspiroval původním postupem autora. Nejdříve proběhne rozmístění uzlů do prostoru pomocí určení vztahů mezi jednotlivými uzly, a tím se určí jejich vzájemná pozice. V následující části jsem zvolil úplně jiný přístup a ten uvedu níže až v detailnějším popisu. Tato metoda nepotřebuje následné opravování a zkracování chodeb. Tím pádem jsem snížil postup o některé části. A to (3),(4) a (6). Ty zmíním na konci pro úplnost.

4.3.2.1 Použitý postup generování

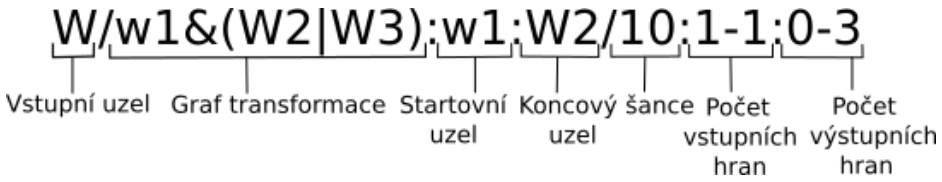
1. směrový graf aktivit s omezením růstu
2. transformace grafu do prostoru
3. převedení prostoru na 2D mřížku
4. odebrání prázdných místností

4.3.2.2 Směrový graf aktivit s omezením růstu

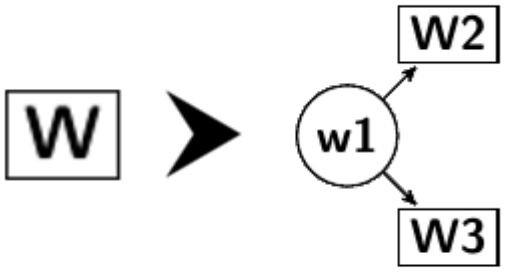
V mém případě generování grafu začíná vždy pouze s grafem obsahujícím jeden startovní uzel. Podle úrovně, která byla zvolena, se načte soubor popisující gramatiku, tím je určeno nastavení a obtížnost úrovní. Po načtení gramatiky se předá první graf, obsahující pouze startovní uzel, generující třídě. Ta vybere náhodně jeden neterminální uzel. Podle jeho symbolu, počtu vstupních a výstupních cest získá z gramatiky náhodně jeden graf transformace. Tento graf je pak připojen do původního grafu pomocí hran odebíraného uzle na jeho počáteční/koncový uzel grafu transformace. Transformace jsou prováděny dokud graf vlastní uzly s neterminálními symboly nebo dosáhne hodnot nastavených v parametrech. Například maximální počet uzlů v grafu.

4.3.2.2.1 Gramatika Ukládá se do textového souboru, kde každá definice transformace je umístěna do jednoho řádku. Každý řádek/transformace obsahuje vstupní symbol, graf transformace, interval počtu vstupních/výstupních hran, startovní a konečný uzel. Zároveň má uložené číslo, které určuje šanci transformace vzhledem k ostatním transformacím.

Samotný graf transformace je převážně definován dvěma znaky. Pro znak ‚&' platí když A&B, pak A je rodičem B. Pro znak ‚|' platí když A|B, pak A má stejného rodiče jako B. Příklad transformace 4.3 a její vizualizace viz 4.4.



Obrázek 4.3: Definice transformace

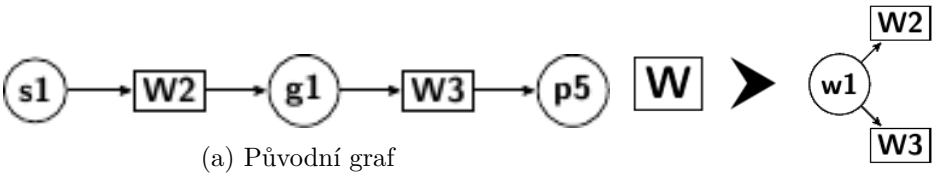


Obrázek 4.4: Vizualizace transformace

4.3.2.2.2 Aplikace transformace Ve zmíněných dokumentech bylo používáno více druhů tvoření transformací a jejich aplikování. Použil jsem tu, která byla použitá ve stejném dokumentu jako jsem se rozhodl použít metodu. A ta používá transformace založené pouze na transformaci jednoho uzlu na jiný nebo na více uzlů. Rozhodl jsem se tak z jednoduchosti použití a z obtížnosti nastavení gramatiky v opačném případě.

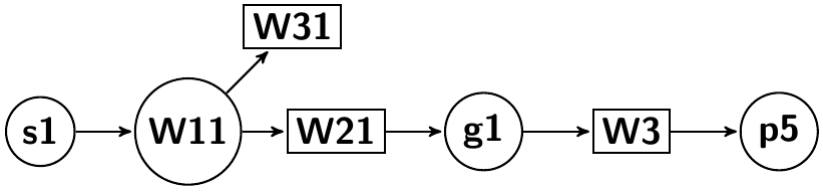
Nejdříve se vezmou vstupní/výstupní uzle původního uzlu. Vstupní se propojí se startovním uzlem grafu transformace a výstupní s koncovým uzlem. Pak se odebere propojení původního uzlu a dojde k jeho odstranění.

Příklad aplikace transformace na 4.5.



(a) Původní graf

(b) Transformace



(c) Graf po provedení transformace (b) na grafu (a) a uzlu W2

Obrázek 4.5: Vizualizace aplikování transformace

4.3.3 Zaručení planárního grafu

Jelikož pracujeme s orientovaným acyklickým graf, pak není planárním pokud obsahuje $K_{3,3}$ jako podgraf. Jediné co musíme zjistit v našem případě je nemožnost vzniku podgrafu $K_{3,3}$. Jelikož není možné se připojit transformací, k již existujícímu uzlu jenom v případě spojení počátku/konce grafu, pak není možné takovýto graf vytvořit pomocí naší definice transformace. Zároveň není možné přidat pouze hranu a tím dojít k většímu množství propojení. Jelikož se pouze zachovává minulé propojení před odebráním původního uzlu a to je pak použito k připojení nového a nevznikají žádné nové propojení, pak není možné vytvořit $K_{3,3}$. Jedinou možností by bylo přidání celého podgrafu $K_{3,3}$ nebo jakéhokoliv jiného neplanárního podgrafu.

Takže, náš graf začíná vždy jedním vstupním uzlem. Graf obsahující jeden uzel je vždy planární. Zároveň vycházíme z toho, že gramatika může obsahovat vždy pouze planární grafy, které mají jeden startovní a koncový uzel.

Náš předpoklad je, že graf, který transformujeme je planární. To znamená, že spojujeme vždy dva planární grafy a to tak, že vstupní hrany připojíme k startovnímu uzlu a výstupní ke koncovému uzlu. Protože původní propojení neporušovalo planaritu grafu, pak nemůže porušovat ani po jejím propojení s jiným planárním grafem.

4.3.4 Transformace grafu do prostoru

Jelikož omezuji generování, aby se vytvářeli pouze čtyři cesty a graf zůstává planární viz 4.3.3. Můžeme začít s transformací grafu do prostoru. Zde vycházím s [Linden et al., 2013]. Kde autor nejdříve provádí rozmístění do prostoru bez fyzického umístění, hlídající si určité podmínky, aby mohl být graf umístěn a nedošlo k překřížení cest.

Po hlubším prozkoumání metody a její funkčnosti jsme zjistili, že autor provádí vykreslování pouze jedním směrem, což autor v textu přímo nezmiňuje. Dělá to z důvodu zachování kružnic v grafu. Původně vyplývalo z algoritmu, že je možné, aby se graf rozrůstal všemi směry. To, ale bohužel bylo vyvráceno. Jelikož jsme, ale potřebovali, aby se finální prostor rozrůstal více směry a nebyl přímočarý. Tak jsme se rozhodli přidat úpravu a to přidání možnosti směru vytváření, ale zase nemůžeme změnit směr kdykoliv, pouze když uzel není součástí kružnice.

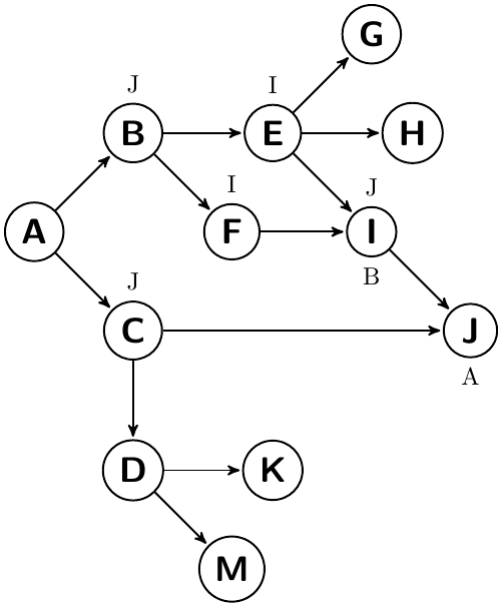
4.3.5 Předzpracování grafu

Do každého uzlu grafu si uložíme jeho nejbližšího následníka, aby jsme věděli jestli může změnit směr a zároveň věděli do kterého spojení uzlů směřuje. Dále potřebujeme uložit data o předchůdci do uzlu uzavírajícím kružnici, abychom následně mohli zjistit odkud cesta směřuje a nedošlo k překřížení více viz detailněji později v sekci 4.3.6.

Tyto informace získáme, tak že si nejdříve vytvoříme topologické uspořádání. Následně umístíme prázdný zásobník do prvního uzlu. A pak pokud uzel má jednu vstupní hranu, tak do něj přidáme předchozí uzel. Pokud má více vstupních uzlů, tak dochází ke spojování (ukončení cyklu). To probíhá, tak že se najde stejný uzel v těchto dvou a více zásobnících neboli místo jejich rozdělení. Abychom věděli, kterým uzlem byla rozdělena určitá kružnice si ukládám předchůdce do bodu sjednocení. Uzel sjednocení si zároveň ukládám do všech, které byli výše v zásobníku nežli uzel rozdělení (byli obsaženy v tomto cyklu) a zároveň všechny odstraním. Tím pádem zůstane v zásobníku pouze uzel vyskytující se před tímto cyklem.

Pokračuji druhým uzlem v pořadí. Když procházím uzel, tak nejdříve získám všechny zásobníky jeho vstupních uzlů navýšené vždy o jejich hodnotu. Pokud je pouze jeden, pak pouze uložím. V opačném případě dochází ke spojování(ukončení kružnice). To probíhá, tak že se najde stejný uzel v těchto dvou a více zásobnících neboli místo jejich rozdělení. To si uložíme jako předchůdce

do aktuálního uzlu, abychom věděli, kterým uzlem byla rozdělena určitá kružnice. Uzel sjednocení se zároveň ukládá do všech, které byli výše v zásobníku nežli uzel rozdělení (byli obsaženy v tomto cyklu) a zároveň všechny odstraním z aktuálního zásobník. Tím pádem zůstane v zásobníku pouze uzel vyskytující se mimo tuto kružnici. Na 4.6 je vidět příklad grafu po předzpracování.



Obrázek 4.6: Výsledek předzpracování - nad uzly jsou následníci a pod předchůdci

4.3.6 Rozmístění jednotlivých uzlů

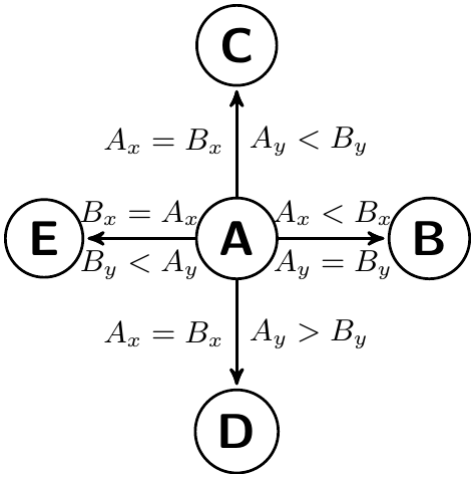
Teď jsem připraven nastavit směry (vzájemné vztahy uzlů) pro všechny hrany grafu viz 4.7. Ty prakticky určují směr hrany neboli pozici vzhledem k ostatním uzlům. V totmto příkladu se musí C nacházet severně od A.

4.3.6.1 Definice směru

Směr části grafu pouze určuje, že nesmí být vytvořena cesta v opačném směru, a že kus grafu vede určitým směrem, aby mohli být uzavřeny cykly bez jakéhokoliv problému. Takže, když je směr například doprava, tak se nesmí vytvořit cesta vedoucí zpět. Na první pohled to vypadá, že to asi ani nejde, ale směr může vést doprava, ale hrana i nahoru a dolů.

4.3.6.2 Průběh

Začínám prvním uzlem v topologickém uspořádání a zvolím pro něj náhodný směr. Postupuji dál pomocí topologického uspořádání a řeším jednotlivě všechny



Obrázek 4.7: Rozmístění uzlů pomocí vzájemných vztahů

uzle. Nejdříve vstupní a následně výchozí hrany.

4.3.6.2.1 Vstupní hrany Pokud aktuální uzel má více vstupních hran, pak dochází k uzavření cyklu. V tomto momentu se nám bude hodit odkaz na předchůdce (místo, kde došlo k rozdělení). Ten využijeme k zjištění, která cesta má být připojena shora, zespoda či přímo. Zpětně prohledáváme, dokud nenalezneme předchůdce a podle směru, ve kterém byl nalezen propojíme.

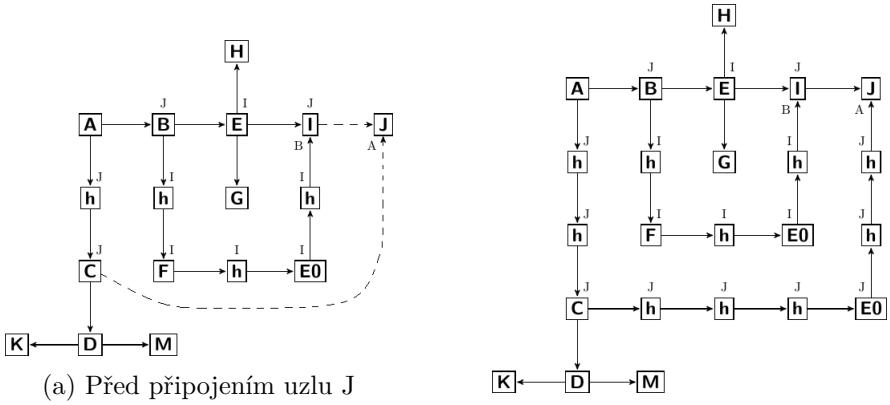
4.3.6.2.2 Výchozí hrany Pro daný uzel se zjistí všechny vycházející uzly, které se nespojují do jednoho místa. Pokud mají následovníka, tak se musí umístit náhodně do směru a zároveň vedle sebe ti se stejným následníkem. V některých případech se musí vytvořit uzel navíc, aby bylo možné toho dosáhnout. Například směr vede doprava, ale vstupní cesta do uzle vede zespoda (nahoru). Když se rozděluje na tři další uzle, pak to není možné udělat bez přidání dalšího uzlu. Pokud uzel, ale nemá následovníka, pak je jedno jakým směrem ho umístíme, ale musíme mu nastavit nový směr.

4.3.7 Převedení prostoru na 2D Grid

Nejdříve se vytvoříme 2D pole, která má možnost se zvětšit pokud je to nutné. Následně umístíme první uzel topologického uspořádání doprostřed pole. A pokračujeme umísťování dalších uzlů v uspořádání. Pokud je místo volné, pak uzel můžeme umístit. V opačném případě musíme uvolnit místo a to posunutím překážejících uzlů pryč. Je tady, ale víc možností a problémů. Nelze vždy posunout směrem, kterým se snažíme postupovat.

Rozhodujeme se vzhledem k tomu, zda má uzel následníka. Když nemá, pak můžeme vždy posunout směrem, kterým uzel umísťujeme. V opačném případě není možné vždy posunout tímto směrem. Jelikož by mohla nastat

situace viz 4.8 a dojít k chybě. V tom případě, když má uzel následníka, a zároveň je směr části grafu stejný, jako směr umístění. Pak je nutné posunout jedním ze směrů kolmých na původní směr. Viz příklad , kde se G muselo posunout směrem nahoru při přidávání uzle I.



(a) Před připojením uzlu J

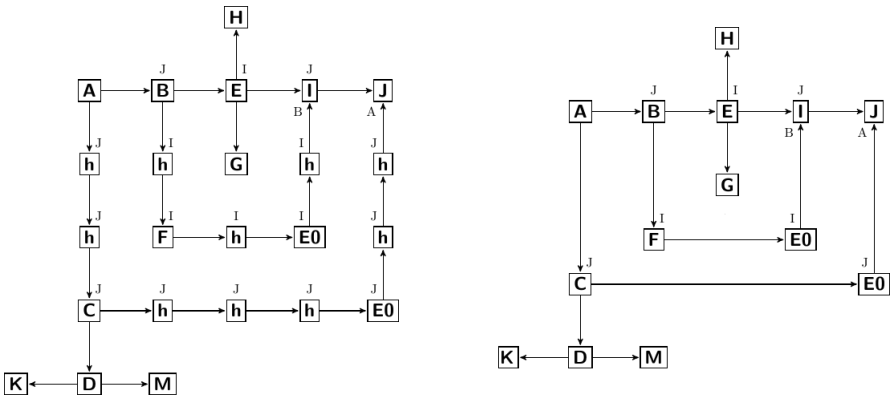
(b) Po připojení uzlu J

Obrázek 4.8: Posunutí a následné připojení uzlu J

4.3.8 Příprava pro generování 3D geometrie

Protože, jsme potřebovali přidat informaci do všech míst v 2D mřížce, musíme je následně odebrat. Jinak by bylo možné umístit na místo obsahující cestu a nedošlo by k odsunutí pryč. Pouze se smažou uzle a změní se propojení cest.

K odebrání prázdných místností proběhlo po prvním experimentu, kde hráčům přišlo nepříjemné se rozhlížet, když nebylo možné stejně nikam odbočit. Připravený prostor je vidět na 4.9.



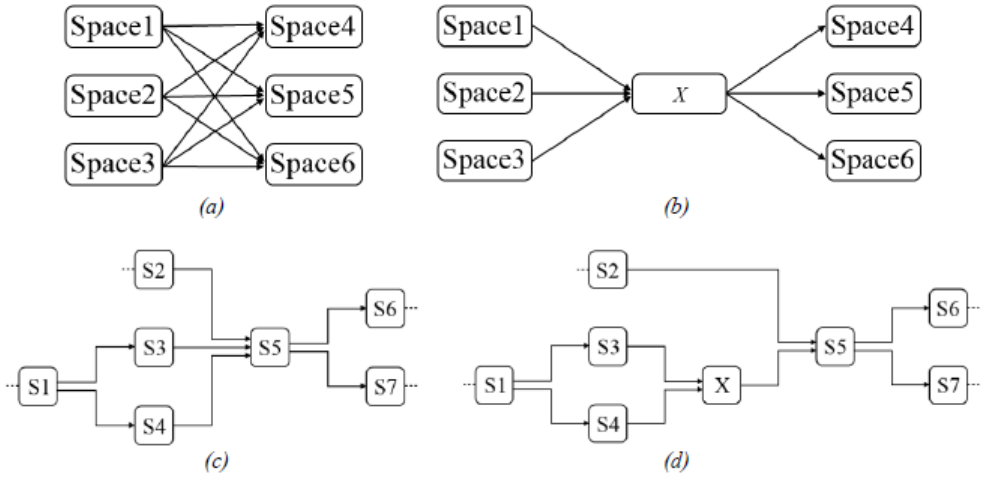
Obrázek 4.9: Odebrání prázdných místností z grafu

4.3.9 Původní implementace autora

Zde naznačím, jak bychom implementovali části, které jsme k našemu případu nepoužili.

4.3.9.1 Planarizace

Autor [Linden et al., 2013] provádí kontrolu a následnou opravu grafu na planární graf. Nejdříve nachází problémová místa a podgrafy, ve kterých dochází k porušení planárního grafu. Jelikož graf je rovinný právě tehdy, když neobsahuje dělení K_5 nebo dělení $K_{3,3}$ jako podgraf. Protože se jedná o orientovaný acyklický graf, pak není možné aby vznikl K_5 graf, protože každý uzel kromě prvního musí mít aspoň jednu vstupní hranu. Podgraf, který způsobuje neplanaritu se odstraní přidáním uzlu mezi tyto hrany viz 4.10.



Obrázek 4.10: a) graf $K_{3,3}$, b) vyřešení grafu $K_{3,3}$ přidáním uzlu, c) příklad, kde S5 má více hran než 4, d) vyřešení (c) pomocí přidání uzlu X viz [Linden et al., 2013]

4.3.9.2 Snížení počtu hran na čtyři na místnost

Hledáme uzly, které mají více hran než čtyři, postupným průchodem od startovního uzlu. Zjistíme, na které straně dochází k problému. Pokud je více vstupních, pak nalezneme největšího společného předchůdce (v opačném případě společné následovníky). To můžeme udělat velmi podobně, jako jsme vyhledávali následovníky a to nejdříve vytvořit topologické uspořádání a pak procházet po vstupních hranách zpět k počátku. Ukládat si všechny uzle do zásobníku, a pokud z uzle vychází více hran (předek má více výstupních hran), pak zjistím zdali spojuje dané uzle. Pokud mezi uzle se stejným předchůdcem/-

následovníkem přidám další uzel (prázdný), pak tím snížím počet hran viz (c) na 4.10.

4.3.9.3 Shrnutí průběhu

1. Topologické uspořádání.
2. Procházíme a hledáme místo, kde dochází k překročení počtu možných hran
3. Hledá předka/následníka obsahujícího co nejvíce uzlů, se kterými je propojený uzel, který má více hran než je možné.
4. přidá se uzel, tak aby se nezničila původní strukturu. A to tak, že umístí spolu uzle, které mají společného předka/následníka, který byl nalezen v (3).
5. Zkontrolujeme, jestli stále není převýšen počet hran. Pokud je, pokračujeme(3).
6. Protože, tím mohly být narušeny počty hran pro předešlé uzle, pak se musí rekurzivně pokračovat zpět k startovnímu uzlu.

4.3.9.4 Následné opravení dlouhých cest

Rozmísťování, které používá [Linden et al., 2013] vede k vzniku dlouhý chodeb. A to převážně pro Y souřadnice, které rozmísťuje autor jako první.

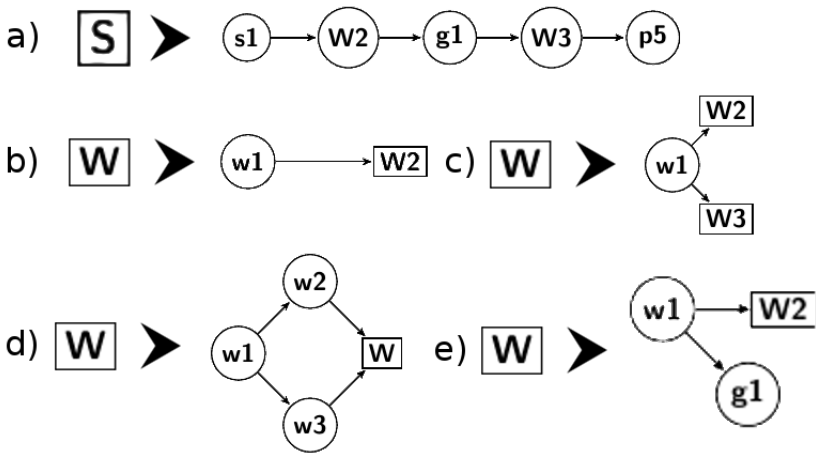
Algoritmus hledá místa, kde dochází k velkému natažení cest a snaží se je posunout jedním směrem v ose Y a směru umožňující zkrácení(nahoru/dolu). Ostatní uzle jsou odstrčeny pryč z cesty a následně se algoritmus snaží umístit je co nejblíže zpět. Tento proces pokračuje dokud bylo pohnuto se všemi vrstvami Y nebo již nezůstala žádná dlouhá hrana.

4.3.10 Nastavení obtížnosti

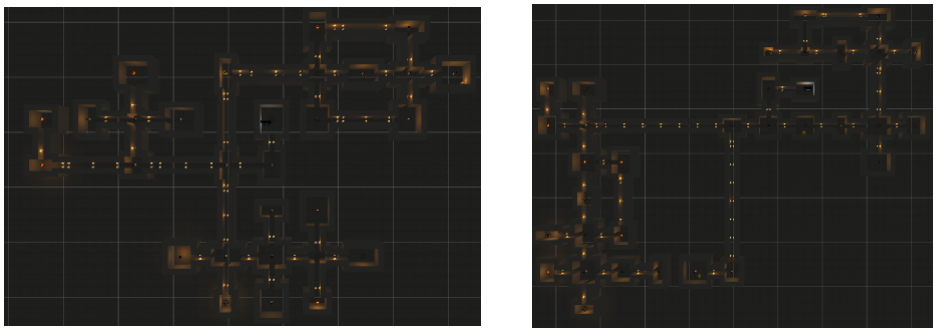
Obtížnost je ovlivněna převážně nastavením gramatiky a nastavením parametru určujícím maximální velikost grafu v uzlech/místnostech. Ten se náhodně vybere z rozmezí definovaného pro určitou obtížnost. Všechny parametry a nastavení metod jsou zobrazeny v tabulce 4.2. Zde pro ukázku zmíním nastavení pro jednoduchou obtížnost viz 4.11. Na gramatiku pro střední a těžkou obtížnost je možno se podívat v přiložených souborech viz tabulka 4.2.

Tabulka 4.2: Parametry a nastavení pro generování pomocí gramatiky

Název	Datový typ	Easy	Medium	Hard
počet místností	interval	(15,20)	(25,30)	(40,45)
gramatika	textový soubor	easygrammar	mediumgrammar	hardgrammar



Obrázek 4.11: Příklad gramatiky (použita pro jednoduchou obtížnost)



Obrázek 4.12: Příklady výsledků metody - gramatika

4.3.11 Výsledky

4.4 Rozdělování prostoru (Space partitioning)

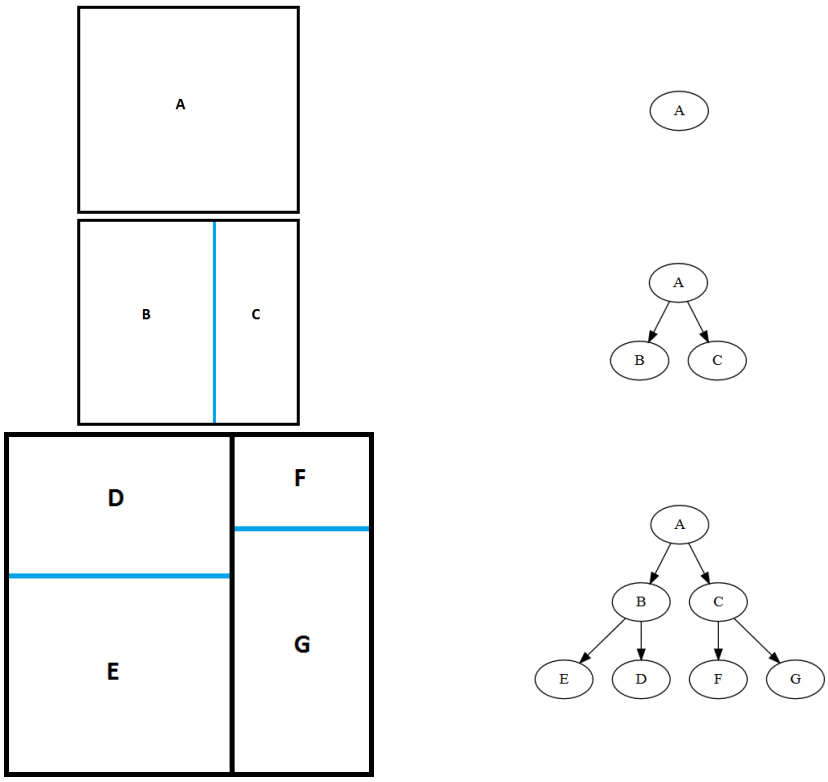
Rozhodl jsem se použít nejtypičtější metodu rozdělování a to na dvě nerovnoměrně velké části. Možností bylo použít jiné množství dělených částí na 4 a více, ale to nevede k lepším výsledkům, pouze ke komplikaci algoritmu a ztrátě jeho významu vzhledem k zjednodušení propojování místností a cest.

Algoritmus má tři fáze rozdělení prostoru, vytvoření místností v listech a propojení prostorů. Poslední fází je vygenerování reálné geometrie, ta probíhá

ve všech algoritmech stejně pomocí rozhraní, o tom se zmíním později v sekci 4.1.1.

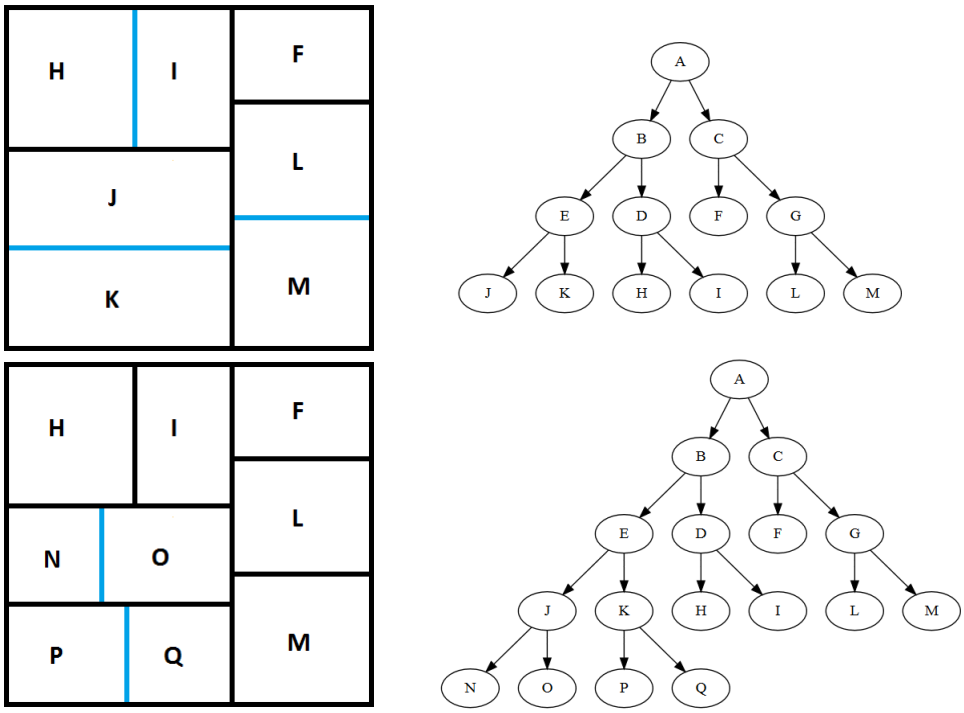
4.4.1 Dělení prostoru

V první fázi dostane algoritmus jako vstup prostor (kořen stromu) s určitou výškou a šírkou. Následně se prostor dělí vzhledem k aktuální šanci na dělení vertikálně nebo horizontálně a jsou přidávány jako potomci děleného uzle. Pokud proběhne vertikální dělení, pak se sníží jeho šance. Mohly by vznikat prostory, kterou jsou až příliš často děleny jedním směrem, a docházet k úzkým a podlouhlým prostorům. Ty by byly ve finále stejně eliminovány tím, že se dělí, dokud jde dělit aspoň jedním směrem, ale vedlo by to k jedné dlouhé cestě (vertikální/horizontální). Dalším problémem by mohlo být rozdělení na jedné straně velmi malé části a na druhé příliš velké. To je řešeno omezením maximálního dělení na $\frac{1}{4}$ na každé straně a pokud $\frac{1}{4}$ je menší než minimální velikost místnosti, pak musí být děleno více u středu. Dělit se dá, dokud velikost místnosti není menší než dvakrát minimální místnost.



Obrázek 4.13: Výběr místnosti k propojení pomocí stromu

Pro lepší efektivnost se při dělení uzle ukládají do zásobníku po vrstvách stromu (kořen, dva jeho potomci ,..). Ten se pak může rovnou použít k pro-



Obrázek 4.14: Dělení prostorů

pojování místností.

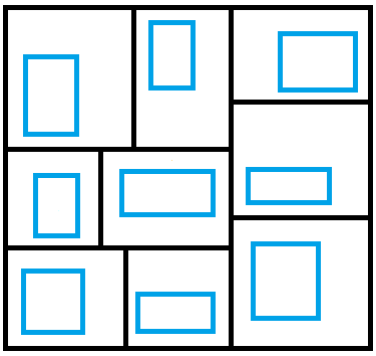
4.4.2 Vytvoření místností v listech

Když už máme prostor rozdělený, musíme vytvořit v každém listu místnost (upravit rozměry listu). Protože propojujeme i cestami do z, pak musíme uvolnit okraj o šířce zdi + $\frac{\text{šířce cesty}}{2}$ na obou stranách. Zároveň musíme vytvářet místnosti velké vzhledem k velikosti původního kusu a tím zajistit jak lepší možnosti propojení cest tak zachování struktury rozděleného prostoru.

4.4.3 Propojení prostorů

Propojování prostorů probíhá od listů ke kořenu. Všechny uzle po dvojicích a po vrstvách, jak již bylo zmíněno. Popis toho, jak se propojí místnosti, není přímo definován v algoritmu. V našem případě se propojuje dvěma typy cest rovnými a do z (z-cesty). Typem z propojuji pouze v krajním případě, když není jiná možnost.

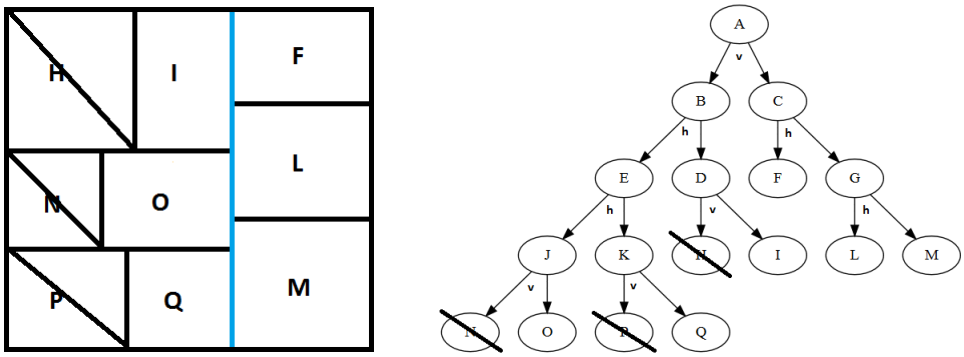
Algoritmu začíná tím, že získáme následující vrstvu. A procházíme uzle po dvou. Pokud jsou oba uzle listy, pak je můžeme jednoduše vyřešit jenom tím, že je zkusíme propojit přímo, a pokud to nejde, propojíme pomocí z-cesty.



Obrázek 4.15: Vytvořené místnosti v rozdělených částech

Pokud nejsou, pak se prohledává strom, a naleznou se všechny listy, které se dají propojit jak pro levý uzel tak pro pravý.

4.4.3.0.1 Naleznutí uzlů k propojení To se dělá tak, že pokud tyto dva uzly byly rozděleny například vertikálně. Pak vezmu levý uzel a procházím ho směrem k listům. Pokud je potomek rozdělený vertikálně, pak vstupuji do obou uzlů a pokračuji. Pokud ne, tak beru pouze pravý podstrom. Stejně tak pro pravý, akorát když narazíme na vertikální dělení, tak místo zahození levého podstromu, zahodíme pravý. Detailněji můžete vidět na 4.16.



Obrázek 4.16: Výběr místnosti k propojení pomocí stromu pro B a C

4.4.3.1 Spojení uzlů do intervalů

Když máme k dispozici levé a pravé uzly, spojíme je do intervalů, které jsou propojené, aby jsme snížili časovou náročnost a zároveň přidali možnost propojit s cestou.

Protože jsou uzle uspořádány zleva doprava, můžeme rychle zjistit, zdali se dají spojit do intervalu. Vezmeme první uzel, zleva zkusíme, jestli z něj vede cesta (vertikálně - dolů, horizontálně doprava). Pokud uzel na druhé straně

cesty obsahuje tento uzel, pak můžeme přidat do intervalu. Mohlo by se zdát, že může přidat, každý když vede cesta dolů/doprava, ale cesta může vést mimo vymezenou plochu a nebo odbočit pokud je to T-cesta (cesta navazující na jinou cestu).

4.4.3.2 Průnik intervalů a vytvoření cesty

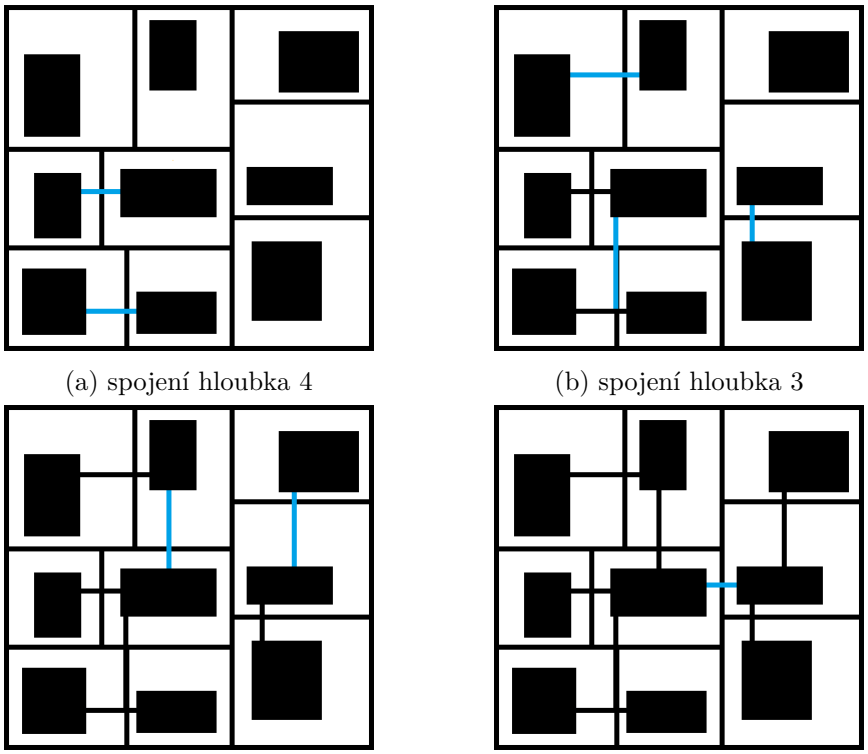
Když mám intervaly pro obě strany, pak vyberu náhodně jeden ze strany obsahující méně intervalů a zamíchám intervaly na opačné straně pro větší náhodnost. A začnu zjišťovat jejich průnik. Pokud jsem jej nalezl, tak již nehledám dále. Do průniku těchto intervalu bodnu na náhodnou pozici a umístím tam cestu. Pokud to ale není možné, tak získám průnik těchto objektů (to do čeho narazím na obou stranách). Je-li interval neprázdný, tak vytvořím cestu. Jinak vymažu tento interval s průniku. A zkouším bodnout znovu, dokud nevytvořím cestu. Jakmile bude průnik prázdný, musí algoritmus propojit pomoci z-cesty. To dělá ve dvou případech, buď když nenašel průnik na začátku, nebo ho nenašel na konci. Když nebyl na začátku žádný průnik, pak vezme nejbližší co našel. Bodne do středu jejich mezery a najde dva nejbližší uzle a posune střed mezi ně a zkusí najít znovu. Pokud se již nic nemění, pak našel dva nejbližší uzly a nemůže vzniknout překřížení (cesty s jinou místností, než do které směřuje). Pokud je to na konci, pak náhodně bodne někam do (původního) průniku. Najdou se nejbližší uzle v intervalech a pak se provádí to samé jako u minule zmíněného problému.

4.4.3.3 Vytvoření cyklu

Aby algoritmus vytvářel komplikovanější prostor, obsahuje algoritmus šanci na vytvoření dalšího propojení. Ta je omezena parametrem kolik maximálně propojeni může být vytvořeno. Normálně budeme používat pouze dva na vytvoření cyklu. Více není potřeba, ale algoritmus s tím umí pracovat (ani by nebylo ve většině případů ku prospěchu). Cyklus se vytváří tak, že se buď pokračuje dál, a to v tom případě, pokud je průnik pořád nenulový, a nebo se skončí, vybraný interval se smaže a začne se pracovat s dalším náhodným stejně jako v předchozím případě.

4.4.3.4 Umístění klíčů/zámků

Klíče se umísťují náhodně v průběhu vytváření cesty (ne pro kořeny v tom případě by to nemělo smysl). Vytvářejí se pouze, když se propojuje jednou cestou, aby se zabránilo možnosti místo obejít bez problémů a nutnosti hledat klíč. Než popíši, jak se bude umisťovat klíč, musím zmínit, kde bude umístěn hráč a portál. Hráč bude umístěn náhodně do nejvíce levého/pravého listu a portál na opačnou stranu do pravého listu, tím je zaručena velká vzdálenost mezi nimi. Teď zpět k umístění zámků, ten se umístí na jednu stranu chodby.



(a) spojení hloubka 4

(b) spojení hloubka 3

(c) spojení hloubka 2

(d) spojení hloubka 1

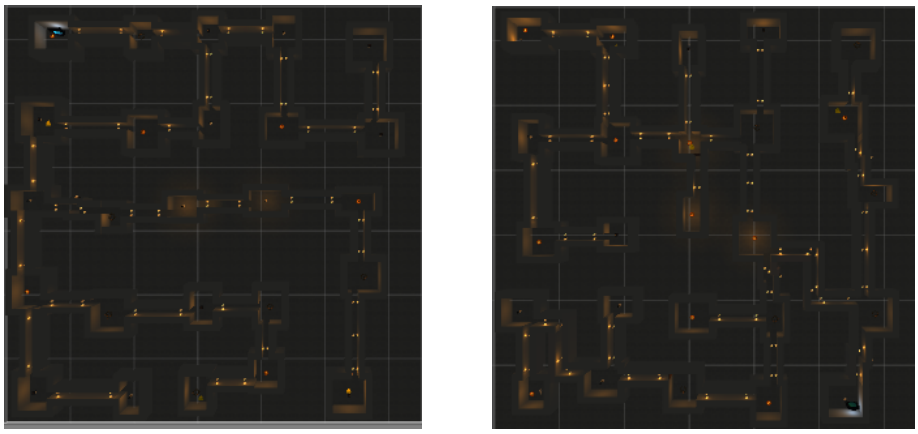
Obrázek 4.17: Propojování místností po vrstvách

A klíč pak na náhodně na stranu, kde je hráč do jednoho z listů. Pro příklad hráč je na pravé straně, tak se vezme levý uzel ke spojení a začne se náhodně (vlevo/vpravo) postupovat k listu a do jednoho z nich se umístí klíč. S umísťováním peněz mi algoritmus nějak neumožňuje chytře pracovat. Ty jsou umisťovány až po doběhnutí algoritmu, náhodně do listů.

Tabulka 4.3: Vstupní parametry a nastavení

název	datový typ	easy	medium	hard
rozměry prostoru	(šířka,výška)	(50,50)	(65,65)	(90,90)
min. poč. propojení	int	1	1	1
max.poč propojení	int	1	2	2
interval délky pro odbočku	(min,max)	(1,5)	(1,8)	(1,10)
šance gold	int	20	20	20
šance klíč/zámek	int	0	15	20
šance dal. propojení	int	0	15	20

4.4.4 Výsledky



Obrázek 4.18: Příklady výsledků metody - dělení prostoru

Kapitola 5

Výsledky experimentů

5.1 Kvalitativní studie

5.1.1 Zjištění

Prvního experimentu se zúčastnilo 17 lidí. Zde uvidíme pouze pár z těch nejužitečnějších experimentů a ostatní budou přidány do přiloženého pdf.

Lidé si všímali dlouhých rovných chodeb, a pokud bylo méně odboček ve výsledku se hráč nemusel výrazně moc rozhodovat. Ale pokud tam byla delší chodba a hráč nemohl odbočit jinam než pokračovat dál, tak to účastníkům výrazně nevadilo. Proto byla vybrána jako jedna charakteristika „klikatost“ (1 málo - hodně 5) prostoru. A tím se myslí, že prostor není přímý a neobsahuje dlouhé rovné cesty. Další charakteristika vyšla z poznatků, kde se hráčům nelíbilo, že měli málo možností se rozhodnout v některých úrovních, v nichž nebylo moc rozhodování po cestě k cíli, a tím pádem nebylo těžké se v prostoru orientovat nebo v některých případech byl hráč zmatený a nevěděl jestli již daným prostorem prošel. A proto byla přidána vlastnost „orientace“ (1 dobrá - špatná 5), byla zde možnost ještě přidat počet rozcestí (málo - hodně), ale přišlo mi lepší to nechat spojené jako orientace. Z více důvodů a jedním je i ten, že by to bylo směrodatné pouze u nejlehčí obtížnosti a schopnost orientace je výstižnější a méně subjektivní. Dalším a očekávaným byla obtížnost, kterou zmínili všichni účastníci, což bylo převážně první proč jim přišla úroveň lepší, a to z důvodu větší obtížnosti. Dále mluvili subjektivně o tom, že je bavila první nebo druhá úroveň více, proto byla přidána „zábavnost“.

Finální kvantitativní experiment proto bude obsahovat 4 charakteristiky (zábavnost, orientace, klikatost a obtížnost) viz obrázek 5.1. Účastníci prvního experimentu se také můžou účastnit druhého. Zároveň jsme jim poslali náš návrh charakteristik, aby jsme zjistili, zdali s nimi souhlasí, a jestli by některé neodebrali nebo nepřidali. Účastníci souhlasili, akorát někteří si nebyli jistí co znamená klikatost a proto byla přidaná informace o přesném znění této charakteristiky, aby nedošlo k nedorozumění a k ovlivnění experimentu. Jelikož

jsme umožnili možnost zrušit odeslání dat experimentu, proto kontrolujeme zda hráči převážně vyplňují dotazníky. Tuto možnost jsme tam dali hlavně z důvodu nemožnosti odeslat zprávu v neschopnosti se připojit k internetu.

Obrázek 5.1: Zvolené charakteristiky

5.1.2 Opravy a chyby

V průběhu prvního experimentu a po něm byly následně upraveny nastavení metod a drobné chyby v algoritmech a přidány možnosti nastavení ovládání kvůli názoru hráčů na špatnou rychlost otáčení a ovladatelnost. Zároveň bylo zjištěno, že některé zařízení mají problém s velkým množství „pochodní s ohněm“ (particle systemů) a byla přidána možnost se tomu vyhnout a ušetřit na náročnosti. Také došlo k přidání zobrazení severu jako kontrolního bodu.

5.1.3 Nastavení pro Druhý experiment

Aby bylo jednodušší pro hráče poznat, kde se nacházejí, byl nastaven větší rozsah velikosti místností, a zároveň byly přidány do místností různé typy stropních lustrů/lamp. Také bylo upřesněno nastavení velikostí místností, aby bylo rozmezí stejné u všech algoritmů. Přibližně jsme se snažili nastavit obtížnost na určitý počet místností a učinit tak pro všechny úrovně. Nedalo se toho úplně dosáhnout, protože například dělení prostoru dosahuje velmi podobných hodnot vytvořených místností. Zatímco agent má velký rozptyl a není možné ho úplně kontrolovat a může se stát, že se agenti rychleji zahrabou. Pro grammar-based bylo nastavené určité rozmezí počtu místnosti(5). V tomto ohledu pouze u agenta dochází k většímu rozsahu počtu místnosti a není tímto směrem úplně stabilní.

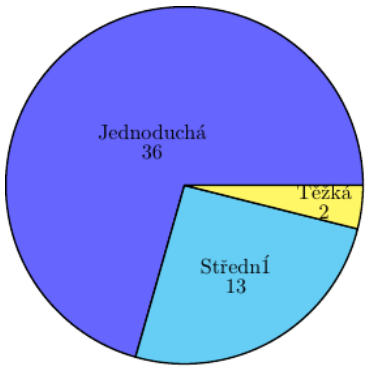
5.2 Kvantitativní experiment

Z prvního experimentu, jsme si odnesli 4 charakteristiky viz níže nebo již zmíněný obrázek 5.1.

- obtížnost(1 lehká - těžká 5)
- klikatost(1 málo - hodně 5)
- orientace(1 lehká - těžká 5)
- zábavnost(1 málo - hodně 5)

Celkem si hru zahrálo 25 lidí a vyplnilo se 51 dotazníků. To znamená, že hráč hrál průměrně 2 úrovně(2.04). K odmítnutí odeslat dotazník došlo pouze v 6 případech.

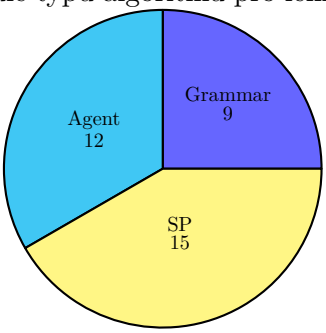
Očekávali jsme větší účast, ale kvůli časové náročnosti her a nutnosti instalovaní aplikace, bylo toto číslo výrazně nižší. Zároveň byl očekáván větší počet her na účastníka. Jelikož první experiment, byl časově náročnější, tak mohlo dojít k odrazení účastníků prvního experimentu od druhého. Kde byla výrazně snížena celková obtížnost pro všechny typy obtížností. Dále jsme pozdě zjistili, že by bylo dobré získávat data již při vytvoření úrovně. Jelikož, ale experiment již začal, nemohli jsme do něj zasahovat. Zároveň by jsme neměli jistotu, že všechna data budou odeslaná nebo by jsme museli kontrolovat pořád přístup k internetu. Což by mohlo hráče ještě více odradit. Teď můžeme jenom diskutovat nad tím, zda hráči nebyli schopni dokončit určité typy úrovní či obtížností.



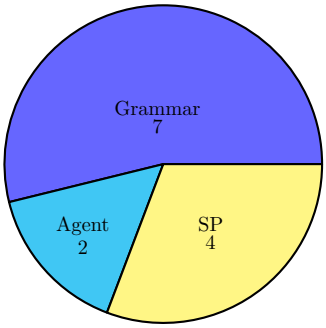
Obrázek 5.2: Celkový počet experimentů podle obtížnosti

V této sekci samostatně zhodnotím pouze střední a lehkou obtížnost, jelikož máme pouze 2 záznamy pro těžkou úroveň. To mohlo nastat z důvodu časové náročnosti úrovně. Obě metody trvali více něž 5 minut. Ta delší skoro 10 minut. Rozdíl mezi těmi to dvěma experimenty mohl být ovlivněno pouze

schopnostmi hráče a nedá se z toho více vypozorovat. Dalším možným důvodem mohl být začátek účastníků na jednodušší úrovni. Jelikož průměrně odehráli 2 hry, pak se k těžké obtížnosti většinou nedostali. Počet dotazníků podle typu algoritmu pro lehkou a střední obtížnost je vidět níže na 5.3 a 5.4.



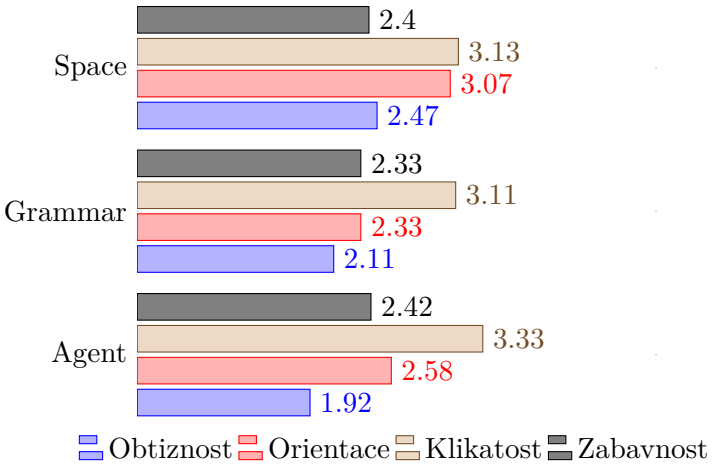
Obrázek 5.3: Lehká obtížnost



Obrázek 5.4: Střední obtížnost

5.2.1 Vyhodnocení pro jednoduchou obtížnost

K lehké obtížnosti jsme získali 36 vyplněných dotazníků viz 5.3. Když se podíváme na získaná data. Prvním rozdílem je u dělením prostoru, kde hráči udávali výrazně horší schopnost orientace a vyšší obtížnost. Zatímco zábavnost se výrazně neliší a nezdá se, že by korelovala z jinou charakteristikou. Klikatost se drží u střední hodnoty pro všechny hodnoty. Její výsledky budou pravděpodobně razantnější pro střední obtížnost, jelikož se tam můžou vlastnosti metody více projevit.



Obrázek 5.5: Jednoduchá obtížnost - průměrná hodnota charakteristik

Vzhledem ke gramatice a dělení prostoru by jsme mohli předpokládat, že koreluje obtížnost s orientací. To ale neplatí pro agenta, ale to může mít jinou

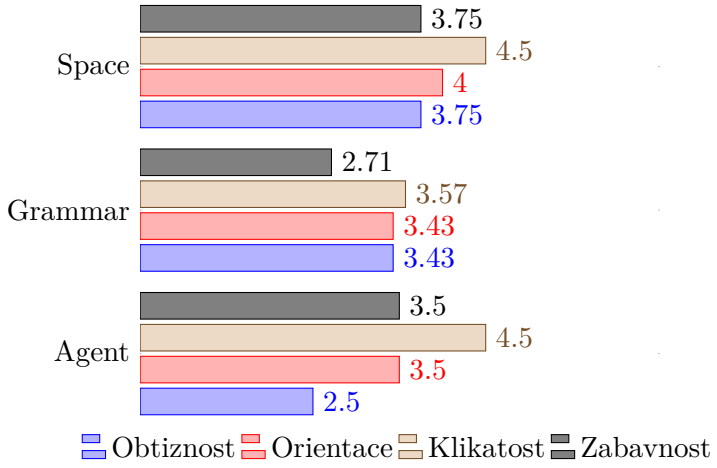
příčinu. Když se podíváme na tabulku 5.1, pak můžeme vidět rozdíl v délce času agenta. Nemůžeme, tedy s jistotou tvrdit tuto korelaci, ale nemůžeme ji ani vyvrátit. Když se podíváme přímo na čas a obtížnost, tak vidíme, že čas je závislý na vyplněné obtížnosti.

Tabulka 5.1: Porovnání času a peněz pro jednoduchou obtížnost

Název	Agent	Gramatika	Prostorové dělení
průměrný počet mincí na úroveň	2.8667	2.4444	3.2500
procento sebraných mincí	63%	83%	74%
průměrné trvání úrovně	2:06	3:17	3:01

5.2.2 Vyhodnocení pro střední obtížnost

Jelikož, ke střední obtížnosti máme pouze 13 dotazníků, navíc nejsou rovnoměrně rozprostřeny(agent pouze 2 záznamy), není možné brát vyhodnocení dat jako vysoce určující. Zároveň nastala situace, kdy počet dat pro gramatiku přesahuje poloviční většinu. Pravděpodobnost, takové situace je velmi mála a snižuje věrohodnost získaných informací.



Obrázek 5.6: Střední obtížnost - průměrná hodnota charakteristik

Pokud se podíváme na průměrná data ze všech výsledků. Pak je výrazně nižší klikatost pro generování pomocí gramatiky, i když je nadprůměrná 3,57). Pro gramatiku se dostala zábavnost těsně podprůměr o proti ostatním, které byli hodnoceny o dost zábavněji o proti jednoduché obtížnosti. Z dat není výrazné co bylo důsledkem výrazného poklesu. Možnými důvody může být menší klikatost nebo přísnější hodnocení hráčů. Jelikož tento typ vychází z nejvíce vstupních dat, je spíše pravděpodobnější příčinou snížení klikatosti.

Agentovi stále zůstává obtížnost výrazně nižší než u ostatních metod, ale není zde náznak vztahu mezi jinými charakteristikami. Může to být pouze výchylka hodnot, protože máme pouze dvě vstupní data. Pokud se zaměříme na čas agenta je prakticky poloviční a to bude důsledkem nižší obtížnosti.

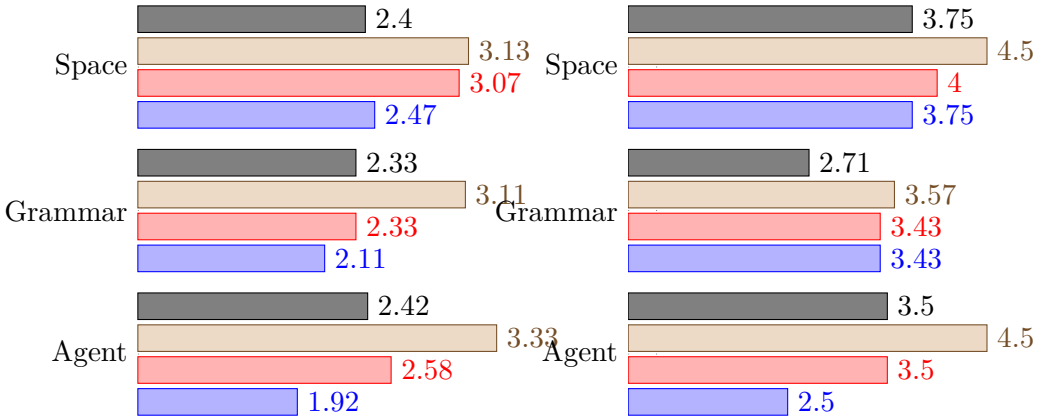
Tabulka 5.2: Porovnání času a peněz pro střední obtížnost

Název	Agent	Gramatika	Prostorové dělení
průměrný počet mincí na úroveň	5.5000	2.0000	6.5000
procento sebraných mincí	58%	86%	68%
průměrné trvání úrovně	1:41	5:03	3:04

5.2.3 Celkové výsledky pro jednoduchou a střední obtížnost

Střední obtížnost vyšla zábavnější, díky vyšším hodnotám všech ostatních charakteristik a delší době hraní. Předpokládám, že to je tím, že hráčům přišla střední obtížnost jako vyšší výzva a jednoduchá obtížnost nebyla dost obtížná, aby mohla být zajímavá.

Pro střední a jednoduchou obtížnost platí, že agent má stále nejlehčí obtížnost(z důvodu času). Zároveň pořád vychází nejzábavnější dělení prostoru a to díky vyšší obtížnosti, horší orientaci a optimálnímu času (kolem 3 minut). Zatímco gramatika dopadla výrazně hůře, díky snížení klikatosti prostoru.



Obrázek 5.7: Porovnání jednoduché a střední obtížnosti

Závěr

Shrnuti

Porovnali jsme kolem 15 metod a vybrali 3 metody, nejlépe splňující naše požadavky hry a to „Gramatikou řízené generování“, „Agentem řízené generování“ a „Dělení prostoru“. Navrhli jsme experiment zjišťující charakteristiky, kterých si hráči všímají a hodnotí při hraní hry. Zjištěné charakteristiky byli orientace, obtížnost, klikatost a zábavnost. V další fázi byl proveden kvantitativní experiment, kde bylo získáno 51 dotazníku od 25 lidí. Po analýze dat bylo zjištěno, že nejlépe splnila požadavky metoda „dělení prostoru“. Ta dosáhla stabilních výsledků ve všech směrech a vyšší zábavnosti než ostatní metody.

Dělení prostoru

Metoda splnila všechny požadavky a je vhodná pro použití pro tento případ hry/prostoru. Překvapující byla stálost a vysokost obtížnosti, která přesahovala ostatní metody. Jelikož není možné zajistit vznik odboček a můžou být vytvořeny v některých případech klikaté chodby bez odboček.

Agentem řízené generování

Generování pomocí agenta dosáhlo velmi podobných výsledků jako dělení prostoru, pouze nastávalo k rychlejšímu dokončení úrovní hráči. Tato situace mohla nastat, kvůli vygenerování menšího prostoru. Jelikož je tato metoda nejméně stabilní na počet místností, které vytváří a nedá se to přesněji specifikovat na rozdíl od ostatních. Vzhledem k počtu procent (63% a 58%) sebraných mincí je možné, že hráči nalezli dříve portál, a pak už nepokračovali. U ostatních metod jsou počty sebraných mincí vyšší.

Pokud by hráč věděl, že se jedná o agentem řízeném generování mohla by se také zvýšit jeho šance na rychlejší vítězství. Jelikož nejdříve se vytváří hlavní cesta k cíli a klikaté cesty se vytvářejí pouze, když není možnost umístit

místnost. To se u prvního agenta většinou nestane, a proto rozhodnutí jít rovnou cestou, by mělo vést rychleji k cíli.

Metoda splňuje všechny požadavky a je vhodná pro použití pro tento případ.

Gramatikou řízené generování

Pro generování pomocí gramatiky došlo k výraznému snížení klikatosti pro střední obtížnost, která negativně ovlivnila její hodnocení zábavnosti. To se mohlo pravděpodobně ještě více projevit pro těžkou obtížnost, ale na to jsme bohužel neměli dost dat.

Vytváření delších chodeb vzniká při uvolňování prostoru, pokud není možné umístit uzel na určené místo a musí se posunout kus grafu pryč. Pokud by jsme chtěli dosáhnout lepších výsledků. Museli by jsme použít jiný časově náročnější přístup rozmisťování uzlů a to například viz[Valtchanov and Brown, 2012a], který ale nesplňuje naší časové požadavky.

Metoda je akceptovatelná pro menší úrovně, ale není úplně vhodná pro tento typ prostoru/hry. Jelikož původní autor umisťoval pouze jedním směrem a možnost přidání směru nestačilo pro kompletní uspokojení podmínek.

Budoucí pokračování

V následující práci bylo dobré zjistit, jak by si vedli metody a jakých výsledků by jsme byli schopni dosáhnout při před-generování, a tím pádem pro časově náročné metody. Dalším pokračováním by bylo zaměření se na kombinování a možnosti využití více metod v jednom případu. Zjišťování dat v průběhu hry a možnosti reagovat na rozhodnutí hráče ve hře.

Literatura

Joris Dormans. Adventures in level design: Generating missions and spaces for action adventure games. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, PCGames '10, pages 1:1–1:8, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0023-0. doi: 10.1145/1814256.1814257. URL http://doi.acm.org/10.1145/1814256.1814257.

Ken Hartsook, Alexander Zook, Sauvik Das, and Mark O. Riedl. Toward supporting stories with procedurally generated game worlds. In Sung-Bae Cho, Simon M. Lucas, and Philip Hingston, editors, *CIG*, pages 297–304. IEEE, 2011. ISBN 978-1-4577-0010-1. URL http://dblp.uni-trier.de/db/conf/cig/cig2011.html#HartsookZDR11.

Remco Huijser, Jeroen Dobbe, Willem F. Bronsvoort, and Rafael Bidarra. Procedural natural systems for game level design. In *Proceedings of the 2010 Brazilian Symposium on Games and Digital Entertainment*, SBGAMES '10, pages 189–198, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4359-8. doi: 10.1109/SBGAMES.2010.31. URL http://dx.doi.org/10.1109/SBGAMES.2010.31.

Roland van der Linden, Ricardo Lopes, and Rafael Bidarra. Designing procedurally generated levels. In *Proceedings of IDPv2 2013 - Workshop on Artificial Intelligence in the Game Design Process, co-located with the Ninth AAAI Conference on Artificial Intelligence in Interactive Digital Entertainment*, pages 41–47, AAAI Press, Palo Alto, CA, oct 2013. AAAI, AAAI Press. URL http://graphics.tudelft.nl/Publications-new/2013/LLB13a. ISBN 978-1-57735-635-6.

Chongyang Ma, Nicholas Vining, Sylvain Lefebvre, and Alla Sheffer. Game level layout from design specification. *Computer Graphics Forum*, 33(2): 95–104, 2014.

Ruben M. Smelik, Tim Tutenel, Rafael Bidarra, and Bedrich Benes. A survey on procedural modeling for virtual worlds. *Computer Graphics Forum*, 33 (6):31–50, 2014. URL http://graphics.tudelft.nl/Publications-new/2014/STBB14. doi: 10.1111/cgf.12276.

Julian Togelius, Noor Shaker, and Mark J. Nelson. Constructive generation methods for dungeons and levels. In Noor Shaker, Julian Togelius, and Mark J. Nelson, editors, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2015.

Valtchan Valtchanov and Joseph Alexander Brown. Evolving dungeon crawler levels with relative placement. In *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, C3S2E '12, pages 27–35, New York, NY, USA, 2012a. ACM. ISBN 978-1-4503-1084-0. doi: 10.1145/2347583.2347587. URL http://doi.acm.org/10.1145/2347583.2347587.

Valtchan Valtchanov and Joseph Alexander Brown. Evolving dungeon crawler levels with relative placement. In *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, C3S2E '12, pages 27–35, New York, NY, USA, 2012b. ACM. ISBN 978-1-4503-1084-0. doi: 10.1145/2347583.2347587. URL http://doi.acm.org/10.1145/2347583.2347587.

R. van der Linden, R. Lopes, and R. Bidarra. Procedural generation of dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(1):78–89, March 2014. ISSN 1943-068X. doi: 10.1109/TCIAIG.2013.2290371.

Příloha A

Game Design Document (GDD)

A.1 Executive Summary

A.1.1 High Concepts

The game is placed in a room labyrinth in 3D space. The goal of the game is to get through the labyrinth to find the portal. On the way through the labyrinth, the player has to find keys to open various doors that block his way to the portal. On the way to the portal, the player can try to pick up as many heaps of gold as possible to increase his score. He has to deside if it's better to try to find more gold or go to the portal already and get better finish time.

A.1.2 Game Description

This game is a type of labyrinth 3D game where the point is to find the way out of the labyrinth. In this case, that means getting to the portal. On the way through the labyrinth, the player should try to find as much gold as he can to get better score. But collecting the gold is not a requirement.The player can escape through the portal without picking up any gold. To make it more challenging for the player to find all the gold, heaps should be place at different places and should primarily not be placed on the main path player takes to get to the portal. Because of the time factor, the player has to choose when he should stop finding heaps of gold and enter the portal.

A.1.3 Key Features

- unlimited number of levels
- different difficulties
- score (time/(found gold +1))
- well formed control of player

- complicated levels

A.1.4 System requirements(Technology)

Game will be created in Unity 4.6 or newer, because of new UI tools.

A.2 Risks

- Too much time required for generating levels
- Competive products

A.3 Milestone Overview

A.3.1 Milestone 1

- game menus(main,difficulty and rating menu)
- game panel(restart,menu,quit)
- game rating request

A.3.2 Milestone 2

- game environment
- save and load enviroment

A.3.3 Milestone 3

- implementation of all procedural generation algorithms

A.4 Marketing

A.4.1 Target Audience

Game will be intended for players of 13 years of age and older that play games on the go and for short periods of time. The game will be suited for players that prefer gameplay over graphics.

A.4.2 Target Platform

Mobile with android operation system (API 9 and higher are supported by Unity)

A.4.3 Target Rating

The game will not contain any violence, blood, gambling or crude humor so the game will be proper for players 10 years old and older. Target ESRB rating will be E10+ rating.

A.4.4 Target Genre

A relaxing 1st person puzzle game.

A.5 Revenue Projection

A.5.1 Experiment

Game will be created in two versions. One for each experiment. The difference will be just in the experiment scene that will show up at the end of the game.

A.5.2 Future plans

Game will be created in two versions. First will be free with advertisement with 10 generated levels. And second for money (90 cents) with a lots of premade levels and when players reach the last level, new levels will be generated.

A.5.3 Competive products

- Labyrinth 3D by MARATGAMES. On appstore - It's a 1st person 3D labyrinth game and it's normal labyrinth only contains walls with no other options like doors and the controls are very difficult to use.
- Labyrinth 3D by Mobabu. On appstore, (uploaded on 8.2.2015) - it's a 3D labyrinth from third-person perspective and it's a basic labyrinth except with the option to jump and gain a quick peek at the rest of the labyrinth. Good graphics and different layouts for levels (walls and floor), but according to user reviews the player controls are also very unwieldy.

A.5.4 Legal analysis

- In the game, free objects from Unity store will be used. It has to be checked for which purpose they can be used before choosing to use them in the game.
- textures from `http://cgtextures.com/`
- generated textures from `http://bg.siteorigin.com/`

A.6 Game World

A.6.1 Game space

A.6.1.1 Overview

The labyrinth will be created from rooms and hallways it will not be a typical labyrinth created only from walls. Each room will be interconnected by hallways or only by a door if they are close. There don't has to be a door between each room. Some of doors needs to be open by key. Keys can have different colors and they are associated only with one door (not the same color as key door will be only brown). If player clicks on the door and door not opens. He has to start look for a key to them. Player can see only the room he is actually in. In some rooms will be placed heap of gold. Heaps should be placed in ways going different than the main path to the portal and in the dead-end alleys, or at least some of them should be. Heaps can be of two time small and big and can have different amount of golds associated with them (when player picks gold amount of gold is given to his total picked up gold). The player and portal position will be chosen randomly and the player position has to be a long distance from the portal.

A.6.1.2 Structure

Rooms are of rectangular shape and can have at most four doorways (one on each side).

Between two rooms can be place only one door maximally. Below are showed all options for placing doors between two rooms. If hallways are connecting more than two rooms each room can possibly have a door in doorway.Hallways are always straight and rectangular. Hallways can be connected together to create more complicated hallways and can be z-type (below is picture of possible z-type hallway). Hallways and Rooms can contain any decorations. Active objects as heap of golds, keys, portal (only one) can be placed only in rooms.

A.6.1.2.1 Example of possible door and hallway structure

A.6.1.3 Decoration

Rooms will have five types of chandeliers randomly placed in each room (one in room). There will be option to add torches to room to use instead of chandeliers, but torches will be mostly used for hallways.

A.6.1.4 Physics

The game will have only one floor, and the player is not able to jump, so only two things to be careful of are when the player looks up and walks he should

not be able to fly and he can't go through walls and doors or fall through floor.

A.6.1.5 Models and Textures

A.6.1.5.1 Models

- portal
- chandeliers (5 types)
- heap of gold
- door (only one used in different prefabs)
- keys(only one used in different prefabs)

A.6.1.5.2 Textures

- wall
- floor
- heap of gold (on display to show how many already picked up/how many are in the map)
- door
- portal
- ceiling

A.6.2 Levels

The game will have 3 different difficulties (easy, medium and hard). Levels will have option to generate level of each difficulty.

A.6.2.1 Easy

95 percent of players should be able to pick up all heaps and finish level.

A.6.2.2 Medium

65 percent of players should be able to pick up all heaps and finish level.

A.6.2.3 Hard

40 percent of players should be able to pick up all heaps and finish level.

A.7 Camera

The game will be played from a first-person point of view.

A.8 Game Engine

Game will use Unity 4.6 engine or newer version which has new UI tools.

A.9 Game interface

A.9.1 Menu layout/design

A.9.1.1 Game Panel

When player opens the start panel(android menu). The game will be paused and in the middle of the screen will be shown message “Game is paused!” after player clicks on the button for second time, the game will be resumed.

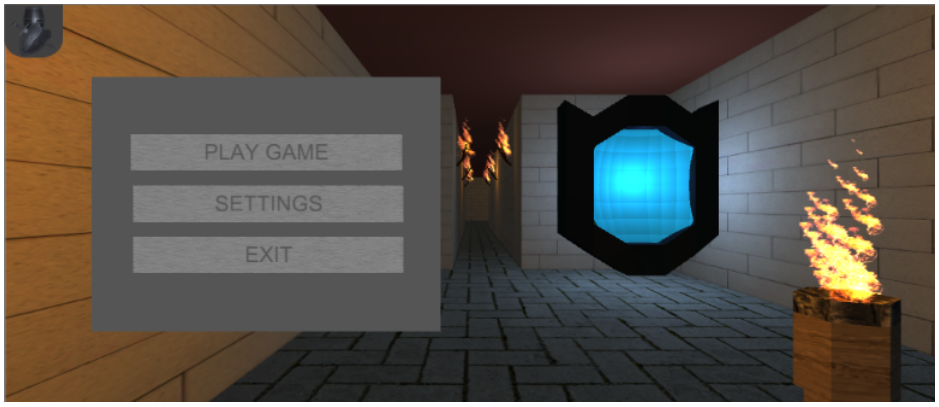
Obrázek A.1: Herní scéna

A.9.1.2 Main menu - Experiment 1

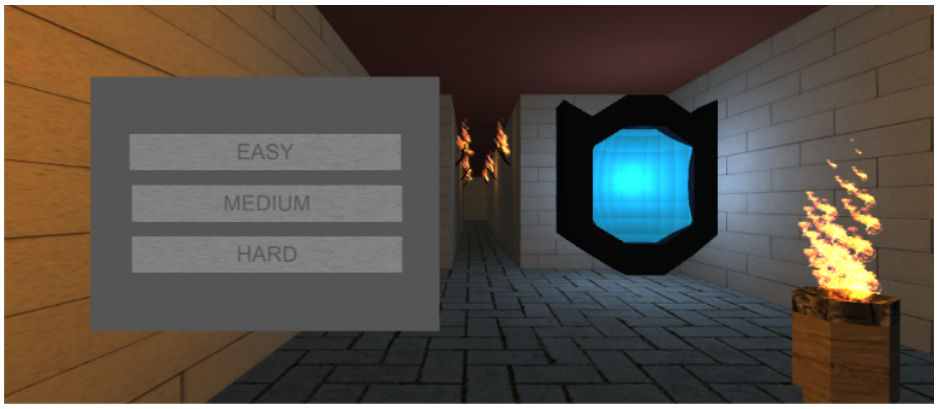
Game will have 3 methods for generating levels but if the player simply chooses to hit the first button (“PLAY GAME”) it will randomly choose from among them.

A.9.1.3 Difficulty menu

After picking the type of generation, "Difficulty menu"will show up with 3 different difficulties to pick from.



Obrázek A.2: Hlavní menu



Obrázek A.3: Menu obtížnosti

A.9.1.4 Send Rating form - Experiment 1

Form will send id of experiment to the server and all information about the last played level and player score.

A.9.1.5 Send Rating form - Experiment 2

Form will send all characteristics that will be known after first experiment.

A.9.1.6 Settings form - Default

Setting allow player to set speed of rotation. Turn on/off particle system for torches and possibility to avoid using map. And full setting of the map possibilities.

Obrázek A.4: Odeslání prvního experimentu

Obrázek A.5: Odeslání druhého experimentu

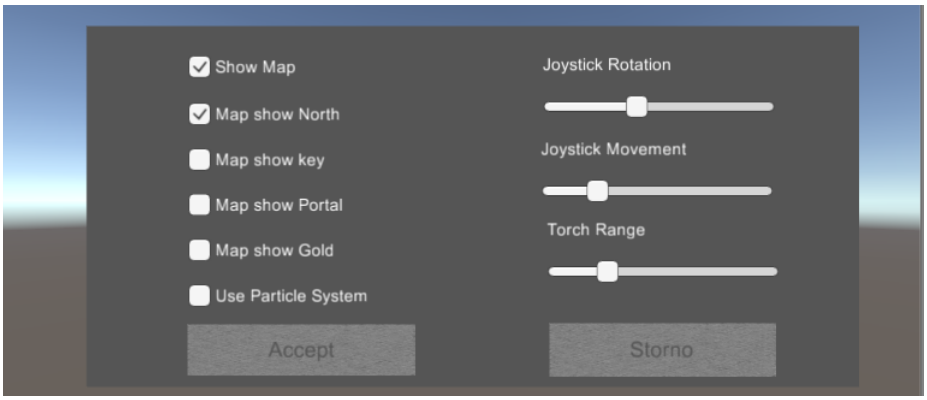
A.9.1.7 Settings form - used for experiments

Setting allow player to set speed of rotation. Turn on/off particle system for torches and possibility to avoid using map.

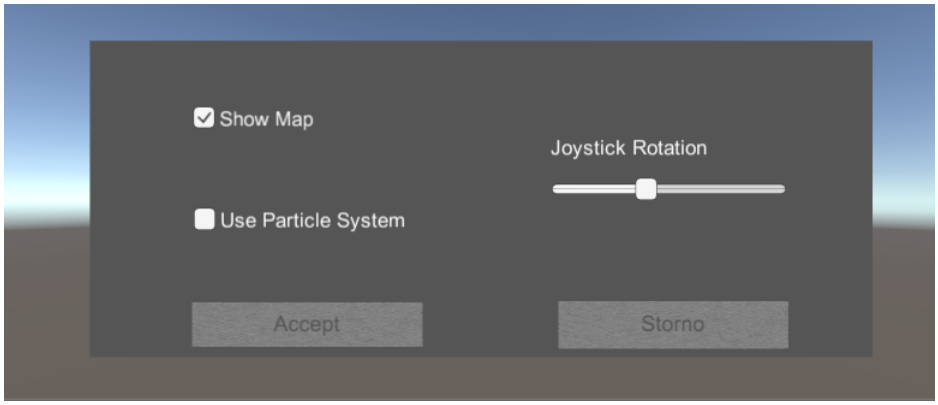
A.9.2 Control and Game possibilities

The game will have 2 joysticks. The first will be placed on the left side and will control player movement. The one on the right side will allow player to look around. The control of view will let him look anywhere except himself. It will be possible for player to set his joystick rotation and movement in settings menu.

Player interact with objects (heap of golds and doors) by clicking on them. And he finish levels by colliding with portal. With Other objects (decorations) player can 't do other things than collide.



Obrázek A.6: Úplné nastavení



Obrázek A.7: Částečné nastavení pro experiment

Příloha **B**

Přiložené CD

- `readme.txt` ..... soubor s popisem obsahu CD
- `ExperimentLabyrinth3D.apk` ..... spustitelný apk soubor
- `experiment data` ..... adresář dat exprimentů
- `src` ..... adresář zdrojových kódů
  - `zdrojové kódy` ..... implementační zdroje
  - `thesis` ..... adresář LaTeX obsahující kód práce
- `text` ..... adresář textu práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS