Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Procedurálne generovanie mapy pre RPG hry

Bakalárska práca

2015
Jakub Pospíchal

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Procedurálne generovanie mapy pre RPG hry

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	RNDr. Jozef Šiška

Bratislava 2015
Jakub Pospíchal

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Jakub Pospíchal
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Procedurálne generovanie mapy pre RPG hry *Procedural generation of maps for RPG Games*

Cieľ: Vytvoriť a implementovať systém pre procedurálne generovanie máp pre RPG hry. Systém by mal umožňovať jednoduché generovanie náhodných máp podľa požiadaviek používateľa (PJ/DM).

Anotácia: Tvorba máp pre RPG hry je často jednou z časovo najnáročnejších činností. Rôzne kampane a rôzne situácie vyžadujú rôzne štylizované a štruktúrované mapy, ktoré pánovi jaskyne umožnia zmysluplne do nich umiestniť všetky potrebné prvky. Procedurálne generovanie umožňuje jednoduchú ale opakovateľnú tvorbu náhodných máp, ale cielená tvorba mapy, ktorá by spĺňala rôzne netriviálne požiadavky, vyžaduje zložitejšie techniky.

Kľúčové slová: Procedurálne generovaný obsah, RPG hry

Vedúci: RNDr. Jozef Šiška, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, PhD.

Dátum zadania: 28.10.2013

Dátum schválenia: 28.10.2013 doc. RNDr. Daniel Olejár, PhD.

garant študijného programu

.....

študent

.....

vedúci práce

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Poďakovanie

Chcel by som poďakovať predovšetkým môjmu školiteľovi RNDr. Jozefovi Šiškovi za usmernenie pri tvorbe tejto práce a za rady a odbornú pomoc pri riešení danej problematiky.

Abstrakt

Procedurálne generovanie je rozšírenou technikou využívanou predovšetkým v počítačovej grafike. Táto technika uľahčuje prácu dizajnérom a zvyšuje rôznorodosť generovaného obsahu. V tejto práci sme si ukázali rôzne postupy, akými táto technika vytvára grafické objekty a aplikovali sme ich na problematiku tvorby máp do RPG hier. Vytvorili sme aplikáciu umožňujúcu generovať mapy na základe netriviálnych požiadaviek zadaných užívateľom. Ukázali sme, ako vhodne navrhnúť štruktúru generovania. Implementovali sme algoritmus, ktorý pomocou generátora pseudonáhodných čísel generuje jaskyňu s viacerými miestnosťami, ktoré obsahujú rôzne predmety. Nakoniec si užívateľ môže takto vygenerovanú mapu ručne prispôsobiť. Týmto sme značne ušetrili časovú náročnosť tvorby mapy do RPG hry.

Kľúčové slová: Procedurálne generovaný obsah, RPG hry

Abstract

Procedural generation is a common technique widely used in computer graphics for its simplicity and the ability to increase diversity of the generated content. In this thesis I present to you various processes of Procedural Generation of creating graphical objects which I then applied to the issue of map creating in RPG games. I created an application that allows generating maps based on User´s nontrivial requirements. Then I explained how to effectively outline the structure of generating. I implemented an algorithm which by using pseudo-random numbers generates a cave with multiple rooms each with several objects in it. This kind of map can eventually be adjusted manually by the User. This considerably speeds up the map creating process in an RPG game.

Key words: Procedural generated content, RPG games

Obsah

Úvod 1

Dungeons and dragons 2

1.1 Princíp hry 2

1.2 Počítačová verzia 2

Procedurálne generovanie 4

2.1 Generovanie v počítačových hrách 5

2.2 Dynamické generovanie sveta 6

2.3 Náhodnosť 6

2.4 Mersenne Twister 7

2.5 Procedurálne generovanie jaskýň 8

Tvorba Softvéru 11

3.1 Špecifikácia 12

3.2 Návrh riešenia 15

3.3 Návrh programu 17

3.3.1 Trieda MapPanel 17

3.3.2 Trieda ProcGen 19

3.3.3 Trieda Generátor 20

3.4 Implementácia 20

3.4.1 Trieda MapPanel 21

3.4.2 Trieda Generator 24

3.4.3 Trieda ProcGen 25

3.5 Validácia 29

3.5.1 Možné vylepšenia 29

Záver 30

Použitá literatúra.....31

Prílohy.....32

Zoznam obrázkov

Obrázok 1 Perlinov dvojrozmerný šum 10
Obrázok 2 Jaskyňa vyrobená celulárnym automatom 10
Obrázok 3 Rozloženie miestností 26
Obrázok 4 Pospájanie chodbami 27
Obrázok 5 Rozmiestnenie predmetov 28
Obrázok 6 Jaskyňa po záverečných úpravách 28

Úvod

Tvorba máp do RPG hier je často jednou z časovo najnáročnejších činností. Rôzne situácie a rôzne kampane si vyžadujú rôzne mapy, ktoré budú spĺňať netriviálne požiadavky. Procedurálne generovanie máp umožní náhodné generovanie týchto máp. V tejto práci sa budeme venovať generovaniu máp do hry s názvom "Dungeons and dragons".

Vytvoriť náhodný generátor máp dnes už nie je nič zložitého. Avšak ak chceme, aby naša aplikácia generovala obsah, ktorý spĺňa určité netriviálne kritéria, dostávame sa pred problém, ktorý je oveľa zložitejší. Existuje viacero metód a postupov ako docieliť stav, ktorý bude spĺňať naše kritéria a niektoré z nich využijeme.

V tejto práci sa budeme venovať predovšetkým vhodným návrhom kritérií a parametrov máp, ktoré si môže užívateľ zadefinovať a spôsobom ako ich realizovať. Kritéria nemusia byť konkrétne. Môžu len vyjadrovať určité vlastnosti mapy. To zjednodušuje užívateľovi zadávanie parametrov v prípade, že nemá konkrétne podmienky, ako má vyzerať mapa, ale má len hrubú predstavu.

Ďalšia téma, ktorej sa budeme v tejto práci venovať, je problematika tvorby miestnosti a jaskýň. V tejto časti sa budeme snažiť čo najefektívnejšie vytvárať miestnosti podľa požiadaviek. Pre jednoduchosť užívateľského prostredia, ktoré ponúka náš generátor je možné pri vytváraní jaskýň jednoducho vymenovať počet a typy miestností. V prípade, že užívateľ má záujem konkrétnejšie definovať miestnosti, môže jednotlivé parametre miestností meniť, od veľkostí až po jednotlivé predmety, ktoré sa v nej nachádzajú.

V poslednej časti sa budeme venovať implementácií metód a algoritmov, ktoré náš generátor využíva. Vytvárať budeme 2D mapy, ktoré budú vznikať pomocou procedurálneho generovania a budú spĺňať požiadavky užívateľa.

Kapitola 1.

Dungeons and dragons

1.1 Princíp hry

D&D je spoločenská hra pre viacero hráčov z kategórie „fantasy“. Hra neobsahuje žiaden plán ani mapu, všetko si pripravuje jeden z hráčov nazývaný “Pán jaskyne”, v skratke PJ. PJ je tvorcom hry a vymýšľa dobrodružstvá a k nim prislúchajúce mapy, ktorými následne ostatní hráči prechádzajú. Keďže táto hra neobsahuje žiaden herný plán, tak sa hrá iba na úrovní ústneho podania. PJ a ostatní hráči medzi sebou komunikujú a všetky akcie čo chcú, aby ich postava v hre urobila, rozprávajú PJ-ovi a ten ich akcie vyhodnocuje a výsledok naspäť povie hráčom. Hra obsahuje mnohé mechanizmy, ktorými sa riešia rôzne situácie ako napríklad boj. Tieto mechanizmy závisia predovšetkým od zdatnosti postáv a hodu kockami. Ostatní hráči si na začiatku vyberú svoju postavu, tým sa myslí rasa, povolanie a podobne, a iba pomocou komunikácie s PJ riešia rôzne úlohy a dobrodružstvá, ktoré si pripravil PJ. Avšak pri situácií ako je boj, je treba, aby si PJ nakreslil mapu, na ktorej sa boj bude odohrávať. PJ si musí mapu dobre premyslieť a navrhnúť podľa požiadaviek zvoleného dobrodružstva. Nakreslením mapy uľahčí sám sebe prácu a pomôže hráčom lepšie si predstaviť situáciu, ktorú si pre nich pripravil. Príprava a hlavne vyhodnocovanie niektorých mechanizmov hry môže byť zdĺhavá a značne tým spomaľuje celkový priebeh hry. Pravidlá tejto hry majú množstvo rôznych foriem, ktoré sa neustále vylepšujú. Delia sa do rôznych kategórií, podľa skúseností hráčov a podľa zvolenej náročnosti.

1.2 Počítačová verzia

Hráči tejto spoločenskej hry musia mať v prvom rade dobre vyvinutú predstavivosť. Bez toho si len ťažko túto hru hráč užije. Výzor svojej postavy a postavy ostatných hráčov či okolité prostredie môže PJ opísať, ale aj v takom prípade to zostáva predovšetkým na hráčovu predstavivosť. Niektoré veci je však zbytočne držať v pamäti. Napríklad atribúty svojej postavy

alebo obsah inventára sa vždy napíšu na papier ktorý ma PJ pri sebe. V praxi to vyzerá tak, že po určitom čase má PJ okolo seba veľké množstvo papierov a keď potrebuje niečo rýchlo nájsť, tak mu to trvá dlhú dobu. Z tohto dôvodu sme vytvoril jednoduchú aplikáciu, ktorá umožňuje PJ všetky tieto informácie udržiavať v elektronickej podobe. PJ môže v tejto aplikácií vytvárať nové postavy, presne tak isto ako mu to určujú pravidlá, ale nemusí pritom nič počítať. Stačí, keď len zadá číslo, ktoré mu padlo na kocke. Tak isto si môže postavy manažovať, pridávať predmety do inventára, prípadne meniť ich vlastnosti. V prvom rade však aplikácia uľahčuje tvorcovi hry kreslenie máp a vyhodnocovanie bojov. Tvorca hry si môže nakresliť mapu, na ktorej budú postavy hráčov riešiť rôzne úlohy. Mapa je dvojrozmerná, zložená z potrebného množstva štvorcov, ktoré predstavujú jedno políčko, na ktorom môže byť postava. V prípade boja nemusí PJ všetky parametre postavy zadávať do vzorcov a následne vyhodnocovať výsledok boja, všetky tieto potrebné veci spraví aplikácia namiesto neho a on len napíše hodnoty, ktoré danému hráčovi padli na kocke. Tým sa urýchli a zjednoduší celý priebeh tejto situácie. Do tejto pripravenej aplikácie budeme taktiež implementovať všetky algoritmy a metódy procedurálneho generovania, ktorým sa budeme venovať v našej práci.

Kapitola 2.

Procedurálne generovanie

Procedurálne generovanie je dnes už široko používaný termín spájajúci sa predovšetkým s počítačovou grafikou a to najmä s modelovaním terénu. Procedurálne generovanie vytvára objekty pomocou algoritmov namiesto toho, aby ich modeloval dizajnér ručne a v prípade potreby ich iba upraví. Termín "procedurálne" odkazuje na proces, výpočet, ktorý vykonáva určitá funkcia, procedúra. Táto procedúra môže generovať nielen povrch nášho terénu, ale aj textúry či tvary jednotlivých objektov. Procedurálne generovanie sa môže využiť nielen na modelovanie, ale aj napríklad na vytváranie hudby. Táto technika sa v neposlednom rade využíva aj vo filme. Program s názvom MASSIVE využíva tieto techniky spolu s umelou inteligenciou na tvorbu masívneho počtu ľudí. Bol vytvorený pre film Pán prsteňov na tvorbu veľkej masy bojujúcich vojakov.

Procedurálne generovanie a jeho techniky sú známe mnoho rokov, pričom niektorí vývojári ich využívajú vo väčšom množstve. Vývojári spoločnosti Bethesda Game Softworks využívajú procedurálne generovanie mapy pre svoju RPG sériu hier s názvom The Elder Scrolls. Táto populárna séria vznikla v roku 1994 a procedurálne generovanie sa využíva od roku 1996 v titule s názvom The Elder Scrolls II : Daggerfall. Séria týchto hier má charakteristický znak akým je veľkosť a otvorenosť tohoto sveta, čiže mapy, po ktorých sa hráč pohybuje. Vývojári už od roku 1996 procedurálne generujú tieto mapy, čím prispeli k určitému zviditeľneniu tejto techniky a jej metód. Následne mnoho spoločností využili výhody procedurálneho generovania, ako napríklad Avalanche Studios alebo Hello Games na generovanie rozsiahleho územia. V uplynulých rokoch značne narástol záujem o procedurálne vygenerovaný obsah a organizuje sa mnoho konferencií, kde sa odborníci týmito problémami zaoberajú.

V dnešnej dobe tieto techniky nabrali nový rozmer a to najmä tým, že vývojári pomocou nových a zložitejších metód generujú napríklad príbeh alebo úlohy do hier.

2.1 Generovanie v počítačových hrách

Prvé počítačové hry boli ešte veľmi obmedzované možnosťami hardware-u. Pamäťové obmedzenia sa v mnohých prípadoch stali veľkým problémom, a preto vývojári hier, ktoré obsahovali veľké mapy, museli nájsť efektívnejšie riešenie. Pamäte nedosahovali dostatočné kapacity, aby sa na nich mohlo držať väčšie množstvo dát, ktoré obsahovali mapy. To donútilo vývojárov nájsť nový spôsob uchovávania máp, čím zmenšili požiadavky na pamäť aj za cenu zvýšeného zaťaženia procesoru. Mapy tak značne zredukovali svoje veľkosti.

Dnes hry obsahujú oveľa väčšie množstvo údajov, ktoré sa týkajú objektov ako algoritmických mechanizmov. Vo veľkých otvorených svetoch ako je napríklad Grand Theft Auto, kde je svet jedno veľké mesto spolu s okolím, je každá jedna budova vymodelovaná ručne. Od tvaru, veľkosti až po textúry je navrhnutá a vymodelovaná dizajnérmi. S narastajúcimi možnosťami počítačov však narastali aj očakávania trhu. Vyžadujú sa čoraz detailnejšie prostredia a objekty, ktoré vytvárať každý zvlášť by bolo neprijateľné, hlavne z hľadiska časovej náročnosti.

Nároky hráčov sú čoraz vyššie nielen na detailnosť textúr či tvarov objektov, ale aj na unikátnosti jednotlivých objektov. Kedysi, ak chceli tvorcovia hier spraviť väčšie mesto, tak napríklad stačilo vytvoriť pár typov áut, ktoré sa často opakovali. Dnes už trh očakáva istú úroveň autentickosti a tú je možné dosiahnuť, iba ak náš svet bude dostatočne rôznorodý. Táto požiadavka na rôznorodosť opäť mnohonásobne zvyšuje nároky na dizajnérov.

Tieto a podobné faktory donútili tvorcov, aby prenechali väčšie množstvo práce na počítač. Zo začiatku sa iba prenášali povinnosti z dizajnérov na programátorov tým, že sa vytvárali algoritmy na tvorbu týchto objektov. Tento prístup má nevýhodu, lebo vytvoriť algoritmus na tvorbu dostatočne rôznorodých objektov je taktiež dosť náročné. Riešenia niektorých problémov si ukážeme a aplikujeme na náš svet v nasledujúcich kapitolách.

2.2 Dynamické generovanie sveta

Dynamické generovanie je jednou s prvých techník na generovanie sveta za behu programu. Táto technika značne odľahčuje pamäť, lebo mapy sa neuskladňujú v pamäti, ale generujú sa postupne pomocou procedurálnych mechanizmov. Hra s názvom Elite (1984), ktorá zo začiatku mala obsahovať svet s neuveriteľnými 2^{48} rôznymi galaxiami a každá z nich má 256 solárnych systémov, patrí medzi najznámejšie príklady otvoreného sveta. Uchovávať také množstvo dát o svete v pamäti by bolo naozaj neprijateľné, tak sa tvorcovia rozhodli pre túto techniku.

Metóda dynamického generovania je založená na využití jedného čísla s názvom "seed", ktorý po celú dobu generovania zostáva konštantou. Seed dávame na vstup rôznym algoritmom, ktorý z neho potom generujú svet. Mapa nikdy nezostáva v pamäti a generuje sa vždy vzhľadom na pozíciu hráča. Nezapisuje sa ani na disk a časti mapy, ktoré sú už nepotrebné, zahadzujeme.

Táto metóda má aj svoje nevýhody. Objekty, ktoré sú dlhé, napríklad cesty alebo rieky, sú veľmi ťažko modelovateľné touto technikou. Problém je v tom, že musíme vedieť informácie o blízkom okolí. Napríklad, keby sme chceli robiť rieku, tak musíme vedieť výškové rozdiely okolia, keďže rieka musí tiecť od vyššie položeného miesta na nižšie. Ale my nikde neuchovávame tieto informácie. Potrebovali by sme tak opäť nechať náš generátor vypočítať pomocou seed-u výšku okolia, ale tieto procesy značne zaťažujú procesor. Preto sa tieto objekty nerobia touto technikou.

Ďalšia väčšia nevýhoda tejto metódy sa ukáže, ak sa rozhodneme meniť iba určité detaily v našom svete. Každý jeden objekt je tvorený jednotným algoritmom. Ak by sme spravili zmenu na algoritme, odrazí sa to na celom svete.

2.3 Náhodnosť

Prvok náhody je nevyhnutnou podmienkou, ktorú musí spĺňať generovaný obsah, aby bol považovaný za procedurálny. Napriek tomu, že je nutnou podmienkou, nie je postačujúcou na to, aby sa obsah dal považovať za procedurálne generovaný. Napríklad hra tetris obsahuje prvok náhody, nie je však považovaná za procedurálne generovanú. Na generovanie náhodných čísiel existujú dve základné metódy. Prvá, fyzikálna generuje čísla na základe

fyzikálnych javov. Generátory využívajúce túto metódu nazývame „Pravé generátory náhodných čísel“. Druhá metóda, takzvaná výpočtová, generuje čísla pomocou rôznych algoritmov. Postupy, kde používame algoritmy, ktoré deterministicky generujú čísla, vytvárajú v skutočnosti iba pseudonáhodné čísla. Avšak metódy generujúce čísla pomocou algoritmov sú pre bežné účely, ako sú napríklad počítačové hry, dostatočne náhodné. Aj pri generovaní pseudonáhodných čísel môžeme postupovať viacerými spôsobmi - pomocou generátorov alebo hash funkciou. Generátor vyrába postupnosti a každý n-tý prvok vypočíta pomocou predchádzajúceho prvku. Ak by sme chceli vybrať práve n-tý prvok musíme vypočítať všetkých n-1 predchádzajúcich prvkov. Hash funkcia na rozdiel od toho vracia čísla nezávisle od predchádzajúcich. Na základe vstupu vráti vždy rovnaké pseudonáhodné číslo. Nie vždy je však takéto riešenie vhodné práve pre procedurálne generovanie. Obe dve metódy je možné použiť na dynamické generovanie. Generátor nastavíme pomocou „random seed“, čísla, ktorý určí jeho počiatočný stav. Platí, že pre rovnaký seed dostaneme vždy rovnakú postupnosť. Keďže generátory produkujú konečné postupnosti, každý z nich má vlastnú periódu, ktorá označuje najdlhšiu neopakujúcu sa sekvenciu. Od kvalitného generátora sa samozrejme očakáva, aby čísla v postupnosti boli rovnomerne rozdelené.

2.4 Mersenne Twister

Mersenne Twister je jeden z najznámejších a najpoužívanejších pseudonáhodných generátorov, ktorý bol navrhnutý v roku 1997 pánmi Makoto Matsumotom a Takuji Nishimurom. Meno tohto generátora je odvodené od Mersenovho prvočísla, čo je prvočíslo, ktoré je možné zapísať v tvare $2^n - 1$. Zároveň toto prvočíslo určuje veľkosť periódy daného generátora. V našom prípade použijeme štandardnú verziu tohto generátora, MT19937, ktorý je založený na prvočísle $2^{19937} - 1$. Generátor nám bude produkovať 32-bitové celé čísla, ktoré budeme využívať na určenie všetkých možných parametrov našej mapy.

Výhody Mersenne Twister:

- Veľmi veľká perióda, až $2^{19937} - 1$.
- Generátor prešiel veľa testami na štatistickú náhodnosť.

Mersenne Twister má aj svoje nevýhody. Medzi tie patrí určite rýchlosť, ktorá už nespĺňa dnešné štandardy. Existujú aj rôzne iné varianty tohto generátora, ktoré zvyšujú rýchlosť až dvojnásobne, ale pre naše účely je štandardná verzia postačujúca. Ďalšou nevýhodou je aj zbytočné zaťažovanie cache pamäte.

Keďže tvorba takéhoto generátora je zložitá a z ďaleka presahujúca našu tému bakalárskej práce, nebudeme sa bližšie zameriavať na algoritmické detaily generátora a využijeme už predprogramovaný generátor. Nám stačí vedieť ako si ho inicializujeme a ako budeme zadávať potrebné požiadavky. Potrebujeme vedieť, ako mu zadať seed, ktorý nám nastaví generátor do počiatočného stavu a ako si od neho pýtať ďalšie pseudonáhodné čísla. Keďže rovnaký seed nám nastaví tento generátor vždy na rovnaký stav, tak po opätovnom spúšťaní nám bude dávať rovnaké postupnosti čísiel.

2.5 Procedurálne generovanie jaskýň

Pri procedurálnom generovaní terénu je potrebné si ako prvé určiť metodológiu, ktorou budeme postupovať. Existujú dve základné metodológie, teleologická a ontogenetická. Pri teleologickom prístupe sa vytvorí fyzikálny model prostredia a následne sa simulujú rôzne javy. Tento prístup napodobňuje procesy v prírode. Napríklad, ak chceme vytvoriť mapu sveta, rozdelíme si ju na niekoľko častí a tie budú reprezentovať tektonické dosky. Následne budeme simulovať ich pohyb, čo spôsobí, že budú do seba narážať a následne nám tak zvrásnia terén. Potom necháme simulovať dážď a ten na základe vrásnenia vytvorí moria a podobne.

Príklady teleologických algoritmov:

Genetický algoritmus. Vychádza z jednoduchej schémy :

- Na začiatku máme počiatočnú populáciu
- Opakujeme tri fázy:
    1. Vyhodnotíme zdravie skupiny
    2. Zabijeme slabých jedincov
    3. Modifikujeme jednotlivcov a nahradíme nimi predošlých.

Tento jav napodobňuje genetický vývoj. Opakovaním týchto troch fáz postupne zlepšuje životnosť skupiny. Je potrebné si dobre reprezentovať jedincov populácie tak, aby sme ich vedeli správne modifikovať. Napríklad ako reťazec, ktorému pri modifikácií odoberieme kúsok a nahradíme ho iným.

Algoritmus Dažďovej kvapky simuluje eróziu povrchu. Keď dažďová kvapka dopadne na povrch, bude stekať z vyššie položeného miesta na nižšie. Pri prvom dotyku s povrchom zníži jeho výšku a v bode, kde sa usadí naopak zvýši. Týmto procesom sa zmenšujú výškové odchýlky terénu.

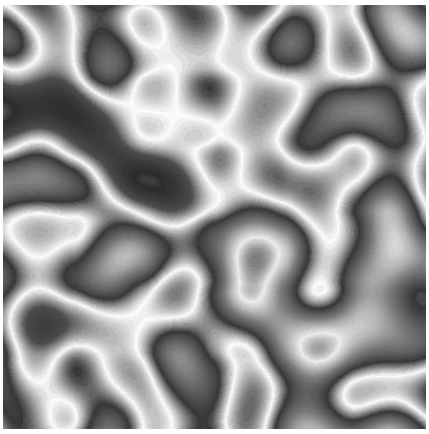
Rôzne modely ako napríklad „Reaction-Diffusion System“, „Fluid Dynamics“ alebo „Fire Propagation“ spadajú tiež pod teleologickú metodológiu. Simulujú rôzne fyzikálne javy pomocou viacerých zložitejších algoritmov.

Druhou metodológiou je ontogenetický prístup. Tento prístup nesimuluje javy, ale len napodobňuje ich výsledky. Namiesto dlhého procesu simulácie len odhadne výsledok, ktorý rôznymi algoritmami napodobní. Tento prístup je oveľa viac rozšírený a využívaný hlavne aplikáciami, akými sú napríklad počítačové hry.

Príklady ontogenetických algoritmov:

Perlinov šum, „Perlin noise“ je jedna z najznámejších a najvyužívanejších algoritmov. Využíva sa predovšetkým na vytváranie textúr materiálov ako je napríklad mramor alebo voda, ale je možné ho využiť aj na zvrásnenie terénu. Využíva generátor pseudonáhodných čísel, ktorý určuje hodnoty nadmorskej výšky terénu v určitom bode. Keďže generátor produkuje čísla príliš náhodne, vykonáva sa medzi nimi interpolácia, ktorá vyhladí tento šum. Výhodou tohto algoritmu je, že pomocou neho môžeme dostať terén akéhokoľvek typu, od

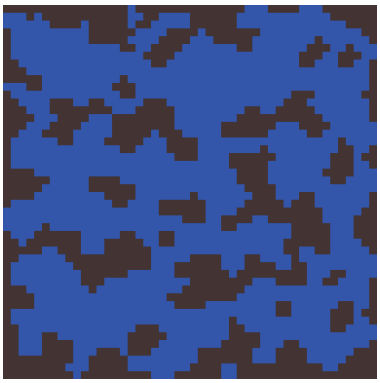
rovinatého až po hornatý. Príklad Perlinovho dvojrozmerného šumu vidíme na obrázku č. 1.



Obrázok 1 Perlinov dvojrozmerný šum

Presúvanie stredného bodu „midpoint displacement“ je metóda využívaná predovšetkým na určenie výšky bodu v teréne. Na začiatok si treba určiť dva okrajové body na mape. Algoritmus postupuje rekurzívne tak, že vezme dva susedné body, vypočíta ich priemer a k nemu pripočíta náhodné číslo a výsledok je výška daného bodu.

Celulárny automat „Cellular Automata“ je model reprezentujúci určitý dynamický systém, ktorý je reprezentovaný štvorcovou mriežkou buniek. Tento systém je diskrétny v priestore, čiže každá z týchto buniek reprezentuje jednu z hodnôt. Systém obsahuje pravidlá, ktoré určujú každej bunke, akú ma mať hodnotu v závislosti od susedných buniek. Je diskrétny aj v čase, čo znamená, že čas je rozdelený na kroky. V každom kroku sa bunka zmení podľa okolia tak, ako jej to určujú pravidlá. Má mnoho využití napríklad v matematike, fyzike či teoretickej biológií. Využitie má aj v generovaní jaskýň. Jaskyne generované týmto algoritmom pripomínajú klasické jaskynné systémy, tie čo poznáme z prírody. My sa však v tejto práci budeme zaoberať iným druhom jaskýň, v doslovnom preklade žalárom. Na obrázku č. 2 je znázornená jaskyňa vyrobená pomocou celulárneho automatu.



Obrázok 2 Jaskyňa vyrobená celulárnym automatom

Žalár v RPG hrách je jaskyňa, ktorá predstavuje komplex miestností pospájaných chodbami. Tieto miestnosti obsahujú rôzne predmety v závislostí od konkrétnej hry. My si na tomto príklade ukážeme základnú myšlienku procedurálne generovaných obsahov, ich výhody a metódy riešenia aplikované na jednoduché 2D mapy.

Existujú dva základné prístupy ako generovať takúto jaskyňu. Prvý prístup vygeneruje ako prvé miestnosti a tie spojí s chodbami. Druhý prístup generuje ako prvé rozsiahlu sieť chodieb a až potom vygeneruje miestnosti využívajúc pritom „Bludiskové algoritmy“. Prvý prístup je oveľa rozšírenejší a to predovšetkým kvôli svojej jednoduchosti. Nezaručuje však prepojenie každej miestnosti s chodbami. Z tohto dôvodu sa používajú rôzne algoritmy na kontrolu, či sú všetky miestnosti prepojené. Tvar miestností môže byť obyčajný obdĺžnik, alebo môže mať nepravidelný tvar generovaný rôznymi procedurálnymi technikami, prípadne vytvorený podľa nejakých vzorov, ktoré ručne vytvorili vývojári a miestnosti budú kopírovať tento vzor, ktorý potom náhodne otočíme a uložíme na mapu.

Kapitola 3.

Tvorba Softvéru

V tejto kapitole si navrhneme a následne vytvoríme náš vlastný pseudonáhodný generátor na vytváranie jaskýň, využívajúci metódy procedurálneho generovania. Tým, že využijeme tieto metódy, získame okrem iného obrovskú znovuhrateľnosť. Ak by sme robili mapy ručne, mali by sme pri opakovanom hraní tu istú. Avšak pri generovaní sa nám vždy vygeneruje iná. V takomto prípade si PJ iba nechá vygenerovať istý počet máp a potom z nich vyberá. Týmto sa mu uľahčí celková tvorba mapy a ušetrí čas. V praxi PJ ako prvé však vytvára dobrodružstvo a mapu potom prispôsobuje. Napríklad, vymyslí dobrodružstvo, v ktorom figuruje malá jaskyňa, v ktorej na konci nájde hráč truhlicu s pokladom. On vie, že nemá zmysel vytvárať rozsiahly komplex miestností. Z takéhoto dôvodu by mu veľmi uľahčilo prácu, keby vopred mohol zadať generátoru požiadavku na tvorbu mapy s malým počtom miestností. A tomu sa budeme v tejto práci venovať. Ukážeme si na príkladoch, aké požiadavky má zmysel

ponúkať, aký vplyv môžu mať na celkovú mapu a aký majú vplyv na ostatné požiadavky. Navrhneme si program, ktorý okrem bežného generovania bude dohliadať na tieto požiadavky a ukážeme si štruktúru postupného generovania našej jaskyne.

Vývojový proces aplikácie bude zložený zo štyroch fáz, zo špecifikácie, návrhu, implementácie a validácie. Začneme špecifikáciou, kde si určíme základné požiadavky, ktoré má náš program spĺňať. V návrhu si popíšeme základný algoritmus, ktorým budeme generovať jaskyne a vytvoríme si návrh tried. V implementačnej časti si ukážeme realizáciu niektorých metód a tried. V poslednej fáze si skontrolujeme, či aplikácia spĺňa všetky podmienky stanovené v špecifikácií.

3.1 Špecifikácia

Náš program bude uľahčovať prácu tvorcom hry Dungeons and Dragons pri tvorbe máp tým, že bude generovať mapy podľa ich požiadaviek. Mapy budú dvojrozmerné, štvorčekové, kde každý štvorček predstavuje jedno políčko, základnú bunku mapy. Keďže vytvárame jaskyne, stačí nám mať dva základné typy políčok, kamennú stenu a voľný priestor. Štvorček reprezentujúci voľný priestor bude môcť ešte obsahovať rôzne predmety alebo nábytok. Náš generátor musí v prvom rade vedieť náhodne vygenerovať mapu bez akýchkoľvek požiadaviek. V takom prípade si náhodne zvolí sám počet miestností, ich atribúty a rozmiestnenie. Ale aj v prípade, že sa predom nedefinovala žiadna požiadavka, musí mať užívateľ možnosť pomeniť všetky možné nastavenia a prispôsobiť túto náhodne vygenerovanú mapu svojím požiadavkám. V prípade, že užívateľ má premyslené niektoré vlastnosti mapy, môže ich definovať pred generovaním a zvyšok nechá na generátor a ten mu vygeneruje mapu s danými vlastnosťami. Hlavným prvkov, bude „seed“, celé číslo, ktoré bude reprezentovať mapu, resp. nastavenie počiatočného stavu generátora. Ostatné voľby, požiadavky užívateľa, budú len doplňujúce informácie pre tento generátor. Požiadavky budú rozdelené do rôznych kategórií na základe úrovne ich detailnosti. Tieto úrovne si teraz rozpíšeme.

- **Základné parametre mapy** budú určovať veľkosť a tvar mapy. Užívateľ zadá rozmery X-ovej a Y-ovej osi. Treba však obmedziť najmä minimálnu hodnotu

veľkosti oboch rozmerov mapy, aby bolo možné vôbec vygenerovať mapu s požadovanými vlastnosťami.

- **Pôdorys jaskyne** bude predovšetkým daný miestnosťami a to ich počtom, veľkosťou, rozmiestnením a spôsobom ich prepojenia pomocou chodieb. Rozmiestnenie a veľkosť miestností bude závisieť od generátora bez toho, aby sme mohli jednotlivé miestnosti posúvať alebo zväčšovať. Počet miestností je však atribút mapy, ktorý by užívateľ mal vedieť určiť, keďže môže mať predstavu, akú jaskyňu bude potrebovať. Zo skúseností tvorcovia si často predstavia, koľko rôznych miestností potrebujú, ale ich presné umiestnenie alebo veľkosť pre nich nezohráva dôležitú úlohu. Počet miestností môžu zadať dvoma spôsobmi. Môžu zadať presný počet miestností, ktorý chcú mať na mape, alebo si budú môcť vybrať zo štyroch intervalov. Intervaly budú vyjadrené slovne: „Few“, „Some“, „Many“, „A lot of“. Predstavujú intervaly <3,4>, <4,6>, <6,10>, <10,15>. Mnohokrát tvorcovia jaskyne postupujú podľa určitých vzorov. Napríklad chcú, aby v strede bola jedna veľká miestnosť a naokolo malé alebo, aby boli miestnosti pospájané v tvare bludiska a podobne. Preto program bude poskytovať tri základné vzory, typy jaskýň, podľa ktorých generátor rozmiestni jednotlivé miestnosti a určí ich veľkosť. Typ „Sun“ bude predstavovať jaskyňu, kde bude jedna hlavná miestnosť v strede s veľkými rozmermi a naokolo budú menšie. Typ „Labyrinth“ bude predstavovať niekoľko stredne veľkých miestností, ktoré budú náhodne pospájané. Posledný typ „Trough“ bude niekoľko malých miestností postupne pospájaných.

- **Typy miestností** bude môcť definovať užívateľ podľa svojich potrieb. Budú preddefinované tri základne typy, z ktorých si tvorca bude môcť vybrať. Miestnosť môže predstavovať jedáleň, spálňu alebo zbrojnicu. Každý typ miestností bude mať svoj špecifický nábytok a vybavenie, ktoré generátor rozloží po miestnosti. Ak sa užívateľovi nebude pozdávať rozloženie nábytku, tak môže opakovane generovať rozloženie nábytku po miestnosti bez toho, aby to malo vplyv na ostatné miestnosti. Keďže aplikácia slúži iba ako pomôcka pre tvorcu hry, netreba vytvárať veľké množstvo nábytku, ale len základné vybavenie. Samozrejme, množiny typov

nábytku nemusia byť disjunktné a môžu sa niektoré opakovať vo viacerých typov miestností. Budeme mať sedem základných predmetov, posteľ, stôl, stolička, ohnisko, skriňa, polica na zbrane a truhlica. Ohnisko bude v jedálni ale aj vo veľkých spálňach. Stôl a stolička bude v jedálni a v zbrojnici. Je treba dodržiavať, aby sa vždy generovali stôl a stolička v tesnej blízkosti. Posteľ bude iba v spálni. Skriňa sa bude vyskytovať vo všetkých troch miestnostiach. Polica na zbrane bude zase len v zbrojnici. Truhlica sa bude generovať v spálni v blízkosti postelí a v zbrojnici. Počet predmetov v miestnosti bude závisieť predovšetkým od veľkosti miestností. Predmety, ako napríklad posteľ, sa bude vyskytovať vo väčšom počte. Na rozdiel od toho skríň alebo truhlíc bude len zopár. Okrem počtu budú tak isto zvolené rozumné polohy týchto predmetov, pričom stále zachováme náhodnosť. To znamená, že skrine a police budú pri stene miestnosti, stoličky pri stoloch a truhlice vo väčšine prípadov pri posteliach. Ohnisko bude prevažne v strede miestností. I keď bude presne určené, že skriňa je vedľa steny, to kde presne v miestnosti bude, necháme na náhodu, to už zabezpečí generátor.

- **Záverečné úpravy** sa budú týkať predovšetkým premiestňovania jednotlivých kusov nábytku a predmetov, prípadne menenie jednotlivých štvorcov mapy. Tu budeme môcť zväčšovať alebo zmenšovať miestnosti a vytvárať alebo meniť chodby pomocou pridávania voľných štvorcov. Tvorca bude môcť odoberať, pridávať alebo presúvať predmety po miestnosti. Táto úroveň bude mať rovnaké možnosti tvorby mapy ako klasické kreslenie bez použitia procedurálneho generovania.

Ako sme už spomínali užívateľ bude mať možnosť určiť si všetky požiadavky ešte pred samotným generovaním. Bude si môcť určiť presný počet miestností, ich typy a vzor, aký ma jaskyňa spĺňať. Po vygenerovaní bude mať stále možnosť všetky svoje požiadavky pomeniť.

Ak si užívateľ zvolí mapu, ktorá rozmermi presahuje plochu, na ktorej sa mapa zobrazí, bude si môcť mapu po obrazovke presúvať. Na to použijeme klávesy „w“, „a“ „s“ „d“.

Po vytvorení mapy si bude možné ju uložiť dvojakým spôsobom. Prvý, klasický spôsob, bude ukladať mapu štvorček po štvorčeku. Druhý spôsob bude využívať procedurálne generovanie a uloží iba potrebné nastavenia generátora. Rovnako bude možné dvojako načítať

uloženú mapu. Pri ukladaní sa bude musieť užívateľ sám rozhodnúť, ako si svoju mapu uloží. Pre načítanie mapy bude možná len jedna voľba. Ak bude v úrovni pre generovanie, bude môcť otvoriť iba súbory uložené ako procedurálne generované. Ak bude v úrovni pre úpravu mapy, bude môcť užívateľ načítať iba mapu uloženú klasickým spôsobom.

3.2 Návrh riešenia

Predtým, ako sa pustíme do návrhu samotného programu, navrhneme si, ako generovať mapy pomocou procedurálneho generovania. Máme dva základné postupy ako generovať mapy pomocou generátora, aby spĺňali stanovené podmienky. Prvý spôsob je, že budeme generovať mapy náhodne a po vygenerovaní skontrolujeme, či sú všetky podmienky stanované užívateľom splnené. Výhoda tohto postupu je jednoduchosť implementácie. Je určite ľahšie kontrolovať výsledok generátora ako generovať s obmedzeniami. Nevýhodou je zaťaženie hardwaru. Je totiž možné, že kým by sa nám vygenerovala jaskyňa taká, aby spĺňala podmienky, muselo by sa predtým zbytočne vygenerovať veľmi veľa nepotrebných. Druhá možnosť je generovať tak, aby sme rovno spĺňali požiadavky. Tým značne odľahčíme hardware a generujeme podstatne rýchlejšie, ale na implementáciu je to náročnejšie ako predchádzajúca metóda. V našej práci budeme postupovať podľa druhej možnosti.

Aby sme mohli využiť výhody procedurálne generovaného obsahu je potrebné si generovanie mapy rozdeliť na rôzne úrovne. Generovanie po úrovniach má tú výhodu, že ak sa rozhodneme zmeniť na našej mape jeden parameter, ktorý bude na istej úrovni, nebude to mať vplyv na úroveň vyššie. My si ju rozdelíme na tri úrovne.

1. Pôdorys. Na tejto úrovni vygenerujeme prázdne miestnosti prepojené chodbami.
2. Miestnosti. Ak sa užívateľ rozhodne zmeniť typ miestnosti.
3. Úpravy. Ak už máme rozmiestnené predmety po miestnosti, ale niektoré chceme vlastnoručne popresúvať.

To znamená, že aplikácia bude generovať mapy stupňovito tak, že ako prvé vygeneruje pôdorys a až potom typy miestnosti a ich vybavenie. Akcie vykonané o úroveň vyššie budú

ovplyvňovať úrovne pod ňou. Ak sa užívateľ rozhodne meniť typ miestnosti nebude to mať vplyv na celkový pôdorys jaskyne. V opačnom prípade ak zmení typ miestnosti a následne sa rozhodne zmeniť celkový vzor jaskyne, tak sa všetky zmeny o úroveň nižšie stratia. Preto je dôležité, aby aj užívateľ dodržoval túto hierarchiu.

Nestačí však iba rozdelenie na tri úrovne. Ak budeme meniť typ jednej miestnosti je potrebné, aby sme ostatné zachovali. Preto si našu jaskyňu reprezentujeme ako graf strom. Náš strom má v koreni pôdorys našej jaskyne. Jeho potomkovia sú miestnosti, ktoré sú v pôdoryse resp. ich typy. Ku každej miestnosti prislúcha určitý zoznam predmetov, resp. štvorcov reprezentujúcich miestnosť. Ten tvorí potomkov danej miestnosti. Ak si budeme takto reprezentovať celú jaskyňu, zmeny na rovnakej úrovni nebudú mať vplyv na ostatných súrodencov. Zmena typu jednej miestnosti ovplyvní iba túto jednu miestnosť a invaliduje celý podstrom vrcholu, reprezentujúceho túto miestnosť.

V prípade, že užívateľ nebude definovať všetky potrebné atribúty mapy, náhodne ich doplníme. Na toto dopĺňanie využijeme obyčajnú random funkciu, nie náš generátor. Keď už budú všetky atribúty určené, vygenerujeme mapu nasledujúcim algoritmom.

1. Opakuj, pokým nebudeme mať správny počet miestností
    - Vygeneruje miestnosť, urči jej veľkosť
    - Umiestni ju na plochu
    - Ak sa pretína s inou miestnosťou, zahoď ju
2. Vyrob chodby medzi miestnosťami
3. Urči každej miestnosti typ
4. Priraď ku každej miestnosti zoznam predmetov, ktorý bude obsahovať
5. Každému predmetu v miestnosti urči pozíciu

Po vygenerovaní mapy budeme čakať na jednotlivé zmeny, ktoré bude chcieť vykonať užívateľ. V prípade, že zadá požiadavku na zmenu miestnosti, zmeníme danú miestnosť na typ, aký požaduje a zopakujeme pre ňu štvrtý a piaty krok.

3.3 Návrh programu

V tejto kapitole sa budeme venovať návrhu nášho programu. Programovať budeme v programovacom jazyku java. Navrhneme si každú jednu triedu, prípadne aj rôzne ďalšie metódy potrebné na splnenie požiadaviek na program spomenutých v špecifikácií.

Základom našej práce budú tri triedy:

1. MapPanel -bude reprezentovať mapu a poskytovať grafické rozhranie.
2. ProcGen -bude obsahovať všetky potrebné metódy na procedurálne generovanie.
3. Generator -bude reprezentovať samotný generátor pseudonáhodných čísel.

Keďže triedu na tvorbu jednoduchej mapy už máme, triedu na tvorbu procedurálne generovanej mapy a triedu obsahujúci generátor doprogramujeme do našej aplikácie a triedu na tvorbu máp pozmeníme, aby bol vhodný na naše účely. Triedy budú medzi sebou navzájom komunikovať a predávať si potrebné dáta. Všetky tri triedy si teraz rozpíšeme.

3.3.1 Trieda MapPanel

Trieda MapPanel vytvára grafické rozhranie na tvorbu mapy. Je potomok triedy JPanel. Na začiatok si uvedieme čo všetko už obsahuje a neskôr navrhneme jej rozšírenie o ďalšie potrebné triedy. MapPanel obsahuje tri triedy:

- DrawPanel
- SelectionPanel
- Square

Po grafickej časti MapPanel obsahuje dva objekty. Vľavo DrawPanel a napravo SelectionPanel. DrawPanel slúži na vykresľovanie mapy. SelectionPanel obsahuje možnosti textúr, ktoré môžu mať jednotlivé štvorčeky mapy. Jeden štvorček mapy je reprezentovaný triedou Square. Tieto základne fakty nám zatiaľ stačia pre návrh programu. Podrobnejšie sa k nim vrátime v implementačnej časti.

V našej práci budeme iba dopĺňať aplikáciu o nové funkcie, čo znamená, že všetky predošlé funkcie musia byť zachované. Z tohto dôvodu nemôžeme automaticky pri voľbe „Tvorba mapy“ nútiť užívateľa, aby vždy využíval generovanie máp. Preto ako prvé, čo sa

zobrazí pri tejto voľbe, je nové okno, ktoré dá užívateľovi možnosť si vybrať, či chce tvoriť mapu ručne, alebo pomocou nášho generátora. Toto jednoduché okno dá na výber dve možnosti „Normal mode“, predstavujúci tvorbu mapy pomocou klikania a menení jednotlivých štvorčekov, a „Generation“, čo bude náš spôsob generovania mapy. Po zvolení sa automaticky stratí a pokračuje sa podľa zvoleného módu. Po výbere módu sa otvorí nové okno, kde užívateľ zadá rozmery mapy. Následne sa vytvoria objekty tried DrawPanel, SelectionPanel a Square. Ak by bol zvolený mód na generovanie máp, vytvorí sa nová inštancia triedy ProcGen, ktorej pošleme referenciu na našu triedu MapPanel a tak isto jej pošleme mapu, ktorá je zatiaľ prázdna. Trieda DrawPanel zostane nezmenená. Jej funkcia je rovnaká pre oba módy. Triedy SelectionPanel a Square sa však musia prispôsobiť zvolenému módu. Vzhľadom na to, že tieto dva módy poskytujú navzájom úplne odlišné možnosti úpravy mapy, musíme tomu prispôsobiť aj obsah triedy SelectionPanel. Tá podľa zvolenej možnosti prispôsobí svoj vizuálny obsah.

Trieda MapPanel bude priamo komunikovať iba s triedou ProcGen. Po stlačení tlačidla na generovanie mapy, MapPanel predá všetky atribúty vytvorenej inštancií triedy ProcGen a tá vygeneruje mapu. Po jej vygenerovaní sa DrawPanel prekreslí a užívateľ rovno uvidí, čo generátor vyprodukoval. Od ProcGen si potom naša trieda vypýta zoznam miestností. Miestnostiam sa budeme bližšie venovať v návrhu triedy ProcGen. Takto získame zoznam miestností a všetky ich potrebné atribúty a sprístupní sa manažment miestností. Po jeho stlačení sa zobrazí nové okno a tam sa vypíšu všetky miestnosti, ich typy a možnosť ich pripadne zmeniť na iný typ. Je potrebné si všetky tieto zmeny zapísať a zapamätať.

Teraz, keď už máme mapu s takým pôdorysom a typom miestností akým sme chceli, môžeme pristúpiť k ďalšej, poslednej úrovni úpravy. Na to bude slúžiť posledné tlačidlo, ktoré prerobí SelectionPanel na taký istý panel ako v „Normal mode“, kde budú možnosti zmeny jednotlivých políčok. Okrem všetkých typov textúr tam budú aj možnosti, ako pridať nový predmet, odobrať alebo premiestniť predmet, ktorý tam vložil generátor.

Teraz už máme hotovú mapu a potrebujeme si ju uložiť. Na to, aby sme boli schopní správne uložiť mapu, musíme si uložiť presne všetky nastavenia generátora a následne zmeny v takom poradí, v akom sme ich robili. Ukladať mapu budeme do textového súboru postupne. Do prvého riadku si uložíme všetky potrebné nastavenia generátora. Do druhého riadku si zapíšeme všetky zmeny typov miestností presne v tom poradí, v akom sa udiali. Do tretieho riadku sa zapíšu zmeny, ktoré sa vykonali v poslednej fáze, kde sa menili jednotlivé políčka. Na tento účel si vytvoríme dve triedy. Trieda ChangesOpt bude predstavovať zmenu typu

miestnosti a trieda ChangesOfSquares zmenu jednotlivých políčok. Zmeny miestností ovplyvňujú generátor, čiže je potrebné si ich pamätať v poradí, v akom sa udiali.

Ako posledné ešte doprogramujeme nové okno, ktoré sa zobrazí užívateľovi a dá mu možnosť definovať všetky parametre mapy, ešte predtým, ako sa mu zobrazí hlavná plocha. Bude si tam môcť definovať počet miestností, ich typy a typ pôdorysu jaskyne.

3.3.2 Trieda ProcGen

Táto trieda bude reprezentovať samotné procedurálne generovanie. Od triedy MapPanel dostane všetky potrebné atribúty, ktoré má mapa spĺňať. Keď ich dostane, začne ju generovať. Ako prvé pri generovaní vytvorí inštanciu triedy Generator. Potom tento generátor nastaví pomocou získaného „seed-u“. Takto nastavený generátor nám teraz bude poskytovať postupnosť pseudonáhodných čísel. Všetko, čo trieda ProcGen vygeneruje, rovno zakreslí do mapy. Postupovať bude presne podľa algoritmu uvedeného vyššie. Vygenerujeme postupne miestnosť po miestnosti, pričom hodnoty ich veľkosti a umiestnenie si vypýtame od generátora. Musia však spĺňať kritéria, ktoré im stanovuje typ pôdorysu. Pri umiestňovaní na plochu kontrolujeme, či sa nepretínajú. Informácie o jednotlivých miestnostiach si musíme neustále držať v pamäti. Preto si vytvoríme triedu Room, ktorej inštancie budú predstavovať jednotlivé miestnosti. Miestnosť vytvárame zatiaľ bez akýchkoľvek predmetov. Následne sa vytvoria chodby medzi jednotlivými miestnosťami tak, že nájdeme bod medzi dvoma miestnosťami a z oboch miestností vedieme úsečku, buď vertikálnu alebo horizontálnu, k danému bodu, čím tieto dve miestnosti spojíme. Zmyslom procedurálneho generovania je odľahčiť pamäť, preto si nebudeme pamätať nič iné okrem miestností. Chodby vygenerujeme a rovno zakreslíme do mapy. Teraz už máme hotový celý pôdorys jaskyne a môžeme pristúpiť k ďalšej úrovni.

Na druhej úrovni meníme typy miestností a s tým súvisiace predmety v miestnostiach. Typ miestnosti bude ďalšou triedou, ktorú nazveme Type. Náhodne určíme každej miestnosti jej typ z množiny určenou v špecifikácií. Potom každej miestnosti pridelíme inštanciu triedy Type a tá bude reprezentovať jej typ. Ku každému typu miestností prislúcha iný nábytok, predmety. V triede Type sa podľa veľkosti miestnosti určí množina predmetov, ktoré sa priradia danej miestnosti. Teraz musíme predmety rozumne rozmiestniť po miestnosti. Podľa špecifikácie má každý predmet určité miesto, kde sa bude vyskytovať. To znamená, že napríklad skriňa bude vždy pri stene, ale presnú pozíciu určíme pomocou generátora.

Vypýtame si ďalšie čísla z postupnosti, ktoré určia pozície jednotlivých predmetov, pričom budú dodržiavať stanovené podmienky. Predmety si tak isto nebudeme pamätať. Rovno ich budeme zakresľovať do mapy. Po tomto kroku už máme hotovú mapu.

Trieda ProcGen nám teda vypracovala prvé dve úrovne. Keďže tretia úroveň sa venuje úprave jednotlivých políčok, nebudeme už nič generovať a tretiu úroveň prenecháme na MapPanel.

3.3.3 Trieda Generátor

Trieda Generátor bude reprezentovať pseudonáhodný generátor. Názov triedy môžeme počas implementácie zmeniť na názov konkrétneho generátora. Programovať túto triedu by bolo nad rámec našej práce, preto použijeme už existujúcu implementáciu. Táto trieda bude výhradne komunikovať iba s triedou ProcGen. Od triedy ProcGen dostaneme „seed“ a tým nastavíme generátor a pripravíme ho na používanie. Mnoho generátorov ponúka veľa rôznych funkcií a veľa možných výstupov, rôznych dátových typov, celých alebo reálnych čísel, prípadne čísla v určitých intervaloch. Nám bude stačiť jeden jediný celočíselný dátový typ a to int. V prípade potreby si aj tak môžeme matematicky odvodiť z neho napríklad číslo z určitého intervalu alebo podobne. Nášmu generátoru musíme teda vedieť zadať „seed“ a vyberať z danej postupnosti čísla. V jave je zabudovaný generátor pseudonáhodných čísel, ktorý ponúka okrem iného aj tie dve funkcie. Avšak môže sa stať, že na dvoch rôznych počítačoch by generátor s rovnakým „seed-om“ generoval inú postupnosť. Preto si implementujeme iný generátor a bližšie sa naň pozrieme v implementačnej časti, kde si popíšeme možnosti nami zvoleného generátora.

3.4 Implementácia

V tejto časti sa budeme venovať implementácií tried a ich metód obsiahnutých v návrhu nášho programu. Niektorým dôležitým metódam sa budeme venovať bližšie, aby sme ukázali konkrétny postup v riešení našej problematiky. Ostatné metódy, ktoré priamo nesúvisia s procedurálnym generovaním si iba načrtneme, prípadne opíšeme algoritmus.

Našim cieľom nie je vytvoriť samostatnú aplikáciu, ale doplniť už existujúcu. V takom prípade, keď modifikujeme aplikáciu, treba vedieť ako funguje, čo môžeme využiť a ako správne upraviť jej obsah. Preto si teraz stručne zhrnieme ako funguje.

Aplikácia D&D je naprogramovaná v jazyku java. Skladá sa z ôsmich tried:

1. DD – táto trieda sa vytvára ako prvá hneď pri spustení a inicializuje MainFrame
2. MainFrame – potomkom triedy JFrame a reprezentuje hlavné okno aplikácie. Toto okno je stále a mení sa len jeho obsah. Pri spustení sa ako obsah automaticky pridá trieda MainPanel.
3. MainPanel – potomkom triedy JPanel. Obsahuje úvodné okno a základný panel možností, z ktorých si môže užívateľ vybrať. Obsahuje šesť tlačidiel, kde prvých päť reprezentuje triedy uvedené nižšie a posledný je výstup z aplikácie. Pri voľbe jednej z tried automaticky okno zmaže svoj aktuálny obsah a pridá si novú, užívateľom zvolenú triedu.
4. MapPanel - potomkom triedy JPanel. Trieda MapPanel slúži na robenie máp. Keďže práve túto triedu budeme upravovať, bližšie si ju rozoberieme neskôr.
5. HManagment - potomkom triedy JPanel. Trieda obsahujúca všetky potrebné metódy na správu postáv.
6. Hedit - potomkom triedy JPanel. Slúži na tvorbu postáv.
7. Battle - potomkom triedy JPanel. Vyhodnocuje boje.
8. Creature - potomkom triedy JPanel. Slúži na tvorbu príšer.

Nás bude zaujímať najmä trieda MapPanel. Ostatné triedy fungujú ako samostatné celky, ktoré s našou prácou nebudú mať nič spoločné, preto sa nimi už nebudeme zaoberať.

3.4.1 Trieda MapPanel

Pri implementácií tejto triedy sme postupovali presne podľa vyššie uvedeného návrhu. Avšak pre čitateľa by to mohlo prísť trochu mätúce, tak v tejto triede si ako prvé rozpíšeme metódy, ktoré sme doprogramovali do tejto triedy. Potom si rozpíšeme zmeny v už existujúcich metódach, prípadne triedach, ktoré MapPanel obsahuje. To znamená, že nepopíšeme implementáciu chronologicky, ako to uvádza návrh a ako sa skutočne udiala. Konvencia zápisu bude v tvare „návratový typ“ „Názov metódy“ „(typ vstupného parametra)“

Doprogramované metódy:

- **void OptPanel()** - Táto metóda vytvorí okno, na ktorom sa zobrazia dve tlačidlá poskytujúce dve možnosti, „Normal mode“ a „Generation“. Po výbere si zapamätáme voľbu užívateľa. Nezávisle od výberu užívateľa sa následne zavolá metóda SizePanel().
- **void SizePanel()** - Vytvorí nové okno, na ktorom sa zobrazia dve textové polia, do ktorých si užívateľ zadá celkovú veľkosť mapy.
- **void ProcOptPanel()** - ak si užívateľ zvolil režim generovania zobrazí mu táto metóda okno, v ktorom budú všetky možnosti, parametre mapy, ktoré si môže zvoliť ešte pred samým generovaním. Následne sa zavolá metóda SetMapPanel.
- **void generate() –** Táto metóda spúšťa generovanie. Vytvorená inštancia triedy ProcGen od tejto metódy dostane všetky atribúty mapy, ktoré si užívateľ zvolil a spustí samotné generovanie.

Okrem metód sme pridali do triedy MapPanel aj ďalšie triedy.

- **ChangesOfSquares** – reprezentuje zmenu jedného štvorca. Túto triedu využívame pri ukladaní máp. Obsahuje súradnice štvorca, ktorý sa zmenil a jeho novú textúru. Zoznam týchto zmien si neustále pamätáme. Ak sa jeden a ten istý štvorec mení viackrát stačí nám si zapamätať poslednú zmenu. Pri výpise tejto zmeny do textového súboru používame formát „XXYYZZ“, kde XX a YY sú súradnice zmeneného štvorca a ZZ je jeho textúra.
- **ChangesOpt** – reprezentuje zmenu jednej miestnosti. Pamätá si identifikačné číslo miestnosti a typ, na aký sme ju menili. Tu si musíme pamätať všetky zmeny. Každá takáto zmena totiž ovplyvnila generátor a pri opätovnom vykresľovaní mapy je potrebné simulovať generovanie presne tak isto. Pri výpise tejto zmeny do textového súboru používame formát „XXY“, kde XX znamená identifikačné číslo miestnosti a Y typ, na ktorý bola zmenený.

Ukladanie a načítavanie vygenerovanej mapy nie je v princípe také jednoduché. V normálnom režime, keď si mapu vyrobíme klasickým kreslením štvorček po štvorčeku, si ju uložíme jednoducho zapísaním ich textúr v danom poradí. Nie je to priestorovo efektívne, avšak jednoduché na implementáciu. Uložiť vygenerovanú mapu, znamená uložiť nastavenia generátora a následne jednotlivé zmeny. Nastavenia generátora sme si zapamätali ešte predtým, ako sme vôbec generovali mapu. Využili sme už existujúci MenuBar, v ktorom už možnosť na uloženie mapy bola naprogramovaná. Pomocou metódy savePG() sme dorobili ďalšiu možnosť

do menu a to možnosť uložiť mapy vo formáte vhodný pre generátor. Tak, ako sme uviedli v návrhu, textový súbor, v ktorom je mapa uložená, sa skladá z troch riadkov. Do prvého sme uložili nastavenia generátora. Do druhého všetky inštancie triedy ChangesOpt a do tretieho všetky inštancie triedy ChangesOfSquares. Teraz sa bližšie pozrieme na triedy a metódy, ktoré sme museli pre náš účel modifikovať.

Trieda Square sa veľmi meniť nemusela. Inštancia tejto triedy si okrem svojej absolútnej polohy na mape pamätá aj relatívnu polohu voči obrazovke pomocou nového páru premenných. Keď už vieme meniť ich relatívne súradnice, ľahko sme doprogramovali posúvanie obrazu po mape. Následne stačilo upraviť metódy paintSquare, kde sme pridali nové textúry a metódou ChangeSquare sme zabezpečili premenu textúry daného štvorca. Túto funkciu voláme, keď prekresľujeme jednotlivé štvorce. Vždy, keď sme prekreslili štvorec, vytvorili sme novú inštanciu tried ChangesOfSquares. Pozrieme, či sme už tento istý štvorec menili, ak áno, nahradíme zmenu novšou, ak nie, tak pridáme do zoznamu zmien.

Trieda SelectionPanel, ktorá reprezentuje pravý panel so všetkými možnosťami, prešla najväčšími zmenami. Tá musí svoj obsah prispôsobovať zvolenému režimu. V režime úprav obsahuje JButton-y, ktoré reprezentujú jednotlivé textúry. Ak sa niektorý z nich stlačí, voľba sa zapamätá a pri kliknutí na štvorec sa postupne zavolajú metódy changeSquare a paintSquare. V režime na generovanie obsahuje tento panel nasledujúce komponenty:

- Textové pole na „seed“
- Počet miestností si môžeme presne definovať alebo si vybrať jednu zo slovných možností
- Výber z možností pôdorysu pre našu jaskyňu
- Tlačidlo, ktoré náhodne zvolí všetky predchádzajúce atribúty.
- Tlačidlo, ktoré sprístupní možnosť zmeniť jednotlivé miestnosti.
- Tlačidlo, ktoré vygeneruje jaskyňu podľa atribútov.
- Tlačidlo, ktoré ukončí mód generovanie a sprístupní úpravy jednotlivých polí.

SelectionPanel sa v tomto prípade stará, aby trieda ProcGen dostala všetky potrebné atribúty na tvorbu mapy. Keďže trieda Square je zdieľaná s ProcGen nedostávame od ProcGen žiadne spätné údaje. Na zobrazenie všetkých miestností, ich typov a s možnosťou ich zmeniť, využíva triedu RoomManager, ktorej inštanciu robí až keď je už mapa vygenerovaná.

Trieda RoomManager je potomok triedy JFrame. Vytvorí okno, v ktorom je vypísaný celý zoznam miestností danej mapy a ponúka ich zmenu na iný typ, alebo len zmeniť rozmiestnenie predmetov. Zoznam miestností je zoznam inštancií triedy Room. Pri každej zmene typu či zmene rozmiestnenia predmetov vytvorí novú inštanciu triedy ChangesOpt a zapíše ju do zoznamu takýchto zmien.

Metóda open má na starosti načítanie novej mapy. Ak sme v režime generovania, tak vieme správne otvoriť mapu vygenerovanú pomocou procedurálneho generovania a ak sme v režime úprav, tak mapu uloženú klasickým spôsobom. Načíta sa zvolený súbor a postupne sa simuluje generovanie mapy tak, ako sme to robili pri jej tvorbe. Ako prvé nastavíme generátor a spustíme ho. Potom načítame druhý riadok a vytvoríme tak nový zoznam inštancií triedy ChangesOpt a odsimulujeme ho pomocou generátora. Následne si vytvoríme nový zoznam inštancií triedy ChangesOfSquares, ktoré sme načítali z tretieho riadku. Potom prehodíme režim generovanie na režim úprav.

3.4.2 Trieda Generator

Skôr ako sa budeme venovať triede ProcGen, implementujeme si triedu Generator. Vzhľadom na to, že ťažisko našej práce spočíva v triede ProcGen, je potrebné všetky ostatné triedy a metódy, ktoré využíva, implementovať skôr ako ju, aby sme ich mohli hneď využívať a vidieť naše výsledky. Ako sme v návrhu spomenuli, neprogramovali sme túto triedu, ale využili sme už existujúcu. Zvolili sme si generátor Mersenne Twister, ktorý sme následne priložili k nášmu programu ako samostatnú triedu MersenneTwister. Nami zvolená implementácia v jazyku java, ktorú vytvoril pán Sean Luke, nám poskytuje všetky potrebné funkcie. Bližšie informácie sa môžete dozvedieť na jeho stránke http://cs.gmu.edu/~sean/research/ . Využívame len dve základné metódy:

- void setSeed(int) – nastaví generátor pomocou parametra.
- int getInt() – vráti číslo z postupnosti.

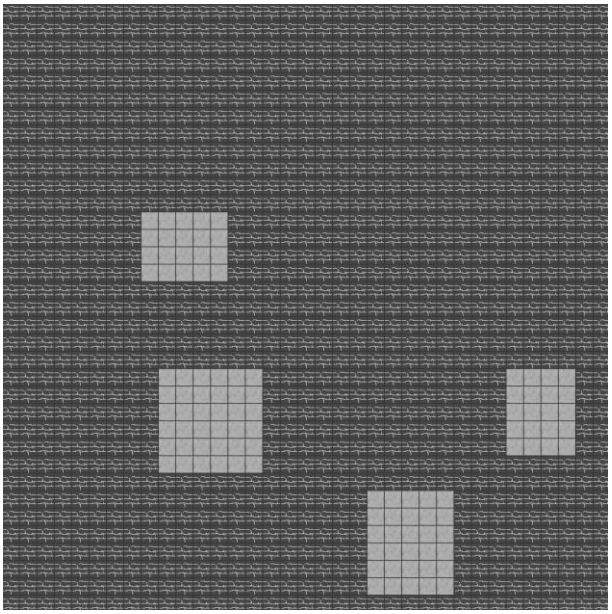
3.4.3 Trieda ProcGen

Trieda ProcGen, tak ako je spomenuté aj v návrhu, slúži na generovanie máp pomocou techník a metód procedurálneho generovania. Rozpíšeme si metódy a triedy chronologicky tak, ako sme ich implementovali. Konvencia zápisu zostáva taká istá ako v časti MapPanel.

- void setSquares() - uloží mapu.
- void setSeed(int) – uloží „seed“ a zároveň vytvorí novú inštanciu triedy MerseneTwister. To znamená, že vytvorí nový generátor a nastaví ho podľa hodnoty „seed“.
- void setCountOfRoomsExactly(int) – uloží užívateľom zvolený počet miestností.
- void setCountOfRooms(int) – keďže sme chceli, aby užívateľ nemusel definovať presný počet miestností, tak si môže vybrať jednu zo slovných možností predstavujúci rôzne intervaly. Podľa možnosti sa určí interval a pomocou generátora si náhodne vyberie číslo z intervalu. Intervaly sa pretínajú, preto pre niektoré „seed-y“ sa môže stať, že dve susedné možnosti dajú rovnaký počet miestností.
- void setPattern(int) – uloží užívateľom zvolený typ jaskyne.
- int getAbsInt() – vráti získanú absolútnu hodnotu z generátora .
- int getAbsInt(int) – vráti získanú absolútnu hodnotu z generátora v intervale od 0 po vstupný parameter – 1.
- int getInt(int) - vráti získanú hodnotu z generátora v intervale tak, ako predošlá metóda.
- int getInt() – vráti získanú hodnotu z generátora.
- void generate () – táto metóda je volaná z MapPanel vo chvíli, keď už bude všetko pripravené na generovanie mapy. Táto funkcia na základe zvoleného typu jaskyne zavolá metódy sun(), lab() alebo through().

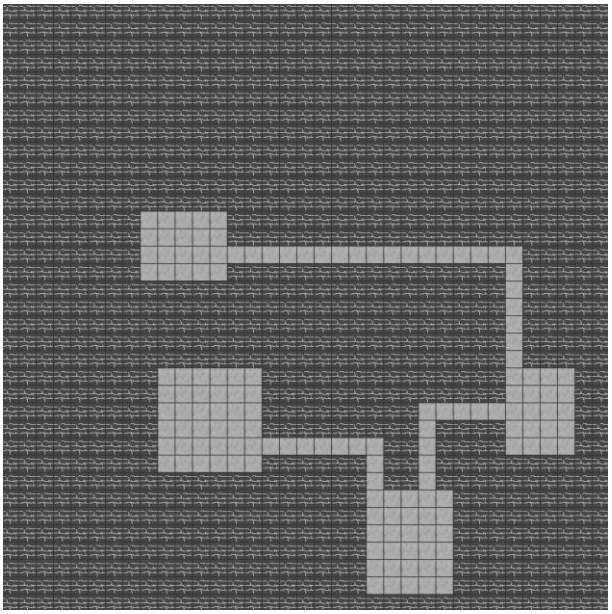
Metódy sun(), lab() a throug() sa líšia prevažne konštantami súvisiacimi s veľkosťou miestností a podobne. Preto k nim budeme teraz pristupovať ako k jednej metóde. Ako prvé sa v nich urobí potrebný počet inštancií triedy Room pomocou metódy makeRoom. Miestnosť na mape je presne daná ľavým horným štvorcom miestnosti a jej rozmermi. Metóda makeRoom vytvorí miestnosť pomocou generátora tak, že si od neho pýta čísla z postupnosti. Ak napríklad potrebuje definovať šírku miestnosti, predelí toto číslo maximálnou veľkosťou, ktorú môže mať miestnosť a zvyšok po delení je šírka našej miestnosti.

Pomocou metódy intersection() skontrolujeme, či sa miestnosti nepretínajú alebo nevyčnievajú z mapy. Keď poznáme ľavé horné body miestností a ich rozmery ľahko vieme skontrolovať, či sa dané dve miestnosti nepretínajú. Ak áno, jednu z nich zahodíme a vytvoríme novú. Po tomto kroku máme hotové miestnosti a môžeme si ich vykresliť. Výsledok doterajšieho postupu môžeme vidieť na obrázku č.3.



Obrázok 3 Rozloženie miestností

Teraz, keď už máme miestnosti, pospájame ich chodbami. Na to slúži metóda makePath. Tá slúži na pospájanie dvoch konkrétnych miestností. Pomocou generátora si určí bod medzi nimi. Potom od oboch miestností vedie úsečku k tomu bodu a tým ich spojí, ako môžeme vidieť na obrázku č.4.

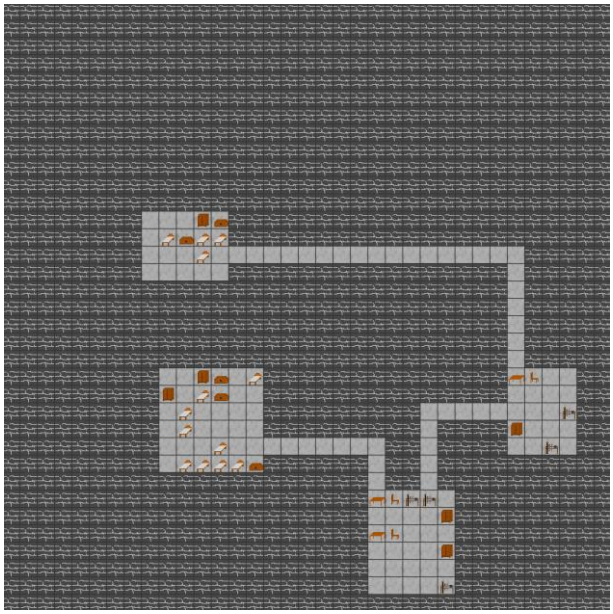


Obrázok 4 Pospájanie chodbami

Inštancie triedy Room, ako bolo viac krát predtým spomínané, reprezentujú miestnosti. Sú to jediné objekty z triedy ProcGen, ktoré si neustále držíme v pamäti. Nemôžme ho po vytvorení hneď zakresliť do mapy a zabudnúť. V takom prípade by sme nevedeli efektívne premieňať typy miestností. Preto si zoznam týchto inštancií držíme v pamäti až do chvíle, keď prepneme z režimu generovania do režimu úpravy jednotlivých políčok. Inštancia si okrem svojej presnej pozície na mape pamätá aj svoj typ, ktorý je reprezentovaný inštanciou tried Type. Má tri základné metódy:

- void clearRoom() – nastaví všetky políčka miestnosti na prázdne políčka.
- void roomSave() – uloží samú seba do mapy.
- void setType() – priradí k sebe jednu inštanciu triedy Type. Tým si určí svoj typ.

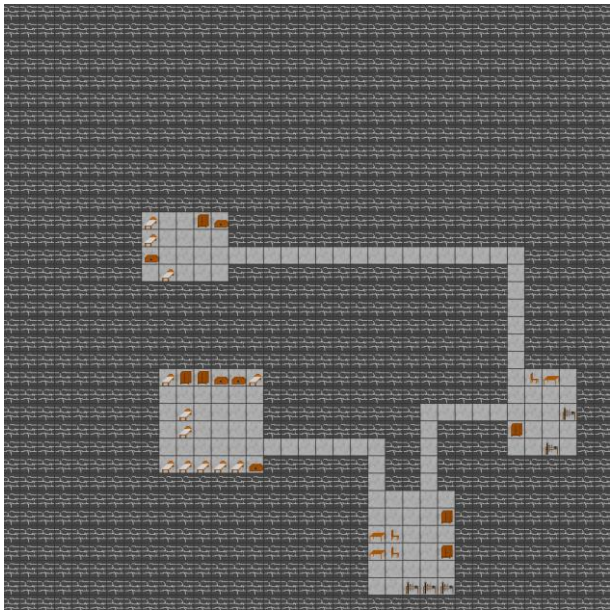
Trieda Type predstavuje typ miestnosti a zároveň k danej miestnosti priradí zoznam predmetov, ktorý následne rozmiestni po miestnosti. Podľa typu, aký si určila miestnosť sa zavolajú nasledujúce tri metódy: makeKitchen, makeLivingRoom, makeArmory. Každá z týchto troch metód určí danej miestnosti zoznam predmetov, ktorý je v nej. Keďže počet predmetov v miestnosti závisí od jej veľkosti, riešime to percentuálne. Keď už máme zoznam predmetov prislúchajúci miestnosti zavoláme metódu setLocation. Tá je spoločná pre každý typ miestnosti a každému predmetu určí polohu a rovno ju zapíše do mapy. Po tomto si už žiaden predmet nemusíme pamätať. Na obrázku č.5 vidíme rozmiestnenie predmetov po miestnostiach.



Obrázok 5 Rozmiestnenie predmetov

Poslednou dôležitou metódou je changeType(int,int). Táto metóda je volaná vždy z MapPanel. Slúži na to, aby sa daná miestnosť zmenila na daný typ.

Ak sa nám nepáči aktuálne rozmiestnenie predmetov, prepneme do režimu úprav a premiestnime určité predmety, alebo ich vyhodíme. Obrázok č.6 zobrazuje našu jaskyňu po týchto úpravách.



Obrázok 6 Jaskyňa po záverečných úpravách

3.5 Validácia

Aplikácia spĺňa všetky body určené špecifikáciou. Program prešiel sériou testov, ktorými sme overovali správnosť vygenerovaných máp. Všetky požiadavky vždy splnil. Sú dodržané požadované podmienky na stabilnosť. To znamená, že program dodržuje predchádzajúce úrovne generovania. Pri zmene jednotlivých miestností zachováva zvyšok mapy rovnaký. Pri načítaní súboru s nastaveniami generátora a zmenami na mape generuje rovnakú mapu. Veľkosť mapy, ktorú sme vygenerovali, je oveľa menšia, ako keby sme mapu zapísali po jednotlivých štvorcoch. Čím sme využili najväčšiu výhodu procedurálne vygenerovaného obsahu. Avšak pri opakovanom menení typov miestností alebo jednotlivých polí nám takto uložená mapa narastá. Preto je dôležité zbytočne neopakovať tie isté zmeny.

3.5.1 Možné vylepšenia

Aplikáciu je možné vylepšiť viacero spôsobmi. Môžeme rozšíriť paletu možností a pridať viacero nových podmienok, ktoré dajú užívateľovi väčšiu moc nad generovaním. Tak isto môžeme doprogramovať rôzne iné algoritmy, ktoré by generovali zaujímavejšie pôdorysy. Použitím napríklad bludiskových algoritmov alebo celulárneho automatu by vznikli úplne iné typy jaskýň. Keďže v našej práci sme iba demonštrovali metódy a výhody procedurálne generovaného obsahu, neponúkli sme veľa možností, ako ovplyvniť generovanie. Ďalším vylepšením by mohlo byť pridanie ďalších úrovní generovania. Napríklad pridávanie rôznych príšer alebo aj vygenerovanie rôznych úloh. Tým sa myslí to, že by sme mohli vygenerovať nejaký jednoduchý príbeh, podľa ktorého by hráč musel postupovať a následne na to vyrobiť prislúchajúcu mapu.

Záver

V tejto práci sme aplikovali metódy procedurálneho generovania na tvorbu máp, konkrétne jaskýň do RPG hry. Generovanie bolo ovplyvnené netriviálnymi požiadavkami užívateľa. Výroba mapy v takomto prípade trvala omnoho kratšie a bola jednoduchšia.

Metódy procedurálneho generovania sa ukázali ako efektívne na vyrábanie a ukladanie máp. Pomocou procedúr sme vygenerovaný obsah nemuseli nikam ukladať a tým sme odľahčili pamäť. Pri ukladaní mapy nám stačilo uložiť si nastavenia generátora, prípadne zopár zmien jednotlivých polí. Uložené mapy mali v našom prípade niekoľkokrát menšie veľkosti.

Rozdelenie generovania na jednotlivé úrovne zabezpečilo, že zmeny vykonávané na jednotlivých úrovniach nemali vplyv na zvyšok mapy a tým sme vedeli stabilne meniť jednotlivé miestnosti. Ukázali sme, že aj náhodne vygenerovaný obsah môžeme jednoducho ovplyvniť bez toho, aby sme museli zadať čo i len jednu konkrétnu požiadavku.

Použitá literatúra

[1] Roland van der Linden, Ricardo Lopes, Rafael Bidarra: Procedural generation of dungeons. 2013 [Online].

http://graphics.tudelft.nl/~rafa/myPapers/bidarra.TCIAIG.2014.pdf

[2] Bob Nystrom: Rooms and Mazes: A Procedural Dungeon Generator, 2014 [Online].

http://journal.stuffwithstuff.com/2014/12/21/rooms-and-mazes/

[3] Pavel Tišnovský: Perlinova šumová funkce a její aplikace, 2007 [Online].

http://www.root.cz/clanky/perlinova-sumova-funkce-a-jeji-aplikace/

[4] Michael Cook: Generate Random Cave Levels Using Cellular Automata, 2013 [Online].

http://gamedevelopment.tutsplus.com/tutorials/generate-random-cave-levels-using-cellular-automata--gamedev-9664

[5] Noor Shaker, Antonios Liapis, Julian Togelius, Ricardo Lopes and Rafael Bidarra: Procedural Content Generation in Games, 2013 [Online].

http://pcgbook.com/

[6] Procedural Content Generation Wiki. [Online].

http://pcg.wikidot.com/

[7] Makoto Matsumoto: Mersenne Twister. [Online].

http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html

Prílohy

Ako príloha je priložené CD so zdrojovým kódom vytvorenej aplikácie.