VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PROCEDURÁLNÍ GENEROVÁNÍ VOXELOVÝCH MODELŮ

PROCEDURAL GENERATION OF VOXEL MODELS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. TOMÁŠ HYPEŠ**
AUTHOR

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

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Hypeš Tomáš, Bc.**
Program: Informační technologie Obor: Počítačová grafika a multimédia
Název: **Procedurální generování voxelových modelů**
Procedural Generation of Voxel Models
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte techniky procedurálního generování a zobrazování voxelových modelů pomocí OpenGL.
2. Navrhněte aplikaci demonstrující procedurální metody.
3. Implementujte navrženou aplikaci.
4. Zhodnoťte, proměřte a vytvořte demonstrační video.

Literatura:

- Dle doporuření vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2 + kostra aplikace s několika metodami procedurálního generování.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Milet Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 6. listopadu 2018

Abstrakt

Tato práce pojednává o technikách procedurálního generování a jeho využití při tvorbě voxelových modelů. Využity byly techniky jako Perlinův šum, Voroného diagram, L-systémy apod. Tyto znalosti jsou následně využity pro vytvoření generátoru světa pro počítačovou hru s otevřeným světem. Tato hra poskytuje hráči možnost tento svět modifikovat a využít jeho kreativitu např. při stavbě budov. Hra ovšem neposkytne hráči všechny možnosti zadarmo, ale např. pro stavbu budovy si bude muset nejdříve najít a natěžit materiál. Hra byla napsána v programovacím jazyce C++ s využitím knihoven Boost, SDL a OpenGL.

Abstract

This thesis deals with procedural generation techniques and its use in the creation of voxel models. The techniques that have been used are Perlin Noise, Voronoi diagram, L-systems etc. This knowledge is then used to create a world generator for computer game with open world. This game provides players with the ability to modify this world and use its creativity, for example, in building construction. The game, however, will not give to the player all options for free, but for example for build, he or she will first have to find and mine the material. The game has been written in programming language C++ with the use of libraries Boost, SDL and OpenGL.

Klíčová slova

Procedurální generování, voxel, Perlinův šum, Voroného diagram, L-systém, počítačová hra, otevřený svět, C++, SDL, OpenGL

Keywords

Procedural generation, voxel, Perlin noise, Voronoi diagram, L-system, computer game, open world, C++, SDL, OpenGL

Citace

HYPEŠ, Tomáš. *Procedurální generování voxelových modelů.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Milet

Procedurální generování voxelových modelů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Mileta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Hypeš
20. května 2019

Poděkování

Rád bych tímto poděkoval vedoucímu práce, Ing. Tomáši Miletovi, za konzultace, návrhy a připomínky k tvorbě této práce.

Obsah

Kapitola 1

Úvod

Tato práce se zaměřuje na procedurální generování voxelových modelů. Procedurální generování slouží k vytváření nejrůznějšího obsahu za pomoci algoritmů a prvku náhody. Tyto generátory pak slouží jako náhrada za lidskou tvorbu, protože na rozdíl od lidí dokáží v krátkém čase vygenerovat velké množství dat.

Voxelové modely představují v počítačové grafice takové třírozměrné modely, které jsou sestaveny pomocí fragmentů uspořádaných v pravidelné mřížce. Tato struktura modelů dovoluje intuitivnější dekompozici prostoru, lepší možnosti komprese dat a jednodušší modifikace modelu uživatelem.

V praktické části této práce bude vytvořena hra s otevřeným světem, který bude implementován pomocí voxelového modelu. Tato hra poskytne hráči možnost tento svět modifikovat a využít jeho kreativitu např. při stavbě budov. Hra ovšem neposkytne hráči všechny možnosti zcela zdarma, ale např. pro stavbu budov si bude muset nejdříve najít a natěžit materiál.

V kapitole 2 budou podrobněji popsány hlavní termíny této práce – v podkapitole 2.1 bude popsané procedurální generování a v podkapitole 2.2 voxelový model. Dále budou popsány různé techniky procedurálního generování, a to konkrétně šumy, Voroného digramy a L-systémy.

V kapitolách 3 a 4 bude popsán návrh samotné hry a generátoru. Bude zde podrobněji popsána struktura světa a základní logika hry. V návrhu generátoru bude popsáno, jakým způsobem budou použity techniky z kapitoly 2 pro generování herního světa.

Implementace samotné hry bude popsána v kapitole 5, kde budou popsány jednotlivé knihovny a spustitelné soubory hry. Také budou popsány nejdůležitější datové struktury a implementované algoritmy. V kapitole 6 bude popsána implementace generátoru hry podle návrhu z kapitoly 4.

V kapitole 7 bude provedeno měření výkonu hry samotné a generátoru světa. V závěru (kapitola 8) bude celá práce zhodnocena a navržena případná budoucí rozšíření generátoru.

Kapitola 2

Teorie

Tato kapitola obsahuje seznámení se základními pojmy, které se týkají tématu této práce, a dále obsahuje několik základních technik procedurálního generování, které byly využity v praktické části této práce.

2.1 Procedurální generování

Procedurální generování [6] představuje algoritmické vytváření obsahu, a to buď zcela bez zásahu uživatele, nebo pouze s jeho minimální účastí. Cílem je získání dat, která se dále využijí např. v různých simulacích, počítačových hrách, filmovém průmyslu, apod. Pod těmito daty si můžeme představit např. generování textur různých materiálů, generování vegetace, krajiny, případně i celého vesmíru, generování budov, vesnic, měst, i celých civilizací, atd.

V těchto algoritmech se využívají nahodné prvky (generátory pseudonáhodných čísel), které dovolují za pomoci stejných pravidel generovat různý obsah – např. pokud bychom generovali texturu dřevěného materiálu, tato textura může po každém provedeném generování vypadat odlišně. Správný algoritmus generátoru by samozřejmě měl zajistit, že vygenerovaná textura bude vždy vypadat jako dřevo – generátor musí obsahovat pravidla, která popisují, jak má výsledek zhruba vypadat, ale určité nuance přenechává náhodě, např. v případě dřeva velikost, tvar a zakřivení letokruhů.

Prvek náhody mimo generování různých výsledků při opakovaném spuštění také dovoluje generování obrovských struktur, jejichž velikost není závislá na množství pravidel obsažených v generátoru. Např. požadujeme vytvoření modelu vesmíru obsahující hvězdy, planety, asteroidy atd. Když využijeme procedurální generování, nejsme omezeni velikostí tohoto vesmíru a teoreticky můžeme generovat i nekonečný vesmír. Nekonečné množství dat v konečném čase samozřejmě získat nemůžeme. Ale pokud budeme předpokládat, že v našem modelu vesmíru se bude pohybovat pozorovatel, stačí, pokud budeme generovat pouze jeho okolí, nemusíme generovat celý model. Při pohybu tohoto pozorovatele budeme následně generovat další části modelu. Pak se tento pozorovatel může po našem modelu libovolně pohybovat a nikdy se nemůže stát, že narazí na jeho okraj.

Procedurální generování nám dovoluje poskytnout cílovému uživateli obrovské množství obsahu, který by jinak musel vytvářet člověk, který svou uměleckou tvorbou dokáže vyprodukovat pouze omezené množství. Procedurální generování s využitím náhody nám ale nabízí prakticky neomezené množství obsahu, protože jak bylo již řečeno, při každém dalším spuštění nám může vrátit nový unikátní výsledek. Představme si například, že pro potřeby animovaného filmu potřebujeme vytvořit model lesa, ve kterém se následně bu-

dou odehrávat scény. Aby bylo možné vytvořit realistický les, není vhodné, aby jednotlivé stromy byly identické, protože v reálném lese vypadá každý strom trochu jinak – není možné v jednom lese nalézt dva stejné stromy. Je tedy nutné každý strom vymodelovat zvlášť, což je pro člověka mnoho práce, která se navíc musí zaplatit. Ovšem pokud použijeme procedurální generování, vytvoříme jeden algoritmus, který dokáže vygenerovat strom, a ten nám následně díky zapojení prvku náhody dokáže dodat libovolné množství různých stromů.

2.2 Voxelový model

Voxel je složenina anglických slov *volumetric element* – objemový prvek. Představuje částici, ze kterých je v počítačové grafice sestaven 3D model – voxelový model. Tyto částice jsou obvykle uspořádány do pravoúhlé mřížky a jedna částice má tedy tvar krychle. Jedná se o převedení pojmu pixel (*picture element* – obrázkový prvek) z 2D grafiky, který má tvar čtverce, do 3D.

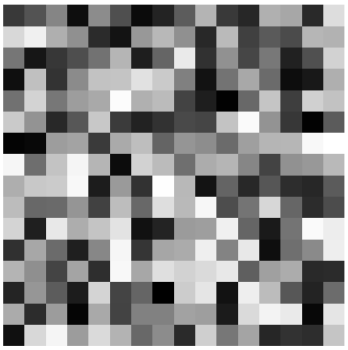
Takto vytvořený model potom můžeme buď vykreslovat přímo použitím techniky sledování paprsků (angl. *ray tracing*), nebo můžeme jednotlivé stěny voxelů převést na polygony a ty vykreslit klasickým způsobem jako trojúhelníky. V tomto případě můžeme použít optimalizaci, že nemusíme vykreslovat ty strany voxelů, které přímo sousedí se stranami sousedních voxelů a tudíž nemohou být vidět z vnějšku objektu, který je z těchto voxelů složen.

2.3 Šumy

Šumy v kontextu procedurálního generování představují diskrétní nebo spojitý n-dimenzionální prostor hodnot. Ty pak slouží jako náhodný základ pro další výpočty v algoritmech procedurálních generátorů např. pro generování textur materiálů nebo výškových map.

Hodnotový šum

Nejjednoduším šumem je tzv. hodnotový šum [1]. Tento šum vznikne vytvořením n-rozměrné ortogonální mřížky, která je naplněna náhodnými hodnotami. Na obrázku 2.1 se nachází příklad hodnotového šumu v dvourozměrném prostoru, který je vyjádřen jako šedotónový obrázek.



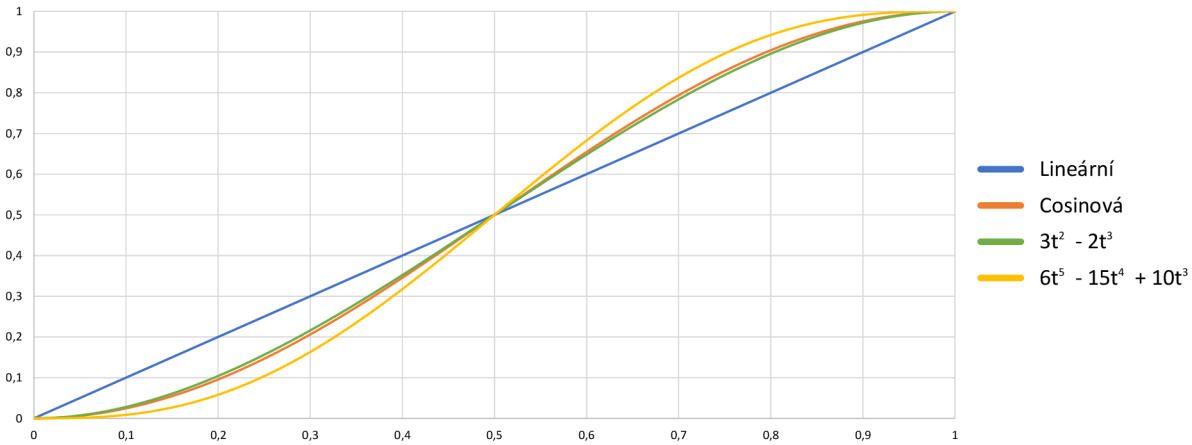
Obrázek 2.1
Hodnotový šum

Obrázek 2.2
S lineární interpolací

Obrázek 2.3
S nelineární interpolací

Pokud potřebujeme na tento šum nahlížet jako na spojitý prostor, je vhodné hodnoty na pozicích ležících mezi definovanými v mřížce počítat pomocí interpolace, a to tak, že pro požadovanou pozici budeme uvažovat 2^n nejbližších bodů tvořící vrcholy hyperkrychle[1] daného prostoru, ve které se tato pozice nachází. Tuto množinu vrcholů následně budeme postupně redukovat tak, že vždy z daného prostoru odstraníme jednu dimenzi. Odstraněním jedné dimenze dostaneme vždy dvojice vrcholů, které leží na stejné pozici. V každé dvojici sloučíme hodnoty vrcholů do jedné pomocí váženého průměru, jehož váhy budou představovat vzdálenosti od původních bodů v ose představující odstraněnou dimenzi. Jakmile odstraníme všechny dimenze, dostaneme právě jednu hodnotu, která představuje výsledek na dané pozici. Příklad hodnotového šumu s interpolovanými hodnotami se nachází na obrázku 2.2.

Interporlací získáme tzv. koherentní šum, tzn. že mezi libovolnými dvěma body v prostoru se hodnoty šumu mění plynule – spojitě. Ovšem tato interpolace má jistý nedostatek – vždy na rozhraní mezi hyperkrychlemi dochází k ostré hraně mezi barevnými přechody. Tento nedostatek lehce odstraníme tím, že při výpočtu interpolace vzdálenost upravíme tak, aby se její hodnota na rozhraní hyperkrychlí neměnila zlomově – tzn. potřebujeme tuto vzdálenost upravit tak, aby u minimální (0) a maximální (1) hodnoty byla její derivace rovna 0. Na grafu nacházejícím se na obrázku 2.4 jsou znázorněny některé funkce, které pro tento účel můžeme využít. Na obrázku 2.3 je příklad hodnotového šumu, u kterého byla pro úpravu vzdáleností při interpolaci použita funkce $6t^5 - 15t^4 + 10t^3$, kde t je původní lineární hodnota vzdálenosti.



Obrázek 2.4: Graf znázorňující různé funkce pro úpravu hodnoty vzdálenosti v interpolaci

I přesto, že díky vhodné interpolaci jsme získali z hodnotového šumu spojitý koherentní šum, není takový šum pro použití v procedurálním programování příliš vhodný, protože ve výsledném šumu je stále snadno možné pouhým okem rozeznat původní pravoúhlou mřížku. Pokud chceme generovat např. různé přírodní jevy, které tuto mřížku v sobě neobsahují, není pro tyto účely použití hodnotového šumu vhodné.

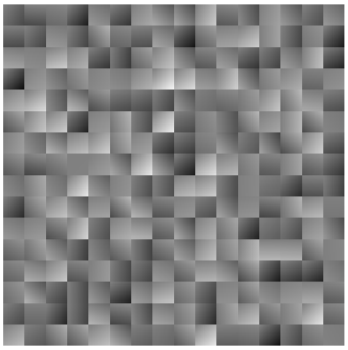
[1] Hyperkrychle je zobecnění krychle do prostorů o libovolném počtu dimenzí. V 1D se jedná o úsečku se 2 vrcholy, ve 2D o čtverec se 4 vrcholy, ve 3D o krychli s 8 vrcholy, ve 4D o teserakt s 16 vrcholy atd.

Perlinův šum

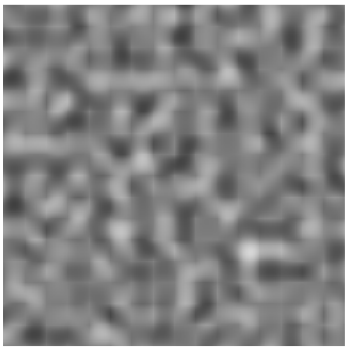
Perlinův šum [1] objevil Kenneth H. Perlin v 80. letech minulého století v důsledku své frustrace nad dosavadní strojovou podobou počítačové grafiky. Tento šum poprvé představil v článku *An Image Synthesizer* [9] v roce 1985.

Tento šum je stejně jako hodnotový šum založen na n-rozměrné ortogonální mřížce, která ale nyní obsahuje n-rozměrné vektory – gradienty. Tento gradient nepředstavuje konečnou hodnotu v daném bodě, ale pouze normálu na spojité n-ární funkci definované pomocí těchto gradientů. Abychom sestrojili tuto funkci, pro výpočet gradientů ležících mezi definovanými v mřížce použijeme opět interpolaci, stejně jako v případě hodnotového šumu popsaném v předchozí podkapitole.

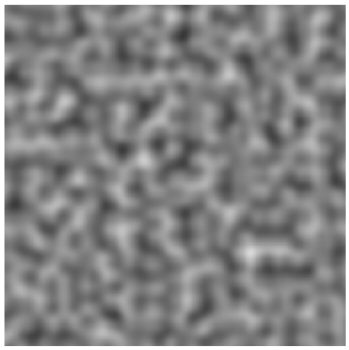
Výpočet hodnoty Perlinova šumu v libovolném bodě probíhá podobně, jako u hodnotového šumu. Nejdříve nalezneme 2^n nejbližších bodů v mřížce představující hyperkrychli, v které se daný bod nachází. Následně pak pro každý vrchol vypočteme hodnotu tak, že od pozice našeho bodu odečteme pozici vrcholu a provedeme skalární součin s vektorem gradientu. Tyto hodnoty ve vrcholech hyperkrychle zredukujeme na jeden použitím interpolace – tato interpolace je opět popsána v podkapitole o hodnotovém šumu.



Obrázek 2.5
Gradientní šum



Obrázek 2.6
S lineární interpolací



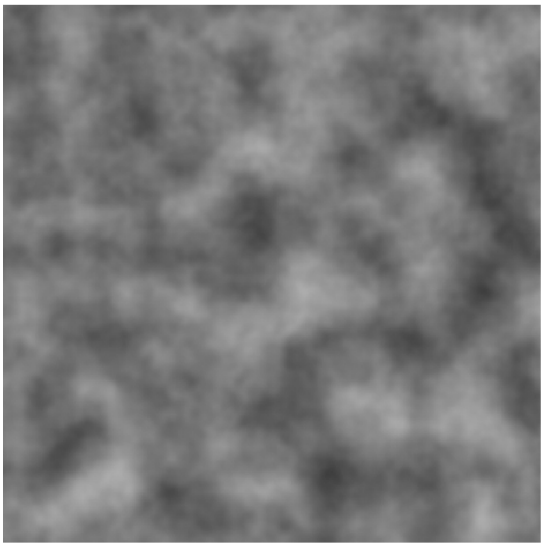
Obrázek 2.7
S nelineární interpolací

Na obrázku 2.5 se nachází ukázka 2D Perlinova šumu, ve kterém nebyla použita interpolace mezi gradienty, ale jako výsledná hodnota se použila vypočtená hodnota pouze jednoho vrcholu hyperkrychle. Na obrázku 2.6 už byla použita interpolace, ale pouze lineární. Opět se zde nachází ostré zlomy na hranicích jednotlivých hyperkrychlí, díky kterým je snadno viditelná původní mřížka. A konečně na obrázku 2.7 byla použita nelineární interpolace, která odstranila ostré zlomy mezi přechody. V tomto šumu již není možné rozeznat žádné stopy po původní mřížce a jedná se již o použitelný koherentní šum.

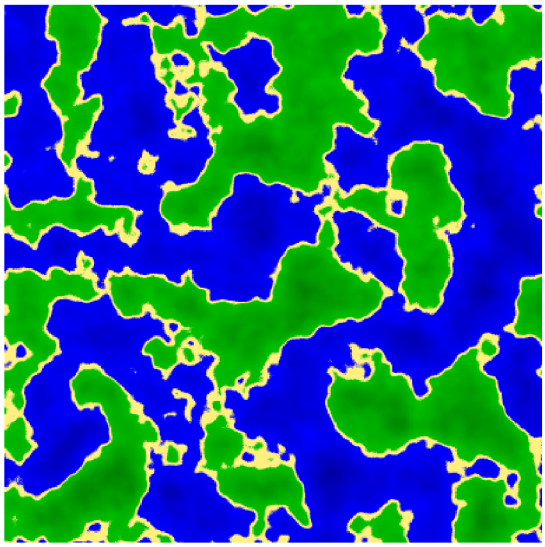
Tento šum ale stále obsahuje jeden nedostatek. Když si výsledek tohoto šumu podrobněji prohlédnete, můžete si všimnout určitých pravidelností, a to, že mezi minimy a maximy tohoto šumu je velmi podobná vzdálenost a navíc tyto extrémy mají navzájem velmi podobnou velikost.

Tento nedostatek lze odstranit složením více šumů dohromady – v každém bodě se sečtou hodnoty všech dílčích šumů. Tyto dílčí šumy se nazývají oktávy. Jednotlivé oktávy mají různá měřítka mřížky a různou amplitudu (maximální absolutní hodnotu). Asi nejlepšího výsledku dosáhneme v případě, kdy každá následující oktává má 2× větší vzdálenost mezi hodnotami v mřížce a 2× větší amplitudu. Takto složený šum se nazývá fraktální [1] – teoreticky se může daný šum skládat z neomezeného množství šumů a při neomezeném

přiblížení tohoto šumu se nám mohou objevovat další a další detaily, stejně jako je tomu v případě fraktálů.



Obrázek 2.8: Fraktální šum



Obrázek 2.9: S obarvením

Na obrázku 2.8 se nachází ukázka fraktálního šumu. Tento šum již vypadá velmi přirozeně a může nám např. připomínat kouř nebo mraky. Obrázek byl vygenerován s paletou barev představující stupně šedi. Pro vygenerování obrázku 2.9 byl použit stejný šum, ale jiná barevná paleta. Zde nám tento šum může připomínat pevninu s vodními plochami, což je zároveň i jeden z případů, pro který je možné Perlinův šum využít. V tomto případě by jeden Perlinův šum nebyl použit pouze pro oddělení pevniny od moře, ale také jako výšková mapa pro vytvoření kopců a hloubky dna moře. Toto je na obrázku znázorněno různými odstíny zelené, respektive modré.

2.4 Voroného diagram

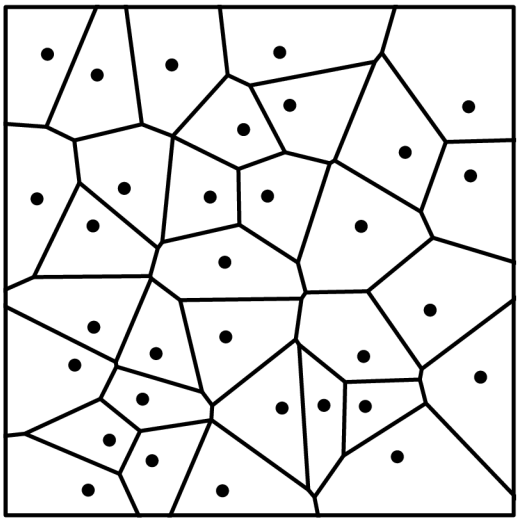
Voroného diagram [5] je pojmenován podle ukrajinského matematika Georgije F. Voronoje. Jedná se o metodu dekompozice prostoru na navzájem výlučné oblasti podle definovaných řídících bodů. Prostor je rozdělen na stejný počet oblastí (buněk), jako je počet těchto bodů. V každé buňce se pak nachází právě jeden řídící bod. Pro všechny body v této buňce pak platí, že nejbližším řídícím bodem je ten, který se nachází uvnitř dané buňky.

Na obrázku 2.10 je ukázka rozdělení 2D prostoru omezeného čtvercem pomocí Voroného diagramu. Nachází se zde 32 náhodně rozmístěných řídících bodů, které daný čtverec rozdělují na 32 buněk.

Voroného diagram lze použít v prostoru o libovolném počtu dimenzí. Pro určení vzdálenosti mezi dvěma body se obvykle používá Euklidovská vzdálenost v Euklidově prostoru, je ale možné použít i jiné metriky.

2.5 L-systémy

L-systémy [10] definoval maďarský biolog Aristid Lindenmayer a nezkráceně se nazývají Lindenmayerovy systémy. Původně se jedná o prostředek pro modelování růstu rostlin.



Obrázek 2.10: Voroného diagram

L-systém definuje pravidla, která určují, jak má stonek rostliny růst, kde se má rozvětvit, nebo na kterém místě má vzniknout list. Tato pravidla jsou opakovaně aplikována na model rostliny a jednotlivé iterace představují postupný růst.

L-systémy lze využít i mimo biologii pro generování nejrůznějších objektů. Často se využívají pro generování fraktálů, kdy je na počátku jednoduchý tvar, který se aplikováním pravidel stává složitějším. Po provedení nekonečného množství iterací bychom pak získali fraktál. V praxi nám ale stačí provést konečný počet iterací, protože v případě fraktálů každá další iterace vnáší do modelu menší změny vzhledem k jeho měřítku a tudíž po relativně málo iteracích již nejsme schopni rozlišit vygenerovaný útvar od pravého fraktálu.

V počítačové grafice se L-systémy kromě generování rostlin (tráva, stromy, květiny, ...) využívají také ke generování nepřírodních struktur jako jsou různé stavby, sítě cest nebo chodeb apod.

D0L-systémy

Základní variantou jsou tzv. D0L-systémy [7] – deterministické bezkontextové L-systémy. Definovat je lze jako trojici $G = (\Sigma, P, S)$, kde:

- Σ je konečná množina symbolů (abeceda).
- P je množina přepisovacích pravidel.
  $P \subseteq A \times B$, kde: $A = \Sigma$, $B = \Sigma^*$
  Přepisovací pravidla se zapisují ve tvaru $A \to B$.
- S je počáteční řetězec (axiom).
  $S \in \Sigma^+$

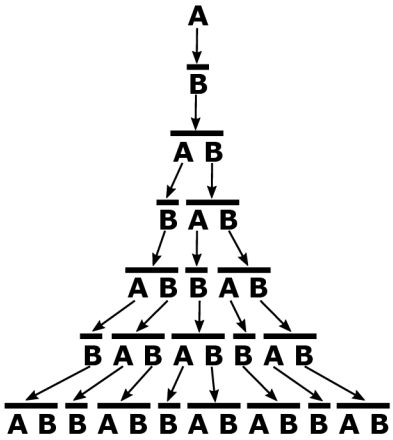
Pro všechny symboly a z abecedy Σ, pro které neexistuje přepisovací pravidlo v P, u kterého by se symbol a nacházel na levé straně, se uvažuje implicitní přepisovací pravidlo ve tvaru $a \to a$, které představuje identitu – po aplikování tohoto pravidla zůstane řetězec nezměněný. Díky tomu se nemusí uvažovat terminální a neterminální symboly, jako je tomu v případě bezkontextových gramatik.

Derivace řetězce se nazývá operace, kdy jsou všechny symboly řetězce nezávisle na sobě (paralelně) nahrazeny řetězci podle přepisovacích pravidel v P, a to tím způsobem, že symbol na levé straně pravidla je v řetězci nahrazen řetězcem pravé strany pravidla. Výsledný řetězec po provedení n derivací se nazývá n-tá iterace systému.

Jako příklad uvažujme systém $G_{fib} = (\{\mathrm{A}, \mathrm{B}\}, \{\mathrm{A} \rightarrow \mathrm{B}, \mathrm{B} \rightarrow \mathrm{AB}\}, \mathrm{A})$. Tento systém generuje Fibonacciho posloupnost, pro kterou platí, že každý následující prvek je součtem dvou předchozích. V tomto systému jsou hodnoty posloupnosti ukryté v délkách řetězců jednotlivých iterací. Tabulka 2.1 demonstruje, jak vypadají řetězce generované systémem v prvních 7 iteracích. Na obrázku 2.11 se pak nachází grafické znázornění jednotlivých derivací pro získání lepší představy, jakým způsobem jsou symboly přepisovány.

Iterace	**Řetězec**
0.	A
1.	B
2.	AB
3.	BAB
4.	ABBAB
5.	BABABBAB
6.	ABBABBABABBAB
7.	BABABBABABBABBABABBAB

Tabulka 2.1: Jednotlivé iterace systému G_{fib}



Obrázek 2.11: Grafické znázornění

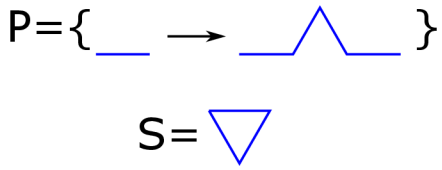
Želví grafika

Jednou z možností, jak graficky reprezentovat řetězce L-systémů, je tzv. želví grafika (angl. *turtle graphics*). Jedná se o představu, kdy se želva pohybuje v písku a zanechává za sebou stopu. Může vykonávat tyto tři pohyby: lézt vpřed, otočit se doprava a otočit se doleva. Tyto tři pohyby (instrukce) budeme reprezentovat symboly F, + a −, přičemž délka pohybu vpřed bude definována konstantou d a úhel otočení oproti aktuálnímu směru bude dán konstantou α. Želva vždy vykonává svůj pohyb z poslední pozice, ve které se nacházela, a vpřed leze vždy směrem, kterým je otočena. Interpret, který bude interpretovat řetězec generovaný L-systémem, bude tedy muset uchovávat dvě informace, ze kterých bude vycházet při interpretaci následujícího symbolu, a to aktuální pozici a směr.

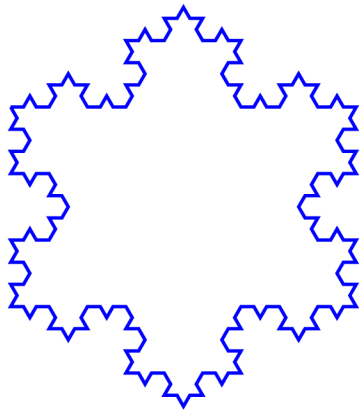
Nyní uvažujme systém $G_{koch} = (\{\mathrm{F}, +, -\}, \{\mathrm{F} \rightarrow \mathrm{F}-\mathrm{F}++\mathrm{F}-\mathrm{F}\}, \mathrm{F}++\mathrm{F}++\mathrm{F})$ a konstanty $d = 1$ a $\alpha = 60°$. Tento systém generuje fraktálový útvar zvaný Kochova vločka. Vygenerovaný řetězec obsahuje instrukce, jak tento obrazec nakreslit. Obrázek 2.13 znázorňuje grafickou reprezentaci třetí iterace tohoto systému. Na obrázku 2.12 je graficky znázorněno přepisovací pravidlo a počáteční řetězec.

Pomocí těchto třech instrukcí lze kreslit pouze souvislou křivku. Aby bylo možné kreslit komplexnější grafiku, můžeme např. zavést sémantiku, že všechny symboly reprezentované velkými písmeny budou interpretovány jako pohyb s kreslením čáry, a všechny symboly reprezentované malými písmeny budou představovat pohyb bez kreslení.

Dalším rozšířením želví grafiky jsou tzv. závorkové L-systémy. Abeceda je rozšířena o symboly [a]. Symbol [představuje vložení aktuálního stavu (pozice a směr) na zásobník.



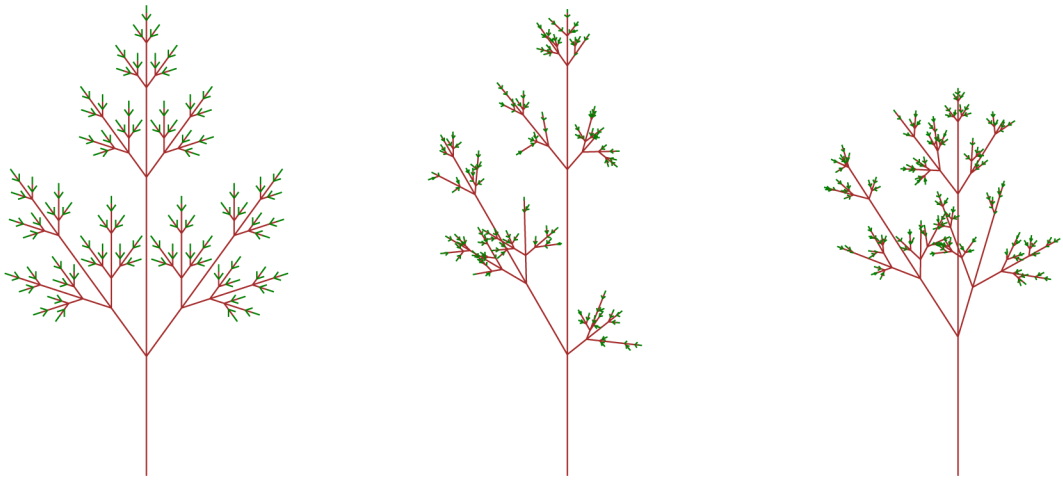
Obrázek 2.12: Systém G_{koch}



Obrázek 2.13: 3. iterace systému G_{koch}

Při přečtení symbolu] se aktuální stav změní na stav nacházející se na vrcholu zásobníku a ze zasobníku se tento stav odebere. Toto rozšíření zjednodušuje návrh L-systémů, ve kterých chceme provést např. rozvětvení: řetězec nejdříve bude obsahovat instrukce pro popis jedné větve, následovat bude návrat do stavu před touto větví a popis druhé větve.

Uvažujme systém G_{tree} = ({W,L,+, −, [,]}, {L→W[+L][−L]WL,W→WW},L). Tento systém generuje strom, viz obrázek 2.14 (první zleva). Prvním pravidlem dojde k rozdělení na tři větve – nejdříve pravá, potom levá a nakonec prostřední. Po podřetězci, který popisuje postranní větev, dojde k navrácení stavu před větvením a pokračuje se další větví. Symboly L se následně v další iteraci budou přepisovat podle stejného pravidla a důsledkem toho budou další vnořené závorky uvnitř vnějších – to je důvod, proč je pro interpretaci nutný zásobník.



Obrázek 2.14: Stromy generované systémem G_{tree}

Parametrické systémy

Parametrické L-systémy rozšiřují základní L-systémy o parametry symbolů. Každý symbol má konečný počet parametrů a jednotlivé parametry mohou nabývat libovolné hod-

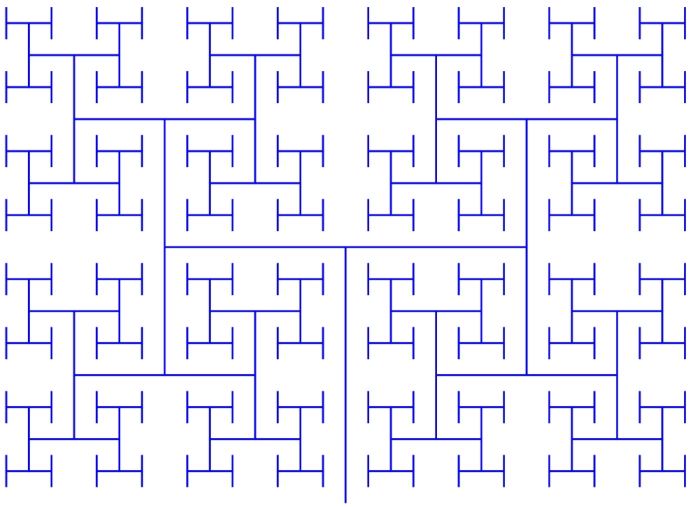
noty z oboru reálných čísel. Pro zápis symbolu s parametry se používá závorková syntaxe: $a(x_1, x_2, \ldots, x_n)$, kde a je symbol a x_i jsou jeho parametry.

Za parametry symbolů lze dosazovat také proměnné a aritmetické vzorce, které jsou následně vyhodnocovány. Toho se využívá v přepisovacích pravidlech. Např. mějme pravidlo: $\mathrm{A}(a, b) \rightarrow \mathrm{B}(2a)\,\mathrm{C}(6/b)$. Toto pravidlo se použije pro všechny symboly A s libovolnými parametry nacházející se ve vstupním řetězci derivace. Pokud se bude v řetězci nacházet např. symbol $\mathrm{A}(1, 2)$, pomocí zmíněného pravidla se přepíše na řetězec $\mathrm{B}(2)\,\mathrm{C}(3)$.

Jako ukázku uvažujme systém:

$$G_H = (\{\mathrm{A}, \mathrm{B}, +, -, [\,,]\,\}, \{\mathrm{A}(l) \rightarrow \mathrm{B}(l)\,[\,+\mathrm{A}(l/\sqrt{2})\,]\,[\,-\mathrm{A}(l/\sqrt{2})\,]\,\}, \mathrm{A}(1))$$

Parametr symbolů A a B udává délku čáry. Úhel otočení je zde 90°. Tento systém generuje fraktálový obrazec zvaný H-strom, jehož 8. iteraci si můžete prohlédnout na obrázku 2.15. Po každém rozvětvení je následující větev kratší. Toto zkrácení je dáno vzorcem $l/\sqrt{2}$ v přepisovacím pravidle, kde l je délka předchozí větve.



Obrázek 2.15: 8. iterace systému G_H

Další možností, jak využít parametry symbolů, jsou podmíněná přepisovací pravidla. Pravidlo může obsahovat výraz (podmínku), ve kterém mohou vystupovat parametry symbolu na levé straně pravidla – pak se toto pravidlo použije pouze tehdy, kdy je tento výraz vyhodnocen jako pravdivý. V opačném případě se buď použije jiné pravidlo se stejným symbolem na levé straně, nebo v případě, že takové pravidlo neexistuje, se použije implicitní pravidlo představující identitu.

Stochastické systémy

U deterministických L-systému platí, že při opakovaném generování je každá n-tá iterace totožná s n-tou iterací z předchozího generování. V procedurálním generování ale vyžadujeme určitý prvek náhody, abychom pomocí relativně malého množství pravidel byli schopni vygenerovat velké množství unikátního obsahu. K tomu slouží stochastické L-systémy (0L-systémy).

Nejjednodušším způsobem, jak přidat do L-systémů prvek náhody, je při interpretaci, kde pomocí náhody pozměníme délku čáry a úhel otočení. Výsledek zde již vypadá po

každém generování trochu jinak, ale tato nahodilost není dostačující, protože pokud bychom tímto způsobem generovali např. strom, jeho topologie bude vždy stejná – vždy bude mít stejný počet větví a rozvětvení.

Lepším způsobem zavedení náhodilosti do L-systémů je náhodný výběr přepisovacího pravidla. Pokud pro jeden symbol existuje více přepisovacích pravidel, u kterých je tento symbol na levé straně, z těchto pravidel se vybere náhodně jedno a to se použije. U pravidel navíc může být definována jejich pravděpodobnost při výběru, díky čemuž docílíme toho, že některá pravidla budou vybírána častěji, než-li jiná.

Ukázka použití stochastického L-systému se nachází na obrázku 2.14, na kterém prostřední a pravý strom byl vygenerován náhodně. Použitý L-systém vychází ze systému G_{tree}, do kterého bylo navíc přidáno přepisovací pravidlo L $\rightarrow$ L (identita). Toto přidané pravidlo způsobuje, že se náhodně volí, zda-li v aktuální derivaci dojde v daném místě k rozvětvení (použije se původní pravidlo pro L), nebo ne (použije se přidané pravidlo). Navíc je zde přidána nahodilost délky čáry a úhlu otočení.

Otevřené systémy

Dalším rozšířením L-systémů je komunikace s prostředím – volba přepisujícího pravidla závislá na okolí. První možností je rozhodování na základě okolních symbolů v řetězci – tzn. že povýšíme bezkontextové L-systémy, které odpovídají bezkontextovým gramatikám, na kontextové L-systémy odpovídající kontextovým gramatikám. Rozlišujeme dva druhy kontextových L-systémů. Prvním jsou tzv. 1L-systémy, které se při výběru přepisovacího pravidla rozhodují pouze podle symbolů nacházející se před přepisovaným symbolem. Druhým jsou tzv. 2-L-systémy, které berou v potaz i symboly nacházející se za právě přepisovaným.

Druhou možností je komunikace L-systému s vnějším okolím – např. detekce překážek v prostoru. Kontextové stochastické L-systémy s komunikací s prostředím [7] se nazývají otevřené L-systémy.

Kapitola 3

Návrh hry

V rámci této práce vytvořím počítačovou hru, ve které se hráč bude pohybovat ve světě tvořeným voxely (bloky), který bude navíc moci v rámci hry modifikovat. Inspirací pro tuto hru byla hra Minecraft [2] od švédské firmy Mojang [3].

V této kapitole popíši návrh této hry a v následující kapitole 4 generátor světa, na který se tato práce zaměřuje.

3.1 Popis hry

Ve hře bude hráči nabídnut zdánlivě neomezený[1] svět. Tento svět bude již při prvním spuštění obsahovat vygenerované prostředí pomocí generátoru světa, které se bude skládat z krajiny (lesy, hory, moře, ...), podzemí (jeskynní komplexy, nerostné suroviny, ...) a různých dalších struktur (budovy, vesnice, tvrze, ...). V tomto světě se bude hráč pohybovat pomocí své virtuální postavy představující lidskou bytost. Herní svět bude pozorovat z prvního pohledu[2] této postavy a pohybovat se v něm bude chůzí, během, výskoky, případně pádem. Hra bude typu sandbox[3], tzn. že bude moci tento vygenerovaný svět libovolně měnit, což je zároveň hlavní podstatou této hry. Hra bude mít za cíl nabídnout hráči možnost využití vlastní kreativity, kterou může využít při modifikacích herního světa – úpravy krajiny, stavby budov, hloubení chodeb, apod. Ovšem tyto možnosti hra neposkytne hráči zadarmo – aby mohl např. postavit budovu, bude muset nejdříve v daném světě nalézt suroviny a vytěžit je. A aby mohl některé materiály efektivně těžit, bude si muset na to vyrobit požadované nástroje.

3.2 Základní bloky

Základním prvkem hry je tzv. blok, který v modelu světa představuje voxel, viz kapitola 2.2. Blok je krychle o velikosti $1 \times 1 \times 1$ metr, jejíž hrany jsou rovnoběžné s osami. Bloky jsou základním kamenem celého herního světa, ve kterém jsou uspořádány do trojrozměrné mřížky. Každý blok je možné lokalizovat pomocí třírozměrných souřadnic vyjádřenými celými čísly,

[1] Z implementačních důvodů bude velikost světa omezená, ale prostor, po kterém se bude hráč moci pohybovat, bude tak obrovský, že by hráč při běžném hraní nikdy neměl narazit na omezení.

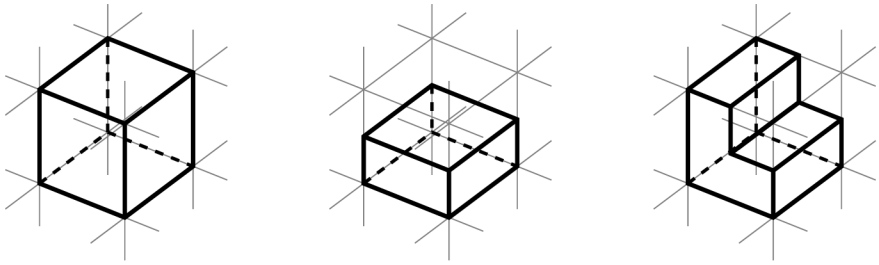
[2] Pohled první osoby v kontextu počítačových her znamená, že hráč prostředí hry sleduje na své obrazovce prostřednictvím virtuální kamery umístěné v prostoru očí jeho herní postavy.

[3] Sandbox (česky „pískoviště“) je typ hry, která hráči poskytuje nástroje, kterými může modifikovat prostředí herního světa.

a zároveň na každé pozici ve světě, která je definována celočíselnými souřadnicemi, se nachází právě jeden blok.

Hra bude obsahovat velké množství druhů bloků. Každý druh bloku má vlastní specifický vzhled a vlastnosti. Většina bloků představuje vyplněnou krychli určitým materiálem, např. kámen, hlína, písek, dřevo apod. Většina takových bloků je neprůhledná a představují pro hráče tzv. zeď – hráč jimi nemůže projít ani propadnout, a pokud se o to pokusí, hra jeho pohyb zastaví před hranicí tohoto bloku. Některé takové bloky mohou být i částečně průhledné, jako např. listí nebo sklo.

Speciálním druhem bloku je vzduch, který je zcela průhledný a pro hráče nepředstavuje zeď – v rámci těchto bloků se může libovolně pohybovat. Tento blok zastupuje ve hře situaci, kdy by se na dané pozici nenacházel žádný blok.



Obrázek 3.1: Tvary materiálových bloků, zleva: krychle, deska, schody

Některé materiálové bloky dovolují kromě krychle ještě dva další speciální tvary, viz obr. 3.1. Prvním z nich je tzv. deska (angl. „slab“), nebo také půlblok. Tento tvar představuje kvádr o velikosti $1 \times 1 \times 0,5$ metru, který může být orientován na libovolnou světovou stranu[4]. Pro hráče se tento blok chová jako zeď pouze v té polovině, která představuje materiál. Druhá polovina bloku se chová jako vzduch a hráč se v této polovině může libovolně pohybovat.

Druhým speciálním tvarem některých materiálových bloků jsou tzv. schody. Jejich tvar vznikne složením půlbloku s orientací dolů, případně nahoru a půlbloku orientovaného do některé horizontální světové strany[5]. Blok se pro hráče opět chová jako zeď pouze v místech, která představují materiál.

3.3 Kolize hráče se zdmi

Než budeme pokračovat výčtem dalších druhů bloků, více si přiblížíme pojem zeď. Zdí je v tomto kontextu myšlen kvádr, jehož hrany jsou rovnoběžné s osami. Tyto kvádry omezují prostor v herním světě, ve kterém se může hráč pohybovat. Hráč má kolem sebe také obalující kvádr, který je závislý na jeho aktuální pozici. Slouží pro detekci, zda-li nedochází ke kolizi hráče a zdi. Pokud herní logika detekuje, že by k této kolizi mohlo dojít, musí zajistit, aby se tak nestalo – např. při chůzi hráče zastaví těsně před zdí, aby nedošlo ke kolizi, a nebo nedovolí hráči propadnout kvádrem zdi, na kterém se hráč aktuálně nachází a představuje pro něj podlahu.

[4]Herní svět obsahuje celkem 6 světových stran: východ/západ podle osy X, sever/jih podle osy Z a nahoru/dolů podle osy Y.

[5]Herní svět obsahuje celkem 4 horizontální světové strany: východ/západ podle osy X a sever/jih podle osy Z.

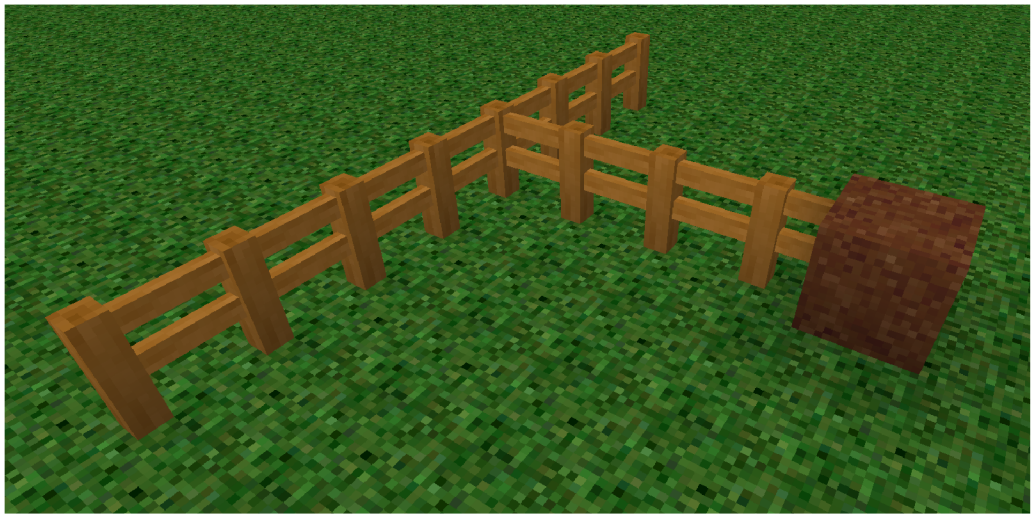
Vlastností každého bloku je množina kvádrů zdí závislá na druhu bloku, případně dalších vlastností bloku. Materiálové bloky s triviálním krychlovým tvarem mají jeden kvádr, který přesně odpovídá hranicím bloku. Půlbloky mají v této množině podobný kvádr, jako krychlové bloky, ale jedna strana tohoto kvádru je posunuta do středu bloku v závislosti jeho orientace, aby odpovídal přesně prostoru, ve kterém se nachází materiál. Schody mají složitější tvar, který musí být vyjádřen dvěma kvádry, a to opět tak, aby odpovídaly prostoru vyplněném materiálem. Blok vzduchu má tuto množinu prázdnou.

Aby mohl hráč chodit po schodech, nebo vystoupit na půlblok orientovaný dolů, který je stejně vysoký jako jeden schod, je logika zdí rozšířena o další pravidlo. Pokud se před hráčem nachází zeď do výšky $0,5$ metru a nad touto zdí je dostatek prostoru, aby se do něj hráč vešel (jeho obalový kvádr nekolidoval se žádným kvádrem zdi), je v tomto výjimečném případě hráči umožněno vstoupit do zdi a do té doby, dokud hráč tímto způsobem koliduje se zdí, ho hra postupně posouvá nahoru, dokud se hráč nedostane mimo kolizi této zdi a nestojí na daném schodě, respektive půlbloku. Z pohledu hráče se potom tento pohyb jeví tak, jako kdyby po daných schodech vystupoval.

Pokud je zeď před hráčem vysoká 1 metr (právě 1 blok), již tuto zeď neprojde pouhou chůzí, ale může pro její překonání využít výskok, který je nastaven přesně tak, aby hráči umožnil vyskočit do této výšky. Vyšší zeď již překonat nedokáže. Skok je také možné použít v kombinaci s rozeběhem pro překonání propasti.

3.4 Další druhy bloků

Mimo bloky materiálů popsané v podkapitole 3.2, kterých by měla být většina, budou ve hře další různě speciální bloky. Blokem zvláštním především z pohledu zdí jsou ploty, které si můžete prohlédnout na obrázku 3.2. Jeden blok plotu odpovídá jednomu sloupku, který se nachází přesně uprostřed bloku a jeho výška odpovídá výšce bloku. Pokud se na vedlejší pozici nachází další blok plotu, mezi těmito sloupky se objeví spojnice. Případně, pokud je ve vedlejším bloku nějaký blok materiálu, se zde objeví spojnice o poloviční délce směřující do tohoto bloku – toto chování je vhodné pro situace, kdy by hráč potřeboval propojit plot např. se stěnou budovy.



Obrázek 3.2: Bloky plotu

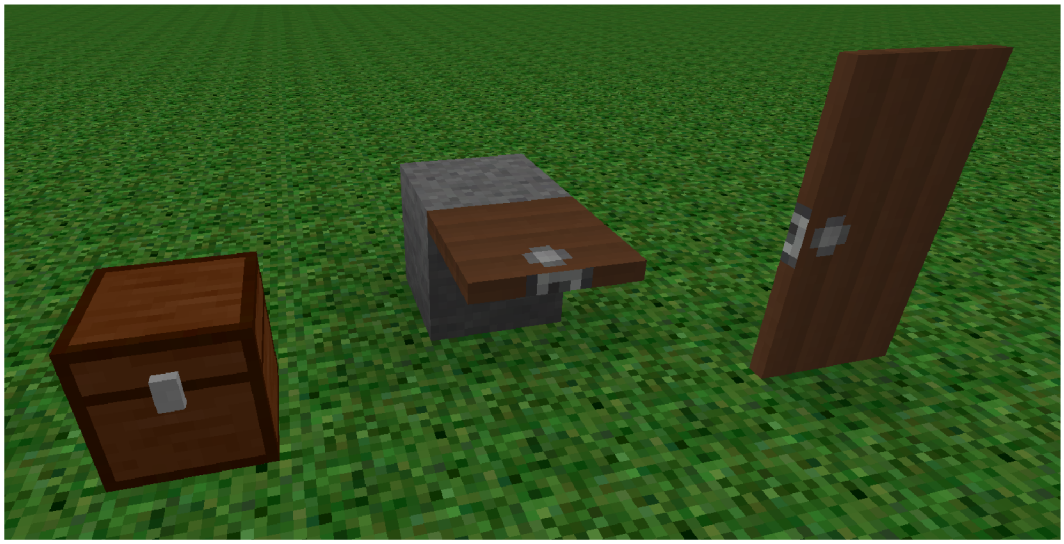
Každý blok plotu má jeden kvádr zdi představující sloupek. Dále může mít až další čtyři kvádry představující spojnice do okolních bloků. Tloušťka i výška těchto spojnic z pohledu kvádrů zdí je stejná, jako sloupků, ačkoliv tyto spojnice se vykreslují uzší a nižší. Je tomu tak z toho důvodu, pokud by se hráč pohyboval podél plotu a spojnice by byla užší, zarazil by se o sloupek, jehož kvádr zdi by vstupoval do prostoru, ve kterém se hráč pohybuje.

Další zajímavostí je to, že tyto kvádry zdí přesahují až do bloku nad plotem. To je z toho důvodu, aby hráč nemohl plot přeskočit, protože hráč je schopen vyskočit do výšky jednoho bloku a právě tak je plot vizuálně vysoký. Ovšem plot nemůže být vizuálně vyšší, protože by mohl nastat problém v případě, kdy by se nacházel nějaký blok nad ním, u kterého by tento přesah mohl způsobovat problémy.

Podobným blokem, jako ploty, jsou skleněné tabule. Chovají se velmi podobným způsobem, jako ploty, ale jejich zdi nepřesahují hranice bloku.

Dalším neobvyklým blokem jsou dveře. Ve hře budou dva druhy dveří – klasické a padací. Oba druhy dveří si můžete prohlédnout na obrázku 3.3. Klasické dveře jsou zajímavé především z toho hlediska, že se skládají ze dvou bloků – jsou dva bloky vysoké, protože téměř dva bloky je vysoká i postava hráče. U těchto dveří musí být logika hry ošetřena tak, aby nemohl nastat stav, kdy by jedna část dveří chyběla. Tedy pokud jsou dveře ve hře postaveny, musí být postaveny oba bloky zároveň, a naopak, pokud dojde ke zničení jednoho z bloků, musí být automaticky zničen i ten druhý.

Dveře mají jeden kvádr zdi, který odpovídá přímo místu, kde se dveře zobrazují. Hráč bude mít ve hře možnost tyto dveře otevírat a zavírat pomocí pravého tlačítka myši. Tím změní vlastnost bloků a tím i pozici kvádru zdi. Dveře se během otevírání, respektive zavírání animují, ale kvádr zdi se změní ihned.



Obrázek 3.3: Zleva: truhla, padací dveře, dveře

Posledním blokem, který zobrazuje obrázek 3.3, je truhla. Ta může být jednoduchá jako je na obrázku, nebo dvojitá – přes dva bloky horizontálně. Na obrázku to není patrné, ale jsou o něco menší, než je velikost bloku a při otevření se víko truhly animuje – kvádr zdi se ovšem v tomto případě nemění. V případě dvojité truhly nemusí herní logika ošetřovat zničení celé truhly naráz – při zničení půlky truhly se jednoduše druhá polovina změní na jednoblokovou truhlu.

Dále jsou ve hře i vedle vzduchu další bloky, které mají množinu zdí prázdnou. Je to např. tráva, sazenice stromu, nebo pochodeň. První dva zmíněné vyžadují pro svoji existenci pod sebou blok hlíny a opět herní logika musí zajistit, aby tento stav vždy platil. Pochodeň lze postavit na zem, nebo na stěnu. Pochodně jsou navíc zdrojem trvalého světla, o kterém více v podkapitole 3.5.

3.5 Světlo

Ve hře jsou dva druhy světla: sluneční a trvalé. Každý blok obsahuje hodnoty intenzity slunečního a trvalého světla, které mohou nabývat celočíselných hodnot v intervalu $\langle 0; 15 \rangle$. Zdrojem trvalého světla je např. pochodeň, nebo láva. Přímý zdroj slunečního světla ve hře chybí. Zdrojem slunečního světla je prakticky každý blok, který má sluneční světlo na maximální hodnotě intenzity[6] a zároveň blok o pozici výše má stejnou, nebo vyšší hodnotu. Zdroj světla je tedy definován rekurzivně bez ukončující podmínky. V praxi ale tato skutečnost nevytváří problém, protože herní logice stačí, že zkontroluje, zda je zdrojem blok o pozici výše, ale už nekontroluje, zda-li tento blok opravdu může být zdrojem slunečního světla – nedochází tedy k samotné rekurzi. Za to, že je sluneční světlo v daném světě ve všech blocích správně, zodpovídá generátor světa.

Chování světla ve hře je specifické a je značně odlišné od chování fyzického svězla. Světlo se do okolních bloků šíří tak, že vždy při přechodu mezi bloky sníží svoji hodnotu o 1, dokud neklesne až na hodnotu 0. Výjimkou je sluneční světlo, které se při maximální hodnotě šíří směrem dolů beze změny.

Do některých bloků se světlo šířit nemůže. Jedná se především o materiálové bloky v klasickém krychlovém tvaru. Do některých bloků se světlo může šířit pouze z určitých světových stran a toto platí i v opačném směru – pokud se nějaké světlo v daném bloku nachází, nemůže se touto stranou šířit dál. Např. u půlbloků se světlo nemůže šířit stranou odpovídající orientaci bloku – strana, na které se v bloku nachází materiál. Podobně je tomu u schodů, kde se světlo nemůže šířit dvěma stranami – opět se jedná o strany, u kterých je po celé ploše strany bloku materiál.

Pokud ve světě dojde ke změně nějakého bloku a tato změna má za následek, že rozprostření rozšířeného světla podle pravidel má vypadat jinak, herní logika musí zajistit přepočítaní tohoto světla.

3.6 Tekutiny

Ve hře se budou nacházet dva druhy tekutin – voda a láva. Pro tekutiny nejsou vyhrazeny speciální druhy bloků, ale tekutina může být vlastností různých bloků. Ne všechny bloky mohou obsahovat tekutinu – zda-li se může v bloku nacházet tekutina je závislé na druhu bloku. V jednom bloku se může nacházet pouze jeden druh tekutiny. Tekutina v bloku se může nacházet v různém množství. Toto množství může nabývat celočíselných hodnot v intervalu $\langle 0; 7 \rangle$. V bloku se může nacházet zdroj tekutiny, který vždy obsahuje množství 7. Pokud se v bloku nachází tekutina, která není zdrojem, jedná se o proud.

Podobně jako u světla, ani chování tekutin neodpovídá přímo chování tekutin v reálném světě. Šíření tekutin probíhá podobně, jako u světla, ale na rozdíl od světla se do této logiky navíc zapojuje gravitace. Tzn. že tekutina se nešíří do svých světových stran, ale může téct pouze v horizontálním směru, a to pouze v případě, kdy se pod tekutinou nachází pevný

[6] Liší se podle bloku, např. ve vodě je tato hodnota nižší, než mimo ní.

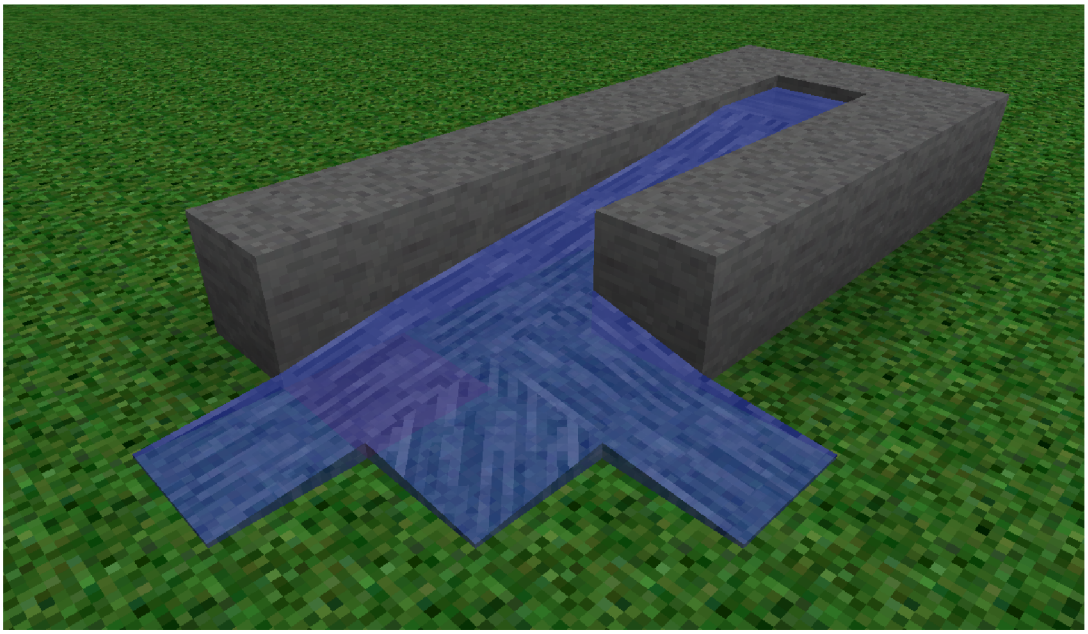
blok – blok, ve kterém se nemůže tekutina nacházet. Pokud se pod tekutinou takový blok nenachází, tekutina se nemůže šířit žádným horizontálním směrem, ale šíří se pouze směrem dolů – vytváří vodopád.

V případě, kdy se tekutina může šířit horizontálně a v některém směru se nachází v dosahu šíření díra (do které tekutina vteče a vytvoří vodopád), začne se šířit pouze tímto směrem. Pokud se díra nachází ve více směrech, začne se tekutina šířit do směru, ve kterém je tato díra blíž. Pokud se nejbližší díry nachází ve více směrech, tekutina se začne šířit všemi těmito směry.

U vody se v případě horizontálního šíření vždy v dalším bloku sníží její množství o 1, v případě lávy o 2 – tzn. že voda se může šířit do větší vzdálenosti, než-li láva. V případě vodopádu se tekutiny šíří v maximálním množství, tj. 7.

Zdroje tekutin se mohou ve světě nacházet vygenerované, nebo je může hráč vytvářet pomocí předmětu (např. kbelík s vodou). Také se mohou vytvořit automaticky při šíření tekutiny, a to v případě, kdy dojde k šíření přímo z alespoň dvou zdrojů stejné tekutiny do téhož bloku.

Pokud nastane kontakt mezi vodou a lávou, v bloku, kde nastane k takovému kontaktu, vznikne příslušný materiálový blok, a to následovně: Pokud se voda bude šířit do zdroje lávy, vznikne blok obsidiánu. Pokud se láva bude šířit do zdroje vody, vznikne blok kamene. Pokud se setká proud vody a lávy, vznikne blok kamení.



Obrázek 3.4: Voda

Na obrázku 3.4 si můžete prohlédnout grafické zobrazení vody ve hře. Výška vykreslené hladiny je závislá na množství tekutiny v bloku. Aby při vykreslení tekutiny nevznikaly schody způsobené skokovými změnami množství mezi bloky, je výška vykreslené hrany hladiny v daném bloku závislá na množství tekutiny v sousedním bloku. Tímto způsobem vznikne náklon hladiny, který je možné vidět na obrázku.

3.7 Biomy

Biom představuje ve hře jeden typ krajiny, např. louka, les, hory, poušť, oceán apod. Svět generovaný generátorem, který bude vytvořen v rámci této práce, bude rozdělen na oblasti, kde ke každé bude přiřazen určitý biom. Biomy jsou zajímavé především z pohledu generátoru, budou tedy podrobněji popsány v následující kapitole 4.

Z pohledu vlastní hry se jedná o vlastnost bloku. Jeden blok nemusí obsahovat právě jeden biom, ale může obsahovat množinu biomů, kde každý biom má definovanou svoji váhu a součet těchto vah musí být roven 1 – založeno na fuzzy logice[7]. Tato vlastnost je využita k plynulým přechodům mezi jednotlivými biomy.

Nejvíce je této vlastnosti využíváno v generátoru, což bude popsáno v kapitole 6.1. V samotné hře je využita při vykreslování. V některých biomech má vegetace a voda odlišný odstín. Např. v poušti veškerá vegetace má nažloutlý odstín, což má navozovat pocit, že je na poušti vyšší teplota a vegetaci se zde nedaří. A např. bažina má naopak tmavší odstín, což má zde navodit pocit, že jde vyšší vlhost, a navíc voda je zbarvena do zelena, aby její podoba připomínala močál. Díky fuzzy vlastnosti biomů je pak přechod odstínů mezi biomy plynulý.

Další využití biomů prozatím hra mít nebude, ale v budoucnu by se mohly určité věci v herní logice typem biomu řídit.

3.8 Entity

Entity na rozdíl od bloků nejsou zarovnány do mřížky, ale naopak se mohou nacházet na libovolné pozici, a mohou mít libovolnou velikost. Navíc se mohou v rámci světa pohybovat. Ve hře jsou prozatím tři druhy entit: hráči, *falling blocks* (česky „padající bloky“) a *dropped items* (česky „upuštěné předměty“). V budoucnu mohou být do hry přidány další druhy entit, např. NPC[8] postavy.

Vlastnosti a herní logika je závislá na druhu entity. Každá entita má obalový kvádr, který se nachází v relativní pozici vůči pozici samotné entity. Ten slouží k detekci kolize se zdmi, ovšem některé druhy entit mohou mít tyto kolize vypnuty, případně mohou být vypnuty v závislosti na aktuálním stavu entity. Určité druhy entit také mohou obsahovat množinu kvádrů zdí stejně jako bloky – pro jiné bloky tak mohou tvořit zeď.

Padající bloky

Některé materiálové bloky mají vlastnost *fallable* (schopné padat), tzn. že na tyto bloky působí gravitace. Ve hře budou tři materiálové bloky s touto vlastností: písek, štěrk a sníh.

V případě, kdy se pod tímto blokem objeví volný prostor (blok o pozici níže má množinu zdí prázdnou), žačne blok „padat“. Aby toto bylo možné, tak se tento blok transformuje na entitu – blok samotný se změní na vzduch a na jeho místě se vytvoří entita typu *falling block*, která bude po grafické stránce vypadat úplně stejně, jako původní blok, aby hráč tuto změnu nepostřehl.

Herní logika spočívá v tom, že se tato entita začne pohybovat směrem dolů a postupně se její rychlost bude zvyšovat, aby tento pohyb připomínal volný pád s gravitačním zrychlením.

[7] Fuzzy logika (česky mlhavá logika) je založena na tzv. fuzzy množinách a na rozdíl od výrokové logiky nepracuje pouze s hodnotami 0 a 1, ale na celém intervalu $\langle 0; 1 \rangle$.

[8] NPC je zkratka z anglického *non-player character* (nehráčská postava). Jedná se o postavu v počítačové hře, kterou neovládá člověk, ale je řízena algoritmicky, případně umělou inteligencí.

Jakmile se entita bude nacházet na pozici bloku, pod kterým se již nenachází volný prostor, transformuje se zpět na blok – entita se ze hry odstraní na blok na dané pozici se změní na blok této entity.

Hráči

V této kapitole jsme se již seznámili s entitou hráče, která reprezentuje herní postavu a je ovládána člověkem. Hráč je široký $0,5$ metru a vysoký $1,75$ metru, jeho obalový kvádr má tedy rozměry $0,5 \times 1,75 \times 0,5$ a je umístěn tak, že v horizontální rovině se pozice hráče nachází přesně ve středu tohoto kvádru a je o $1,5$ metru výše, než spodní stěna. Pozice entity se tedy v tomto případě nachází přibližně v pozici očí postavy. V tomto místě potom také bude umístěna virtuální kamera.

Herní logika této entity bude uzpůsobena tak, aby mohla být ovládána člověkem. Hráč se po světě bude moci pohybovat chůzi, nebo sprintem. Pokud se ocitne v místě, kde se pod ním nenachází žádná zeď, začně padat, což znamená, že se jeho rychlost pádu nastaví na kladnou hodnotu. Během pádu se pak tato hodnota postupně zvyšuje, což způsobuje vyšší rychlost – gravitační zrychlení. Hráč bude také moci skákat. Aby mohl vyskočit, musí být jeho rychlost pádu rovna 0 – tzn. že nepadá a tedy se pod ním nachází kvádr zdi (podlaha), od které se může odrazit. Výskok se provede tak, že se rychlost pádu nastaví na zápornou velikost – jedná se tedy o pád opačným směrem. Dále tento pohyb pokračuje stejně jako u pádu – rychlost pádu se zvyšuje a jakmile překročí hodnotu 0, hráč se opět začne pohybovat směrem dolů – jeho verikální pohyb během výskoku odpovídá parabole.

Předměty

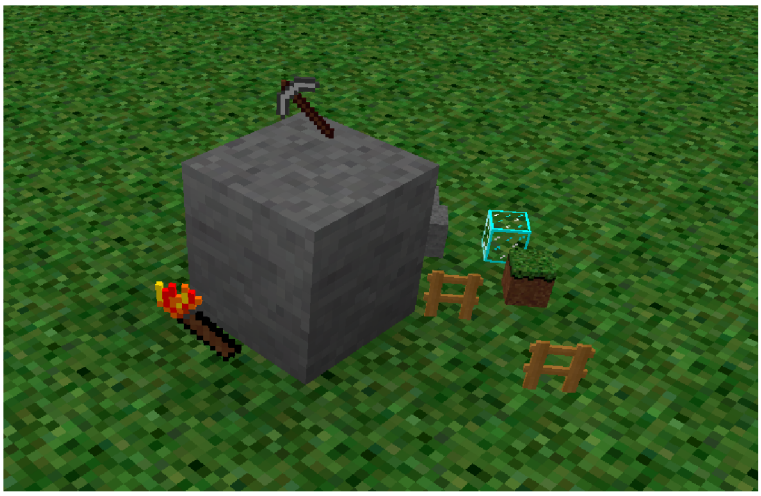
Hráč bude moci ve světě ničit (těžit) bloky. Vytěžení bloku bude určitý čas trvat v závislosti na druhu bloku. Bloky budou moci být zničeny i dalšími způsoby – např. pokud dopadne padající blok na pochodeň (pochodeň nemá žádný kvádr zdi, tudíž toto může nastat), nebo pokud hráč zničí půlku dveří – automaticky bude zničena i ta druhá, protože ve světě nemůže blok jedné půlky dveří existovat bez té druhé.

Při zničení bloku dojde k vytvoření entity *dropped item* – z bloku vypadne předmět. Druh a množství předmětu je závislý na druhu zničeného bloku – z některých bloků dokonce nemusí vypadnout žádný předmět.

Ve hře existuje velké množství druhů předmětů – stejně jako je tomu v případě bloků. Většina těchto předmětů představuje určitý druh bloku a obvykle z tohoto bloku vypadává. Některé druhy předmětů nelze umístit do světa jako bloky, jsou to např. nástroje, nebo předměty určené pouze k výrobě jiných předmětů (železný ingot, diamant, ...).

Jedna instance předmětu v sobě může obsahovat větší množství téhož předmětu. Maximální množství v jedné instanci se liší podle druhu předmětu. Instance předmětu s množstvím větším než 1 se nazývá štos (anglicky *stack*).

Entita *dropped item* představuje jednu instanci předmětu nacházející se v prostoru herního světa. Ukázku těchto entit si můžete prohlédnout na obrázku 3.5. Na tyto entity působí gravitace a při jejich vytvoření mají určitý výchozí pohybový vektor. Jiným způsobem se samy ve světě pohybovat nemohou. Pokud se v jejich blízkosti ocitne entita hráče, začnou být tímto hráčem přitahovány a jakmile se přiblíží k jeho bezprostředné blízkosti, dojde k tzv. sebrání předmětu hráčem – tzn. že se tato entita odstraní ze světa a příslušný předmět se vloží do hráčova inventáře.



Obrázek 3.5: Entity *dropped items*

Předměty, které má hráč u sebe a představují blok, může hrát použít tím, že je umístí do světa – předmět se odstraní z jeho inventáře (případně se sníží jeho množství) a ve světě se na daném místě objeví blok. Toto položení bloku je instantní bez žádné časové prodlevy.

Kapitola 4

Návrh generátoru světa

Generátor světa má za úkol vytvoření počáteční podoby světa. Tento svět následně bude využit ve hře, kde bude sloužit jako prostředí pro hráče, ve kterém se bude moci pohybovat, hledat a těžit suroviny, stavět budovy, případně upravovat vygenerovanou krajinu dle svého uvážení.

Generátor, který bude vytvořen v rámci této práce, bude generovat přírodní krajinu rozdělenou na biomy. Budou se zde nacházet vodní plochy v podobě moří a oceánů, dále pevnina v podobě ostrovů, případně kontinentů. Tato pevnina bude obsahovat nejrůznější biomy jako louky, lesy, bažiny, pouště, hory atd. Každý takový biom se bude lišit výškami a tvarem kopců, vegetací (tráva a strom), materiálem tvořící povrch atd. V podzemí se budou nacházet rozsáhlé komplexy jeskyň a budou zde také ložiska surovin, které bude moci hráč těžit.

4.1 Základní členění krajiny

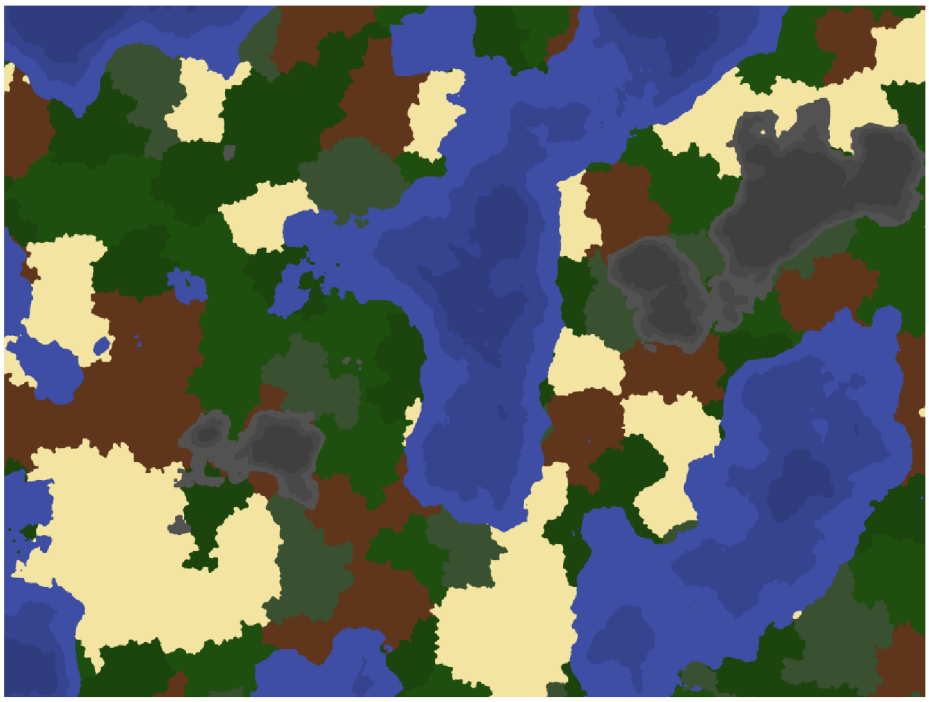
Základním členěním krajiny bude rozdělení prostoru na oblasti s oceánem a pevninou. K tomuto bude použit dvourozměrný Perlinův šum, který generuje hodnoty v intervalu $\langle -1; 1 \rangle$. Určením vhodné hodnoty pro rozdělení tohoto intervalu na dvě části určíme množství pevninské a vodní plochy. Potom na všech pozicích ve světě, u kterých hodnota z daného Perlinova šumu spadá do první části intervalu, se bude nacházet moře, a na zbylých pevnina.

Dále bude pevnina i oceán dále dělen na oblasti, které budou reprezentovat jednotlivé biomy – ty již byly uvedeny v kapitole 3.7. U oceánu se bude jednat o tři biomy, a to: moře, oceán a hluboký oceán. Tyto tři biomy se budou lišit vzdáleností od pevniny a hloubkou mořského dna – moře bude bezprostředně sousedit s pevninou a bude mělké, hluboký oceán bude naopak ve velké vzdálenosti od pevniny a bude hluboký. Podmínku vzdálenosti snadno splníme, pokud pro oddělení těchto tří biomů využijeme stejný šum, který jsme již použili pro oddělení pevniny od oceánů, a to tak, že část intervalu představující oceán rozdělíme na další tři části. Perlinův šum je koherentní, což nám zajistí, že moře bude ležet vždy mezi pevninou a oceánem, a stejně, že oceán bude ležet mezi mořem a hlubokým oceánem.

Pevnina se bude dělit na štítové hory a další biomy. Štítové hory se budou nacházet ve vnitrozemí. Ke splnění této podmínky opět můžeme použít stejný Perlinův šum, ve kterém nyní rozdělíme část intervalu náležícího pevnině. Štítové hory se budou také dělit na tři biomy: nízké, střední a vysoké. Toto rozdělení provedeme totožným způsobem jako u oceánů.

Zbylé pevninské biomy se již nebudou rozdělovat tímto Perlinovým šumem, protože jejich pozice nebude závislá na pozici pevniny a oceánu. Pro rozdělení těchto biomů použijeme Voroného diagram, který dělí prostor na souvislé oblasti. Při rovnoměrném rozmístění řídících bodů jsou navíc tyto oblasti podobně velké. Nevýhodou tohoto diagramu je, že tvar těchto oblastí představují konvexní polygony, což není příliš vhodný tvar pro oddělení různých typů krajin, protože vypadá příliš uměle.

Tento problém lze vyřešit např. použitím více Voroného diagramů s různými měřítky. Nejdříve rozdělíme prostor diagramem velmi hustým rozmístěním řídících bodů, přičemž hustotu zvolíme takovou, aby se při aplikaci na mřížku bloků ztratila pravidelnost polygonů tvořící jednotlivé oblasti. Následně když budeme pro určitou pozici počítat, ve kterém biomu se nachází, nejdříve vypočteme, v jaké oblasti se nachází u tohoto diagramu s největší hustotou. Pro další výpočet místo dané pozice použijeme pozici řídícího bodu této oblasti a budeme pokračovat diagramem s nižší hustotou. Takto iterativně provedeme výpočet přes všechny diagramy, až se dostaneme k diagramu s nejnižší hustotou. Pro každou oblast v tomto diagramu předem vygenerujeme, který biom se zde bude nacházet a podle vypočtené oblasti přiřadíme daný biom původní pozici, pro který jsme prováděli výpočet.



Obrázek 4.1: Rozdělení prostoru na biomy

Na obrázku 4.1 se nachází ukázka rozdělení krajiny na jednotlivé biomy způsobem, který byl popsán v této podkapitole.

Pro účely využití v dalších částech generátoru bude pro rozmístění biomů použit filtr *blur*[1] za účelem zjemnění přechodů mezi biomy. To nám zajistí plynulou změnu typu krajiny mezi jednotlivými biomy. Tento filtr bude aplikován několikrát s různým poloměrem, protože každá část generátoru bude potřebovat různý stupeň velikosti přechodů.

[1]Filtr *blur* provede zjemnění přechodů mezi hodnotami v mřížce – výsledek bude rozostřený.

4.2 Terén

V této podkapitole bude popsán způsob generování tvaru povrchu – tzn. kopce, skály, mořské dno apod. Každý biom má definovanou svoji počáteční výšku. Aby nedošlo ke skokovým změnám výšky mezi biomy, je využito jemných přechodů mezi biomy vytvořených filtrem *blur*.

Pro základní zvlnění povrchu bude opět použit dvourozměrný Perlinův šum. Jeho amplituda je rovna 1. Pro každý biom bude definován koeficient, kterým budou hodnoty tohoto šumu násobeny za účelem změny amplitudy – změny výšky kopců. Opět, aby na hranicích biomů nedocházelo ke skokovým změnám, je využito jemných přechodů tak, že koeficient v dané pozici se vypočte váženým průměrem, kde jako váhy se použijí váhy jednotlivých biomů. Takto vypočtená hodnota se přičte k počáteční výšce a pro většinu biomů již představuje finální výšku.

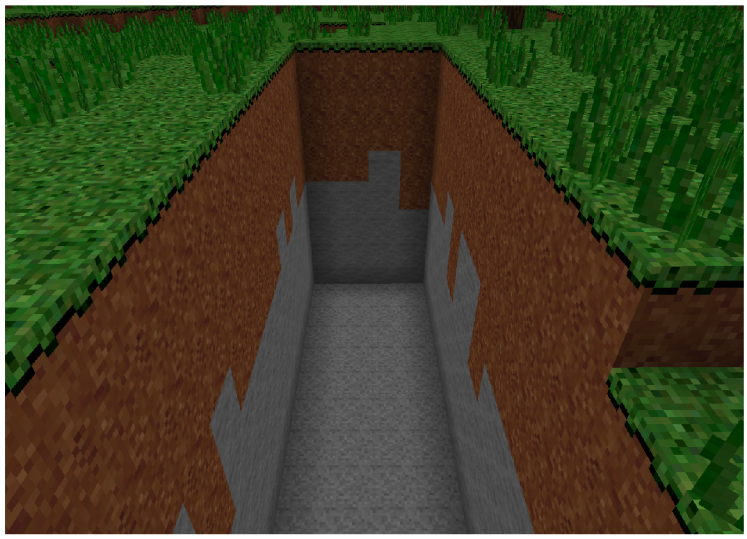
Pro generování hřebenů ve štítových horách bude použit Voroného diagram, a to tak, že výška v daném bodě bude přímo úměrná vzdálenosti od nejbližšího řídícího bodu [8]. Problém s pravidelností oblastí v tomto diagramu bude tentokrát vyřešen pomocí dvourozměrného Perlinova šumu, ze kterého se nyní použijí hodnoty gradientů, které budou interpolovány stejným způsobem, jako u hodnotového šumu. Tyto gradienty vynásobené vhodnou konstantou budou složit k horizontální změně pozice, ve které se následně bude počítat výška pomocí Voroného diagramu. Tato úprava pozice nám způsobí zakřivení hranic mezi jednotlivými oblastmi diagramu – tedy hřebeny hor. Takto vypočtenou hodnotu výšky dále přičteme k počáteční výšce a výšce kopců vypočtenou Perlinovým šumem.

Dalším zvláštním biomem s rozdílným terénem budou skalnaté hory. Zde bude použit trojrozměrný Perlinův šum, který nyní nebude udávat výškovou mapu, ale země a vzduch budou odděleny rozdělením intervalu hodnot tohoto šumu stejně, jako v případě rozdělení pevniny a oceánů v podkapitole 4.1. Pro omezení zdola a zhora se bude hodnota rozdělující tento interval měnit v závisloti na výšce, a to tak, že s výškou se bude zvětšovat část intervalu představující vzduch a zmenšovat část intervalu představující zemi. A naopak v druhém směru se bude zvětšovat část intervalu představující zemi a zmenšovat část představující vzduch. Tímto vytvoříme podobný povrch, jako v případě dvourozměrného Perlinova šumu, ale s tím rozdílem, že nyní se v tomto terénu mohou vyskytovat skály a převisy, které nemohou vzniknou ve výškové mapě. Z důvodu návaznosti na terén v okolních biomech je výška, do které je třírozměrný šum omezen, násobena váhou biomu skalnatých hor, což způsobí na okrajích tohoto biomu postupné vymizení skal a převisů, což zajistí plynulý přechod.

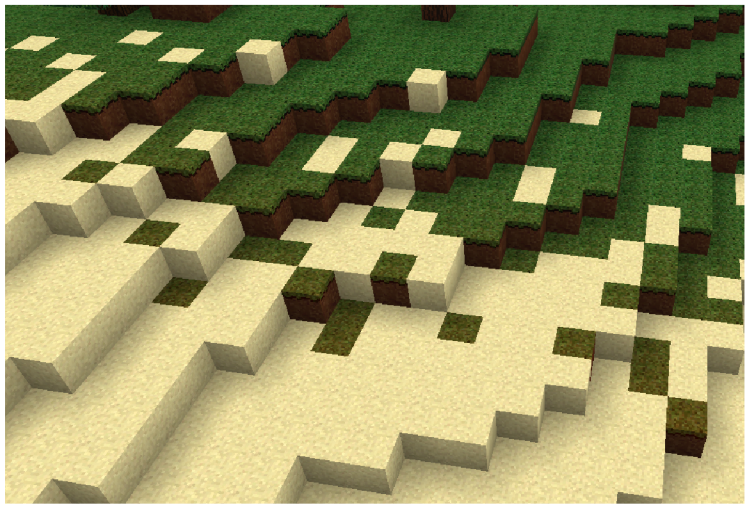
4.3 Povrch

Jako základ pro terén popsaný v předchozí podkapitole bude použit blok kamene, který bude sloužit jako materiál především pro podzemí. Na povrchu se pak budou generovat další bloky v závislosti na konkrétním biomu – na louce hlína s trávou, na poušti písek apod.

Každý biom bude mít vlastní sekvenci bloků, které se budou nacházet těsně pod povrchem – např. u louky bude nejblíže povrchu blok hlíny s trávou a pod ním několik bloků hlíny s tím, že u pár spodních bude určitá pravděpodobnost, jestli na té pozici opravdu budou, nebo tam místo nich bude kámen. Tato pravděpodobnost se bude pro každou pozici počítat zvlášť. Tímto se docílí nepravidelnosti, kterou si můžete prohlédnout na obrázku 4.2.



Obrázek 4.2: Snímek ze hry znázorňující průřez povrchu v biomu louky



Obrázek 4.3: Přechod mezi písečnými a travnatými bloky na okraji pouště

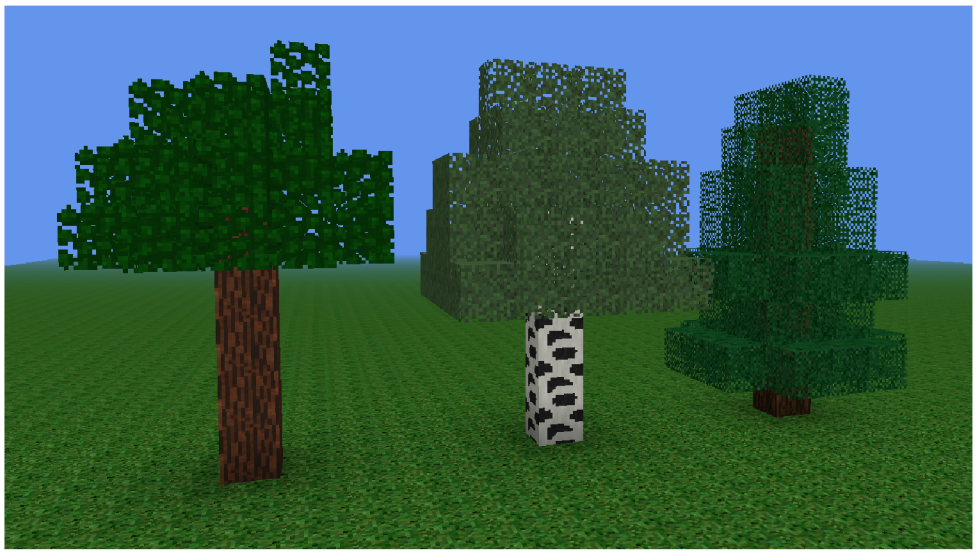
V případě skalnatých hor nastává u převisů situace, kdy se nad sebou vyskytuje 2 a více povrchů. Tato sekvence bude v tomto případě počítána pro každý povrch zvlášť.

Pro každý biom bude zpravidla definována jedna taková sekvence. Biomy štítových hor budou mít těchto sekvencí několik a budou závislé na výšce povrchu – v nejnižších pozicích bude tráva, která se později bude měnit ve štěrk a u vrcholků pak v kámen. Aby nedocházelo k nevzhlednému rovnému přechodu mezi těmito povrchy v určité výšce, než se pro danou pozici vypočte konkrétní sekvence bloků podle výšky povrchu, k této výšce se přičte náhodná hodnota za účelem vytvoření nepravidelnosti tohoto přechodu, podobně jako je tomu u spodních bloků těchto sekvencí.

Mezi jednotlivými biomy obvykle bude docházet k ostrému přechodu mezi těmito sekvencemi bloků. Jelikož u většiny biomů se bude na povrchu nacházet blok hlíny s trávou, nebudou tyto přechody příliš znatelné. V případě pouštního biomu, u kterého se na povrchu nachází bloky písku, by docházelo k ostrému přechodu mezi bloky hlíny s trávou a

bloky písku. Proto v tomto případě je pro zjemnění tohoto přechodu opět použita náhoda, a to tak, že u pouštního biomu se použije sekvence tohoto biomu pouze v případě, kdy generátor náhodných čísel vygeneruje číslo v intervalu $\langle 0; 1)$ nižší, než je váha tohoto biomu v dané pozici. Díky tomuto opět dojde k „rozbití“ tohoto přechodu. Tento případ zachycuje obrázek 4.3.

Na povrchu se bude dále generovat vegetace – tráva a stromy. Trávu představuje jeden blok s prázdnou množinou zdí – jedná se tedy především o dekorativní prvek. Tento blok si můžete prohlédnout na obrázku 4.2, kde se nachází po okrajích díry. Tráva bude náhodně rozmístěna s tím, že pro její rozmístění se použije dvourozměrný Perlinův šum. Hodnota vypočtená z tohoto šumu se použije pro určení pravděpodobnosti, zda se na určité pozici bude tráva nacházet nebo ne – díky tomu vzniknou oblasti s vyšší a naopak oblasti s nižší hustotou trávy. Tato pravděpodobnost bude také dále upravována podle daného biomu – např. na louce bude podstatně více trávy, než na horách. Blok trávy vyžaduje, aby na pozici pod ním se nacházel blok hlíny, případně blok hlíny s trávou. Generátor tedy nejdřív musí zkontrolovat, že tam tento blok opravdu je, případně jestli ho tam v budoucnu opravdu vygeneruje.



Obrázek 4.4: Stromy, zleva: dub, bříza, smrk

Stromy se budou skládat z většího počtu bloků a budou celkem tři druhy stromů: duby, břízy a smrky. Každý druh stromu se bude skládat ze dvou druhů bloků – kmen a listí. Kmen stromu bude představovat sloupec bloků kmenu a koruna se bude skládat z bloků listí. Koruna bude mít tvar paraboloidu orientovaného směrem dolů a omezeného ze spodu tak, aby koruna stromu nesahala až dolů. Prvek náhody zde bude použit pro výšku kmene, rozměry koruny a její výšky od země (spodního konce kmene). Dále bude náhoda použita pro tvar koruny, aby vzhledem k jejímu paraboloidnímu tvaru nevypadala příliš pravidelně. To bude dosaženo tím, že na pozicích bloků, které se budou nacházet na okraji daného paraboloidu, budou umístěny bloky listí podle pravděpodobnosti, z jak velké části tyto bloky zasahují do paraboloidu. U smrků navíc bude od spodu každá druhá vrstva koruny užší, aby celkový tvar koruny více připomínal tvar koruny smrku. Ukázka všech tří druhů stromů se nachází na obrázku 4.4.

Na vrcholcích štítových hor se budou generovat bloky sněhu. Blok sněhu může mít jednu z osmi výšek, přičemž ta nejvyšší odpovídá výšce celého bloku. Sníh bude v horách generován náhodně tak, že s výškou bude stoupat pravděpodobnost výskytu a větší výšky sněhu.

4.4 Vodní plochy

Voda se bude generovat do výšky o y-ové hodnotě 0 nad povrchem, jehož generování popisuje podkapitola 4.3, a bude představovat moře, oceány, případně jezera. Pod výškou 0 se budou nacházet podzemní komplexy, ve kterých se tímto způsobem voda generovat nebude. Je ale možné, že z těchto vodních ploch se voda do podzemí rozteče po spuštění vygenerovaného světa ve hře, protože podzemní komplexy budou moci ústit na povrchu, a to i pod hladinou. Více o podzemí popisuje podkapitola 4.6.

4.5 Suroviny v podzemí

V podzemí se budou nacházet suroviny, např. uhlí, železo, zlato apod. Tyto suroviny budou reprezentovány příslušnými bloky, které když hráč vytěží, může z nich získat požadovanou surovinu. Bloky jedné suroviny se budou ve světe vyskytovat v shlucích nazývaných ložiska. Ložiska budou generována tak, že nejdříve se náhodně určí jejich pozice, do které se umístí první blok. Od tohoto počátečního bloku budou generovány další sousední bloky. Generování bude probíhat tak, že vždy se náhodně zvolí blok z bloků, které jsou již vygenrovány, a od kterého se bude generovat nový blok. Následně se náhodně zvolí strana, na které se nový blok vygeneruje, pokud jeho vygenerování bude na dané pozici možné – pokud se např. na této pozici nebude vyskytovat jeskyně, povrch, nebo jiná překážka. Budou definované určité pravděpodobnosti, kdy se bude moci sousední blok vygenerovat tak, aby s předchozím blokem sousedil buď stěnou, hranou, nebo pouze rohem.

4.6 Jeskyně v podzemí

Generovat se budou tři druhy jeskyní: chodby, dómy a trhliny. Chodby budou tunely o průměru několika bloků a dlouhé několikset bloků. Některé budou mít svůj počátek na povrchu a některé budou zcela pod povrchem. Generovat se budou tak, že se nejprve náhodně zvolí počáteční bod a počáteční vektor. Generování trasy chodby pak bude probíhat iterativně tak, že se vždy vezme předchozí vektor, normalizuje se, vynásobí definovanou délkou kroku (průměrná vzdálenost mezi dvěma body trasy) a přičte se k němu náhodný vektor, čímž docílíme náhodného zatáčení chodby libovolným směrem. Bude definována určitá pravděpodobnost, kdy se z jednoho bodu budou moci vygenerovat dva následující vektory současně – tím dojde k rozvětvení chodby a tvar celé jeskyně pak bude připomínat blesk. A také bude definována další pravděpodobnost, že se z daného bodu již další vektor nevygeneruje – tím se chodba ukončí.

Pro vygenerování chodby se mezi jednotlivými body vytvoří válce a v samotných bodech koule o průměru odpovídajícímu požadované šířky chodby. Uvnitř těchto těles se pak budou generovat bloky vzduchu, čímž se ve světě vytvoří jeskyně. Aby taková jeskyně neměla po celé délce stejnou šířku, náhodně se vygeneruje šířka jeskyně na začátku a jiná, užší na konci. Potom se v každém bodě trasy vypočte šířka jeskyně tak, aby se od začátku postupně

zužovala až na hodnotu u konce. V důsledku toho se válce mezi jednotlivými body změní na komolé kužele.

I přes postupné zužování chodby a její nahodné kroucení bude takto generovaná jeskyně stále vypadat příliš pravidelně. Aby se do tvaru jeskyně přidala další nepravidelnost, bude použit Perlinův šum. Ten bude udávat průměr chodby podle dvou rozměrů - vzdálenosti od začátku jeskyně a směru ke stěně v průřezu chodby – čímž docílíme toho, že průřez jeskyně nebude kruhový a navíc se bude postupně jeho tvar měnit. Šum, který pro tento účel potřebujeme, musí mít dva rozměry s tím, že rozměr, který se použije pro směr, se musí periodicky opakovat, aby v určitém místě nevzniknul šev. Šum s touto vlastní můžeme vytvořit tak, že nejdříve vygenerujeme trojrozměrný šum. V prostoru tohoto šumu pak vytvoříme válec a plášť tohoto válce použijeme jako dvourozměrný šum, který nyní bude mít požadované vlastnosti.

Dalším druhem jeskyně bude jeskynní dóm. Bude se jednat o vetší prostor v podzemí, ve kterém se navíc budou vyskytovat krápníky. Tvar dómu bude generován z elipsoidu. K vytvoření nepravidelnosti tvaru bude použit Perlinův šum obdobným způsobem jako u chodeb, akorát s tím rozdílem, že nyní nebude dvourozměrný šum vytvářen z trojrozměrného pomocí pláště válce, ale pomocí pláště koule.

V dómu se budou generovat tři druhy krápníků: stalagmit (rostoucí od spodu), stalaktit (rostoucí od zhora) a stalagnát (spojený stalagmit a stalaktit). Krápníky budou generovány pomocí paraboloidů orientovaných podle druhu krápníku. Krápníky budou úzké (u svého kořene cca 2-4 bloky), proto když nebudou umístěny na celočíselné souřadnice, není nutné jejich tvar znepravidelňovat jiným způsobem, než mapováním na celočíselnou mřížku bloků.

Posledním druhem jeskyň budou trhliny. Ty budou představovat vertikální pukliny, např. vzniklé po zemětřesení, a mohou se buď nacházet zcela v podzemí, nebo i na povrchu. Pro jejich generování se nejprve vygeneruje trasa trhliny, podobně jako v případě chodeb, ale nyní pouze dvourozměrná v horizontální rovině a bez větvení. Také stejně jako u chodeb zde bude proměnlivá šířka trhliny, ale zde bude nejširší část uprostřed a nejužší na koncích.

Pro omezení zdola a zhora budou použity dvourozměrné Perlinovy šumy, pomocí kterých bude vygenerována výšková mapa. Prostor mezi podlahou a stropem se navíc bude směrem ke krajům parabolicky snižovat. Stěny budou náhodně posunuty kolmým směrem na rovinu trhliny, kde rozdíl, o který bude trhlina v daném bodě posunuta, bude určovat třetí dvourozměrný Perlinův šum.

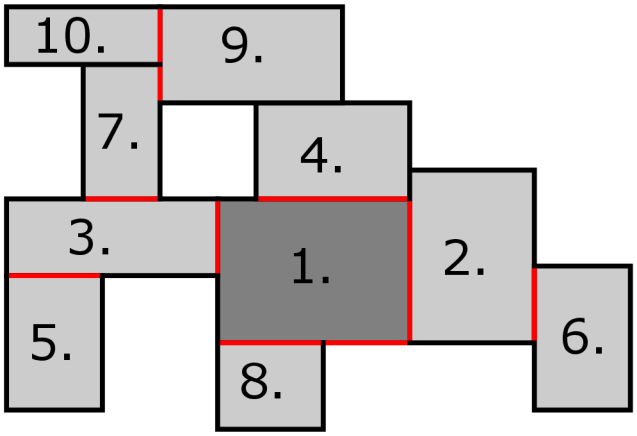
Jeskynní chodby se budou generovat z povrchu s tím, že jejich počáteční vektory budou směřovat dolů s určitou mírou náhodnosti. Bude určitá pravděpobnost, kdy se na konci některé větve chodby vygeneruje jeskynní dóm. Jeskynní dómy se budou v podzemí generovat i samostatně na náhodných pozicích. Z libovolného dómu se pak s určitou pravděpodobností může vygenerovat další chodba náhodným směrem – tentokrát i nahoru. Trhliny budou generovány na náhodných pozicích nezávisle na ostatních jeskynních. Kromě těchto propojení bude často docházet i k náhodným protnutím jednotlivých jeskynní, což je dáno způsobem generování.

V jeskynních dómech se také bude moci generovat s určitou pravděpodobností voda nebo láva, která se následně může roztéct do dalších jeskynní. Se vzrůstající hloubkou bude vyšší pravděpodobnost, že místo vody se vygeneruje právě láva.

4.7 Budovy

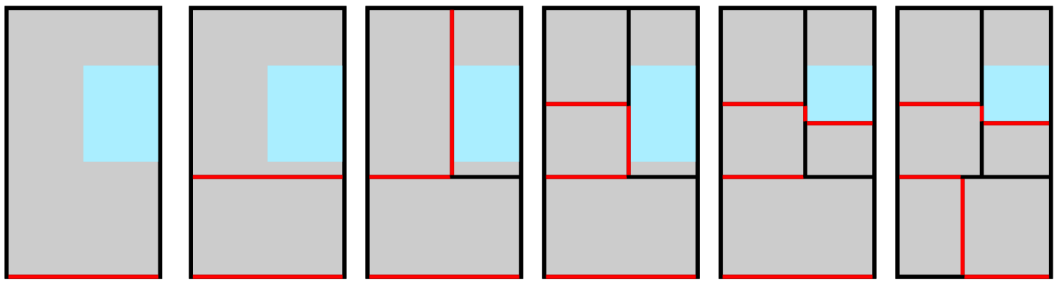
V krajině se budou na náhodných pozicích generovat různě velké budovy. Větší budovy mohou mít netriviální půdorys skládající se z navzájem se dotýkajících obdelníků (viz ob-

rázek 4.5) – jeden takový obdelník představuje tzv. blok budovy. Budovy mohou mít více pater a tento počet se může v různých blocích lišit. Na vrcholu každého bloku se bude nacházet střecha. Každé patro každého bloku bude rozděleno na místnosti. Toto dělení bude nezávislé na ostatních patrech. Ve vnějších zdech se budou generovat okna a ve vnitřních zdech se budou generovat dveře a průchody mezi místnostmi. Ve vnějších zdech přízemí se budou generovat vchodové dveře a každá budova bude mít vždy alespoň jedny. Dále se na vnitřních stranách zdí budou generovat pochodně. Mezi jednotlivými patry se budou generovat schodiště. Dveře, průchody a schodiště bude generované tak, aby vždy existovala cesta mezi vchodovými dveřmi a libovolnou místností v budově – v budově se nebude nacházet žádná izolovaná místnost, do které by neexistoval přístup.



Obrázek 4.5: Demonstrace skládání půdorysu budovy z jednotlivých obdelníků

Budovy budou generovány pomocí otevřených L-systémů – viz kapitola 2.5. Počáteční řetězec bude obsahovat jeden blok budovy. Pomocí pravidel se k tomuto bloku budou přidávat sousední bloky, čímž se bude zvětšovat půdorys budovy. Na vrcholu každého bloku se buď vygeneruje další patro, nebo střecha. V každém patře každého bloku se vygeneruje jedna místnost, která bude zabírat celý prostor bloku. Každá místnost se může rozdělit na dvě vytvořením zdi uprostřed původní místnosti [4]. V každé místnosti se vygenerují její obvodové zdi, podlaha a strop. Pro každé dvě sousední místnosti bude existovat pouze jedna zeď. Ve zdech se budou generovat dveře, průchody, okna a pochodně podle již zmíněných podmínek. V podlahách se budou generovat schodiště – s výjimkou podlah v přízemí.



Obrázek 4.6: Demonstrace dělení místností

Aby byla zajištěna podmínka, že do každé místnosti bude existovat cesta od vchodových dveří, budou během generování dodržovány následující principy. Alespoň jedna zeď každé místnosti bude mít tu vlastnost, že se v ní povinně vygenerují dveře, nebo průchod. Místnost prvotního bloku, který se bude nacházet v počátečním řetězci L-systému, bude mít tuto zeď zvolenou náhodně. V každém novém bloku, který se připojí k půdorysu budovy, se určí jedna zeď, která se dotýká jiného bloku, a této zdi se nastaví povinnost dveří. Demonstrace tohoto postupu se nachází na obrázku 4.5, na kterém jsou červeně zvýrazněné zdi, ve kterých se budou povinně generovat dveře. Stejný princip bude aplikován i při dělení místností, viz obrázek 4.6.

Po přidání patra některému bloku se podobným způsobem přidá podlaze povinnost schodiště do nižšího patra. Pokud některý sousední blok má stejné patro, které se právě vygenerovalo, místo schodiště může dojít k propojení přes zeď, stejně jako v přízemí. Dělením místnosti se místo v podlaze postupně zmenšuje, viz obrázek 4.6, kde místo, na kterém se bude povinně generovat schodiště, je vyznačeno modře. Počáteční velikost a pozice je dána dělením místností v nižším patře. Takto se postupuje proto, aby se schodiště negenerovalo skrz některou zeď.

Při dodržení těchto principů je možné sestrojit strom obsahující všechny místnosti, kde v kořeni se bude nacházet místnost s povinnými vchodovými dveřmi a hrana bude spojovat vždy dvě místnosti, mezi kterými jsou povinné dveře, nebo schodiště. Samozřejmě se dveře mohou generovat i v ostatních zdech a schodiště v ostatních podlahách – ve výsledku se tedy bude jednat o souvislý graf.

Budova se umístí do krajiny tak, že pro prvotní blok se nalezne prostor s relativně rovným povrchem, aby rozdíl minimální a maximální výšky povrchu krajiny v prostoru tohoto bloku nepřesáhl určitou hranici. Při aplikaci pravidla pro rozšíření půdorysu budovy bude též kontrolována výška povrchu. Pokud se bude příliš lišit od výšky počátečního bloku, na vybraném místě se nový blok nevygeneruje. Mezi spodním okrajem budovy a povrchem může na některých místech vzniknout volný prostor. Tento prostor bude zaplněn kamennými bloky představující základy budovy. Dále se může stát, že se některé vchodové dvěře budou nacházet nad nebo pod úrovní okolního terénu. V takových případech bude doplněno schodiště, které bude končit dosažením úrovně terénu.

Kapitola 5

Implementace hry

Hra navržená v kapitole 3 byla implementována v programovacím jazyce C++ ve standardu verze C++17. Vedle standardní knihovny jazyka C++ byla dále využita sada knihoven Boost, konkrétně knihovny Asio, Bimap, Container, DLL, Filesystem a Program Options. Pro grafický výstup byly využity knihovny OpenGL, GLEW, SDL a SDL_image. Vývoj hry byl zaměřen především pro operační systém Microsoft Windows. Knihovny ale byly zvoleny tak, aby zdrojové kódy hry byly nezávislé na platformě. Implementovaná hra dostala jméno *Sixteenia*.

5.1 Základní členění

Základem je knihovna SixteeniaCore, na které jsou jsou závislé všechny ostatní knihovny a spustitelné soubory. Obsahuje jádro hry (backend), nástroje pro ukládání hry, synchronizaci hry po síti, základní nástroje pro generátory světa a další pomocné nástroje. Tato knihovna je závislá pouze na knihovnách Boost. Jádro hry je popsáno v podkapitole 5.2.

Grafická část hry je obsažena ve spustitelném souboru SixteeniaClient. Tento soubor představuje hlavní uživatelské rozhraní. Veškerá grafika je vykreslována pomocí OpenGL. Knihovna SDL zde slouží pouze pro vytvoření okna aplikace, pro vstup z klávesnice a myši, apod. Nachází se zde kód pro vykreslení hry a její ovládání hráčem (frontend). Implementace klienta je popsána v podkapitole 5.3.

Dalším spustitelným souborem je SixteeniaServer. Jedná se o konzolovou aplikaci, která primárně slouží ke spuštění serveru hry, ke které je možné se následně připojit přes síť z klienta (soubor SixteeniaClient). Dále je možné pomocí této aplikace spravovat uložené světy. Tato aplikace může běžet samostatně na počítači, který nemá grafický výstup (typicky server), protože vyžaduje pouze SixteeniaCore a knihovny Boost. Tato aplikace není pro tuto práci podstatná, proto nebude podrobněji popsána.

Posledním spustitelným souborem je SixteeniaMapMaker, který slouží především jako pomocný nástroj při vývoji generátorů světa. Jedná se také o konzolovou aplikaci. Slouží pro generování map (bitmapových obrázků), na kterých je vykreslen vygenerovaný svět z ptačí perspektivy. Generátor map je popsán v podkapitole 5.4.

Generátory světa jsou řešeny jako zásuvné moduly – dynamicky linkované knihovny[1]. Více o implementaci generátorů světa se nachází v kapitole 6.

[1] Dynamicky linkované knihovny jsou knihovny, které se linkují až za běhu aplikace. Na systémech Microsoft Windows se jedná o soubory DLL, na Linuxu to jsou DSO atd.

5.2 Jádro hry

V této kapitole budou popsané základní stavební bloky jádra hry. Hra je velmi rozsáhlá a obsahuje obrovské množství kódu, proto zde budou zmíněny jen základy pro nastínění základního principu fungování samotné hry.

Jádro hry obsahuje velké množství dalších tříd, které v této kapitole nebudou popsány. Jedná se o pomocné třídy pro práci s geometrií, pro rozdělování úloh mezi vlákna, pomocné třídy a algoritmy pro ukládání světa na disk, pomocné třídy pro komunikaci klienta a serveru a mnoho dalšího. Tato práce se zaměřuje na procedurální generování, proto zde nejsou popsány, jelikož jsou to témata nad rámec této práce.

Bloky

Jak již bylo řečeno v kapitole 3.2, základním prvkem hry je tzv. blok. Block představuje struktura `Block`. Veškerá data bloku jsou uložena na 32 bitech. Takto úsporná velikost struktury byla zvolena proto, že v jeden okamžik se v paměti může nacházet řádově desítky až stovky milionů bloků (načtené okolí hráče), proto by ve struktuře každý bajt navíc představoval podstatné navýšení paměťové náročnosti hry. I přesto, že hra obsahuje velké množství druhů bloků, nebyl pro bloky použit polymorfismus z OOP[2], a to z důvodu, aby bylo možné bloky uchovávat v trojrozměrných polích bez nutnosti alokace dalších dat na haldě (v takovém případě by už samotné ukazatele vyžadovaly velký kus paměti).

bity	**Význam**
0-12	druh bloku
13	jednobitová hodnota (přírodní/otevřeno/aktivní)
13-18	propojení s okolními bloky (1 bit pro každou stranu)
14-15	druh rohu u schodů (žádný/vnitřní/vnější)
16-18	orientace bloku (1 ze 6 stran)
16-18	tříbitová hodnota (např. výška sněhu)
19	vertikální otočení
19	druh tekutiny
20-22	množství tekutiny v intervalu $\langle 0; 7 \rangle$
23	označení zdroje tekutiny
24-27	hodnota slunečního světla v intervalu $\langle 0; 15 \rangle$
28-31	hodnota trvalého světla v intervalu $\langle 0; 15 \rangle$

Tabulka 5.1: Data ve struktuře bloku

Tabulka 5.1 obsahuje přehled mapování jednotlivých vlastností bloku na bity ve struktuře. Data každého druhu bloku obsahují druh bloku a hodnoty slunečního a trvalého světla. Existence ostatních vlastností je závislá na druhu daného bloku. Některé vlastnosti se sice v tabulce překrývají, ale ve hře neexistuje žádný druh bloku, který by měl takové vlastnosti, že by se překrývaly.

Struktura obsahuje metody pro čtení a změnu těchto vlastností. Bloky obsahují mnoho dalších vlastností, které ale nejsou uložené v datech, protože jsou závislé na daném druhu bloku – např. kvádry zdí, propustnost světla, tvrdost, možnost zatopení tekutinou atd.

[2]OOP – objektově orientované programování

Svět v této hře je opravdu obrovský. Pro souřadnice bloku je použit 32-bitový integer, tzn. že celý svět je krychle se stranou o velikosti přes 4 miliony km. Proto není nutné, ani technicky možné (např. paměťové nároky), aby během hraní byl v paměti celý svět – vždy je načteno pouze okolí místa, kde se nachází hráč. Aby bylo toto postupné načítání světa možné, je celý svět rozdělen na sektory – jeden sektor představuje třída `Chunk`. Ta představuje krychli o velikosti $32 \times 32 \times 32$ bloků, které jsou zde uloženy v třírozměrném poli. Ve třídě `Chunk` se nenacházejí pouze bloky, ale i všechny další objekty náležící danému sektoru světa, jako např. entity a další data, která ještě budou popsána.

Pro uložení aktuálně načtených objektů třídy `Chunk` slouží třída `ChunkWarehouse`. Struktura této třídy byla inspirována hierarchickým uspořádáním stránek v operační paměti. Nachází se zde tři úrovně tabulek. V nejvyšší úrovni se nachází slovník (objekt třídy `std::unordered_map`), na dalších třírozměrná pole o velikosti $16 \times 16 \times 16$. Tato struktura byla zvolena pro rychlý náhodný přístup k libovolnému objektu třídy `Chunk` nebo bloku. Pole byla zvolena pro konstantní složitost přístupu. Slovník implementovaný hašovací tabulkou má též v ideálním případě konstantní složitost přístupu, ale testováním bylo ověřeno, že tato struktura je rychlejší, než kdyby byly objekty třídy `Chunk` vkládány do slovníku přímo. Jedna položka slovníku je schopná obsáhnout krychli o velikosti strany 8192 bloků, což je více jak dvojnásobek maximální dohlednosti hráče (maximální vzdálenost načtených objektů třídy `Chunk` od pozice hráče). Z toho důvodu ve slovníku bude vždy maximálně 8 položek, tudíž zde nebude docházet k tolika kolizím hašů jako v případě, kdy by ve slovníku byly objekty přímo. Dále byla tato struktura navržena tak, aby bylo možné objekty postupně přidávat a odebírat v závislosti na postupném načítání okolí hráče a jeho pohybu po světě. To je důvod, proč je na nejvyšší úrovni slovník. Kdyby bylo na nejvyšší úrovni pole, muselo by být příliš velké, aby obsáhlo celý svět. Nebo by zde muselo být více úrovní, což by bylo naprosto zbytečné vzhledem k dohlednosti hráče.

O správu načtených objektů třídy `Chunk` se stará třída `ChunkManager`. Uchovává v sobě informace o pozicích jednotlivých hráčů (pokud hra běží na serveru, může se k ní připojit v jednu chvíli více hráčů) a jejich dohledností (podle jejich nastavení). K tomu slouží třída `ChunkSurroundings`, ve které je pozice postupně aktualizována podle pozice hráče. Objekty třídy `Chunk` jsou vytvářeny asynchronně ve vedlejších vláknech programu. Nové jsou generovány generátorem světa, dříve vygenerované jsou načítány z disku a při síťové hře jsou všechny stahovány ze serveru. Všechny tři způsoby jsou časově náročné a tudíž tyto operace není možné provádět v hlavním vlákně. Různé způsoby načítání objektů je schované za rozhraním, které představuje třída `Distributor`. Tato třída obsahuje metodu `request`, pomocí které se vyžádá načtení objektu třídy `Chunk`, a metodu `take`, která vrátí kolekci již načtených objektů z předchozích požadavků.

Třída `ChunkManager` obsahuje metodu `update`, která je periodicky volána. Obsahuje tři hlavní kroky. Prvním krokem je získání načtených objektů z předchozích volání metody a vložení těchto objektů do objektu třídy `ChunkWarehouse`. Druhým krokem je získání pozic, pro které je nutné načíst objekty třídy `Chunk`. Posledním krokem je kontrola, zda některé objekty se již nenachází mimo dohled všech hráčů – v takovém případě dojde k odstranění z paměti. Z časových důvodů se při každém volání zkontroluje pouze část objektů. Odstranění objektu z paměti tedy může nějaký čas trvat, ale vždy k němu nakonec dojde (pokud se mezitím k němu některý z hráčů opět nepřiblíží).

Objekty třídy `Chunk` jsou načítány v takovém pořadí, že nejdříve jsou načteny ty, které se nachází blíže k pozici hráče. Tím je zajištěn efekt, že hráč ihned uvidí své bezprostřední okolí a postupně se mu objevuje krajina kolem něj, dokud není načtena celá oblast podle jeho nastavené dohlednosti. K tomuto účelu je nutné seřazení pozic objektů, které se nacházejí

v okolí hráče. Tento výpočet se provádí ve třídě `ChunkSurroundings`. Jelikož se v okolí hráče mohou nacházet řádově desítky tisíc objektů třídy `Chunk`, je tento výpočet poměrně náročný, proto se neprovádí při každém pohybu hráče, ale pouze až překoná vzdálenost 32 bloků (velikost oblasti ve třídě `Chunk`). Tento výpočet se provádí asynchroně, tudíž k aktualizaci pořadí může dojít až po několika volání metody `update` třídy `ChunkManager`. Chyba, kterou představuje staré pořadí, ale není vzhledem k rychlosti pohybu hráče příliš veliká.

Herní svět

Herní svět zastřešuje třída `World`. Konstruktor této třídy očekává ukazatel na objekt třídy `Distributor` – tato třída se nestará, odkud se budou objekty třídy `Chunk` získávat (z disku nebo ze sítě). V této třídě se vytvoří instance třídy `ChunkManager`, která je zpřístupněna pomocí metody, aby do ní mohla být vkládána instance třídy `ChunkSurroundings` informující o pozicích a dohlednostech hráčů. Třída `World` obsahuje přístupové metody ke všem načteným objektům třídy `Chunk` a ke všem blokům. Také obsahuje další metody s algoritmy pro logiku pokládání a ničení bloků, kolize entit, zdí a tekutin, a další.

Důležitými metodami zde jsou aktualizační metody `asyncUpdate` a `syncUpdate`. První z nich může být volána kdykoliv – jedná se o aktualizace nezávislé na herním čase. Je zde volána metoda `update` třídy `ChunkManager` sloužící pro načítání objektů třídy `Chunk`.

Zavoláním metody `syncUpdate` dojde k posunu herního času o jeden tzv. tick. To by mělo nastat 60× za sekundu. O pravidelné volání této metody se ale stará frontend – backend pouze provádí simulaci herního světa nezávislou na reálném čase.

Vedle posunu herního času také dojde k posunu denního času. Denní čas se opakuje periodicky a jeden denní cyklus trvá 86 400 ticků – je tedy 60× zrychlený oproti pozemskému dennímu cyklu. Den a noc vždy trvají polovinu tohoto cyklu, přičemž přechod mezi nimi vždy trvá 7 200 ticků. V metodě `sunLightCoefficient` je počítán koeficient denního světla podle aktuální denní doby v intervalu $\langle 0; 1 \rangle$. S jeho aktuální hodnotou je násobena každá hodnota slunečního světla v herním světě.

Dojde zde také k aktualizaci stavů bloků (např. šíření tekutin, růst trávy a stromů, atd.) a k pasivním aktualizacím entit (volný pád entity, dopak padajícího bloku a jeho přeměna na klasický blok, atd.).

Instance třídy `World` je vytvářena pomocí třídy `Distributor`, která se stará o získávání objektů třídy `Chunk` a dalších informací o daném světě. Třída `Distributor` je abstraktní. První třída, která z ní dědí, je třída `LocalDistributor`, která je určená pro lokální hru – hra je načítána a ukládána do složky se hrou, ve které je hra spuštěna. Požadavek na objekt třídy `Chunk` může vyřídit jedním ze dvou způsobů. Nejdříve zkusí načíst daný objekt z disku, o toto se stará třída `ChunkStorage`. Pokud objekt nebyl na disku nalezen, znamená to, že zatím nebyl uložen a tudíž ani vygenerován. Proto v takovém případě přichází na řadu generátor světa, který je zapouzdřen ve třídě `GeneratorManager`, která se stará o paralelní generování pomocí generátoru světa, který představuje abstraktní třída `Generator`, která bude dále popsána v kapitole 6

Dalšími odvozenými třídami od třídy `Distributor` jsou třídy starající se o běh hry po síti. Jedná se o třídu `ClientDistributor`, který je vytvářen na klientské straně – na počítači s grafickým výstupem. Požadavky na objekty třídy `Chunk` jsou zde transformovány na zprávy, které se posílají po síti na server a naopak zprávy ze serveru jsou interpretovány jako objekty třídy `Chunk`, případně jsou provedeny jiné akce v závislosti na typu zprávy. Herní svět v tomto případě běží v omezeném režimu, kdy nejsou např. prováděny aktualizace

bloků – ty jsou prováděny pouze na straně serveru a případné změny jsou posílány na klient formou síťových zpráv.

Protipolém ke třídě `ClientDistributor` je třída `ServerDistributor`, která dědí přímo z třídy `LocalDistributor` a běží na straně serveru. Na straně serveru běží hra v plnohodnotném režimu, ale chybí zde frontend v podobě grafického výstupu. Místo toho se ve třídě `ServerDistributor` nachází logika pro komunikaci s klienty. Server např. poskytuje klientům objekty třídy `Chunk`, posílá veškeré aktualizace herního světa a vyřizuje případné požadavky od klienta. Ukládání a načítání dat z disku je zde převzato ze třídy `LocalDistributor`.

Aktualizace bloků

Při každém volání metody `syncUpdate` není možné kontrolovat všechny načtené bloky, aby bylo zjištěno, zda-li je potřeba provést nějakou aktualizaci, protože herní svět obsahuje příliš velké množství bloků na to, aby toto mohlo být provedeno v reálném čase. Aktualizací bloku je myšleno např. šíření tekutin, růst trávy, růst stromů, mizení bloků listí po vytěženém stromu apod. Proto existuje struktura `FutureUpdate`, která obsahuje informace o tom, jaká akce, na jaké pozici a v jakém čase (hodnota počtu ticků v objektu třídy `World`) je nutné provést. Tyto struktury jsou obvykle tvořeny při změně bloku (metoda `setBlock` třídy `Chunk`) nebo při vykonávání jiné akce naplánované pomocí struktury `FutureUpdate`. Tyto struktury jsou vkládány do fronty v objektu třídy `Chunk`, kde jsou řazené podle času. Logika jednotlivých akcí se nachází v metodě `updateFutures`, která je volána metodou `syncUpdate` třídy `World`. Zde jsou postupně odebírány struktury z fronty a prováděny akce, které definují, dokud souhlasí hodnota ticku ve struktuře s aktuálním tickem herního světa. Jakmile tick přestane souhlasit, znamená to, že následující akce je určena pro provedení v budoucnu a provede se tedy v některém z budoucích volání metody `updateFutures`.

Předměty

Předmět reprezentuje třída `Item`, která obsahuje druh předmětu (vnořený výčet `Type`) a množství. Některý druhy předmětů jsou úzce svázány s určitým druhem bloku. Pro tyto druhy vrací metoda `canBeBlock` hodnotu `true` a příslušný druh předmětu a druh bloku mají stejnou binární hodnotu. Třída předmětu také obsahuje velké množství metod, které vracejí vlastnosti předmětu závislé na jeho druhu, podobně jako je to v případě bloku.

Tato třída je navržena tak, že umožňuje využití polymorfismu pro přidání vlastností určitému druhu předmětu. Je implementována jedna rozšiřující třída `BreakableItem`, která předmětu přidává míru poškození. Tato třída je použita pro předměty představující nástroje, jako např. krumpáč, který se používáním ničí.

V inventáři, truhle apod. jsou předměty vkládány do mřížky. Tuto mřížku představuje třída `ItemsGrid`. Např. v případě truhly je tato mřížka vázána na její blok. Mřížka může obsahovat větší množství objektů třídy `Item` a tudíž nemohou být uložené ve struktuře bloku, která všechny vlastnosti bloku ukládá na pouhých 32 bitech. Pro tyto účely existuje abstraktní třída `BlockData`, jejíž objekty se ukládají přímo do třídy `Chunk` a jsou asociovány s konkrétní pozicí bloku – tím je nahrazen chybějící polymorfismus struktury bloku. V herním světě se obvykle nachází malé množství bloků, které mají takové vlastnosti, že je nutné ukládat je do třídy `BlockData`, tudíž je toto řešení výhodnější, než rozšiřování samotné struktury bloku. Pro zmíněný blok truhly existuje odvozená třída `ChestBlockData` obsahující mřížku předmětů, které se nacházejí uvnitř truhly.

Entity

Entitu představuje abstraktní třída `Entity`, od které ostatní třídy dědí. Jsou to třídy `Player` (hráč), `FallingBlock` (padající blok) a `DroppedItem` (upuštěný předmět). Zde se nenachází žádný výčet určující druh entity, protože každý druh entity má vlastní třídu – je zde plně využit polymorfismus. Entit se v herním světě nachází oproti blokům velmi malé množství, proto zde nedává smysl se omezovat na struktury bez dědičnosti.

Na rozdíl od bloků nejsou pozice entit omezeny celými čísly, ale jsou reprezentovány čísly s plovoucí desetinnou čárkou. Tzn. že entity nejsou vázané na blok. Objekty entit jsou vázány na objekt třídy `Chunk`, přičemž během pohybu entity po světě může nastat situace, kdy je přesunuta z jednoho objektu třídy `Chunk` do jiného, protože se přesunula mimo část světa, kterou původní objekt třídy `Chunk` reprezentoval. K přemístění entity slouží metoda `warpTo`, která přesune entitu na pozici specifikovanou parametrem, ale to pouze v případě, že se na dané pozici nenachází žádný kvádr zdi. Podobnou metodou je metoda `moveTo`, která funguje velmi podobně, ale v případě, že na cílové pozici se nachází zeď, tato metoda entitu i přesto posune, a to tak, že ji posune daným směrem od původní pozice k nové o maximální vzdálenost, při které nedojde ke kolizi se zdí. Tyto metody jsou volány z metody `update`, ve které se nachází logika každé entity popsané v kapitole 3.8. Metoda `update` je volána uvnitř metody `syncUpdate` třídy `World`.

Biomy

Hodnoty jednotlivých biomů představuje výčet `BiomeType`. Jak již bylo popsáno v kapitole 3.7, v jednom bloku se může nacházet více biomů v určitém poměru. K tomu slouží třída `FuzzyBiomeBlock`, která uchovává seznam biomů a k nim asociovány jejich váhy. Tato třída má implementovány některé aritmetické operace, aby bylo možné spočítat např. aritmetický průměr z více objektů této třídy.

Objekt třídy `Chunk` obsahuje instanci abstraktní třídy `Biome`, která uchovává hodnoty pro všechny bloky odpovídající části světa objektu třídy `Chunk`. Třída `Biome` obsahuje metodu `getFuzzy`, která vrací objekt třídy `FuzzyBiomeBlock` pro konkrétní blok na daných souřadnicích, a metodu `get`, která vrací hodnotu datového typu `BiomeType` – tato hodnota představuje biom s nejvyšší váhou v objektu třídy `FuzzyBiomeBlock`.

Od třídy `Biome` dědí třídy `MonoBiome`, `HorizontalMultiBiome` a `HorizontalLeveledMultiBiome`. První zmíněna slouží pro případ, kdy ve všech blocích v objektu třídy `Chunk` je stejný biom. Druhá slouží pro případ, kdy ve všech blocích, jejichž souřadnice se liší pouze na ose Y, je biom definován stejným objektem třídy `FuzzyBiomeBlock`. Tato třída je určená pro výpočet biomů v generátorech světa a k tomuto účelu např. obsahuje metodu `blur`, která provede *rozostření* hranic biomů za účelem plynulejších přechodů mezi nimi. Poslední zmíněná třída se chová stejně jako předchozí, pouze hodnoty vah jednotlivých biomů mapuje na 8 bitů za účelem snížení paměťových nároku za cenu snížení přesnosti. Tyto třídy existují jako optimalizace, aby pro každý blok nemusel existovat odpovídající objekt třídy `FuzzyBiomeBlock`. Třída `Biome` je navržena tak, že pro každý blok v objektu třídy `Chunk` může uchovávat unikátní objekt třídy `FuzzyBiomeBlock`, ale žádná odvozená třída, která by splňovala tuto vlastnost, neexistuje – pro generátor vytvořený v rámci této práce není nutná.

5.3 Klient

Tato aplikace představuje vstupní bod pro hráče. Jako jediná má grafický výstup a vyžaduje knihovnu OpenGL, pomocí které je zde vykreslována veškerá grafika. Nachází se zde hlavní menu, vytvoření a spuštění hry, připojení k síťové hře a základní nastavení (jazyk, dohlednost, ...). Na obrázku 5.1 se nachází zachycení obrazovky s hlavní nabídkou této aplikace.

Obrázek 5.1: Hlavní nabídka klientské aplikace

Základní anatomie aplikace

Ve funkci `main` je vytvořen objekt `Window` v jehož metodě `run` se nachází celý životní cyklus okna aplikace. Okno je vytvořeno pomocí knihovny SDL a je v něm inicializován grafický kontext knihovny OpenGL – od této chvíle se o vykreslení veškeré grafiky v okně stará právě OpenGL.

Dále se zde nachází vykreslovací smyčka, která běží po celou dobu běhu aplikace. Ve smyčce jsou nejdříve pomocí knihovny SDL přečteny události (vstup z klávesnice a myši, události okna, ...). Informace získané z událostí jsou následně uloženy do třídy `UpdateData`, která je distribuována do ostatních částí aplikace, kde jsou z ní získávána potřebná data.

Celá aplikace je rozdělena na tzv. obrazovky. V aplikaci existuje několik obrazovek (např. obrazovka hlavního menu, obrazovka nastavení, obrazovka hry, atd.), přičemž aktivní je vždy právě jedna. Všechny tyto obrazovky jsou reprezentovány třídami, které dědí od abstraktní třídy `Screen`. Toto rozdělení aplikace bylo inspirováno např. aplikacemi pro mobilní systém Android, kde podobnou funkci představuje `Activity`.

Třída `Screen` obsahuje dvě důležité metody `update` a `draw`. První z nich je určena k provedení aktualizace obrazovky podle aktuálních hodnot ve třídě `UpdateData` – např. reakce na kliknutí myši. Metoda `draw` slouží k překreslení obrazovky v okně aplikace. Obě metody jsou volány z vykreslovací smyčky.

Implementace ovládacích prvků jako jsou tlačítka, textová pole, zaškrtávací políčka apod. včetně jejich rozložení je inspirována knihovnami jako jsou WinForms, Qt, GTK+ apod. Základem je abstraktní třída `Control`, od které dědí třída každého prvku. Prvky jsou uspořádány ve stromu – návrhový vzor *Composite*. Prvky, které představují vnitřní uzly tohoto stromu, jsou tzv. kontejnery. Existují dva základní druhy kontejnerů – první z nich může obsahovat libovolné množství synovských uzlů a takový kontejner slouží k jejich rozložení v prostoru. Druhý může mít právě jeden synovský prvek, kterému přidává určitou vlastnost, např. rámeček, scrollbar, barva pozadí, rozměry apod. Prvky představující koncové uzly jsou obvykle formulářové prvky jako tlačítko, textový popisek, textové pole apod.

Těmito ovládacími prvky jsou implementovány veškeré nabídky a formuláře, které se v aplikaci nacházejí.

Frontend hry

Samotná hra se nachází na obrazovce implementované třídou `GameScreen`. Nejdůležitější objekty nacházející se v této třídě jsou objekt třídy `GameController`, který se stará o obsluhu backendu hry, a objekt třídy `WorldRenderer`, který se stará o vykreslení hry z pohledu hráče.

Obsluha backendu hry ve třídě `GameController` se nachází v metodě `update`. Při každém zavolání této metody je zavolána asynchronní aktualizace backendu voláním metody `asyncUpdate` třídy `World` z jádra hry. Synchronní aktualizace (metoda `syncUpdate` třídy `World`) je volána tak, aby byla zavolána 60× za sekundu. Vykreslovací smyčka nacházející se ve třídě `Window` negarantuje pravidelnost volání metody `update`, ani četnost tohoto volání. Pravidelnost volání synchronní aktualizace ve třídě `GameController` je zajištěna tak, že ve speciální proměnné je počítán čas od spuštění hry, ke kterému se při každém zavolání metody `update` přičtě čas od posledního volání. Následně je v cyklu volána metoda `syncUpdate` třídy `World` a odečítána délka jedné periody (1/60 sekundy) tak, aby hodnota v této proměnné neklesla pod nulu. Tzn. že synchronní aktualizace se může během jednoho volání metody `update` zavolat vícekrát, ale také se nemusí zavolat vůbec. V případě, že hra běží na stabilních 60 FPS, synchronní aktualizace je zde volána právě jednou.

Dále se v této třídě nachází obsluha pohybu entity hráče podle vstupu z klávesnice a myši od uživatele. Tato obsluha se nachází v metodě `controlPlayer` volané z metody `update`.

O vykreslení veškeré grafiky herního světa se stará třída `WorldRenderer`, ve které jsou jednotlivé části vykreslování přesměrovány do dalších tříd. Herní svět je vykreslován po částech, které odpovídají objektům třídy `Chunk`. O vykreslení jedné takové části se stará třída `ChunkRenderer`. Aby tato třída byla schopna požadovanou část světa vykreslit, je potřeba nejdříve bloky obsažené v objektu třídy `Chunk` přepočítat na odpovídající trojúhelníky tvořící jejich grafiku. Tuto operaci provádí metoda `computeVertices`. Následně je pomocí knihovny OpenGL vytvořen tzv. *vertex buffer*, pomocí kterého jsou tyto trojúhelníky přeneseny do paměti grafické karty. Jakmile se vypočtené trojúhelníky nachází na grafické kartě, tak je možné je vykreslit metodou `draw`. Pokud dojde v objektu třídy `Chunk` ke změně, která má vliv na grafickou podobu dané části světa, je nutné metodu `computeVertices` zavolat znovu.

Výpočet nacházející se v metodě `computeVertices` je časově náročný, proto probíhá ve vedlejších vláknech programu. K přidělování vláken jednotlivým instancím třídy `ChunkRenderer` se stará třída `ChunkRenderManager`, která funguje velmi podobným způsobem

jako třída `ChunkManager` v jádře hry. Také zde jsou přednostně zpracovávány části světa, které se nacházejí v menší vzdálenosti od hráče. I zde je řazení objektů třídy `ChunkRenderer` časově náročné a proto musí probíhat ve zvláštním vláknu, aby nebyla narušena plynulost hry. O řazení se zde stará třída `ChunkRenderQueue`.

V metodě `update` objektu `WorldRenderer` jsou jednotlivé vertex buffery, ve kterých se nachází geometrie zobrazované grafiky, vykreslovány tak, aby byly vždy nejdříve vykresleny ty, které zpracovává jeden shader program, ten je pak vyměněn za jiný a vykreslování pokračuje dalšími buffery.

Ve třídě `ChunkRenderer` jsou generovány vertex buffery pro 3 shader programy – pro vykreslení bloků (statické, bez animací), vody a lávy. Další vertex buffery jsou vytvářeny ve specializovaných třídách. Truhly a dveře jsou spravovány třídou `OpenableRenderer`. Ty jsou speciální svými animacemi, které jsou z této třídy obládány pomocí přepisovatelných vertex bufferů (zapisuje se do dat uložených na grafické kartě čas zahájení animace). Pro vykreslení entit pak slouží třídy `DropedItemsRenderer` (upuštěné předměty) a `FallingBlocksRenderer` (padající bloky).

5.4 Generátor map

Tato konzolová aplikace slouží pro generování bimapových obrázků ve formátu PNG, které představují pohled na vygenerovaný herní svět z ptačí perspektivy (mapu). Jedná se o alternativní dvourozměrnou interpretaci voxelového modelu herního světa. Každý pixel obrázku mapy představuje jednu horizontální pozici bloku, jehož barva odpovídá nejvyššímu bloku, který není vzduch. Mapa může být generována ve třech režimech:

- `simple` – Barva pixelu přesně odpovídá typu nejvyššího bloku.
- `shading` – Stejné, jako `simple`, ale je přidáno stínování kopců podle vypočtené normály (podle výšek okolních bloků) a na vodní hladině je naznačena hloubka odstínem modré barvy.
- `biomes` – Barva pixelu odpovídá biomu nacházejícím se na dané pozici.

Pomocí parametrů lze určit generátor a semínko generovaného světa, případně název složky s uloženým již vygenerovaným světěm. Pomocí této aplikace je možné vygenerované objekty třídy `Chunk` i ukládat – pro přístup k datům je využita třída `LocalDistributor`. Dále je možné zadat pomocí souřadnic pozici a velikost prostoru v herním světě, ze kterého se bude mapa generovat. Více o parametrech je možné zjistit z vestavěné nápovědy, kterou lze získat spuštěním aplikace s parametrem `-h`.

O získávání objektů třídy `Chunk` se zde starají třídy odvozené od abstraktní třídy `ChunkLoader`. První odvozenou třídou je `LocalChunkLoader`, ve které se objekty třídy `Chunk` získávají z objektu třídy `LocalDistributoru` z již dříve uloženého světa. Pokud to není parametrem zakázáno, jsou nově vygenerované třídy zároveň i uloženy. Druhou je třída `GeneratorOnlyChunkLoader`, která pro získávání objektů třídy `Chunk` využívá třídu `GeneratorManager`. V tomto případě nejsou vygenerované objekty nikam ukládány, slouží pouze pro vykreslení obrázku a následně jsou zahozeny.

O vykreslení se starají třídy odvozené z `ChunkDrawer`. Jsou to třídy `SimpleChunkDrawer`, `ShadingChunkDrawer` a `BiomesChunkDrawer`, které odpovídají jednotlivým režimům aplikace. Celou aplikaci zastřešuje třída `MapGenerator`.

Kapitola 6

Implementace generátoru světa

Generátory světa jsou implementovány tak, aby je bylo možné do hry přidávat bez nutnosti kompilace samotné hry. Proto jsou řešeny jako zásuvné moduly[1] pomocí dynamicky linkovaných knihoven. Základem každého generátoru je abstraktní třída `Generator` nacházející se v jádře hry. Jednotlivé generátory pak od této třídy dědí. V knihovně každého generátoru se povinně musí nacházet funkce `createGenerator` vracející ukazatel na novou instanci třídy `Generator` vytvořené operátorem `new`. O připojování jednotlivých generátorů do hry se v jádře hry stará třída `Generators`. V této třídě se prochází soubory ve složce `generators` nacházející se ve složce hry a každý se pokusí připojit jako dynamicky linkovaná knihovna, ve které se hledá funkce `createGenerator`.

Abstraktní třída `Generator` obsahuje vedle metod pro získání informací o daném generátoru a dalších pomocných metod také metody `generate` a `findSpawn`. První zmíněná má za úkol vytvořit instanci třídy `Chunk` (více o této třídě v kapitole 5.2) na základně souřadnic předané v parametru metody. Jedná se o vygenerování části světa o velikosti $32 \times 32 \times 32$ bloků. Každá taková část je generována nezávisle na sobě a metoda `generate` musí být *thread-safe*[2], protože může být volána současně z více vláken. Metoda `findSpawn` má za úkol nalézet souřadnice místa, na kterém se objevují noví hráči ve hře – tzv. spawn. Generátor by měl zajistit, aby se noví hráči neobjevovali např. vysoko nad zemí, ve zdi, ve vodě apod. U složitějších generátorů může být hledání bezpečné pozice netriviální a může vyžadovat volání metody `generate`.

Jelikož je herní svět generován po částech (objekty třídy `Chunk`) a za použití náhodilosti, je nutné zajistit, aby daná nahodilost nezpůsobovala, že jednotlivé části herního světa na sebe nebudou navazovat. Pro generování pseudonáhodných čísel byl použit generátor Mersenne Twister implementovaný ve standardní knihovně jazyka C++. Generátor se incializuje pomocí semínka (angl. *seed*), které určuje generovanou posloupnost čísel – při opětovném generování se stejným semínkem získáme stejnou posloupnost. Dále je nutné pro každý generovaný objekt třídy `Chunk` inicializovat pseudonáhodný generátor zvlášť. Není možné použít stejné semínko pro každý objekt, protože by vygenerované objekty byly všechny stejné. Tento problém je odstraněn tím, že pro každý generovaný objekt je vypoč-

[1] Zásuvný modul, nebo-li plugin (z anglického *plug in* – zasunout) je software, který nepracuje samostatně, ale rozšiřuje funkčnost jiné aplikace, se kterou komunikuje pomocí připraveného API.

[2] Vlastnost kódu *thread-safe* (vláknově bezpečný) je pojem z multivláknového programování, který označuje takovou část kódu, kterou je možné v jeden okamžik spustit paralelně z více vláken a je zajištěno, že při tomto paralelním spuštění nedojde mezi vlákny k žádnému konfliktu, který by program uvedl do nekonzistentního stavu. V takovém kódu musí být např. ošetřen přístup k proměnné společné pro více vláken, např. pomocí zámků.

teno unikátní semínko z hlavního semínka nastaveného v generátoru a pozice generovaného objektu třídy `Chunk`.

V následujících podkapitolách bude popsán generátor nazvaný *Jahodový* (respektive *Strawberry* v anglické mutaci hry). Tento generátor je implementován dle návrhu v kapitole 4.

6.1 Biomy

O generování rozložení jednotlivých biomů v prostoru světa se stará třída `BiomTiles`. Biomy se zde počítají pouze v horizontální rovině – k jejich výpočtu je využívána třída `HorizontalMultiBiome` z jádra hry. Nejprve dojde k rozdělení biomů pomocí 2D Perlinova šumu a Voroného diagramů, jak je popsáno v kapitole 4.1. Tím získáme první fázi rozdělení biomů, které mezi sebou mají ostré hranice. Pro generování povrchu popsané v následující podkapitole jsou nutné velmi jemné přechody mezi biomy, čímž je docílena metodou `blur`, kterou je provedeno *rozostření* s poloměrem 32 bloků. Pro přístup k objektu `HorizontalMultiBiome` s takto jemnými přechody mezi biomy slouží metoda `getBlurred` třídy `BiomeTiles`.

Postup pro vytvoření rozdělení biomů pro objekt třídy `Chunk` je složitější. Vzhledem k vlastnostem povrchu v jednotlivých biomech je nutné provést korekci hranic za účelem získání *hezčího* výsledku. Např. v případě hranice biomu hor a biomu roviny by vzhledem k implementaci povrchu se tato hranice nacházela uprostřed svahu okrajových hor a je žádoucí tuto hranici posunout k jejich úpatí. Začíná se opět objektem `HorizontalMultiBiome` z první fáze, u kterého je také volána metoda `blur`, nyní s poloměrem 8. Následně jsou jednotlivé složky všech objektů třídy `FuzzyBiomeBlock` vynásobeny danými koeficienty závislé na typu biomu. Vysoko položené biomy (např. hory) mají vyšší hodnoty koeficientů a naopak nízko položené biomy je mají nízké. Hodnoty koeficientů byly zvoleny tak, aby výsledek byl co nejuspokojivější. V další fázi dojde k opětovnému zaostření podle biomů s nejvyššími váhami v objektech třídy `FuzzyBiomeBlock`. Výsledek této fáze je podobný jako v první fázi, ale již s posunutými hranicemi biomů. V poslední fázi je opět volána metoda `blur` s poloměrem 8 a výsledek je uložen do objektu třídy `HorizontalLeveledMultiBiome`, případně do objektu třídy `MonoBiome`, pokud je to možné. Tento výsledný objekt je již připraven k uložení do instance třídy `Chunk` a získat jej lze metodou `get`.

6.2 Terén

Základem tvaru a výšky terénu světa v tomto generátoru je výšková mapa. Ta je vypočtena součtem tří složek: základní výšky, 2D Perlinova šumu a Voroného diagramu. Pro výpočet jednotlivých složek zde jsou třídy `SurfaceStaticHeightLayer`, `SurfacePerlin2HeightLayer` a `SurfaceVoronoiHeightLayer`. Pro každý biom je každá složka počítána s jinými parametry, proto každá z těchto tříd obsahuje metodu `addBiome`, která do objektu třídy přidá parametry pro konkrétní biom. Všechny tři třídy dědí od abstraktní třídy `SurfaceHeightLayer`, která obsahuje metodu `apply`, ta vypočte hodnotu výšky terénu na daných souřadnicích. Tato metoda očekává biomy reprezentované objektem třídy `FuzzyBiomeBlock` a hodnota výšky v dané pozici je počítána jako vážený aritmetický průměr podle vah jednotlivých biomů a výšek vypočtených na základě parametrů těchto biomů. Díky tomu, že biomy byly ve třídě `BiomeTiles` rozostřeny s velkým poloměrem, dochází ve výškové mapě na hranicích jednotlivých biomů k plynulým přechodům.

Od třídy `SurfaceHeightLayer` také dědí třída `SurfaceMultiHeightLayer`, která obsahuje kolekci objektů třídy `SurfaceHeightLayer` a v metodě `apply` počítá součet všech výšek vypočtených objekty této kolekce. Všechny tyto třídy dohromady tvoří návrhový vzor *Composite* a tím je zaručena snadná rozšiřitelnost o další typy terénu.

Biom zvaný skalnaté hory (angl. *rocky mountains*) pro výpočet svého terénu vyžaduje 3D Perlinův šum a tento terén nelze vyjádřit pomocí výškové mapy, protože v tomto biomu mohou vznikat převisy a v těchto místech existuje povrch na dvou i více úrovních. Pro výpočet tohoto terénu slouží třída `SurfacePerlin3`, která terén vypočítává způsobem popsaným v kapitole 4.3. Parametrem závislým na typu biomu je zde amplituda, která udává výšku rozmezí nad výškovou mapou, ve kterém je terén vytvářen. Tato amplituda je nenulová pouze v biomu skalnatých hor a na okrajích tedy postupně přechází až do hodnoty 0. Metoda `apply` zde ukládá výsledek přímo do objektu třídy `SurfaceChunk`.

Třída `SurfaceChunk` reprezentuje stejný prostor jako třída `Chunk` a obsahuje trojrozměrné pole, ve kterém je pro každý blok hodnota typu bool udávající, jestli se na daném místě nachází země, nebo vzduch. Toto pole je z optimalizačních důvodů implementováno pomocí třídy `std::bitset`. Třída `SurfaceChunk` slouží pro uložení výsledku generování terénu, který je následně využíván dalšími částmi generátoru.

Třída `SurfaceTile` obsahuje kolekci všech objektů třídy `SurfaceChunk`, které mají stejné horizontální souřadnice. Tato kolekce je zdola omezena tím, že neobsahuje všechny objekty obsahující pouze zemi, a shora tím, že neobsahuje všechny objekty obsahující pouze vzduch. Dále jsou zde předpočítány pro všechny horizontální pozice bloků maximální výšky země a minimální výšky vzduchu pro pozdější využití.

Celý výpočet terénu je spravován ve třídě `SurfaceManager`. Jsou zde vytvořeny instance tříd odvozených od `SurfaceHeightLayer`, nastaveny jejich parametry a vytvářeny objekty třídy `SurfaceTile`.

6.3 Povrch

O sekvence bloků na povrchu závislé na biomu, které byly popsány v kapitole 4.3, se stará třída `SurfaceBlocks`. Obsahuje metodu `apply`, která do objektu třídy `Chunk` zapíše na správná místa bloky, které se nachází na povrchu a těsně pod ním. Pozice, na které budou aplikované sekvence bloků, jsou v této metodě zjišťovány z objektu třídy `SurfaceTile` získáným z objektu třídy `SurfaceManager`. Pro každý blok země je vypočtena hloubka a z metody `type` typ sekvence. Metoda `type` určuje typ sekvence na základě pozice, biomu a náhodné hodnoty vygenerované pro danou horizontální pozici. Tato náhodná hodnota je využita k zanesení nahodilosti do rozmezí jednotlivých vrstev štítových hor a také k nahodilosti přechodu na okrajích pouště. V případě rozšíření generátoru o další biomy může být použita i pro jiné účely. Podle hloubky a typu sekvence je následně vypočten blok, který se na danou pozici zapíše.

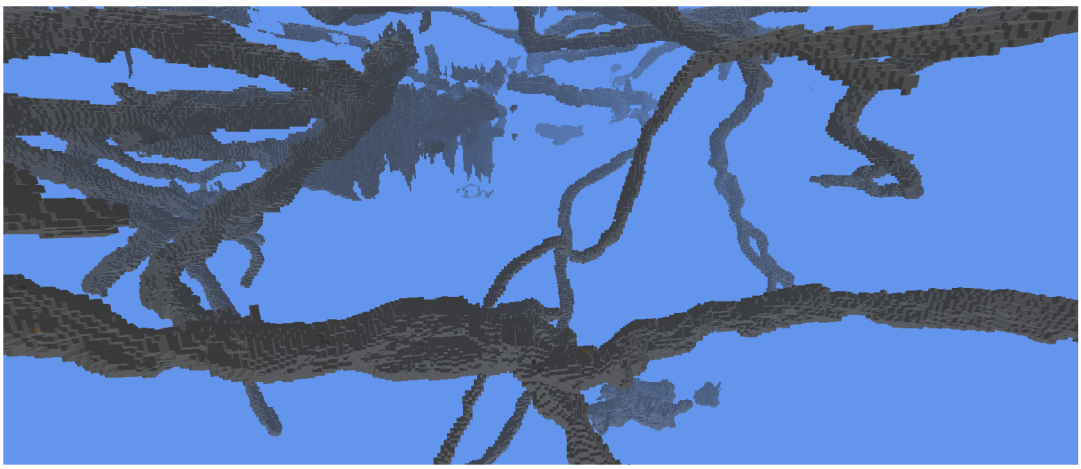
O umístění objektů na povrch zabírající jeden blok se starají třídy `Grass` (bloky trávy) a `Snow` (bloky sněhu). Obě třídy generují tyto bloky způsobem popsaným v kapitole 4.3. Bloky jsou umisťovány o blok výš, než je maximální výška země na dané pozici v objektu třídy `SurfaceTile`.

Stromy jsou umísťovány ve třídě `Trees`. Pozice stromů jsou počítány tak, aby mezi dvěma stromy byla vždy vzdálenost minimálně dva bloky. Samotné stromy jsou generovány třídou `ParabolicTree` nacházející se v jádře hry postupem, který je opět popsán v kapitole 4.3.

6.4 Podzemí

Pro generování jednotlivých druhů jeskyní popsaných v kapitole 4.6 slouží v jádře hry třída `Cave` generující chodby, třída `CaveDome` generující dóm a třída `Ravine` generující trhlinu. Všechny tři třídy obsahují statickou metodu `generate`, u které se v parametrech předají požadované parametry. Metoda `generate` vytvoří objekt dané třídy, který obsahuje metodu `isInner`, která podle pozice bloku určí, zda se daný blok nachází uvnitř prostoru jeskyně, nebo mimo.

V generátoru se o generování jeskyní stará třída `Underground`, ve které se jeskyně generují po dlaždicích o velikosti 512×512 bloků a jeskyně se generují pouze ve výšce v intervalu $\langle -384, 128 \rangle$. V každé takové dlaždici se vygenerují na náhodných pozicích jednotlivé jeskyně. Vygenerované jeskyně se následně mapují na objekty třídy `UndergroundChunk`, která je velmi podobná třídě `SurfaceChunk`. Pro vygenerování jednoho objektu třídy `UndergroundChunk` je nutné použít jeskyně z devíti dlaždic (1 dlaždice ve které se nachází pozice objektu třídy `UndergroundChunk` a 8 sousedních), protože některé jeskyně mohou přesahovat do sousedních dlaždic. Objekty třídy `UndergroundChunk` jsou následně použity pro vygenerování prostoru jeskynní v objektech třídy `Chunk` a pro detekci kolize s jeskyněmi při generování ostatních prvků světa, jako jsou stromy, tráva, budovy apod. Kolize mezi jednotlivými jeskyněmi se v třídě `Underground` neřeší – jeskyně spolu mohou kolidovat, protože tak spolu vytvoří větší komplex jeskyní.



Obrázek 6.1: Síť podzemních chodeb

Na obrázku 6.1 se nachází ukázka, jak může vypadat síť podzemních jeskyní a chodeb.

6.5 Budovy

Budovy jsou v herním světě generovány pomocí L-systémů, a to způsobem popsaným v kapitole 4.7.

Implementace L-systémů

Implementace samotných L-systémů se nachází v jádře hry. Symboly jsou zde reprezentovány pomocí objektů, které dědí od bázové třídy `LSymbol`. Díky tomu mohou v sobě

obsahovat libovolné parametry (členské proměnné), a to buď primitivní datové typy (např. reálná čísla jako je tomu v matematickém modelu popsaném v kapitole **??**), nebo i ukazatele na jiné objekty – a to jak mimo samotný L-systém, díky čemuž může symbol komunikovat s okolím a např. detekovat kolize, tak i na ostatní symboly v rámci jednoho L-systému a díky tomu mohou mezi sebou jednotlivé symboly komunikovat. Tento návrh tak dává implementovanému L-systému obrovskou výpočetní sílu.

Přepisovací pravidla představují třídy dědící od třídy `LRule`. Každé pravidlo v sobě obsahuje symbol, který představuje symbol na levé straně pravidla v matematickém modelu a který je v generátoru použit pro výběr přepisovacího pravidla na základě aktuálně čteného symbolu v řetězci. Třída `LRule` dále obsahuje tři virtuální metody. Metoda `can` určuje, zda lze v daném okamžiku pravidlo provést – tato metoda je určena pro podmíněná pravidla. Metoda `probability` vrací pravděpodobnost použití daného pravidla – toto slouží pro stochastický výběr pravidla v případě, že je možné na čtený symbol z řetězce použít více pravidel. Nejdůležitější metodou je metoda `generate`, která vrací řetězec na pravé straně – provádí samotné přepsaní symbolu řetězcem. Při každém použití pravidla může tato metoda vygenerovat jiný přepisující řetězec na základě parametrů symbolu, kontextu, nebo může použít generátor pseudonáhodných čísel. Všechny tyto tři metody dostávají v parametru objekt třídy `LContext`, který zpřístupňuje metodám čtený symbol ze řetězce a kontext – symboly před a za čteným symbolem (v případě symbolů před lze nahlížet jak do původního řetězce, tak do řetězce generovaného v aktuální iteraci).

O běh samotného L-systému se stará objekt třídy `LGenerator`, kterému je předán vstupní řetězec symbolů a jednotlivá přepisovací pravidla. Zavoláním metody `generate` této třídy se pak provede iterace L-systému – postupně je čten řetězec a vybírány pravidla, nad kterými jsou volány příslušné metody. Výsledný řetězec je pak zapsán zpět do objektu třídy `LGenerator` a objekt je připraven pro další iteraci.

Jazyk pro generování objektů v herním světě

Pro generátor světa byl navržen speciální jazyk, který vstupní řetězec interpretuje jako část voxelového modelu. Základní instrukcí v tomto jazyce je *Block(size, block)*, která v herním světě vytvoří kvádr o velikosti *size* vyplněný bloky *block*. Dále jazyk obsahuje několik instrukcí, které definují transformaci vkládání tohoto bloku do světa. Jsou jimi *Offset(offset)* (posunutí), *Rotate(transform)* (rotace nebo zrcadlení), *Clip(size)* (oříznutí) a *Priority(value)* (mění pořadí vkládání bloků do světa). Tyto instrukce platí vždy pro následující instrukci *Block* a lze je za sebou řetězit. Pokud mají být tyto instrukce aplikované pro více bloků, tyto bloky lze vložit do sekvence začínající instrukcí *Begin* a končící instrukcí *End*. Pro jednotlivé bloky uvnitř takové sekvence mohou být vloženy další transformační instrukce a dokonce zde může být vložena i další sekvence. Jednotlivé transformace jsou aplikovány zleva doprava. Jednotlivé symboly instrukcí jsou v kódu implementovány pomocí tříd *`Instruction`*`VolumetricLSymbol`, kde část *`Instruction`* v názvu každé třídy je nahrazena názvem dané instrukce.

O interpretaci tohoto jazyka se stará třída `VolumetricLInterpret`, která nejdříve provede zparsování řetězce instrukcí. Výsledkem této operace je strom, v jehož listech se nachází instrukce pro vykreslení kvádru tvořený určitým druhem bloku. Ve vnitřních uzlech je pak uložen kvádr obalující všechny synovské uzly a slouží jako optimalizace při zápisu voxelového modelu do objektu třídy `Chunk`, kdy jsou vynechány všechny uzly (a jejich podstromy), kde obalující kvádr má prázdný průnik s oblastí objektu.

Dále je pro tento jazyk definováno několik dalších symbolů, které nejsou součástí sady instrukcí, ale existují pro ně přepisovací pravidla, pomocích který je každý takový symbol přepsán na řetězec instrukcí. Příkladem takového symbolu je např. *Triangle(size, depth, block)*, který má za úkol vložit do světa hranol s pravoúhelníkovou podstavou tvořený daným blokem. Pro tento symbol existuje přepisovací pravidlo definované ve třídě `Triangle-VolumetricLRule`.

Pro všechny tyto symboly existuje tovární třída `VolumetricLAlphabet`. Ta navíc obsahuje metodu `addRulesToGenerator`, která do generátoru přidá všechny přepisovací pravidla pro symboly, které nejsou součástí jazyka.

L-systém samotných budov

Symboly a přepisovací pravidla pro samotné budovy se nachází přímo v generátoru. L-systém budov je navržen tak, aby všechny symboly byly ve výsledku přepsány na symboly instrukcí pro generování voxelového modelu. Všechny třídy reprezentující symbol pro generování budovy mají název ve tvaru `House*LSymbol` a třídy přepisovacích pravidel ve tvaru `House*LRule`.

Generování budovy je rozděleno na několik fází. Na konci každé fáze jsou z L-systému odebrány všechna přepisovací pravidla a vložena nová pro následující fázi. Generování je takto navrženo z toho důvodu, že některé symboly potřebují komunikovat s ostatními symboly a není žádoucí, aby symbol, se kterým chce jiný komunikovat, byl již ze řetězce odebrán (přepsán na jiný řetězec). Příkladem, kdy je nutná komunikace, je např. dělení místností, viz. kapitola 4.7 a obrázek 4.6. Symbol místnosti si uchovává ukazatele na všechny své sousedy, aby byla zachována informace o tom, ve kterých zdech je následně možné, či povinné vytvořit dveře, případně okna, pokud na druhé straně dané zdi žádná místnost není. Když dojde k rozdělení místnosti, musí být zároveň aktualizovány ukazatele na sousedy v okolních místnostech. Tato akce by selhala v případě, kdy by se symbol sousední místnosti již nenacházel v řetězci, ale byl rozgenerován na symboly v následující fázi jako jsou zdi, podlaha apod.

Pro každou budovu je vygenerována množina materiálů, ze kterých se skládá – např. materiál zdí, podlah, střech, atd. Tyto materiály jsou vloženy do symbolu budovy a během generování jsou distribuovány do konkrétnějších symbolů, jako jsou bloky, místnosti, zdi apod. V konečné fázi je materiál předán příslušným instrukcím *Block*.

Vkládání budov do světa

O vkládání budov do herního světa se stará třída `Houses`. Budovy jsou vkládány na náhodné pozice – jedno z možných budoucích rozšíření generátoru je budovy seskupovat do vesnic a měst. Pro budovu je nalezena taková vhodná pozice, že se nachází nad hladinou moře a povrch je v tomto místě relativně rovný – rozdíl výšky v jednotlivých rozích počátečního bloku nesmí být větší než 4 bloky. Pro rozšiřování půdorysu budovy o další bloky je omezeno o funkci, která kontroluje výšku okolního terénu. Výška místa, kde je generován nový blok, nesmí být vyšší více než o 3 a nižší než o 6 bloků oproti výšce počátečního bloku. Ve třídě `Houses` je metoda `apply`, která do daného objektu třídy `Chunk` přidá odpovídající část budovy.

Při vkládání budovy do světa mohou nastat dvě chyby. První je, že v některých místech může pod budovou vzniknout volné místo tím, že v daném místě je terén nižší, než spodek budovy. Metoda `apply` v takovém případě pod budovu přidá bloky kamení. Druhou možnou

Obrázek 6.2: Budova

chybou je, že vchodové dveře se mohou nacházet pod nebo nad úrovní terénu. V takovém případě metoda `apply` dodá na tato místa dřevěné schodiště.

Na obrázku 6.2 se nachází příklad vygenerované budovy, která je jedna z rozsáhlejších – obvykle mají budovy jednu, nebo dvě místnosti.

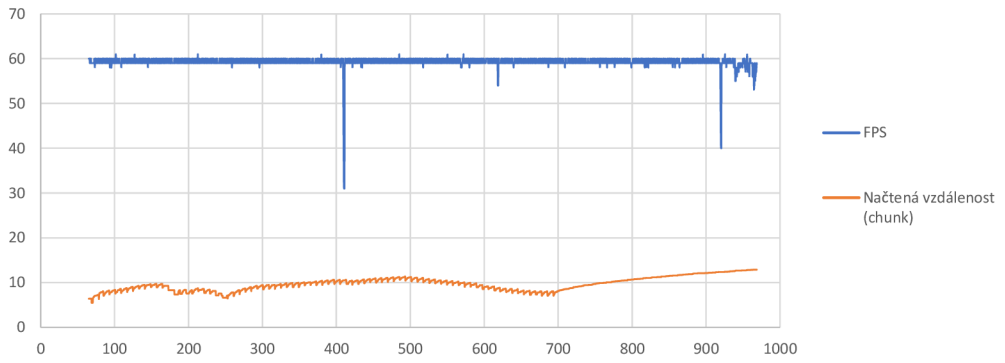
6.6 Třída generátoru

Celé generování je spravováno z metody `generate` třídy `Generator`. Jsou zde vytvořeny instance všech tříd spravující jednotlivé části generování a navzájem jim jsou předány potřebné ukazetele na ostatní objekty. Např. třída `Houses` potřebuje objekt třídy `SurfaceManager`, aby mohla detekovat výšky terénu.

Kapitola 7

Měření výkonu

Během pohybu hráče po herním světě klasickou rychlostí (chůze, sprint) by generátor světa měl být schopen dodávat vygenerované objekty třídy `Chunk` dostatečně rychle na to, aby bylo okolí hráče vždy načteno a nedošlo k situaci, kdy by došel na *konec* vygenerovaného světa, kde by se zasekl a musel čekat, než generátor světa patřičné objekty třídy `Chunk` dodá. V ideálním případě by měl generátor dodávat tyto objekty tak, aby vždy kolem hráče bylo vygenerováno okolí do vzdálenosti v řádu alespoň nízkých stovek bloků, aby nebyl hráči pokažen jeho herní zážitek.



Obrázek 7.1: Měření výkonu hry

Bylo provedeno měření, při kterém hráč prolétával herním světem rychlostí chůze, podobně jako v přiloženém demonstračním videu, ve kterém jsou tyto průlety 8x zrychleny. Herní svět byl nově vytvořený, na disku se tedy nenacházely žádné dříve vygenerované objekty třídy `Chunk` a proto během průletu musely být všechny objekty generované v reálném čase. Během průletu hráče byly zaznamenávány hodnoty FPS (počet vykreslených snímků za sekundu) a načtená vzdálenost v jednotkách *chunk* (1 *chunk* = 32 bloků). Zaznamenané hodnoty jsou vyjádřeny formou grafu na obrázku 7.1, kde se horizontální ose nachází čas v sekundách od spuštění aplikace.

FPS se během celého průletu drželo kolem hodnoty 60, která odpovídá vertikální synchronizaci, až na tři ojedinělé výkyvy. Tyto výkyvy byly pravděpodobně způsobeny ostatními programy běžícím na testovacím stroji. Hodnota FPS začala klesat až ke konci, kdy načtená vzdálenost přesáhla hodnotu 12. Tento pokles nastal z důvodu, že byla již vykreslena příliš velká plocha herního světa a testovací stroj začal přestávat stíhat překreslování.

Generátor světa by neměl mít na FPS žádný vliv, protože generování probíhá v jiných vláknech, než ve kterém běží vykreslovací smyčka.

Načtená vzdálenost se během celého průletu držela kolem intervalu $\langle 8, 10 \rangle$ (cca 250 až 300 bloků), což je dostačující hodnota pro kvalitní herní zážitek. Načtená vzdálenost chvílemi stoupala a chvílemi klesala. To je způsobováno různou náročností generování v různých částech světa, protože na některých místech ve světě se nacházejí složitější struktury, jejichž generování trvá déle. V čase 700 se hráč zastavil a načtená vzdálenost od této chvíle pouze stoupala, protože již nedocházelo ke vzdalování se od načtené části světa.

Kapitola 8

Závěr

Od této práce jsem si sliboval vytvořit pro tuto hru generátor světa, který bude splňovat dva hlavní cíle. Aby generátor tvořil *hezký* svět, kde krajina bude vypadat přírodně (s odhlédnutím od faktu, že je svět v této hře tvořen krychlemi) a vygenerované budovy budou připomínat budovy, které by postavili samotní hráči. Druhý cíl byl, aby generátor byl dostatečně rychlý na to, aby se svět stíhal generovat v reálném čase tak, že hráči při procházce světem nemusí nikdy čekat na to, až se dogeneruje další část světa.

Druhý cíl byl jednoznačně splněn, což potvrzuje kapitola 7, ve které bylo provede měření. Splnění prvního cíle je subjektivní záležitostí. Osobně jsem s výsledkem nad míru spokojen. Mohu říct, že vytvořený generátor naplnil má prvotní očekávání.

Dle mého názoru se v generování krajiny nejvíce povedly hory a to v obou horských biomech: štítové hory a skalnaté hory. Každý z těchto biomů generuje hory rozdílným způsobem a výsledek vypadá také velmi odlišně. V případě štítových hor jsou generovány vysoké hory se sněhovými čepičkami na vrcholcích. Z dálky tyto hory připomínají např. Alpy. Když se hráč k těmto horách přiblíží, krajina kolem nich se postupně zvedá, ubývají postupně stromy a tráva, až je nakonec celý povrch tvořen pouze kameny a štěrkem – nikde se nenachází žádný ostrý přechod z jednoho typu krajiny do druhého. V případě skalnatých hor jsou tvořeny zajímavé skalní převisy a dokonce i skalní brány. Tento biom vyniká mezi ostatními převážně rovinatými biomy. Nevýhodou způsobu generování těchto hor je bohužel fakt, že mohou být vygenerovány kusy skály nepropojené s povrchem – *létající ve vzduchu*.

V případě budov si dovoluji vyzvednout fakt, že do každé místnosti v budově existuje cesta, po které je možné se do ní dostat od libovolných vchodových dveří – budovy nejsou generovány příliš naivním způsobem, který by dovoloval generovat místnosti bez přístupu.

Existuje mnoho způsobů, jak je možné v budoucnu tento generátor světa rozšířit. Jelikož mám v plánu na hře pracovat i nadále, mám celkem jasno v tom, co bych rád přidal. Co se týče povrchu, lze vymyslet mnoho nových biomů generovaných různými způsoby, jako např. ledové krajiny, kaňony, řeky, horská jezera, pralesy, džungle atd. Dále není nutné se omezovat na horizontální krajiny, ale např. v určité hloubce by bylo možné generovat podzemní svět, např. podsvětí, peklo apod. Naopak ve výšce mohou být generovány vzdušné *létající* krajiny, nebo nebe. V případě budov existuje také mnoho možných rozšíření. Uvnitř nyní např. chybí vybavení. Dále by mohly být generovány další druhy místností, které by např. přesahovaly do více pater, velká schodiště, složitější střechy a jejich různé typy apod. Také může být generováno více druhů budov, např. hrady, kostely, městské budovy, atd. Budovy by mohly také tvořit vesnice, případně města, a mohly by být spojeny cestami. Možností je opravdu velké množství.

Literatura

[1] *Noise, being a pseudorandom artist*. [Online; navštíveno 18.5.2019].
URL https://catlikecoding.com/unity/tutorials/noise/

[2] *Minecraft*. 2009, [Online; navštíveno 7.1.2019].
URL https://minecraft.net/

[3] *Mojang*. 2010, [Online; navštíveno 7.1.2019].
URL https://mojang.com/

[4] *Building Rooms*. 2012, [Online; navštíveno 17.5.2019].
URL http://procworld.blogspot.com/2012/03/building-rooms.html

[5] Aurenhammer, F.: *Voronoi Diagrams – a Survey of a Fundamental Geometric Data Structure*. *ACM Comput. Surv.*, ročník 23, č. 3, Září 1991: s. 345–405, ISSN 0360-0300, doi:10.1145/116873.116880.
URL http://doi.acm.org.ezproxy.lib.vutbr.cz/10.1145/116873.116880

[6] Hendrikx, M.; Meijer, S.; Van Der Velden, J.; aj.: *Procedural Content Generation for Games: A Survey*. *ACM Trans. Multimedia Comput. Commun. Appl.*, ročník 9, č. 1, Únor 2013: s. 1:1–1:22, ISSN 1551-6857, doi:10.1145/2422956.2422957.
URL http://doi.acm.org.ezproxy.lib.vutbr.cz/10.1145/2422956.2422957

[7] Žára Jiří; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika*. Brno: Computer Press, druhé vydání, 2004, ISBN 80-251-0454-0.

[8] Milet, T.: *Grafické intro 64kB s použitím OpenGL*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2012.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=9301

[9] Perlin, K.: *An Image Synthesizer*. *SIGGRAPH Comput. Graph.*, ročník 19, č. 3, Červenec 1985: s. 287–296, ISSN 0097-8930, doi:10.1145/325165.325247.
URL http://doi.acm.org.ezproxy.lib.vutbr.cz/10.1145/325165.325247

[10] Prusinkiewicz, P.; Lindenmayer, A.: *The Algorithmic Beauty of Plants*. New York: Springer-Verlag, 1996, ISBN 0-387-97297-8.

Přílohy

Seznam příloh

Příloha A

Obsah paměťového média

`bin/` – Zkompilované soubory hry určené pro platformu Microsoft Windows x64
`map.png` – Mapa předvygenerované oblasti světa ve hře ve složce `bin`
`map_players.png` – Stejné, jako `map.png`, ale navíc jsou na mapě vyznačeny pozice hráčů
`src/` – Zdrojové soubory hry
`src/Sixteenia.sln` – Projektový soubor pro Microsoft Visual Studio 2017
`src/cmake/CMakeLists.txt` – Soubor pro nástroj CMake, nutno spustit z této složky
`projekt.pdf` – Technická zpráva ve formátu PDF
`doc/` – Zdrojové soubory pro vytvoření technické zprávy pomocí LaTeXu
`video.mp4` – Demonstrační video

Příloha B

Stručný manuál

Na přiloženém paměťovém médiu se ve složce `bin` nachází zkompilovaná hra pro platformu Microsoft Windows x64 včetně všech potřebných knihoven s výjimkou OpenGL, která musí být nainstalována v systému spolu s ovladačem grafické karty. Hra vyžaduje OpenGL ve verzi alespoň 3.1 nebo vyšší. Hru lze spustit i v prostředí systému Linux za použití nástroje Wine.

Hra se spouští pomocí souboru `SixteeniaClient.exe`. Po spuštění je nutné zadat jméno hráče (libovolný řetězec), potom vás hra přesměruje do hlavní nabídky.

B.1 Předpřipravený svět

Ve hře je připravený předvygenerovaný svět, ke kterému se dostanete přes tlačítko *Jeden hráč* (hra na lokálním počítači). Ve světě se nachází dva hráči se jmény *Budova* a *Hory*. Toto jméno je nutné zadat při spuštění celé aplikace a je nutné dodržet velikosti písmen. Pokud svět spustíte se jménem *Budova*, objevíte se u jedné z rozsáhlejších budov, které jsou ve světě generovány. V případě spuštění se jménem *Hory* se ocitnete na místě s panoramatickým výhledem na štítové hory. Pozice těchto dvou hráčů jsou vyznačeny na mapě, která se nachází v souboru `map_players.png` v kořenové složce média. V případě, že svět spustíte pod jiným jménem, objevíte se na spawnu (místo, na kterém se objevují noví hráči), který se od vygenerované části nachází ve velké vzdálenosti.

B.2 Ovládání

Pohyb hráče se ovládá klávesami *W*/*S*/*A*/*D*. Sprint lze aktivovat držením klávesy *Ctrl* během pohybu, případně dvojstiskem klávesy *W*. Skáče se pomocí mezerníku. Klávesou *L* lze zapínat/vypínat režim létání, ve kterém se nahoru pohybuje mezerníkem a dolů klávesou *Shift*.

Bloky lze ničit držením levého tlačítka myši a pokládat kliknutím pravého.

Klávesou *F3* lze zobrazit/skrýt debug informace, ve kterých se mimo jiné nachází i souřadnice aktuální pozice hráče.

Klávesou *Esc* je aktivována pauza a zobrazeno herní menu, přes které je hru možné opustit.

B.3 Příkazy

Herní konzoli lze otevřít klávesou *T* nebo */*. V druhém případě zároveň dojde ke vložení znaku `/` do textového pole. Konzoli lze ukončit klávesou *Esc*. Pomocí konzole lze spouštět příkazy. Následuje seznam několika užitečných příkazů.

`/time` *`integer`*
`/morning`
`/noon`
`/evening`
`/midnight`

Tyto příkazy změní denní dobu – vhodné pro *vypnutí* noci.

`/tp` *`integer integer integer`*

Tímto příkazem dojde k teleportaci hráče na zadané souřadnice. Pokud cílová destinace světa není vygenerována, může tento příkaz být proveden s určitým zpožděním.

`/coll`

Vypíná/zapíná kolizi se zdmi. S vypnutými kolizemi lze prolétat libovolnými bloky. Pokud zapnete kolize a nacházíte se uprostřed bloku, obrazovka zčerná a nebudete se moci hýbat. Aby jste se z takové pozice dostali, musíte kolize opět vypnout. Před vypnutím kolizí je vhodné zapnout režim létání, aby jste nepropadli blokem pod vámi.

`/ides`

Zapíná/vypíná instantní ničení bloků levým tlačítkem myši.

B.4 Nastavení

Do nastavení se lze dostat přes hlavní nabídku hry. Mimo jiné zde lze nastavit dohlednost. Pokud se hra seká, je vhodné hodnotu dohlednosti snížit. Naopak zvýšením této hodnoty bude krajina kolem hráče vykreslována do větší vzdálenosti.

B.5 Vytvoření nové hry

Nový svět lze vytvořit tlačítkem pod seznamem existujících světů. Kromě názvu světa zde lze vybrat generátor a semínko, které bude použito pro generování. Cílem této práce bylo vytvořit generátor s názvem *Jahodový*. Ostatní generátory byly vytvářené během studia a testování jednotlivých technik procedurálního generování a jejich aplikace pro generování určitých prvků krajiny, ze kterých je výsledný generátor složen.