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

GENERÁTOR 2D HERNEJ MAPY

MAP GENERATOR FOR 2D GAMES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MAREK NÉMETH**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ STARKA,**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

144940

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Németh Marek**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Generátor 2D herní mapy**
Kategorie:	Počítačová grafika
Akademický rok:	2022/23

Zadání:

1. Nastudujte techniky procedurálního generování.
2. Navrhněte knihovnu na procedurální generování 2D herní mapy pro hry typu Craft the World, Oxygen not Included, Terraria atp.
3. Implementujte knihovnu.
4. Implementujte demonstrační aplikaci, která vizualizuje vygenerovanou mapu.
5. Vytvořte krátké video demonstrující práci.

Literatura:
Po dohodě s vedoucím.

Při obhajobě semestrální části projektu je požadováno:
Body 1,2 a část 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Starka Tomáš, Ing.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	31.10.2022

Abstrakt

Cieľom práce je implementácia procedurálneho generátoru dvojdimenzionálneho herného sveta. Herný svet pozostáva z blokov tvoriacich konečné pole prvkov v mriežke. Generovaný terén obsahuje niekoľko biómov, dva typy jaskýň. Herná mapa obsahuje objekty a pozadia. Druhou časťou je implementácia editoru na vizualizáciu mapy.

Abstract

The goal of this thesis is to implement a procedural generator for a two-dimensional world. The world consists of blocks in an array representing a grid. The generated terrain contains a few biomes and two types of caves. The map contains objects and backgrounds. The second part of the thesis is the implementation of an editor for visualization.

Klíčová slova

procedurálne generovanie, Perlinov šum, editor 2D mapy, engine

Keywords

procedural generation, Perlin noise, 2D map editor, engine

Citace

NÉMETH, Marek. *Generátor 2D hernej mapy.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Starka,

Generátor 2D hernej mapy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáša Starku. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Marek Németh
9. května 2023

Poděkování

Chcel by som poďakovať pánovi Ing. Tomášovi Starkovi za pomocné rady a počas práce.

Obsah

1 Úvod **2**

2 Analýza a porovnanie existujúcich metód a softwaru **4**
2.1 Perlinov šum 4
2.2 Voroného diagramy a Worleyho šum 6
2.3 Editory 9
2.4 Herný engine 11

3 Návrh **12**
3.1 Návrh editorovej časti 12
3.2 Návrh generátoru 13
3.3 Dátové štruktúry 14
3.3.1 Moduly generátoru 15
3.3.2 Moduly editoru 16
3.3.3 Komunikácia modulov 17

4 Implementácia **19**
4.1 Nástroje 19
4.2 Generovanie sveta 19
4.2.1 Generovanie povrchu 20
4.2.2 Generovanie backgoundu 25
4.2.3 Generovanie objektov 25
4.3 Engine 27
4.3.1 Rendering 27
4.4 Editor a interakcie s GUI 30

5 Limitácie, možné rozšírenia a optimalizácie **34**
5.1 Limitácie 34
5.2 Optimalizácie súčastí 34
5.3 Rozšírenia modulov 35

6 Záver **36**

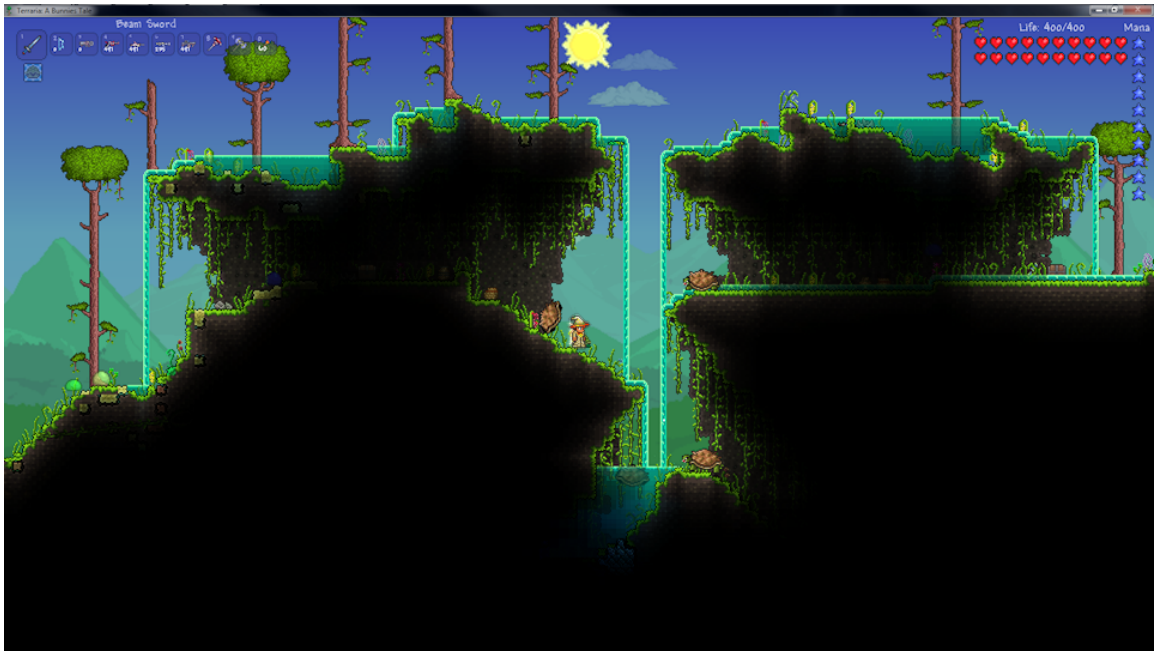
Literatura **37**

Kapitola 1

Úvod

V dobe automatizácie a modernizácie sa s vývojom čoraz výkonnejších počítačov rapídne rozrástlo odvetvie počítačových hier, ktoré sa v posledných rokoch stali veľmi populárne. Grafika týchto hier často dosahuje až neuveriteľnú hĺbku detailov. S týmto odvetvím je často spájané aj procedurálne generovanie. Metódy procedurálneho generovania sa využívajú v mnohých žánroch hier na vytvorenie potenciálne nekonečných arén, svetov a objektov v nich, ako napriklad na obrázku 1.1.

Za vznik procedurálneho generovania sa dá považovať spoločenská hra Dungeons & Dragons [8], ktorá na 'generovanie' úrovní používa náhodné hodnoty z hodov kociek. Tento princíp sa neskôr začal používať v oblasti informatiky na algoritmické vytváranie modelov (namiesto manuálneho). Rozvoj procedurálneho generovania viedol napríklad k vzniku hry Terraria, ktorá v 2D hernej úrovni generuje procedurálne objekty, povrch a jaskyne.



Obrázek 1.1: Procedurálne vygenerovaný povrch a jaskyne v hre Terraria.

Vo svojej práci sa zaoberám generovaním procedurálnych svetov v 2D. V časti 2 je spomenutý vývoj metód, ktoré sú používané pri procedurálnom generovaní, existujúce druhy

editorov a enginov. Pri práci sú použité niektoré z týchto metód s úpravami, ktoré sú popísané v 4.2.

Cieľom tejto práce je vytvorenie knižnice pre generovanie takýchto svetov bez ohľadu na žáner hry. Modul vytvára súbory, ktoré popisujú herný 2D svet. Knižnica generuje rôzne typy biómov[1]a procedurálnych štruktúr.

Druhou časťou práce je implementácia editoru a vykresľovanie mapy do okna spolu s jednoduchým grafickým užívateľským rozhraním. Editor poskytuje možnosť prehliadať vygenerovanú mapu a vykonávať v nej úpravy. Editor používa jednoduchý engine ktorý popisujem v 4.3. V závere práce sú zhrnuté dosiahnuté výsledky, získané skúsenosti a využiteľnosť v hernom priemysle. Uvedené sú tiež možnosti na rozšírenia a budúci rozvoj modulu.

[1]bióm - spoločenstvo rastlín a živočíchov v určitej geografickej zóne s charakteristickým typom vegetácie (napr. ihličnatý les, step, tundra)

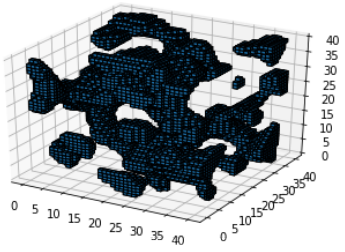
Kapitola 2

Analýza a porovnanie existujúcich metód a softwaru

Táto kapitola sa zaoberá metódami, ktoré sa dajú využiť na procedurálne generovanie a vykresľovanie generovaných objektov. Ďalej sú spomenuté existujúce editory a enginy, ktoré sa používajú na vývoj hier.

2.1 Perlinov šum

Perlinov šum je funkcia na generovanie koherentného šumu v priestore. *Koherentný* šum znamená, že hodnoty šumu sa menia hladko z jedného bodu na nasledujúci. Táto funkcia mala úspech v oblasti počítačovej grafiky, či už sa jedná o CGI[1] vo filmovom priemysle alebo procedurálne generované objekty v hernom priemysle.



(a) Šum vygenerovaný perlinovou funkciou

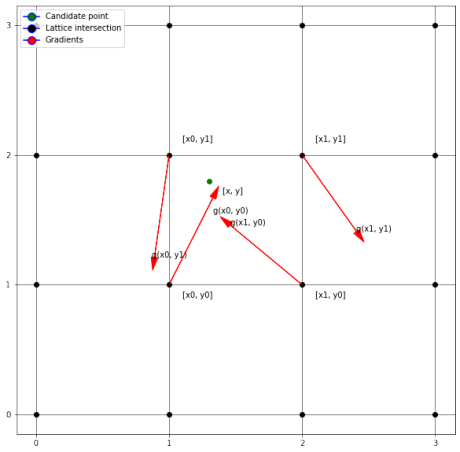
(b) Povrch herného sveta v hre Minecraft[a]

[a] prevzaté z https://www.9minecraft.net/xray-mod/

Obrázek 2.1: Trojrozmerný šum a príklad použitia

Na rozdiel od bieleho šumu (white noise), ktorý generuje pseudonáhdné hodnoty bez korelácie, Perlinova funkcia využíva na výpočet hodnôt n-rozmernú sieť, ktorá v n-rozmernom priestore tvorí tzv. Hyperkocky a gradienty uzlov(jednotlivých vrcholov hyperkociek) v nej.

[1] Computer-generated imagery (počítačom generované obrazy)



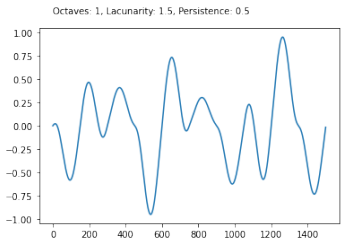
Obrázek 2.2: Princíp siete a náhodných gradientov na výpočet Perlinovho šumu v dvojrozmernom priestore

V $n \geq 1$ rozmernom priestore je každému z uzlov pridelený pseudonáhodný n-rozmerný gradient. V prípade $n = 1$ je tento gradient náhodná skalárna hodnota v rozmedzí -1,1.

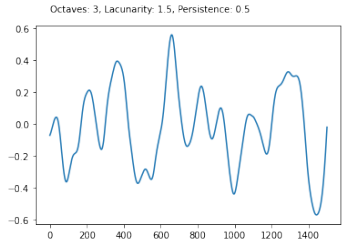
Na výpočet jednotlivých hodnôt kandidátnych bodov (daných množinou koordinátov o veľkosti n) sa využije interpolácia gradientov – skalárnych súčinov medzi gradientmi uzlov hyperkocky, v ktorej sa bod nachádza, a vektormi v danej hyperkocke z kandidátneho bodu do uzlov (obrázok 2.2).

Pri generovaní sa dvojrozmerný alebo trojrozmerný (obrázok 2.1) šum dá použiť na vytvorenie povrchu a jaskýň v priestore. **Oktáva** je jedna z šumových funkcií koherentných šumov, ktoré spojením tvoria Perlinov šum. Tieto šumové funkcie sú nazývané oktávami, pretože každá oktáva má dvojnásobok frekvencie predchádzajúcej oktávy. V praxi funkcia (obr. 2.3) pre výpočet používa nasledovné parametre:

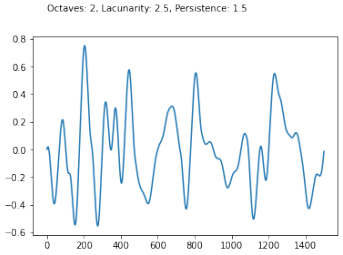
- Počet oktáv, ktorý udáva úroveň detailu pre vygenerovaný šum
- (Ang.) Lacunarity - Koeficient udávajúci rýchlosť rastu frekvencie pre každú oktávu vo funkcii. Frekvencia každej nasledujúcej oktávy je výsledkom násobenia frekvencie predošlej oktávy a koeficientu
- (Ang.) Persistence - Koeficient udávajúci ako veľmi jednotlivé oktávy prispievajú k celkovému tvaru. Amplitúda každej nasledujúcej oktávy je výsledkom násobenia amplitúdy predošlej oktávy a koeficientu



(a) Perlinov šum s prednastavenými parametrami



(b) Dopad zvýšenia počtu oktáv na šum



(c) Perlinov šum s upraveným koefiecientom pre amplitúdy a frekvencie

Obrázek 2.3: Demonštrácie Perlinovho šumu s rôznymi parametrami pre jednu oktávu (obr. 2.3a) na porovnanie so šumom s 2 oktávami pre rovnaké parametre (obr. 2.3b), a šumom s upravenými parametrami (obr. 2.3c).

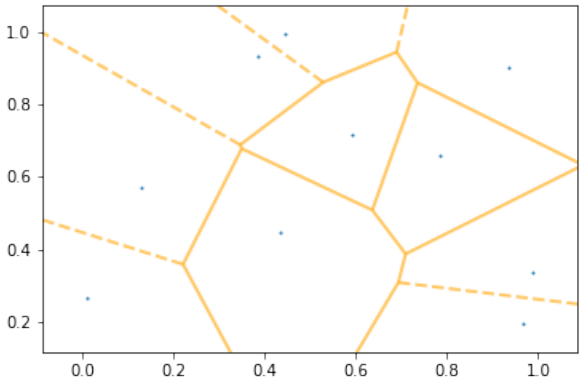
Jeden z najpodstatnejších rozdielov medzi Perlin noise (ďalej ako PN) a simplex noise (ďalej ako SN) algoritmom je spôsob, akým je počítaný šum. PN je založený na (hyperkubickej) sieti náhoných gradientov, zatiaľ čo SN používa sieť rovnostranných trojuholníkov. Výsledkom tohoto vylepšenia je šum, ktorý vyzerá prirodzenejšie a menej kockato oproti PN.

SN používa Lagrangeovu interpoláciu, ktorá je hladšia než lineárna interpolácia v PN. Pri počítaní hodnôt v SN pre vyšší počet dimenzií je vďaka vylepšenej metóde na generovanie náhodných gradientov vyššia, než v PN.

2.2 Voroného diagramy a Worleyho šum

Voroného diagram (tiež známy ako Dirichletova teselácia[4]) je v oblasti geometrie jedna z najzaujímavejších a najužitočnejších štruktúr.

Jednoduchá varianta Voroného diagramu využíva Delaunay trianguláciu[5] n bodov v Euklidovskom priestore (príklad na obr. 2.4).



Obrázek 2.4: Priklad Voroného diagramov pre 10 bodov v Euklidovskom priestore (náhodne rozložených)

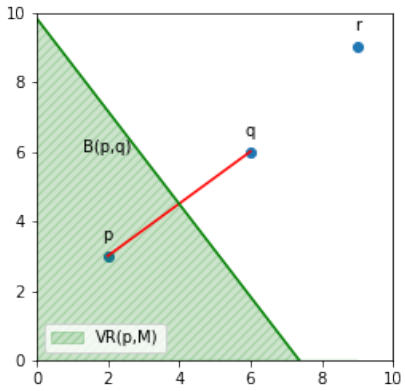
Nech množina M je množina o veľkosti $n \geq 3$ bodov $p, r, q....$ Pre body $p = (p_1, p_2)$ a $x = (x_1, x_2)$ nech $d(p, x) = \sqrt{(p_1 - x_1)^2 + (p_2 - x_2)^2}$ je ich euklidovská vzdialenosť. Pod $\vec{pq}$ sa bude rozumieť úsek priamky $\vec{pq}$ od bodu p po bod q. $\overline{A}$ je komplement(doplnok) množiny A. Pre $p, q \in M$ nech 2.1 je os bodov p a q, je kolmá na $\vec{pq}$. Táto os rozdeľuje polrovinu 2.2 obsahujúcu p od polroviny obsahujúcej q. Potom 2.3 nazývame Voroného región, a teda 2.4 definujeme ako Voroného diagram[2].

$$B(p, q) = \{x | (d(p, x) = d(q, x)\} \tag{2.1}$$

$$D(p, q) = \{x | d(p, x) < d(q, x)\} \tag{2.2}$$

$$VR(p, M) = \bigcap_{q \in M, q \neq p} D(p, q) \tag{2.3}$$

$$V(M) = \bigcup_{p,q \in M, p \neq q} \overline{VR(p, M)} \cap \overline{VR(q, M)} \tag{2.4}$$

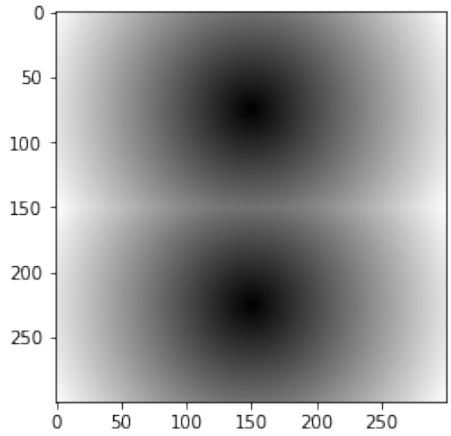


Obrázek 2.5: Rozdelenie dvojrozmerného priestoru na Voroného diagram podľa definície pre $p, q, r \in M$

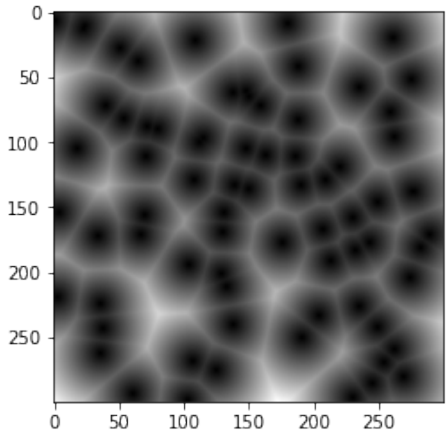
Voroného diagramy majú široké využitie v rôznych oblastiach. V oblasti biológie môžu byť využité na reprezentáciu tvaru buniek[6]. V meteorológií sa Voroného diagramy (tiež označované ako Thiessenove polygóny[1]) dajú použiť na aproximáciu množstva zrážok v oblastiach s malým množstvom hydrometeorologických staníc[7].

Pre túto prácu je relevnatná oblasť počítačovej grafiky, v ktorej diagramy možno využiť na algoritmické vytváranie mozaík na obrázkoch. V obrázku sa vyberie množina bodov diagramu a každý región sa vyfarbí podla farby príslušného bodu. Pri procedurálnom generovaní sú diagramy aplikované pri rozdeľovaní priestoru, kde každá bunka môže mať čiastočne upravené pravidlá alebo algoritmus na generovanie objektov v danej oblasti.

Worleyho šum je podobne ako Perlinov šum (2.1) metóda pomocou ktorej sa dajú generovať textúry. Proces vytvárania Worleyho šumu spočíva v rozptýlení *feature points* v priestore $\mathbb{R}^3$. Ak nieje špecifikované inak, tieto body sú náhodne rozptýlené v priestore. Princíp vytvárania je založený na Voronoi diagramoch (2.2).



(a) Príklad Worleyho šumu v dvojrozmernom priestore pre 2 body



(b) Worleyho šumu s náhodným počtom bodov

Obrázek 2.6: Worleyho šum s bázovou funkciou pre prvý najbližší bod

Nech M je množina *feature points* o veľkosti n, kde každý bod má svoju špecifickú polohu v priestore. Pre akúkoľvek pozíciu x v $\mathbb{R}^3$ existuje bod, ktorý je k tejto pozícií bližsie ako ostatné body. Definujeme teda funkciu $F_1(x)$ ako vzdialenosť od x k najbližšiemu bodu z množiny M.

Ako sa x mení, F_1 sa mení hladko nakoľko vzdialenosť medzi vybranou pozíciou a statického bodu sa mení hladko. Napriek tomu – na špecifických vrcholových pozíciach – vzdialenosť pozície x bude ekvivalentná medzi dvomi bodmi z množiny M(obr. 2.6a). Na tomto mieste je hodnota $F_1(x)$ stále definovaná, nakoľko hodnota je rovnaká bez ohľadu na to, ktorý feature point sa použije (z dvoch najbližších). Funkcia $F_2(x)$ je definovaná ako vzdialenosť pozície x a bodu z množiny M, ktorý je druhý najbližší k tejto pozícií. Podobne vieme definovať $F_n(x)$ ako vzdialenosť medzi x a n-tým najbližším bodom[9].

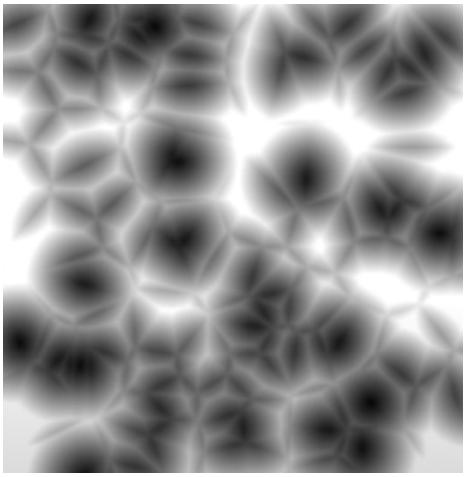
Funkcie F majú zaujímavé vlastnosti:

- F_n je vždy spojitá
- F_n sú neklesajúce
- Z definície funkcie vyplýva, že $F_n(x) \leq F_{n+1}(x)$

Obecne by sa dal výpočet pre n rozmerný priestor vyjadriť nasledovne:

$$F(coord) = min(d(coord, p) \tag{2.5}$$

kde *coord* je n-rozmerný vektor udávajúci pozíciu v n-rozmernom priestore (napr. v trojrozmernom by *coord* symbolizovala x, y, z a podobne), $d(coord, p)$ je funkcia ktorá vypočíta vzdialenosť k p-tému najbližšiemu bodu.



Obrázek 2.7: Worleyho šum s bázovou funkciou pre druhý najbližší bod

Na výpočet vzdialenosti sa bežne používa euklidovská vzdialenosť. Pri vyššom počte dimenzií je však možné použiť aj iné spôsoby, ako napríklad:

- Manhattanská vzdialenosť, ktorá je počítaná v osách pri pravých uhloch. Pre n-rozmerný priestor je obecné vyjadrenie 2.6

$$M_d(p1, p2) = |x2 - x1| + |y2 - y1| + ... + |n2 - n1| \tag{2.6}$$

- Chebysheova vzdialenosť, ktorá vzdialenosť definuje ako maximum medzi vzdialenosťami v jednotlivých dimenziách 2.7

$$C_d(p1, p2) = max(|x2 - x1|, |y2 - y1|, ..., |n2 - n1|) \tag{2.7}$$

Zmena algoritmu na počítanie vzdialenosti môže viest k rôznym variáciam vzorov vo výslednom šume generovaného Worleyho metódou.

Z definíce vyplýva, že bázová funkcia $F_1(x)$ vytvára textúry ktoré pripomínajú Voroného diagramy2.2. Tvary textúr su ovplyvnené tým aká bázová funkcia sa použije. V prípade 2. najbližšieho bodu by sa dalo uvažovať o textúrach pripomínajúcich kryštály(obr. 2.7). V počítačovej grafike sa Worleyho šum dá využiť napiklad na znázornenie vlnenie vodnej hladiny pri daždi[3].

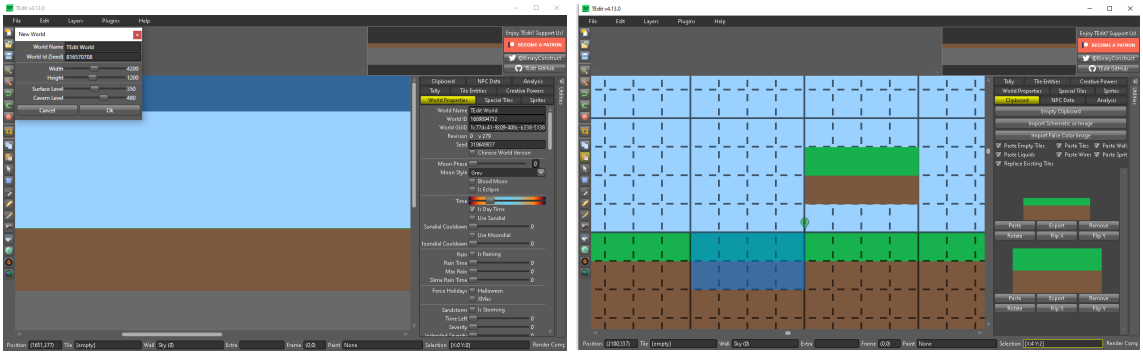
2.3 Editory

Herné editory sú softwarové nástroje, ktoré umožňujú vývojárom hier vytvárať a upravovať materiály ako napríklad levely, charaktery alebo herné objekty. Vytváranie mapy v modernej dobe prebieha často procedurálne. Jedna z najpopulárnejších[2] hier je Terraria[3], ktorá takisto používa na generovanie povrchu procedurálne metódy. Pre úpravy herných úrovní v Terrari existuje veľa externých editorov, ako sú napríklad TEdit[4] alebo TerraMap.

Tieto editory disponujú možnosťou kompletnej úpravy (obrázok 2.8) hernej úrovne, ako aj napríklad pridávanie NPC. Jednou unikátnou vlastnosťou editoru TEdit je vytváranie

[2]https://en.wikipedia.org/wiki/List_of_best-selling_video_games
[3]https://terraria.org/
[4]https://forums.terraria.org/index.php?threads/tedit-terraria-map-editor.43822/

vlastných typov biómov alebo nových druhov blokov. Mapu je možné priblížiť alebo oddialiť, čo bežne hra pri hraní neposkytuje. V iných editoroch je možné zobraziť napríklad detailný popis bloku, ako je napríklad jeho dátová reprezentácia.

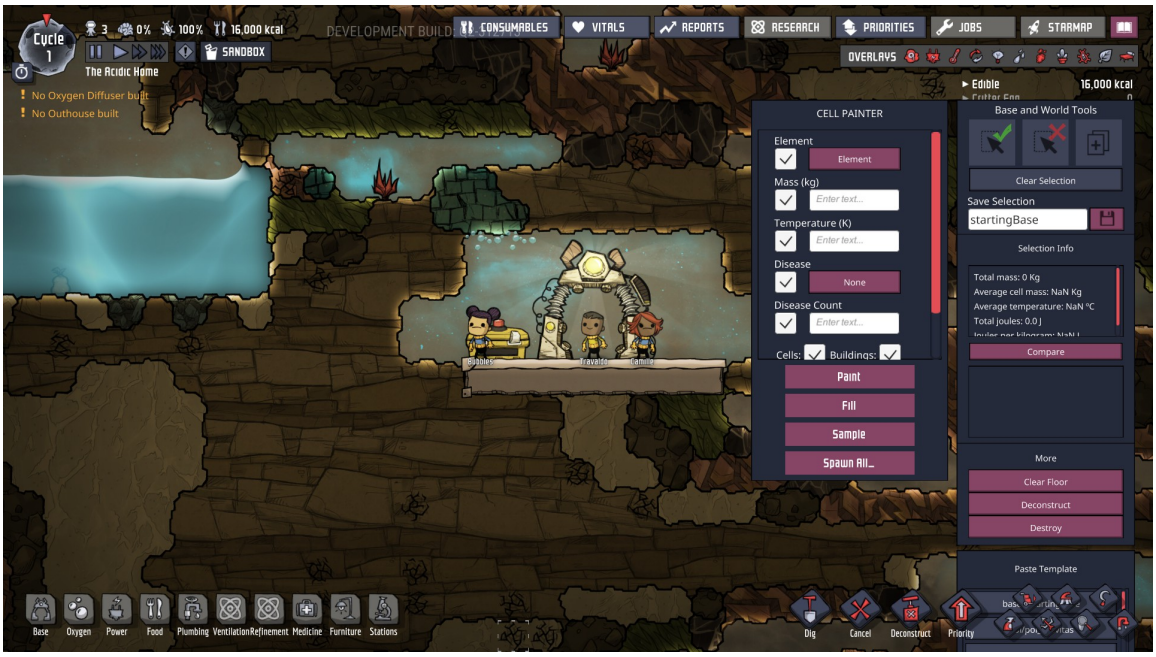


(a) Vytváranie novej mapy v editore so vstupnými parametrami

(b) Náhľad na mapu s detailami a možnosťami úprav

Obrázek 2.8: Ukážka editoru TEdit a jeho možností úprav hernej úrovne

Editory herných úrovní sú často aj súčasťami hier, ako napríklad v her Oxygen Not Included[5]. To poskytuje používateľom priamo experimentovať s jednotlivými charakteristikami mapy a dynamicky ju upravovať počas hrania, ako na obrázku 2.9.



Obrázek 2.9: Interaktívny editor v hre Oxygen not Inlcuded

[5]https://www.klei.com/games/oxygen-not-included

2.4 Herný engine

Engine je software navrhnutý a prispôsobený na vývoj video hier. Herný engine obecne poskytuje vývojárom kolekciu nástrojov pre tvorbu rôznych aspektov hry ako sú herné mechaniky, fyzika alebo grafika. Jednou z hlavných výhod použitia enginu je, že umožňuje vývojárom sústrediť sa na herný obsah namiesto programovania na nízkej úrovni, pretože sa stará o veľa základných technológií potrebných pre vývoj hier, ako sú grafické vykresľovanie, fyzikálna simulácia a sieťovanie. Herné enginy sa líšia vzhľadom na vlastnosti, jednoduchosť použitia a podporované programovacie jazyky. Niektoré herné enginy sú navrhnuté špecificky pre určité druhy hier, ako sú strelecké alebo platformové hry, zatiaľ čo iné sú univerzálne. Medzi najpopulárnejšie[6] herné enginy patria:

- Unity engine - Vývojári môžu používať vizuálne nástroje na tvorbu herných objektov, levelov a iných herných prvkov, a to aj bez znalosti programovania. Umožňuje tvorbu hier v rôznych žánroch, ako sú napríklad arkády, adventúry, RPG, simulátory a mnoho ďalších. Engine poskytuje množstvo nástrojov a funkcií, ktoré vývojárom umožňujú vytvárať rôzne herné mechanizmy, grafiku, zvuk a fyziku.[7]

- Unreal engine - výkonný herný engine, ktorý ponúka vývojárom množstvo nástrojov a funkcií na tvorbu hier. Je populárny medzi hernými vývojármi, ktorí sa sústreďujú na tvorbu hier s akčným žánrom a first-person shooter (FPS) hry. Unreal Engine ponúka mnoho funkcií, ktoré sú špeciálne navrhnuté pre tieto žánre hier, ako sú napríklad nástroje na tvorbu AI nepriateľov, animácií pohybu a bojových mechanizmov.[8]

- CryEngine - Jednou z hlavných výhod CryEngine je jeho vysoká kvalita vizuálu. Engine poskytuje pokročilé nástroje na tvorbu 3D modelov, animácií, svetiel, textúr a zvukových efektov (obrázok 2.10), ktoré umožňujú vytvárať hry s vysokou kvalitou grafiky a vizuálnym zážitkom.[9]

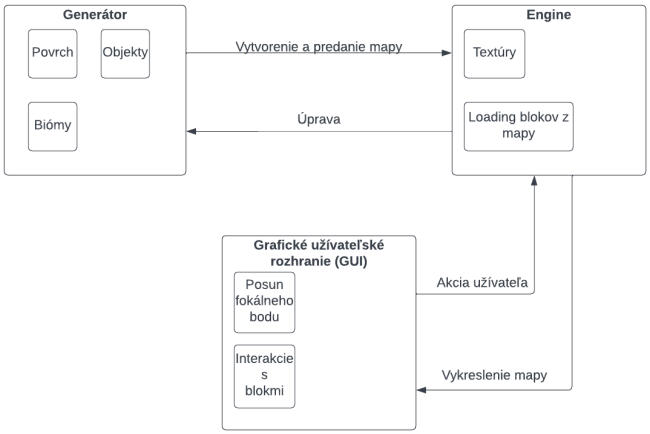
Obrázek 2.10: Video hra Crysis 3[10] používajúca CryEngine.

[6] https://www.incredibuild.com/blog/top-gaming-engines-you-should-consider
[7] https://stepico.com/blog/why-is-unity-the-best-game-engine-pros-and-cons/
[8] https://starloopstudios.com/unreal-engine-5-features-and-how-it-can-improve-gaming/
[9] https://www.cryengine.com/news/view/why-developers-choose-cryengine

Kapitola 3

Návrh

Editor sa skladá z dvoch primárnych častí. Modul generátoru, ktorý zaisťuje základné reprezentácie mapy a editorový modul na real-time rendering mapy do grafického užívateľského rozhrania (ďalej ako GUI) pomocou enginu. Schéma návrhu jednotlivých častí je zobrazená spolu s interakciami medzi jednotlivými časťami na obrázku 3.1 Vzhľadom na to, že sa



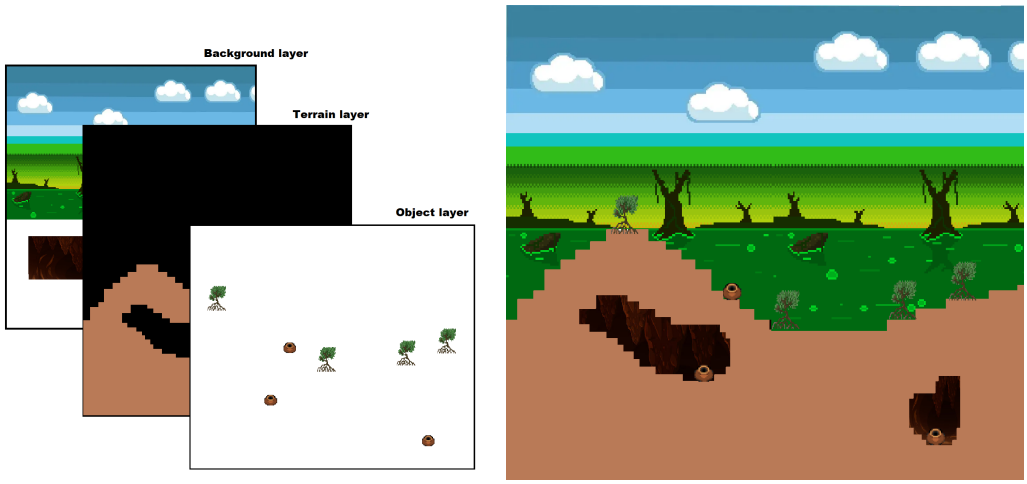
Obrázek 3.1: Schéma editoru. Celá štruktúra má dve hlavné časti (generátor a engine), ktoré sú modulárne. Výstup z enginu je vedený do jednoduchého GUI

práca zaoberá primárne generovaním, je dôraz kladený hlavne na modularitu. Obe hlavné časti sú vo veľkom okruhu rozšíriteľné z hľadiska generačných algoritmov, ale aj textúr a objektov. Pri zobrazovaní mapy je možné definovať vlastné rozmery jednotlivých objektov v blokoch.

3.1 Návrh editorovej časti

Editorová časť pozostáva z načítavania textúr a assetov do enginu ktorý ich aplikuje na mapu a renderuje do GUI. Engine generátoru primárne slúži na dynamické načítavanie jednotlivých layerov mapy z pamäte a rendering ich textúr do GUI. Engine je schopný príjmať vstupy z GUI ako sú napríklad pohyb v mape (zmena fokálneho bodu), úprava blokov alebo block scaling. V pamäti sa vždy nachádza aktuálny stav mapy. Okrem renderingu sa engine stará aj o samotné vytvorenie textúr.

Spojením všetkých vrstiev ako na obrázku 3.2 sa vytvára scéna ktorá je renderovaná do GUI. Engine je schopný jednoduchých úprav v mape ako nahradenie blokov v mape, pridanie a odstránenie pozadí a objektov. Vďaka týmto funkciám je editor čiastočne nezávislý od generátora - mapa dá načítať z lokálneho úložiska v počítači. Jednotlivé možnosti



(a) Princíp spájania vrstiev do finálnej scény (b) Návrh finálnej scény spojením všetkých vrstiev procedurálnej mapy.

Obrázek 3.2: Princíp a demonštrácia vytvorenia scény z jednotlivých vrstiev mapy.

úprav mapy sú rozdelené do dvoch štádií. Prvé je preprocessing, v ktorom je zobraziteľná povrchová vrstva mapy. V tomto štádiu môže používateľ využívať funkcie ukladania blokov do mapy, priblížiť alebo oddialiť mapu a označovať zhluky blokov, ktoré môže následne vymazať alebo skopírovať a uložiť na zvolené miesto. Druhé štádium je postprocessing. Mapa tu obsahuje všetky spomenuté vrstvy a vykresľuje sa do GUI. Okrem úprav v preprocessingovom štádiu tu má používateľ možnosť robiť rovnaké úpravy s objektami (ukladanie, selekcia, kópie, vymazávanie).

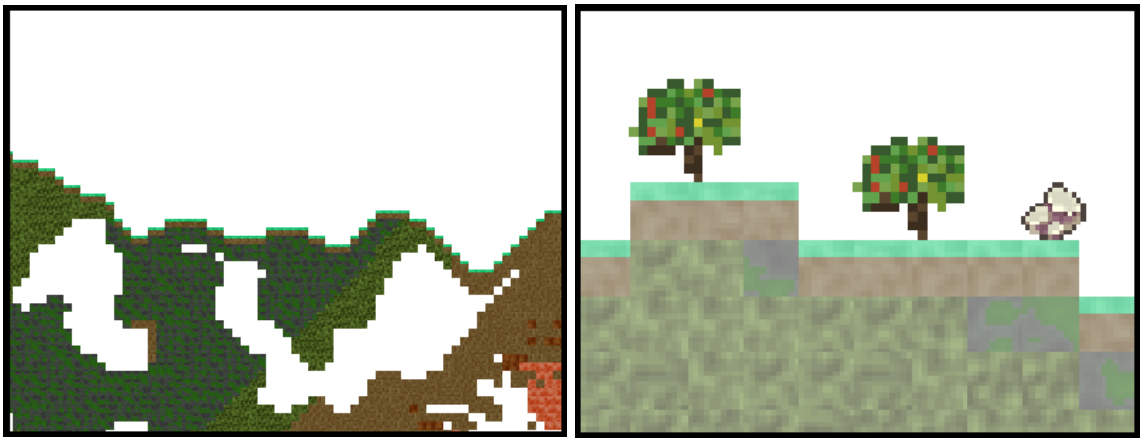
3.2 Návrh generátoru

Generátor slúži na vytvorenie a dynamické upravovanie mapy v pamäti. Mapa je reprezentovaná v pamäti spôsobom aby sa v nej úpravy vykonávali jednoducho. Pre prehľadnosť a jednoduchosť je teda celá reprezentácia mapy rozdelená do troch vrstiev. Povrchová vrstva reprezentuje povrch mapy ako jednotlivé bloky. Objektová vrstva reprezentuje neblokové objekty ako sú napríklad stromy, skaly a podobne. Treťou vrstvou je pozadie mapy. Ďalej sú detailnejšie popísané jednotlivé vrstvy z ktorých sa mapa skladá.

Povrchová vrstva

Pri 2D mape je povrch najdôležitejšou časťou (obrázok 3.3a). Na základe povrchu a jeho rozdelenia sa do mapy môžu pridávať jednotlivé objekty a priraďovať sa pozadia. Keďže táto časť veľmi ovplyvňuje ostatné vrstvy, je dôležité aby sa mapa jednoducho upravovala a reprezentovala v pamäti. Generovanie sa riadi na základe vstupných parametrov od užívateľa. Vzľadom na procedurálne prvky je poskytnutá vysoká granularita vstupných parametrov. Povrch obsahuje prvky ako sú jaskyne, rozdelenie do biómov alebo špeciálne

povrchové štruktúry. Bióm je reprezentovaný ako zhluk blokov v špecifickej oblasti. Generovanie zahŕňa aj pretváranie povrchu v špecifických biómoch. Pre jaskyne sú dostupné dve metódy na vytvorenie odlišných a procedurálnych jaskýň v jednotlivých biómoch. Jaskynné algoritmy sú takisto modulárne a nezávislé od typu mapy.



(a) Ukážka povrchovej vrstvy vykreslenej do okna.

(b) Ukážka objektovej vrstvy položenej na (transparentnej) povrchovej vrstve.

Obrázek 3.3: Návrh vizualizácie jednotlivých vrstiev

Objektová vrstva

Objektová vrstva slúži na reprezentáciu objektov v mape (obrázok 3.3b). Na základe rozdelenia povrchovej vrstvy do biómov má každý bióm vlastnú kolekciu objektov, ktoré sa v ňom môžu nachádzať. Objekty sa v mape generujú na základe špecifických pravidiel, ktoré sa odvíjajú od typu mapy. Ak by bola mapa plne vygenerovaná v podzemí, mohlo by dôjsť k nedostatku generovania povrchových objektov, ako sú napríklad stromy. Preto sú tieto pravidlá navrhnuté tak, aby boli všetky objekty zahrnuté, nezávisle od typu mapy. Každý objekt má predeterminovanú veľkosť, ktorá je upraviteľná formou vstupných parametrov.

Background vrstva

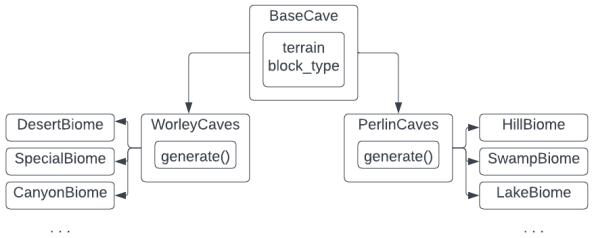
Reprezentácia background vrstvy spolupracuje s povrchovou vrstvou. Primárne táto vrsva slúži na vyplnenie prázdneho priestoru za povrchovou vrstvou. Každý bióm má svoje špecifické pozadie. Pozadia sa môžu odlišovať aj od výšky v mape, alebo toho či sa užívateľ pozerá na jaskyňu alebo iný prázdny priestor.

3.3 Dátové štruktúry

V tejto sekcii sa nachádza návrh dátových štruktúr jednotlivých častí (generátoru a editoru). Ďalej sú rozobraté štruktúry modulov, ktoré časti používajú, ich vstupy a výstupy. Nakoniec je popísaná vzájomná komunikácia mezdi editorom a generátorom.

3.3.1 Moduly generátoru

Generátor je hlavná časť vytvárania mapy ako dátovej reprezentácie mapy. Hlavnými súčasťami sú moduly pre biómy, a jaskynné algoritmy. Každý bióm je reprezentovaný svojou



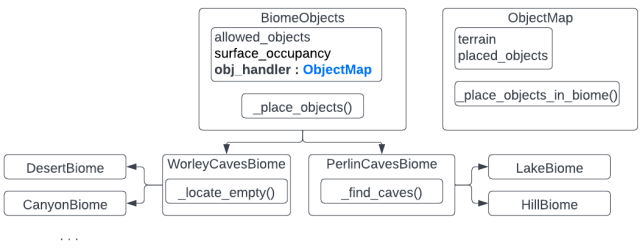
Obrázek 3.4: Objektová reprezentácia štruktúry modulov na generovanie jaskýň v biómoch.

špecifickou triedou, ktorá dedí od obecnej triedy biómu. Štruktúra biómov, ktorá je definovaná na obrázku 3.6, slúži na transformáciu povrchu mapy. Moduly jaskýň sú definované podľa algoritmu (transfromácia podľa Perlinovho alebo Worleyho šumu), ktorý využívajú. Každý bióm používa svoj špecifický algoritmus na vytvorenie jaskynných systémov v ňom podľa schémy 3.4. Na základe spomenutých algorimtov sú rozdelené aj moduly na generovanie objektov v jednotlivých biómoch (obrázok 3.5), keďže v rozličných typoch jaskýň sa definujú rozličné pravidlá na umiestňovanie objektov. Štruktúra objektov je popísaná v sekcií 3.3.2, v časti editoru a textúrovania.

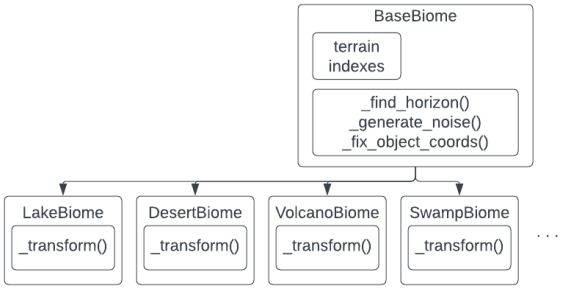
Vstupy a výstupy

Vzhľadom na to že sa jedná o procedurálne generovanie, vstupy a výstupy pre jednotlivé moduly a ich súčasti sú determinované na základe parametrov pri inicializácií generátoru. Generátor môže fungovať ako nezávislá časť a príjmať vstupné parametre z poskytnutého *map_config.json* súboru, alebo príjmať vstupy od užívateľa prostredníctvom editoru (sekcia 4.3). Výstupom generátoru je dátová štruktúra reprezentujúca mapu vo všetkých vrstvách (sekcia 4.2). Obecne sa jedná o príečinok s názvom, ktorý bol definovaný vo vstupných parametroch. Tento priečinok obsahuje súbor s dátovou reprezentáciou povrchu mapy a súbor s popisnými parametrami potrebnými na rekonštrukciu mapy.

Jednotlivé možnosti vstupných parametrov sú dané podľa počtu krokov, ktoré sa majú vykonať na generovanie. Významy jednotlivých parametrov sú uvedené v sekcií 4.2.1. Ge-



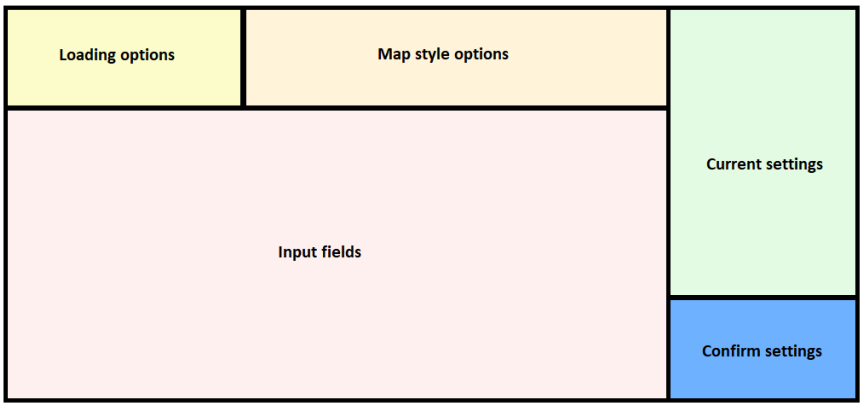
Obrázek 3.5: Štruktúra modelov generujúcich objekty v biómoch na základe typu ich jaskýň.



Obrázek 3.6: Objektová reprezentácia štruktúry biómových modulov.

nerovanie objektov nastáva po uzamknutí zoomu v editore (sekcia 4.4.) Každý objekt je uložený ako obrázok v úložisku assetov. Užívateľom definované rozmery objektov sa môžu nachádzať v konfiguračnom súbore *object_config.json*. V opačnom prípade sú použité preddefinované rozmery.

3.3.2 Moduly editoru



Obrázek 3.7: Layout menu pre generovanie novej mapy. *Inputfields* je placeholder polí pre možné vstupy od užívateľa v závislosti od vybratého typu mapy (ako v sekcii 3.3.1, vstupy z *json* súboru.)

Na rozdiel od generátoru, editor nedokáže pracovať samostatne. Editor sa primárne zaoberá vykresľovaním mapy do okna a možnosťou úpravy mapy užívateľom formou interakcie s oknom. Spojenie týchto dvoch častí vytvára engine, ktorý beží v reálnom čase. Engine používa dve hlavné súčasti:

- Moduly menu, ktoré poskytujú používateľovi interakcie pred načítanim alebo vytvorením procedurálnej mapy
- Modul EditorMap a jeho súčasti na výpočet a transláciu koordinátov z procedurálnej mapy do okna. Okrem toho sa stará o pomocné operácie pri úpravách mapy.

Prostredníctvom menu vie užívateľ poskytnúť vstupné parametre na vytvorenie mapy. Layout (obrázok 3.7) je rozdelený podľa krokov ktoré sa použijú na vytvorenie procedurálnej

mapy. Používateľovi je poskytnutá variácia parametrov ktoré môže využiť. S narastajúcim počtom úprav mapy sa takisto zvyšuje množstvo dostupných vstupov. Z nich je napríklad možné upraviť interval šumovej funkcie, zahrnúť alebo vynechať špecifické biómy a upraviť rozmery objektov.

Editor a textúrovanie

Účel modulu EditorMap je načítať daný výsek mapy z pamäte, prideliť každej dlaždici jej textúry a vrátiť ich enginu na vykreslenie spolu s objektami, ktoré by sa mali na mape nachádzať. Modul pracuje s dátovou reprezentáciou mapy, ktorá je výsledkom generátoru. Kritickými súčasťami sú moduly pre textúry a objekty. Keďže unikátnosť dlaždíc je reprezentovaná celočíselnou hodnotou, textúry sú uložené v triede ktorá reprezentuje dlaždicu s danou hodnotou. Vzhľadom na podporu približovania a oddialenia náhľadu na mapu v okne obsahuje každá trieda textúry vo všetkých dostupných veľkostiach. Každý blok má viacero variánt textúry, ktorá je generovaná procedurálne. Editor podporuje úpravy mapy používateľom v objektovej aj povrchovej vrstve.

Vstupy a výstupy

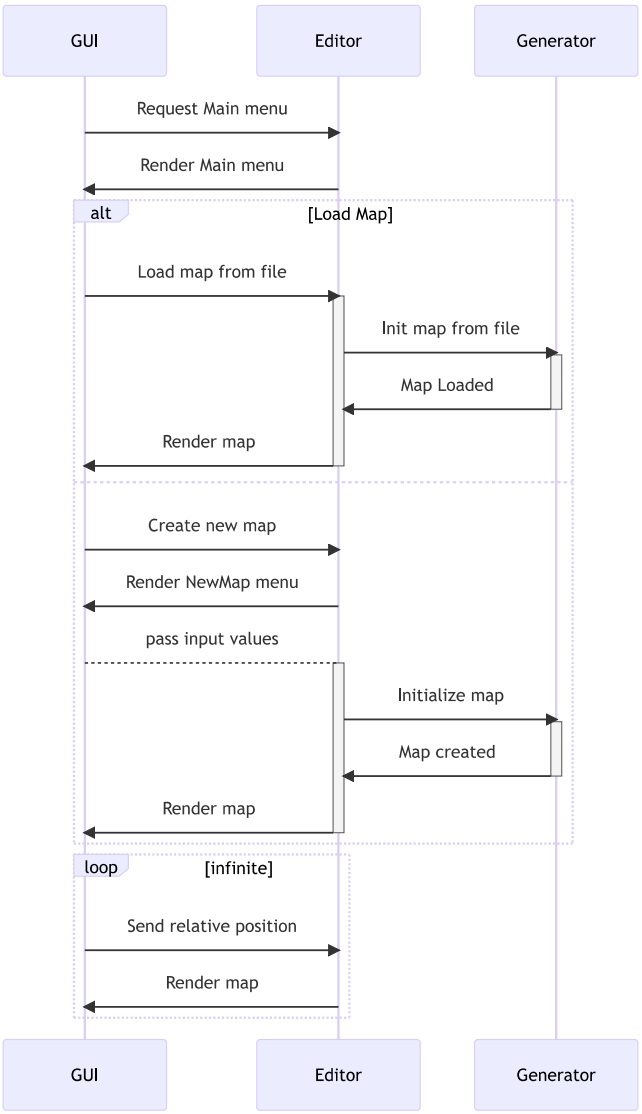
Editor primárne očakáva vstupy od používateľa formou vypĺňania vstupných polí alebo selekcie checkboxov. Alternatívne môže editor použiť vstupný súbor na konfiguráciu prvotnej mapy. Na generovanie editor vždy používa konfiguračný súbor objektov ak je dostupný. Výstupom je vizuálne zobrazenie mapy v okne vo vybranej oblasti. Editor takisto poskytuje možnosť ukladania mapy v jednotlivých štádiách (popísané v 4.4) za pomoci generátoru a jeho súčastí. Mapa sa dá načítať do editoru, ak sa nachádza v úložisku máp.

3.3.3 Komunikácia modulov

Schéma komunikácie je zobrazená sekvenčným diagramom na obrázku 3.8. Užívateľ má možnosť načítať si existujúcu mapu zo súboru alebo vytvoriť novú mapu. V prípade načítania sa predpokladá že vstupný adresár obsahuje všetky nevyhnutné dáta na rekonštrukciu mapy. Pri zvolení vytvorenia novej mapy je uživateľovi v GUI zobrazené menu, v ktorom sú mu poskytnuté polia na vstup inicializačných hodnôt. Po zadaní hodnôt a zvolení možnosti *Create* sa editoru signalizuje, že má pozbierať poskytnuté dáta a pomocou nich inicializuje generátor. Ten si na na základe poskytnutých parametrov vytvorí a upraví mapu (zatiaľ ešte bez objektov) a po dokončení úprav poskytne jej dátovú reprezentáciu Editoru. Editor si z mapy vyberie relevantný výsek mapy, každej dlaždici priradí jej textúru, pozíciu v okne a tieto textúry renderuje do GUI. Po tejto časti nasleduje real-time generácia framu na základe relatívnej pozície v mape formou komunikácie medzi GUI a Editorom. GUI poskytne pozíciu a Editor renderuje povrch.

Výstupom tejto sekvencie je mapa, ktorá je priehliadateľná, má možnosť priblíženia a oddialenia ale neobsahuje objekty. Aktuálny stav mapy povoľuje úpravy blokov (presúvanie, kopírovanie, vymazávanie a podobne). Od momentu čo je mapa renderovaná sa začína rozlišovať absolútna pozícia bloku a relatívna pozícia fokálneho bodu v mape. Absolútna pozícia značí indexy dlažíc mapy, zatiaľ čo relatívna pozícia značí priamo pixel v mape. Translácia súradníc je popísaná v sekcií 4.3.1. Po dokončení úprav má užívateľ možnosť zamknúť si zoom mapy a vygenerovať objekty v mape.

Po ukončení generovania objektov sa v okne vykresľuje finálna verzia mapy. Užívateľovi okrem možností úpravv povrchu pribudnú možnosti úprav objektov. V prípade použitia generátoru ako samostatného modulu je stále možné generovať objekty do mapy, aj bez interakcie s editorom. Generátor za takýchto okolností použije defaultný zoom.



Obrázek 3.8: Sekvenčný diagram komunikácie jednotlivých súčastí na vytvorenie alebo načítanie mapy.

Kapitola 4

Implementácia

V tejto časti je detailne popísaná implementácia modulu generátoru a vykresľovania.

4.1 Nástroje

Práca je implementovaná v jazyku Python. Na vytváranie a základné úpravy mapy je použitá knižnica numpy, ktorá zabezpečuje rýchle a efektívne operácie s n-rozmernými maticami. Vytváranie procedurálnych prvkov zabezečujú šumové funkcie z knižnice noise[1]. Úprava a filtrovanie jednotlivých šumov je dosiahnuté použitím obrázkových filtrov z knižnice scikit-image[2] a scipy[3]. Na vykresľovanie je použitá knižnica pygame[4]. Validácia vstupných parametrov z $json$ súborov je vykonaná pomocou knižnice marshmallow[5].

4.2 Generovanie sveta

V tejto sekcii je popísaný postup vytvárania 2D mapy a jej dátovej reprezentácie. Primárnym vstupným súborom pre generátor je $map\_config.json$. Jeho štruktúra je popísaná nasledovne:

- $width, height$ - povinné parametre, definujú rozmery mapy v blokoch. Vzhľadom na možnosti zoomu je sú minimálne hodnoty nastavené na veľkosť okna, do ktorého sa vykresľuje.
- $seed$ - použije sa pri metódach ktoré používajú náhodne generované čísla. Ovplyvňuje hlavne rozmiestnenie biómov a tvar terénu. Ak nieje poskytnutý tak sa použije náhodná hodnota.
- $file\_name$ - názov súboru, do ktorého sa má uložiť dátová reprezentácia mapy. V prípade že nieje poskytnutý sa použije aktuálny dátum.
- $terrain$ - parameter, ktorý upravuje interval šumovej funkcie použitej na vytvorenie základného povrchu. Default je nastavený na 1.

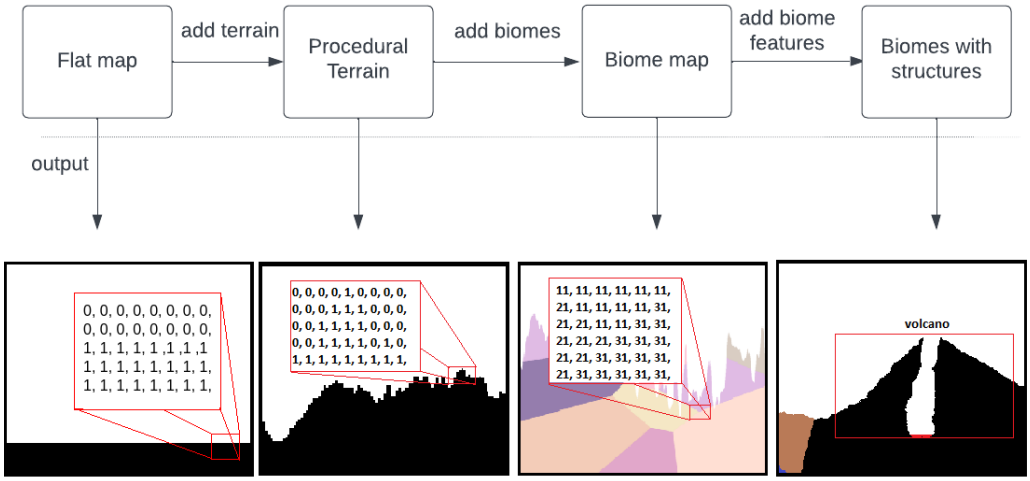
[1] https://pypi.org/project/noise/
[2] https://scikit-image.org/
[3] https://scipy.org/
[4] https://www.pygame.org/news
[5] https://marshmallow.readthedocs.io/en/stable/

- *smoothing* - slúži na vyhladzovanie povrchu po vygenerovaní. Poskytnutá hodnota udáva počet blokov, ktorý sa priemeruje pri použití mean-filteringu na povrch. V prípade 0 je vyhladzovanie vynechané.
- *include_biomes* - slovník hodnôt typu *biome_name* : *biome_description*, kde popis je objekt s kľúčami *include*, ktorý symbolizuje či sa bióm má alebo nemá nachádzať na mape a voliteľným kľúčom *position*, ktorá špecifikuje feature point pre daný bióm. V prípade, že sa tento parameter nepoužije sa generujú všetky biómy s náhodnými pozíciami feature points.
- *transf_biomes* - podobne ako *include_biomes* s rozdielom, že popis je objekt s kľúčom *transf*, ktorý symbolizuje dodatočnú transformáciu biómov. V prípade že sa tento parameter nepoužije sa transformujú všetky biómy.
- *scale* - nepovinný parameter. V prípade použitia generátoru ako samostatného modulu sa tento parameter použije na generovanie objektov a pozadí.

Všetky parametre budú popísané v častiach, ktoré sa ich týkajú. Sekundárnym vstupným súborom je *objects_config.json*, ktorý obsahuje používateľom definované rozmery špecifických objektov. Jeho štruktúra obsahuje páry *object_name* : *size*, kde rozmer je pár hodnôt reprezentujúci výšku a šírku v blokoch.

4.2.1 Generovanie povrchu

Všetky inicializačné parametre sú získané z menu (sekcia 3.3.3) v prípade editoru, alebo zo vstupného json súboru na základe validácie (sekcia 3.3.1.). Generovanie základnej mapy prebieha v štyroch krokoch. Je dôležité poznamenať, že tieto kroky sú na sebe závislé podľa schémy na obrázku 4.1.



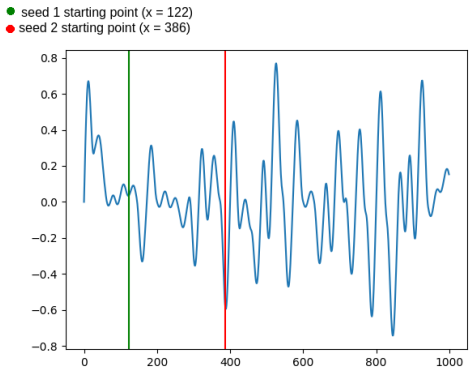
Obrázek 4.1: Postup vytvárania povrchu v jednotlivých krokoch. Výstup je dvojrozmerné pole čísel, ktoré reprezentujú typ bloku. Bloky sú farebne rozlíšené

Generovanie prvotného povrchu

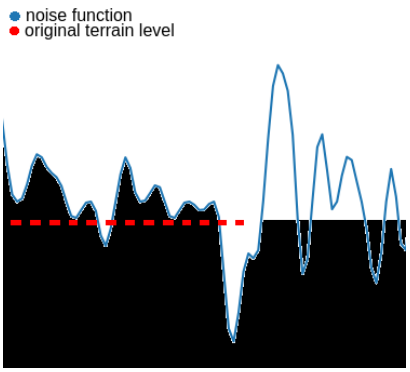
Prvý krok je vytvorenie mapy. Na reprezentáciu mapy je použité dvojrozmerné pole. V pamäti sa vytvorí pole o veľkosti $height \times width$ (inicializačné parametre). Ak bol poskytnutý aj parameter *horizon*, od danej výšky (indexované z hora dole) je vytvorený základný povrch mapy. Jednotlivé dlaždice sú reprezentované ako celé číslo na determináciu obsadenosti pozície.

Aplikovanie procedurálneho povrchu

V prípade že pri generovaní prvotného povrchu nebol poskytnutý parameter *horizon* má táto časť minimálny dopad na aktuálny vzľad mapy, nakoľko sa zameriava na terraformáciu prechodu medzi prázdnym priestorom a plochým terénom. Na vytvorenie povrchu bol využtý PN. Vstupný parameter *terrain* umožňuje upraviť interval šumovej funcie, čo vedie k rozličným tvarom povrchu z hľadiska vertikálneho alebo horizontálneho členenia. Keďže metóda na generovanie šumu nepríjma seed, generuje stále rovnakú funkciu. Procedurálnosť je riešená posunutím v x-ovej osi, v závislosti od inicializačného parametru *seed*. Aj napriek tomu je výsledkom funkcia 'orientovaná' okolo priamky $f(x) = 0$. Na vytvorenie prirodzeného rastu alebo klesania je okrem aplikovania šumu nutné pričitať náhodnú hotnotu v rozpatí $(-2, 2)$. Princíp transformácie povchu je demonštrovaný na obrázku 4.2.



(a) Výber začiatku šumovej funkcie na základe hodnoty seedu.



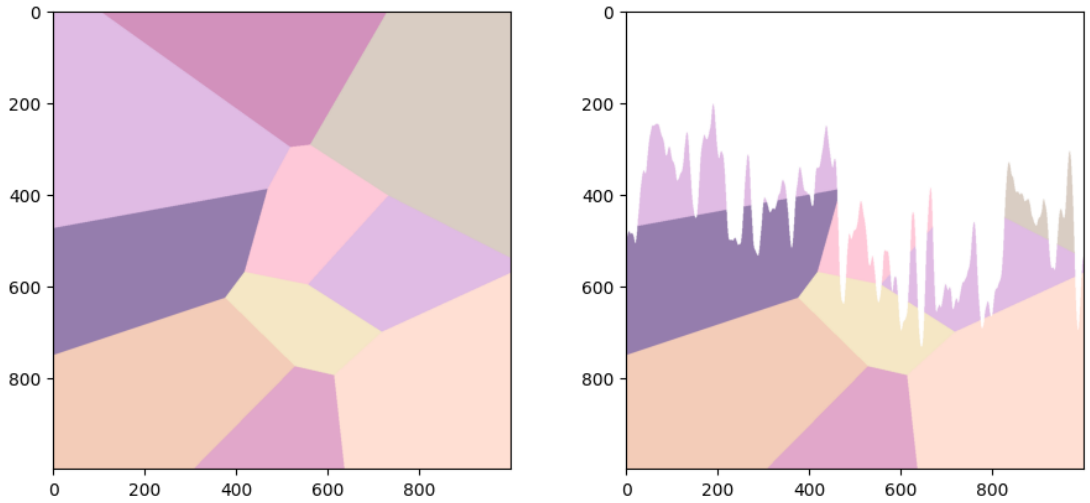
(b) Princíp aplikovania šumovej funkcie na plochý terén

Obrázek 4.2: Proces transformovania plochého terénu na procedurálny.

Rozdelenie do biómov

Biómy sú špecifické oblasti v mape. Odlišujú sa od seba rozdielnou hodnoutou v poli, ktoré mapu reprezentuje. Princíp vytvárania spočíva v rozdelení mapy na voronoi diagramy (obr. 4.3a). Každému regiónu je pridelená konkrétna celočíselná hodnota ktorá reprezentuje základný identifikačný blok biómu. Po rozdelení sa každý blok, ktorý patrí danému biómu, nahradí hodnotou ktorá mu bola pridelená, s výnimkami prázdenho priestoru ako na obrázku 4.3b.

Hranice regiónov sú transformované z priamok do kriviek použitím šumovej funkcie s dĺžkou odpovedajúcou dĺžke priamky otočenej pod daným uhlom podľa rovníc 4.1 a 4.2,



(a) Vizuálna demonštrácia rozdelenia mapy do voroného diagramov.

(b) Voroného diagramy aplikované na terén mapy, vytvárajúci biómy

Obrázek 4.3: Princíp vytvorenia biómov na mape

kde x_n, y_n sú koordináty šumu a x_b, y_b sú nové koordináty hranice medzi biómami. Uhol θ sa dá jednoducho získať pomocou 4.3, kde r reprezentuje euklidovskú vzdialenosť medzi začiatkom hranice s daným biómom a koncom, a x je šírka plochy, na ktorej sa táto hranica nachádza (obr. 4.4). Bloky, ktoré horizontálne susedia s novým vybraným bodom hranice majú sú prepísané na náhodný bloku z typu blokov susediacich biómov. Dôvod prepisu je premiešanie biómových blokov na vytvorenie prirodzenejšieho prechodu. V pripade mapy s definovanou výškou povrchu sa $featurepoints$, na základe ktorých sa Voronoi diagramy generujú, primárne umiestnujú pod úroveň povrchu na zvýšenie pravdepodobnosti zahrnutia všetkých biómov. Na základe vstupného parametru *include_biomes* je možné špecifikovať, ktoré biómy by mali byť v mape zahrnuté spolu s pozíciu $featurepointu$.

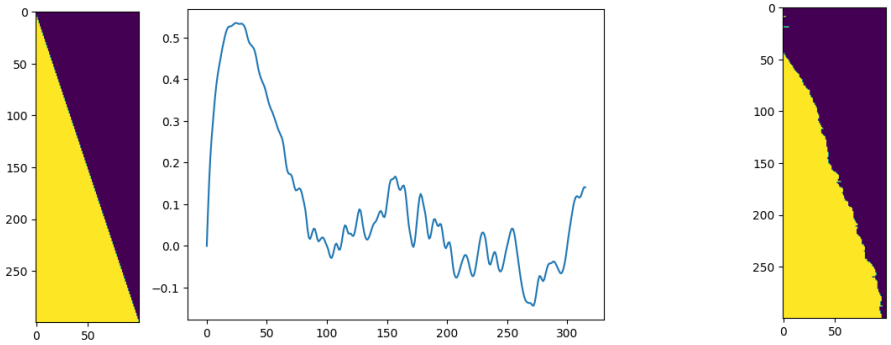
$$x_b = x_n * \cos(\theta) - y_n * \sin(\theta) \tag{4.1}$$

$$y_b = x_n * \sin(\theta) + y_n * \cos(\theta) \tag{4.2}$$

$$\theta = \arccos\left(\frac{x}{r}\right) \tag{4.3}$$

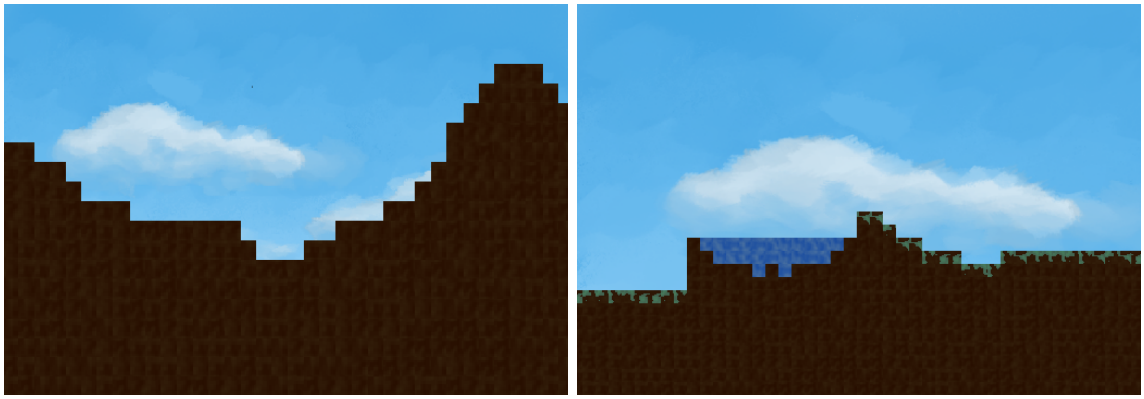
Transformácie jednotlivých biomov

V generátore je definovaných 10 biómov. Niektoré, ako napríklad púštny bióm, netransformujú terén vôbec. Iné, ako napríklad kaňon, upravujú povrch pridávaním štruktúr ako sú sopky, hory, jazerá a pod (príklad na obrázku 4.5). Všetky biómy ktoré transformujú povrch pridávaním štruktúr definujú unikátne pravidlá pre ich generovanie. Tieto pravidlá obecne zahŕňajú položenie, prekrývanie a veľkosť štruktúr v závislosti od veľkosti a typu mapy. Ak mapa nemá vygenerovaný povrch (teda sa nachádza celá pod zemou), táto časť má minimálny dopad. Na základe vstupného parametru *transf_biomes* je možné špecifikovať, ktoré z existujúcich biómov majú prejsť transformáciou terénu. Vzhľadom na procedurálnosť mapy je možné, že vo väčšine máp sa tieto štruktúry nenachádzajú vo veľkom počte



Obrázek 4.4: Transformácia hranice dvoch biómov z priamky (vľavo) pomocou šumovej funkcie (v strede) do výslednej krivky (napravo)

alebo vôbec, pretože ich pravidlá definujú minimálnu veľkosť povrchu terénu na generovanie.



(a) Močiarový bióm pred transformáciou povrchu

(b) Močiarový bióm s transformáciou povrchu a pridaním štruktúr

Obrázek 4.5: Rozdiel medzi netransformovaným močiarovým biómom (vľavo) a transformovaným (vpravo), ktorý obsahuje močiare ako biómovo-špecifické štruktúry

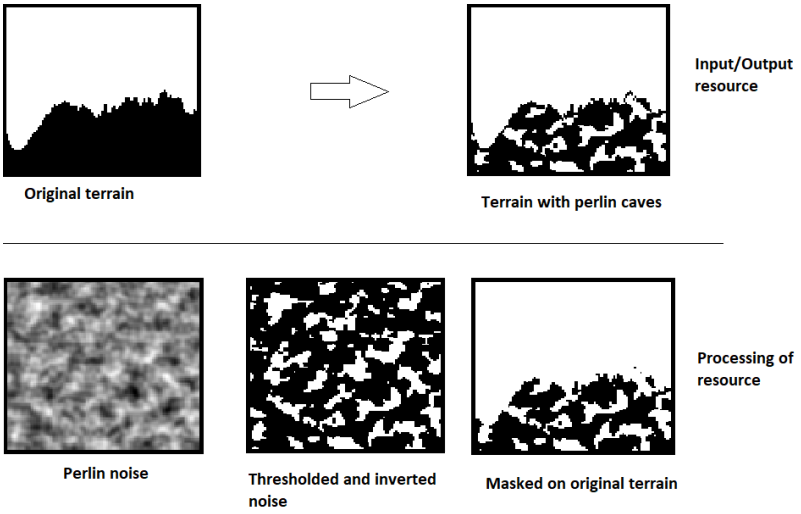
Generovanie jaskýň

Na generovanie jaskynných systémov sú definované dva algitmy. Tie su zaobalené do tried, ktoré ich použijú na transformáciu terrainu. V ďalších kapitolách sa bude referovať terén ako dátová reprezentácia povrchovej vrstvy, a povrch ako reliéf terénu.

Jaskynné systémy založené na PN (ďalej ako PJ) sú rozdelené do dvoch podkategórií:

- *rough* - generuje ostrejšie hrany v jaskyniach a nepravidelnejšie tvary
- *smooth* - generuje hladšie jaskyne, oválnejšie tvary

Rozdiel typu PJ je založený na parametroch šumovej funkcie. Princíp vytvorenia spočíva v generovaní dvojrozmerného šumu o veľkosti terénu v oblasti biómu. Tento šum je nutné spracovať, nakoľko jeho hodnoty sú desatinné čísla ktoré by nereprezentovali žiadny typ bloku. Cieľom je dosiahnuť šumovú funkciu, s celočíselnými hodnotami reprezentujúce blok ktorý identifikuje daný bióm a s nulami, ktoré reprezentujú prázdne miesta. Šum je uložený

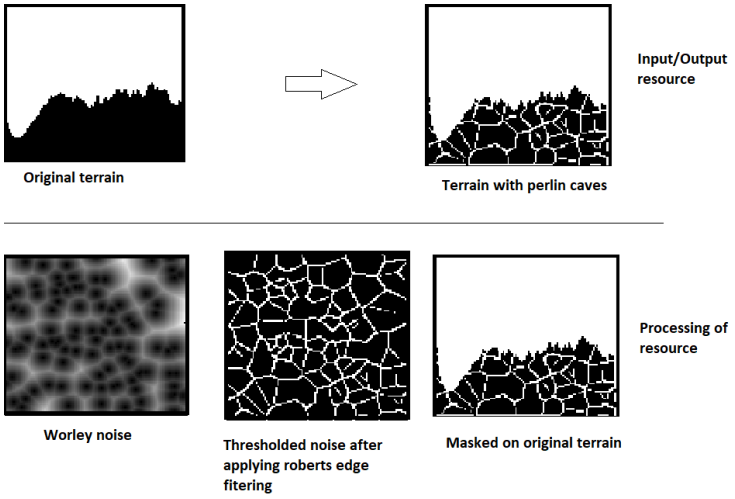


Obrázek 4.6: Vizuálna demonštrácia pridávania PJ.

ako dvojrozmerné pole o veľkosti odpovedajúcej veľkosti biómu, čo poskytuje jednoduchý hromadný prepis hodnôt. Na získanie relevantných hodnôt, ktoré sa dajú transformovať do terénu primomínajúceho jaskyne sa šum prahuje pomocou hodnoty 0.1, takže prepis všetkých hodnôt šumu väčších ako 0.1 je prepísaný na 1 a zvyšok na 0. Táto úprava vytvorí oblasti pripomínajúce jaskyňe tam, kde je hodnota šumu 1. Preto treba tento šum invertovať. Ďalej je treba uvažovať o možnosti, že nie celý bióm sa nachádza pod zemou. Z tohoto dôvodu sú pred aplikovaním šumu na povrch uložené všetky indexy povrchu, ktoré sa vyhľadajú prehľadávaním oblasti biómu zhora nadol, zľava doprava. Pri nájdený prvého nenulového prvku sa uložia indexy bloku a pokračuje sa až do konca biómu. Nakoľko šumová funkcia obsahuje hodnoty 1 tam, kde má transformovaný povrch ostať a 0 tam, kde sa vytvorí prázdne miesto, je nutné ešte všetky nenulové hodnoty prepísať na hodnotu bloku reprezentujúceho bióm. Následne sa vytvorí maska biómu, ktorá v sebe drží indexy všetkých blokov, ktorých hodnot odpovedá hodnote bloku biómu. Táto maska sa aplikuje na šum a hodnoty terénu v bióme sú nahradené šumom. Nakoľko šumová funkcia mohla spôsobiť deštrukciu pôvodného povrchu, na základe uložených indexov blokov povrchu sa terén rekonštruuje. Demonštrácia úpravy biómu je zobrazená na obrázku 4.6.

Jaskyne využívajúce Worleyho šum (ďalej ako WJ) majú komplexnejší proces vytvárania. Na vytvorenie jaskynných systémov sú použité hranice regiónov. Na worleyho šum je aplikovaný robertsov[6] filter na vyhľadanie hrán. Výsledný šum je opäť prahovaný. Následne sa vytvorí maska všetkých bodov, ktoré v šume reprezentujú hrany. Na každý bod je aplikovaný algoritmus, ktorý okolo neho v zvolenom okruhu prepíše hodnoty šumu. Prepis je založený na pravdepodobnosti, ktorá je daná vzdialenosťou od pôvodného bloku, teda čím ďalej sa prepisuje, tým nižšia pravdepodobnosť toho že na danom mieste sa bude nachádzať hodnota 0. Týmto vytvára šum ktorý pripomína tunely. Stavy sú rovnaké ako v prípade použitia PN. Vizuálna demonštrácia úpravy je na obrázku 4.7.

[6]https://homepages.inf.ed.ac.uk/rbf/HIPR2/roberts.htm



Obrázek 4.7: Vizuálna demonštrácia pridávania jaskýň pomocou algoritmu využívajúci worleyho šum.

4.2.2 Generovanie backgoundu

Proces generovania background vrstvy je závislý od biómov a povrchu. Pozadia sú v úložisku assetov ako obrázky. Pre každý bióm existuje pozadie povrchu a pozadie jaskyne, ktorých položenie v mape sa odvíja od najnižšie položeného bloku na povrchu v danom bióme.

Lokálna úroveň jaskyne je daná najnižším povrchovým blokom. V prípade že je táto hodnota 0 je povrchové pozadie vynechané. Keďže hranice biómov sa s vysokou pravdepodobnosťou prelínajú, pozadia sú uložené podľa ich začiatočných a koncových indexov bounding boxu.

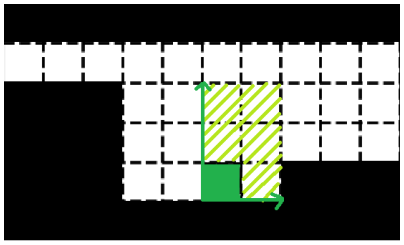
4.2.3 Generovanie objektov

Objekty sa do mapy pridávajú po uzamknutí zoomu (sekcia 4.4). Generovanie objektov prebieha postupne po biómoch, kde každý bióm má svoje pridelené objekty, ktoré sa v ňom môžu nachádzať. Povolené objekty sú reprezentované ako pole reťazcov, ktoré priamo odpovedajú názvu obrázku v úložisku assetov. Všetky objekty maju obecne preddefinovanú veľkosť. Táto veľkosť sa udáva v blokoch, avšak za pomoci vstupov z konfiguračného súboru je možné rozmery jednotlivých objekov meniť. Pri generovaní sa využije (podobne ako pri generovaní backgroundu) lokálna úroveň jaskyne, ktorá sa ráta od výšky horizontu definovanej v inicializačných parametroch. Ak bol parameter *horizon* poskytnutý, generovanie objektov je rozdelené na generovanie objektov na povrchu a generovanie podzemných objektov. V takomto prípade je set povolených objektov roztriedený na povrchové a podzemné. V opačnom prípade sa s celým výsekom mapy pracuje ako s podzemným.

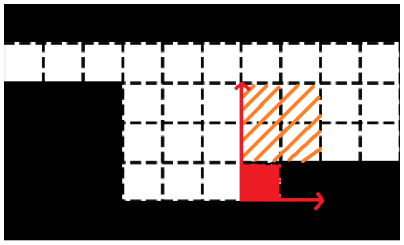
Umiestňovanie povrchových objektov

Obecné pravidlá generovania objektov zahŕňajú nasledovné:

- Objekt musí byť vygenerovaný na mieste, kde sa žiadny iný objekt nenachádza
- Objekt musi byť vygenerovaný tak, aby bol položený na teréne (nemôže sa nachádzať vo vzduchu)



(a) Kandidátny bod vyhovujúci pre položenie daného objektu



(b) Kandidátny bod nevyhovuje pre položenie kvoli terénu.

Obrázek 4.8: Demonštrácia možného (vľavo) umiestnenia objektu s rozmermi 3×2, a nevyhovujúceho miesta (vpravo). Čiastočná výplň symbolizuje obsadené bloky v prípade umiestnenia, celá výplň označuje kandidátny blok.

- Objekt môže byť vygenerovaný len na mieste, kde sa nenachádza terén.

Proces generovania objektov začína transformáciou obrázku do rozmerov odpovedajúcich jednému bloku (v pixeloch).

Na generovanie objektov na povrchu je nevyhnutné najprv lokalizovať všetky bloky, ktoré povrch tvoria. Lokalizácia prebieha rovnakou metódou, ako v časti generovania PJ pri ukladaní a rekonštrukcií povrchu. Trieda, ktorá objekty umiestňuje si drží obsadenosť na mape vo forme dvojrozmerného poľa odpovedajúcemu výseku mapy reprezentujúceho terén biómu. Princíp umiestňovania objektov na povrch je rovnaký v prípade biómu s PJ aj WJ. Princíp umiestňovania spočíva v nájdení vhodného miesta pre náhodne vybraný typ objektu. Miesto je vhodné pre objekt ak spĺňa všetky obecné pravidlá (demonštrácia na 4.8) umiestňovania spolu s obmedzeniami (napríklad pre povrchové objekty alebo objekty špecifické pre iný bióm).

Umiestňovanie podzemných objektov

Umiestňovanie podzemných objektov sa riadi rovnakými pravidlami ako povrchové objekty s detailom, že objekty nemôžu presiahnuť daný objem celej jaskyne. Z dôvodu sú definované dva typy jaskynných algoritmov je nutné navrhnúť meody, ktoré vyhľadajú prázdne miesta v teréne a budú ich vedieť izolovať ako samostatné jaskyne. Na vyhľadávanie prázdnych oblastí v PJ je využitý nasledovný prístup (demonštrácia na obrázku 4.9). Algoritmus na vyhľadávanie preiteruje cez všetky bloky, ktoré sa nachádzajú v danom bióme od úrovne *cavern_level* zľava doprava, zhora nadol. V prípade že predošlý blok bol prázdny, jaskyňa už začala a blok sa pridá mezdi bloky patriace danej jaskyne. V opačnom prípade sa validuje, či blok nad vybraným je prázdny. Ak áno, jaskyne sa spájajú, inak sa vytvára nová jaskyňa s vybraným blokom. Vďaka tomuto algoritmu je možné identifikovať dostupné miesto v danej jaskyni a umiestniť no nej objekty.

WJ sú reprezentované ako tumely, a jedná sa teda o jednu rozsiahlu jaskyňu. Výsek prázdneho priestoru sa tu vyhľadáva pomocou masky. Povrchové bloky sú vyhľadávané pomocou jednoduchého algoritmu 1.

(a) Iterácia vyhľadávania pred spojením jaskýň (b) Iterácia, v ktorej sa jaskyňe spoja do jednej.

Obrázek 4.9: Demonštrácia algoritmu na vyhľadávanie samostatných jaskynných priestorov a spájania jaskýň. Farbami sa odlišujú jednotlivé instancie tried, ktoré jaskyňu definujú.

Algorithm 1 Metóda vyhľadávania povrchových blokov v jaskyniach vytvorených použitím Worleyho šumu

```
floor_blocks ← []
if block[biome_height] is 0 then
    in_cave ← False
else
    in_cave ← True
end if
for height ← biome_height, 0 do
    if in_cave and block[height] is not empty then
        floor_blocks ← block
    else if not in_cave and block[height] is not empty then
        in_cave ← False
    end if
end for
```

4.3 Engine

V tejto sekcií sa nachádzajú implementačné detaily enginu, ktorý slúži na real-time rendering mapy do GUI.

4.3.1 Rendering

Vykresľovanie je rozdelené do dvoch hlavných častí. Prvá časť je vstupná fáza. Pri spustení editoru je používateľovi zobrazené okno, do ktorého sa zobrazí menu. Engine načítava všetky objekty (ako sú tlačidlá, inputy a pod.) menu do okna. Objekty sú v okne vykresľované ako štvorce, s ktorými je možná interakcia.

Menu pre novú mapu obsahuje vstupné polia, do ktorých sa vkladajú hodnoty na inicializáciu mapy, konkrétne procedurálneho terénu. Pomocou tlačidla create engine vyšle signál na generovanie mapy. Počas generovania je vykresľovanie pozastavené. Po dokončení generovacích krokov nasleduje druhá fáza.

Druhou fázou je editor. Počas editorovej fázy engine načítava objekty editorového menu. Okrem neho sa vykresľuje mapa, s ktorou sú možné interakcie popísané v kapitole 4.4.

Mapa sa renderuje v danej pozícii (ďalej ako fokálny bod), ktorá je určená relatívnymi súradnicami. Prepočet na relatívne súradnice sa odvíja od priblíženia (ďalej ako zoom). Základná hodnota zoomu je 1.0, pri ktorej má jeden blok veľkosť 12×12 pixelov v okne. Aby sa zamedzilo zoomu, pri ktorom by počet pixelov nebola celočíselná hodnota, sú tieto hodnoty dané fixne. Relatívna veľkosť jedného bloku v pixeloch je teda daná rovnicou 4.4, kde. 4.5

$$size = \left(\frac{12}{zoom}\right) \tag{4.4}$$

$$zoom \in \{0.0625, 0.125, 0.25, 0.5, 1.0, 2.0, 3.0, 4.0, 6.0, 12.0\} \tag{4.5}$$

V zavislosti od tohoto je teda možné používať relatívnu pozíciu, ktorá sa odvíja od indexov blokov v mape podľa rovnice 4.6, kde *pos* reprezentuje x-ové a y-ové súradnice bloku.

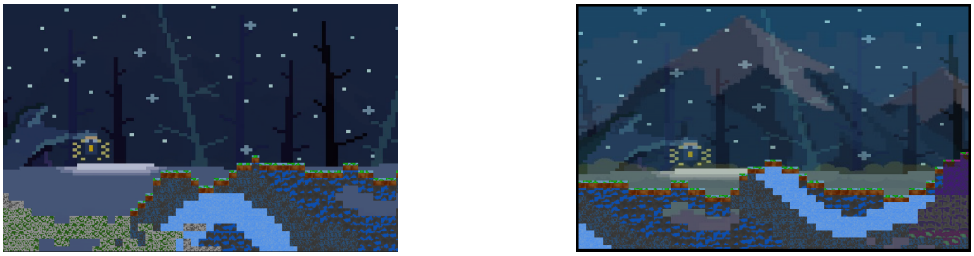
$$pos' = pos * 12/zoom \tag{4.6}$$

Pri posune mapy sa upravuje relatívna pozícia pričítavaním alebo odčítavaním (invertovane z hľadiska pohybu myšou). Pri zmene priblíženia sa prepočítava aktuálna relatívna pozícia na novú, pre minimalizovanie posunu vrámci mapy. Vykresľenie mapy prebieha v troch krokoch:

1. Vykreslenie backgroundu do prázdnej scény.
2. Vykreslenie terénu na backgroundovú vrstvu.
3. Vykreslenie objektov na terén.

Background

Pozadia sú v pamati reprezentované ako slovník objektov, kde pre každé pozadie je definovaná začiatočná a koncová pozícia. Pozícia je v relatívnych súradniciach, preto sa pozadia vykresľujú až po uzamknuti zoomu, spolu s objektami. V pripade, že sa fokálny bod nachádza na mieste, kde je umiestnených viac ako jedno pozadie, generovanie je vyriešené zmiešaním pozadí s upravenou transparentnosťou. Úroveň transparentnosti je priamo úmerná pomeru vzdialeností voči stredu biómu v relatívnych súradniciach - čím ďalej, tým priehľadnejšie (príklad na obrázku 4.10). Pozadia sú pri posune mapou nehybné.



(a) Background snežného biómu v mieste, kde sa neprekrýva s iným

(b) Prechod medzi snežným a horským backgroundom

Obrázek 4.10: Demonštrácia pozadí, ich vykresľovania a prekrývania

Terén

Vykresľovanie terénu pozostáva z načítania všetkých, blokov ktoré by sa mali zobraziť v závislosti od zoomu, a textúrovanie daných blokov. Engine pomocou relatívnej pozície a zoomu vypočíta, ktoré bloky sa budú nachádzať v okolí fokálneho bodu. Nakoľko relatívna pozícia môže udávať aj polohu v strede jedného bloku, je treba brať do úvahy relatívny offset. Výpočet viditeľného framu je podľa rovníc 4.7 a 4.8, kde pos' a pos'' sú súradnice začiatočného a koncového bodu, ktoré ohraničujú načítavanú oblasť. S_w je daná veľkosť okna (pre výpočet x-ovej súradnice sa použíje šírka, na y sa použije výška). S_b je veľkosť jedného bloku v pixeloch v závislosti od zoomu (výpočet podľa rovnice 4.9).

$$pos' = pos - \left(\frac{S_w}{2 * S_b}\right) - 5 \tag{4.7}$$

$$pos'' = pos + \left(\frac{S_w}{2 * S_b}\right) + 5 \tag{4.8}$$

$$B_s = \frac{12}{zoom} \tag{4.9}$$

V hraničné body definujú väčší výsek mapy ako je potrebné, aby nedošlo k prípadu, že sa v rohoch alebo hranách nebude nič generovať. Po získaní začiatočných a koncových súradníc je relatívna pozícia upravená tak, aby rohy a hrany mapy, ktoré sa majú vykresliť, neboli out-of-bounds. Následne sa z mapy vyberie výsek blokov, ktoré sú dané ohraničujúcimi súradnicami. V nich sa identifikujú unikátne typy blokov. Na základe ich hodnôt sa z editoru načítajú textúry pre dané bloky v závislosti od zoomu a absolútnej pozície bloku (sekvencia 4.11.

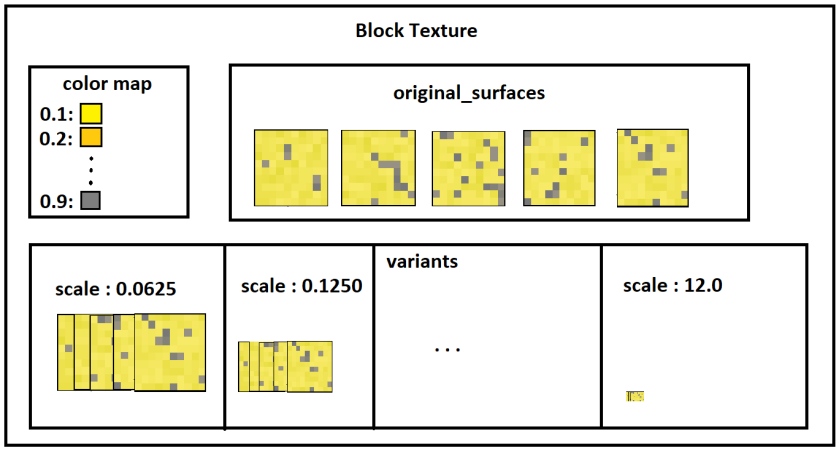
(a) Načítanie prvej unikátnej textúry (blok hliny)

(b) Načítanie nasledujúceho druhu textúry (blok trávy) na povrch biómu

(c) Načítanie následných textúr

Obrázek 4.11: Princíp načítavania blokov a ich textúr do okna.

Textúra pre každý blok je definovaná ako objekt, ktorý si drží originály textúr ktoré sa generujú procedurálne pomocou PN, alebo sú načítané z pamäte. V prípade procedurálneho generovania sa vygeneruje päť rôznych šumov, ktoré sú thresholdované na jedno destainné miesto. Každá textúra má špecificky definované parametre pre generovanie. Po vytvorení šumov sa výsledná textúra vytvorí pomocou priradenia RGB hodnoty každému bloku v závislosti of farebnej mapy, ktorá je pre každú textúru unikátna. Následne sa v objekte vytvoria textúry odpovedajúce originálu vo všetkých dostupných veľkostiach. Štruktúra textúry pre blok je reprezentovaná na obrázku 4.12.



Obrázek 4.12: Vnútorná štruktúra textúry pre blok. V atribúte *original* sú uložené textúrové reprezentácie bloku v základnej veľkosti (12×12 pixelov), *variants* obsahuje transformované textúry do všetkých možných veľkostí v závislosti od zoomu (*scale*)

Po tom, ako engine získa všetky potrebné textúry, im prideľuje absolútnu pozíciu v okne. Prepočet absolútnej pozície bloku do absolútnej pozície v okne je daný rovnicou 4.10.

$$pos_{screen} = \left(\frac{S_w}{2/B_s - (FP_{abs} - X_0 + pos)} \right) * B_s - off_{FP} \tag{4.10}$$

Výpočet bodu od ktorého sa má vykresľovať je daný fokálnym bodom FP, prepočítaným do absolútnych súradníc. X_0 je začiatočný bod ohraničenia, *pos* sú súradnice kontolovaného bloku a off_{FP} značí relatívny offset voči absolútnym pozíciam získaný pomocou 4.11.

$$off_{FP} = mod(FP, \left(\frac{12}{zoom} \right)) \tag{4.11}$$

Textúry sa vrátia ako pole objektov, ktoré je vykreslené do okna.

Objekty

Princíp renderovania objektov je založený na ich relatívnej pozícií. Podobne ako pri vykresľovaní terénu sa použije metóda na vyhľadanie začiatočného a koncového bloku. Ich súradnice sa transformujú do relatívnych súradníc. Následne sa zo zoznamu všetkých existujúcich objektov vyberú tie, ktorých súradnice sa nachádzajú v medziach okna. Z tejto vlastonsti vyplýva že v prípade zmeny zoomu po uzamknutí a vygenerovaní objektov by jednotlivé objekty nemali korektné súradnice.

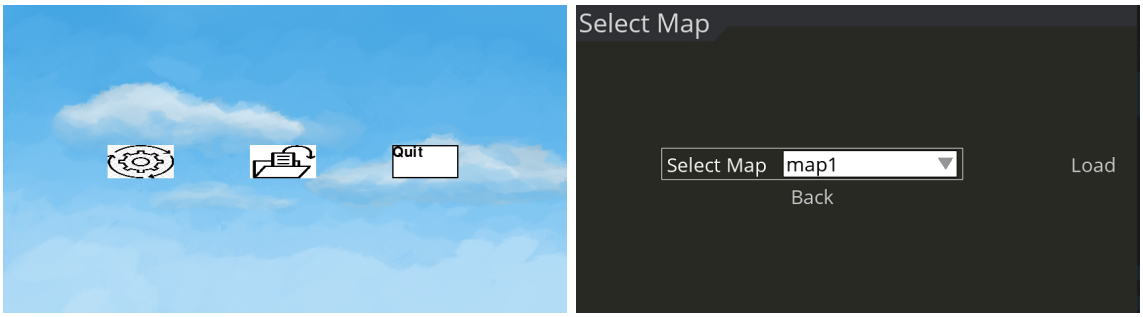
4.4 Editor a interakcie s GUI

Editor je súčasť programu, ktorá vykonáva pomocné operácie pri úpravách mapy. Primárne zahŕňa funckie, ktoré sa majú vykonať na základe vstupu od užívateľa počas renderovania mapy v reálnom čase. Definujú sa dve callback funkcie pre editorovú fázu:

- preprocess callback - Zahŕňa editorové menu na úpravu aktuálneho stavu mapy. Takisto vykresľuje terén mapy do okna.

- postprocess callback - používa engine na renderovanie mapy vo všetkých vrstvách a poskytuje interakcie s mapou na úrovni terénovej aj objektovej vrstvy.

a jedna callback funkcia, ktorá slúži na navigáciu cez menus a interakcie s nimi počas vstupnej fázy (obrázok 4.13).



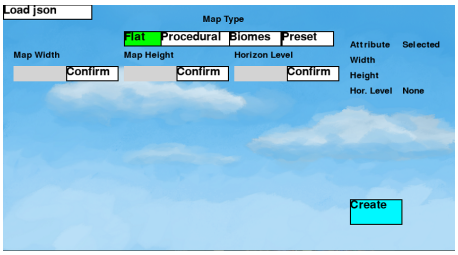
(a) Vykreslenie Main menu a jeho objektov: Tlačidlo (naľavo) na smerovanie do menu pre novú mapu (NewMenu), tlačidlo (v strede) pre menu na načítanie mapy zo súboru a tlačidlo (napravo) na ukončenie programu

(b) Vykreslenie Load menu. V selektore sa používateľovi zobrazia všetky existujúce mapy. Pomocou tlačidla **Load** sa mapa načíta do stavu, v akom bola uložená. Tlačidlo back vráti používateľa do Main menu

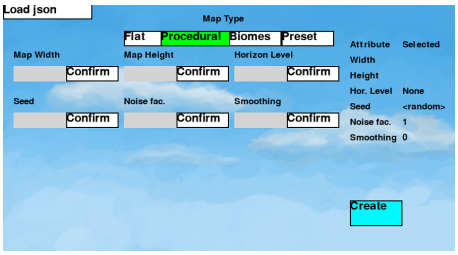
Obrázek 4.13: Menus vstupnej fázy pred inicializáciou generátora

Menus

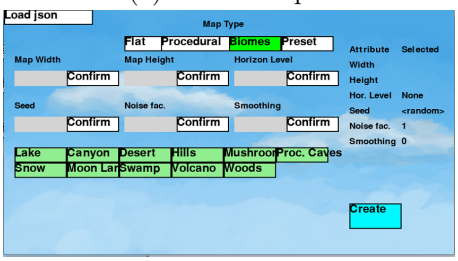
Interakcia s GUI a editorom prichádza najpr vo forme menus. Menu je definované ako



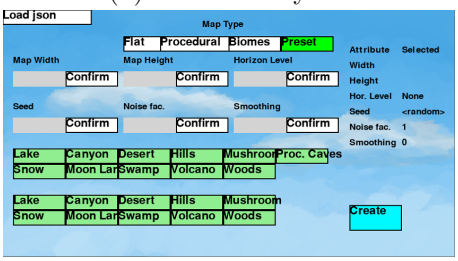
(a) Plochá mapa.



(b) Procedurálny terén



(c) Biómy



(d) Biómové štruktúry

Obrázek 4.14: Stavy NewMenu od ktorých sa odvíjajú generačné kroky na vytvorenie terénu mapy. Počet dostupných vstupov sa každým nasledovným stavom zvyšuje. Tlačidlo **Load json** slúži na načítanie parametrov z konfiguračného súboru.

kolekcia objektov. Obecne objekt v menu obsahuje pozíciu v okne, rozmery (v pixeloch) a alternatívne akciu. Tieto objekty sa potom delia na textové polia, tlačidlá alebo vstupné

polia. Editor v tejto časti spracováva vstupy od užívateľa na základe aktivovaných vstupných polí a tlačidiel a klávesnice. Po kliknutí na tlačidlo *Create* (obrázky 4.14) sa cez menu objekty vyšle signál na pozbieranie vstupných parametrov a v závislosti od nich sa spustí generátor. Parametre sú pozbierané z premmenných menu ktoré boli upravené na základe vstupných polí. Počas generovania mapy je vykresľovanie pozastavené, nakoľko generovací signál je vyslaný priamo do callback funckie menu. Po dokončení generovania sa hlavné menu vypne a inicalizuje sa preprocesové menu. V prípade vybratia možnosti načítania mapy sa používateľovi zobrazí menu načítania, ktoré obsahuje selektor s možnými súbormi mapy. Po načítaní sa inicializuje (v závislosti od stavu mapy) preprocesové alebo postprocesové menu (obrázok 4.15).

(a) Overlay editorového menu vykreslené spolu s mapou (vľavo hore v okne).

(b) Menu nástrojov podľa aktuálneho štádia editoru (preprocesové).

Obrázek 4.15: Editorové menu poskytujúce možnosť uložiť a upravovať mapu.

Editorové menu obecne obsahujú tlačidlo pre návrat, ktoré zahodí všetky doteraz vygenerované súčasti mapy a vráti sa do hlavného menu, tlačidlo pre nástroje a uloženie mapy. Preprocesové menu má tlačidlo **Post process**, ktoré inicializuje backgroundový generátor a generátor objektov, pridá ich do mapy a zmení callback funkciu na postprocesovú.

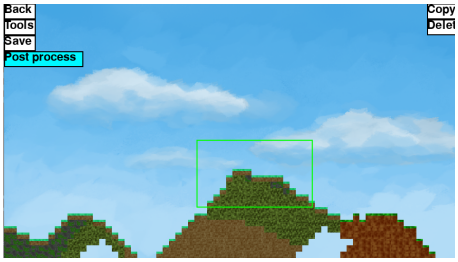
Editor a úpravy mapy

Po vygenerovaní (alebo alternatívne načítaní zo súboru) je mapa pripravená na prehliadanie. Fokálny bod je nastavený na stred mapy v závislosti od definovaných rozmerov. V prípade, že je zoom uzamknutý, sa v mape môžu nachádzať už aj objekty a pozadia, inak sa používateľovi do okna zobrazuje povrch so zoomom 1.0. V preprocesovom štádiu mapy je možné upravovať terén mapy. Používateľovi sa pomocou tlačidla **Tools** zobrazí menu dostupných úprav. v aktuálnom stave sú dostupné nasledovné nástroje:

- Posun mapy - po vybratí sa uzamkne priblíženie a mapu je možné uchopiť a posúvať. Editor túto interakciu spracuje posunutím fokálneho bodu v závislosti od posunu myši. Posun v mape sa realizuje vždy priamim pričítaním relatívneho posunu myši k relatívnej pozícií, čo znamená že pri veľkom priblížení je posun menej výrazný ako pri oddialenej mape.
- Ukladanie blokov - V menu si používateľ može vybrať typ bloku v selektore, obsahujúcom všetky dostupné bloky. Náhľad vybraného bloku je zobrazený vedľa selektoru. Po potrvrdení výberu sa užívateľovi zobrazí mapa a na pozícií myši sa vykreslí vybratý blok s polovičnou transparentnosťou. Používateľ može ukladať bloky do mapy kliknutím alebo držaním a ťahaním myši po mape. Editor prepočíta pozíciu myši

na absolútne súradnice v mape za použitia fokálneho bodu a umiestni celočíselnú hodnotu reprezentujúcu vybraný blok do mapy (demonštrácia na obrázku 4.16d).

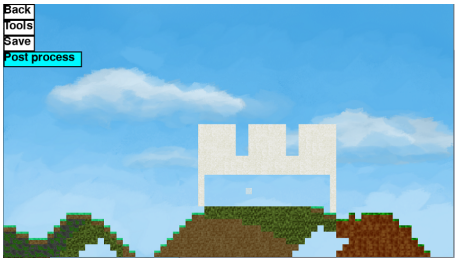
- selekcia blokov - Po vybrani možnosti selekcie môže používateľ označit výsek terénu kliknutím alebo držaním a ťahaním myši po mape (obrázok 4.16a). Označený terén je zvýraznený prázdnym zeleným štvorcom. Po ukončení selekcie má používateľ možnosť kopírovať selekciu (obrázok 4.16b, čo podobne ako ukladanie blokov zobrazí selekciu na pozícií myši s polovičnou transparenosťou a pri kliknutí sa selekcia vloží do mapy. Alternatívne môže používateľ selekciu vymazať (obrázok 4.16c. Túto akciu editor vykoná nahradením oblasti selekcie dvojrozmerným poľom obsahujúcim 0.



(a) Selekcia blokov.

(b) Kopírovanie selekcie blokov

(c) Vymazanie selekcie



(d) Manuálne umiestňovanie blokov (vybraný mramor).

Obrázek 4.16: Demonštrácie úprav mapy v preprocesovom štádiu.

Úpravy v mape neprepisujú hranice biómov. Okrem týchto úprav je možné mapu približovať a odďalovať. Prepočet relatívnych súradní pri oddialení je daný rovnicou 4.12.

$$pos'_{rel} = pos_{rel} * \left(\frac{zoom_{current}}{zoom_{new}} \right) \tag{4.12}$$

Používateľ môže následne uzamknúť zoom mapy, čím sa vyšle signál generátoru na vyplnenie mapy objektami a pridelenie pozadí jednotlivým biómom. Po týchto finálnych krokoch má používateľ okrem vyššie spomenutých úprav možnosti pridávať a upravovať objektovú vrstvu mapy. Editorové menu obsahuje okrem selekcie a vkladania blokov už aj selekciu a vkladanie objektov.

Kapitola 5

Limitácie, možné rozšírenia a optimalizácie

V tejto kapitole budú uvedené limitácie generátoru a editoru, možné optimalizácie jednotlivých súčastí a takisto rozšírenia jednotlivých modulov.

5.1 Limitácie

Procedurálne štuktúry pre špecifické biómy sa často v mapách nenachádzajú z príčiny fixne zadaných obmedzení. Tieto obmedzenia sa týkajú veľkosti dostupného povrchu v blokoch. Generátor aktuálne obsahuje malú variabilitu objektov a typov blokov, preto objekty aj bloky v mape môžu pôsobiť repetitívne. Vzhľadom na to, že je program implementovaný v Pythone, nie je rýchlosť operácií a chodu programu optimálna. Práca s pamäťou je spomaľovaná faktom, že celá mapa sa musí udržiavať v lokálnej pamäti programu. Funkcia na vyhľadávanie prázdnych priestorov v PJ spôsobuje veľmi výrazné spomalenie v biómoch s veľkým počtom jaskýň alebo obecne vo veľkých biómoch. Pri generovaní sa môžu v závislosti od výkonu stroja na ktorom sa program spúšťa, objaviť problémy z hľadiska frameratu. To je pozorovateľné hlavne pri renderovaní väčších oblastí mapy. Hranice biómov majú často aj po spracovaní tvar priamky. To môže byť spôsobené veľkosťami biómov ale aj koeficientami šumovej funkcie ktorá sa na na aplikuje. Algoritmus pre umiestňovanie objektov pracuje čiastočne neefektívne, keďže to, aký objekt sa má nachádzať na danom mieste nieje deterministické. Nakoľko sa jedná o prototyp editoru, menus a interaktívne prvky v nich niesu vizuálne atraktívne. Nástroje sú limitované počtom upraviteľných prvkov v mape (aktuálne iba terén a objekty).

5.2 Optimalizácie súčastí

Primárnym optimalizačným krokom by bolo implementovať program v inom jazyku (C, C++) alebo použiť už existujúci engine. Optimalzácia z hľadiska rýchlosti načítania mapy alebo prácou s ňou má nasledovné možnosti:

- uchovávanie mapy ako *memorymap* - mapa by by sa nenachádzala v lokálnej pamäti programu ale v úložisku, ku ktorému by mal program prístup.

- uchovávanie mapy na GPU - CUDA[1] poskytuje v jazyku python prácu s grafickou kartou. Keďže grafické karty sú prispôsobené na rýchlu prácu s dátami, táto optimalizácia by mohla opraviť problém s frameratom pri veľkom zoome
- chunky a tiling - bloky by bolo možné zhlukovať do chunkov a namiesto celej mapy by sa v lokálnej pamäti udržiaval iba istý počet chunkov ktoré sú najbližšie k fokálnemu bodu. Týmto procesom by sa zredukoval počet prístupov do pamäte, čo by viedlo k lepšiemu výkonu. Rozdelenie do tiling by spočíval v reprezentácií istej oblasti ako n dlaždíc, kde každá dlaždica obsahuje dátovú reprezentáciu daného výseku mapy spolu s menšími dlaždicami a textúrou reprezentujúcou spojenie textúr mapy v danej oblasti. Týmto by sa zredukoval počet načítaných blokov pri volaní callback funkcie.

Za účelom akcelerácie práce generátoru by sa mohla mapa definovať abstraktne vo všetkých vrstvách pred generovanim a vykonať všetky generovacie kroky paralelne, alebo použiť na generovanie jednotlivých častí multiprocessing. Vylepšenie algoritmu na vyhľadávanie miesta pre objekty by sa dal optimalizovať tým, že pre danú pozíciu by bolo dané, aký objekt sa na nej má generovať. Počet objektov by sa týmto spôsobom mohol znížiť. Ďalšou možnosťou by bolo predrozdelenie voľného priestoru na špecifické oblasti a populácia už definovaných oblastí náhodným objektom z kolekcie objektov danej veľkosti. Na menus je vhodnejšie použiť už existujúcu knižnicu, ktorá používateľa osloví po vizuálnej stránke a takisto mu poskytne ľahšiu navigáciu programom. Do editoru by bolo vhodné implementovať funkcie *Undo* a *Redo*.

5.3 Rozšírenia modulov

Program a jeho jednotlivé súčasti sú implementované tak, aby boli triedy rozšíriteľné o novú funkcionalitu. Tieto rozšírenia sú možné vo viacerých oblastiach. Vzhľadom na to, že každý špecifický bióm dedí základné funckie od obecnej triedy, je možné pridávať nové biómy jednoducho definovaním nových tried. To umožní používateľovi nadefinovať nové štruktúry a pridať variabilitu do mapy. Každý bióm používa preddefinovanú kolekciu blokov ktoré môže vo svojej oblasti zahrnúť. Tieto bloky sú - podobne ako biómy - definované špecifickou triedou ktorá dedí od obecnej. Týmto spôsobom je možné nadefinovať si vlastné typy blokov a priradiť ich existujúcim alebo novo vytvoreným biómom. Existujúce bloky používajú procedurálne textúry. Používateľ má možnosť nahrať si vlastné obrázky pre textúry a použiť ich namiesto preddefinovaných pre každý blok. Ďalšiou možnosťou rozšírenia sú algoritmy na vytváranie jaskýň alebo úprava algoritmu na pridávanie variability blokov v biómoch.

Budúce rozšírenia editoru sa budú zaoberať implementácie gravitácie do enginu. S týmto rozšírením prichádza možnosť rozlišovania typov blokov (pevné, sypké alebo kvapalné). Editor by v budúcnosti mohol podporovať základnú umelú inteligenciu NPC (non-player character) a po pridaní príbehových prvkov možnosť spustenia editoru v hernom móde.

[1] https://developer.nvidia.com/cuda-python

Kapitola 6

Záver

Cieľom tejto práce bolo navrhnúť moduly na generovanie 2D hernej mapy použitím procedurálnych metód a implementovať editor pre vizualizáciu a úpravy.

Ako prvé bolo nutné naštudovať techniky procedurálneho generovania. Zameral som sa na šumové funkcie a ich využitie v grafike. Po ich preštudovaní som sa rozhodol využiť Perlinov a Worleyho šum. Ďalším krokom bol návrh modulu generátoru a editoru, ktorý bude schopný mapu vizuálne reprezentovať. Jednotlivé časti som si rozdelil do logických celkov tak, aby boli moduly čiastočne nezávislé a jednoducho rozšíriteľné.

Moduly som sa rozhodol implementovať v jazyku Python. Implementácia splnila všetky cieľe práce. Používateľ má možnosť vygenerovať si 2D mapu samostatne pomocou generátoru, alebo pomocou GUI v editore. Na vygenerovanie má užívateľ poskytnuté vstupy ktoré upravujú výslednú mapu. Editor poskytuje možnosť vizualizácie generovanej mapy a interakcie s ňou (obrázok 6.1).

Práca mi priniesla nové poznatky v oblasti počítačovej grafiky. Umožnila mi preskúmať metodiky procedurálneho generovania. Mojím ďalším postupom bude rozširovanie vytvorených modulov a ich optimalizácia.

Obrázek 6.1: Demonštrácia finálnej mapy

Literatura

[1] Aurenhammer, F. Voronoi Diagrams—a Survey of a Fundamental Geometric Data Structure. *ACM Comput. Surv.* New York, NY, USA: Association for Computing Machinery. sep 1991, sv. 23, č. 3, s. 345–405. DOI: 10.1145/116873.116880. ISSN 0360-0300. Dostupné z: https://doi.org/10.1145/116873.116880.

[2] Aurenhammer, F. *Handbook of Computational Geometry: Voronoi Diagrams*. 1. vyd. Rolf Klein. Prosinec 1999. ISBN 9780080529684. Voľný preklad.

[3] Gawron, M. a Boryczka, U. Procedural rain ripples generated using Worley noise. *Computer Game Innovations*. s. 39.

[4] Green, P. J. a Sibson, R. Computing Dirichlet Tessellations in the Plane. *The Computer Journal*. Květen 1978, sv. 21, č. 2, s. 168–173. DOI: 10.1093/comjnl/21.2.168. ISSN 0010-4620. Dostupné z: https://doi.org/10.1093/comjnl/21.2.168.

[5] Green, P. J. a Sibson, R. Computing Dirichlet Tessellations in the Plane. *The Computer Journal*. Květen 1978, sv. 21, č. 2, s. 168–173. DOI: 10.1093/comjnl/21.2.168. ISSN 0010-4620. Dostupné z: https://doi.org/10.1093/comjnl/21.2.168.

[6] Meineke, F. A., Potten, C. S. a Loeffler, M. Cell migration and organization in the intestinal crypt using a lattice-free model. *Cell Proliferation*. 2001, sv. 34, č. 4, s. 253–266. DOI: https://doi.org/10.1046/j.0960-7722.2001.00216.x. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1046/j.0960-7722.2001.00216.x.

[7] Olawoyin, R. Objective assessment of the Thiessen polygon method for estimating areal rainfall depths in the River Volta catchment in Ghana. *Ghana Journal of Geography*. 2017, sv. 9, s. 151–174. ISSN 2821-8892. Dostupné z: https://www.ajol.info/index.php/gjg/article/view/159544.

[8] Smith, G. An Analog History of Procedural Content Generation. *IEEE Transactions on Games*. IEEE. 2018, sv. 10, č. 3, s. 213–217.

[9] Worley, S. A cellular texture basis function. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, s. 291–294.