

Nástroj pro řízené generování herního světa

Petr Santler

Bakalářská práce
2025



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Petr Santler
Osobní číslo: A22621
Studijní program: B0613A140020 Softwarové inženýrství
Forma studia: Kombinovaná
Téma práce: Nástroj pro řízené generování herního světa
Téma práce anglicky: A Tool for Procedural Generation of Game Worlds

Zásady pro vypracování

1. Vypracujte řešení na zadané téma.
2. Porovnejte existující algoritmy pro procedurální generování herního světa
3. Vyberte a popište vhodný algoritmus
4. Implementujte daný algoritmus do tankového simulátoru UTW
5. Vytvořte dokumentaci pro práci s nástrojem

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. ADAMS, Tarn a SHORT, Tanya X. (ed.). *Procedural generation in game design*. Boca raton: CRC Press, [2017]. ISBN 978-1-4987-9919-5.
2. Hendrikx, M., Meijer, S., Van Der Velden, J., & Iosup, A. (2013). Procedural content generation for games. *ACM Transactions on Multimedia Computing Communications and Applications*, 9(1), 1–22. <https://doi.org/10.1145/2422956.2422957>
3. NYSTROM, Robert. *Game programming patterns*. [London]: Genever Benning, [2014]. ISBN 978-0-9905829-0-8.
4. SHIFFMAN, Daniel. *The nature of code: simulating natural systems with processing*. [Mountain View]: Daniel Shiffman, [2012]. ISBN 978-0-9859308-0-6.
5. LENGVEL, Eric. *Mathematics for 3D game programming and computer graphics*. Third edition. Boston, MA: Course Technology, Cengage Learning, [2012]. ISBN 978-1-4354-5886-4.

Vedoucí bakalářské práce: **Ing. Michal Zapletal**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **27. října 2024**
Termín odevzdání bakalářské práce: **2. června 2025**



L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA BAKALÁŘSKÉ PRÁCE

Beru na vědomí, že

- odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení bakalářské práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – bakalářskou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na bakalářské práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Tato bakalářská práce se zaměřuje na návrh a implementaci nástroje pro řízené generování herních světů s využitím procedurálních technik. Hlavním cílem práce bylo vytvořit funkční, intuitivní a snadno ovladatelný nástroj, který umožňuje rychlé generování herních map podle zadaných parametrů. Výsledný software poskytuje uživateli možnost ovlivnit strukturu generovaného světa a zároveň zajišťuje časovou efektivitu celého procesu. Součástí práce je rovněž podrobná dokumentace popisující ovládání a praktické použití nástroje. V práci jsou dále uvedeny technické detaily implementace, popis použitých algoritmů a příklady výstupů. Nástroj může sloužit jako užitečný prostředek při vývoji herních prototypů i finálních projektů.

Klíčová slova: Unity, Simulátor, Tanky, Procedurální generování, Algoritmus, C#, prefab

ABSTRACT

This bachelor thesis focuses on the design and implementation of a tool for guided procedural generation of game worlds. The main goal of the project was to create a functional, intuitive, and user-friendly application that enables fast map generation based on user-defined parameters. The resulting software allows users to influence the structure of the generated world while ensuring time-efficient operation. The thesis also includes comprehensive documentation describing installation, usage, and practical applications of the tool. Technical details of the implementation, the algorithms used, and sample outputs are presented as well. The tool can serve as a helpful resource in the development of game prototypes as well as complete projects.

Keywords: Unity, Simulator, Tanks, Procedural generation, Algorithm, C#, prefab

Využívám možnosti poděkovat vedoucímu mé bakalářské práce panu Ing. Michalu Zapletalovi za jeho cenné rady, vedení a užitečné podněty, ze kterých jsem po celou dobu čerpal a jenž mi po celou dobu pomáhaly překlenout nástrahy spojené s vytvářením této práce.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Prohlašuji, že při tvorbě této práce jsem použil nástroj generativního modelu AI ChatGPT, dostupné z: <https://chatgpt.com/> za účelem pomoci parafrázování v teoretické části. Po použití tohoto nástroje jsem provedl kontrolu obsahu a přebírám za něj plnou zodpovědnost

OBSAH

ÚVOD.....	11
I TEORETICKÁ ČÁST	12
1 GENEROVÁNÍ TERÉNU	13
1.1 VÝŠKOVÉ MAPY	13
1.2 INTERPOLAČNÍ ALGORITMY	14
1.2.1 Bilineární interpolace	14
1.2.2 Bikubická interpolace.....	14
1.3 GRADIENTNÍ ALGORITMY	15
1.3.1 Perlinův šum.....	16
1.3.2 Simplexní šum.....	16
1.3.3 Fraktální metody	16
1.4 DIAMOND SQUARE ALGORITHMUS	17
1.5 ALGORITMY NA BÁZI AGENTŮ	17
2 GENEROVÁNÍ VEGETACE	19
2.1 ALGORITMY PRO GENERACI BIOMŮ	19
2.1.1 Whittakerův diagram.....	19
2.1.2 Voronoiovův diagram	20
2.1.3 Worley noise	21
2.1.4 Flood fill/Region growing.....	22
2.2 ALGORITMY PRO TVORBU VEGETACE	23
2.2.1 Poisson disk Sampling	23
2.2.2 Clustering	24
2.3 ALGORITMY PRO GENERACI JEDNOTLIVÝCH ROSTLIN.....	25
2.3.1 Modely na bázi gramatiky.....	25
2.3.2 Lindenmayerův systém (L systém)	26
2.3.3 Space colonization	28
3 GENERACE MĚST A CEST.....	30
3.1 ALGORITMY PRO GENERACI BUDOV	30
3.1.1 Cellular automata	30
3.1.2 Agent-based model (ABM).....	32
3.1.3 Wave function collapse	33
3.2 ALGORITMY PRO GENERACI DOPRAVNÍ INFRASTRUKTURY	34
3.2.1 Dijkstrův algoritmus.....	34
3.2.2 Algoritmus A*.....	34
3.2.3 Algoritmus RRT (Rapidly exploring Random Tree)	35
II PRAKTICKÁ ČÁST.....	37
4 VÝVOJOVÉ PROSTŘEDÍ	38

4.1	KNIHOVNY	38
4.2	UNITY ASSETY	38
5	POROVNÁNÍ ZKOUMANÝCH ALGORITMŮ	40
5.1	STANOVENÍ PARAMETRŮ	40
5.1.1	Výběr algoritmu pro generování terénu	40
5.1.2	Výběr algoritmu pro generování biomů	41
5.1.3	Výběr algoritmu pro generování vegetace	42
6	IMPLEMENTACE NÁSTROJE V PROSTŘEDÍ UNITY	44
6.1	VYTVOŘENÍ TERÉNU MAPY	44
6.1.1	Mapa biomů.....	44
6.1.2	Výšková mapa	45
6.1.3	Utvoření finálních výšek biomů	46
6.1.4	Vytvoření mřížky mapy.....	47
6.1.5	Další terénní úpravy	47
6.1.6	Finalizace mapy.....	48
6.2	TVORBA VEGETACE NA MAPĚ	49
6.2.1	Příprava generace vegetace	49
6.2.2	Vymezení ploch generace	49
6.2.3	Generování vegetace	51
7	POPIS NÁSTROJE	52
7.1	HLAVNÍ NASTAVENÍ NÁSTROJE.....	52
7.2	PŘIDÁNÍ ČI ÚPRAVA BIOMU.....	53
7.3	UŽIVATEL	56
7.4	UKÁZKA POUŽITÍ NÁSTROJE	57
	ZÁVĚR.....	61
	SEZNAM POUŽITÉ LITERATURY	63
	SEZNAM OBRÁZKŮ	65
	SEZNAM TABULEK.....	66
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	67
	SEZNAM PŘÍLOH.....	68

ÚVOD

S rostoucími nároky na rozsáhlost, variabilitu a autentičnost herních prostředí se vývojáři stále častěji obracejí k procedurálním metodám generování obsahu. Tyto techniky umožňují vytvářet rozsáhlé a různorodé herní světy bez nutnosti ručního návrhu každého prvku, čímž přispívají nejen k úspoře času, ale i ke zvýšení variability a opakovatelnosti výsledných prostředí. Procedurální generování nachází uplatnění nejen v komerčních hrách, ale také v simulátorech, výuce či experimentálním vývoji.

Cílem této bakalářské práce je navrhnout a implementovat nástroj pro řízené procedurální generování herního světa. Tento nástroj bude integrován do prostředí stávajícího tankového simulátoru UTW a měl by uživatelům umožnit generovat herní mapy dle předem definovaných parametrů.

V úvodních kapitolách bude provedena rešerše zaměřená na existující algoritmy používané při procedurálním generování, přičemž zvláštní pozornost bude věnována metodám generování terénu, vegetace a sídelních struktur. Tyto přístupy budou vzájemně porovnány na základě kritérií jako realističnost výstupu, výpočetní náročnost a možnost řízení výsledků ze strany uživatele.

Na základě tohoto porovnání bude vybrán konkrétní algoritmus či kombinace algoritmů, které se následně implementují do vývojového prostředí Unity. Součástí práce bude také návrh uživatelského rozhraní, ladění parametrů nástroje a vytvoření dokumentace popisující jeho použití.

Záměrem práce je vytvořit prakticky využitelný nástroj, který bude sloužit jako podpůrný modul pro generování prostředí v rámci simulací a vývoje herního obsahu.

I TEORETICKÁ ČÁST

1 GENEROVÁNÍ TERÉNU

Základní aspekty při tvoření herní mapy jsou terén a podloží, na které vývojář přidává další prvky. Pod pojem terén herní mapy, lze zahrnout samotný půdorys, výškový profil, a především drobnější úpravy této mapy, například vodní toky, koryta řek, jeskyně a skaliska. Součástí terénu herní mapy mohou být také biomy, které umožňují generovat různá území dané mapy, rozhodovat mimo jiné o typu vegetace, druhů budov a NPC (Non-Player Character).[1]

1.1 Výškové mapy

Výškové mapy představují jeden ze základních předpokladů pro generování herního terénu. Jsou reprezentovány obvykle černobílými rastrovými obrazy, kde každý pixel nese data o výškové hodnotě povrchu. Čistě bílé body jsou jedním extrémem, například definují nejvyšší body mapy, černé body pak definují extrém opačný, záleží na zvolené implementaci. Tyto hodnoty pro výškovou mapu lze získat z reálného světa např. pomocí lidarů, GPS (Global Positioning System), přímým měřením, nebo vygenerováním náhodných hodnot. Jednou z metod generování náhodných hodnot je šum. Spojení mezi terénem a šumem definuje kniha Procedural content generation in Games takto:[2]

„Šum i většinu aspektů terénu lze efektivně reprezentovat jako dvourozměrné matice reálných čísel. Šířka a výška matice odpovídají osám x a y na obdélníkovém povrchu. V případě šumu se tato reprezentace nazývá mapa intenzity a hodnoty jednotlivých buněk přímo odpovídají jasů příslušných pixelů. V případě terénu hodnota každé buňky představuje výšku terénu (vzhledem k určité základní úrovni) v daném bodě. Tato reprezentace se nazývá výšková mapa (heightmap).“ [2]

Výškové mapy tedy umožňují jednoduchou a efektivní reprezentaci krajiny. Hlavním problémem tohoto přístupu je, že generování náhodných čísel pro zaplnění výškové mapy je nezávislé na okolních výsledcích. V reálném světě výšky sousedících bodů na sobě statisticky závisí a určují i stupeň kolmosti terénu v daném bodě. Proto použití výhradně výškové mapy pro vytvoření výsledné krajiny, způsobí skokové přechody mezi jednotlivými buňkami, a tedy výsledná mapa by byla nerealistická. Otázku, jak vyplnit výškovou mapu řeší matematické algoritmy, které nám pomáhají nalézt vhodné hodnoty mezi body mřížky výškové mapy. [3]

1.2 Interpolační algoritmy

Jedním z řešení, jak maximalizovat realistické zobrazení krajiny, je nahrazení skokového propojení mezi jednotlivými sousedícími body vygenerovanými na výškové mapě, plynulými přechody a jejich propojením. Toto řešení sice částečně naruší realističnost terénu v případě skalisek a útesů, avšak dodá celému profilu mapy vzhled blížící se podobě krajiny. Mezi základní interpolační algoritmy se zařadí bilineární a bikubická interpolace. [4]

1.2.1 Bilineární interpolace

Jedná se o interpolační algoritmus, aplikující lineární interpolace souřadnic ve směru jedné osy a poté ve směru druhé. Zdali je provedena napřed ve směru osy x a poté osy y , nebo naopak není rozhodující. Matematický vzorec pro tuto operaci je následující:

$$f(x, y) \approx f(0, 0)(1 - x)(1 - y) + f(1, 0)x(1 - y) + f(0, 1)(1 - x)y + f(1, 1)xy$$

Příkladně při použití výškové mapy, kde výšky na souřadnicích mřížky $výška_1(0, 0)$ a $výška_2(0, 10)$ budou náhodně vygenerovaná čísla, pak užití interpolace umožní matematicky dopočítat hodnotu všech výšek na spojnici těchto bodů. V tomto modelu bude $výška_3(0, 3)$ rovna 30 % z $výška_1$ do $výška_2$. Interpolace je aplikována jako průměr mezi těmito výškami:

$$výška_3 = výška_1 \times 0.7 + výška_2 \times 0.3$$

Stejný postup je pak aplikován také ve směru druhé osy. Nevýhodou algoritmu se stává jeho jednoduchost. Při tvorbě krajin s minimálními rozdíly elevace bude výsledek, v případě hor a údolí, zobrazovat nereálně rovné svahy, a tedy i úplně špičaté vrcholy. Výsledný terén by se spíše podobal kresbě malého dítěte než reálnému světu.

1.2.2 Bikubická interpolace

Tento algoritmus naopak umožňuje odstranit lineární spoje v místech vrcholů a dolin. Způsobů, jak takového jevu dosáhnout je několik, například Lagrangeův polynom, kubické křivky, nebo kubický konvoluční algoritmus. Při bilineární interpolaci má rovnice průběhu funkce tvar $s(x) = x$. Pro požadované zakřivení do tvaru písmene „S“ je možné lineární rovnici převést do kubického tvaru $s(x) = -2x^3 + 3x^2$.

Při použití stejného zadání souřadnic a výšek bodů jako v bilineární interpolaci pro bod $výška_3(0,3)$, kde $x = 0.3$, $s(x) = 0.216$, bude interpolační rovnice pro výpočet jeho výšky následující:

$$výška_3 = výška_1 \times 0.784 + výška_2 \times 0.216$$

Terén nebude stoupat přímo strmě, ale v počátku a konci svahu se průběh jeho stoupání zpomalí a tedy vyhladí. Takový terén už se bude blížit více reálné krajině, avšak stále bude realismu vzdálený.

1.3 Gradientní algoritmy

Tato rodina algoritmů vychází z myšlenky, že na místo generování samotných výšek a interpolováním svahů mezi nimi, budou generovány náhodné gradienty, mezi kterými proběhne následná interpolace. Pro jeho dosažení je nutné užití mřížky výškové mapy stejně jako u interpolačních algoritmů, avšak s náhodně vygenerovanými body, reprezentanty gradientu, které mění směr a náklon svahu. Těmto algoritmům se obecně říká šumové, například Perlinův šum, Simplexní šum atd. [5]

Generování gradientů na místo výškových hodnot má velké množství výhod. Interpolace pro vyhlazení změn růstu hodnot výšky, na rozdíl od vyhlazení interpolováním změn mezi výškami, pomáhá svahům růst strměji či mělčeji, a tedy více realisticky. Tento algoritmus využívá mřížkově založenou generaci, čímž se vyhýbá trojúhelníkovým efektům způsobeným interpolačně založeným algoritmům, a je tak výpočetně a paměťově více efektivní.[6]

Pro gradientní algoritmy, není pro následnou interpolaci nutné generovat konkrétní čísla na mřížce, výběr gradientu je náhodný. V realizaci se jedná o 2D vektor, (d_x, d_y) , v každém bodě mřížky určující, jak moc bude svah strmý ve směru osy x pro d_x a y pro d_y . Výchozí výšky bodů mřížky jsou 0, avšak nulová hodnota výšky samotné může být v různých bodech svahu. Díky těmto odlišnostem v přechodech mezi vrcholy a nížinami, se zobrazí odlišná výška a hloubka vrcholů i nížin. Počet rozpoznatelných artefaktů je menší a dalšími úpravami se dá prakticky úplně odstranit.[6]

Výška bodů, jež neleží přímo na mřížce, se definuje pomocí čtyř okolních bodů mřížky, tzv. vypočtením „dot“ produktu. Jedná se o skalární součin mezi gradientovým vektorem v každém rohu a odpovídajícím směrem k hledanému bodu. Toto určí, jak moc přispívá daný

roh k výsledné hodnotě. V posledním kroku proběhne hladká interpolace mezi body získanými v předešlých krocích a součtu výsledků.

1.3.1 Perlinův šum

V roce 1983 použil Ken Perlin jako první gradientní šumový algoritmus pro vytvoření krajiny v rámci CGI (počítačově generované snímky) při práci na filmu Tron. Perlinův šum pro nalezení kandidátního bodu identifikuje 2^n rohových bodů čtverce, ve kterém se hledaný bod a jejich gradientní vektory nachází, proto složitost tohoto algoritmu je 2^n .

Přesnější popis hladké interpolace definuje rovnice, kde $n = 1$ a funkce interpoluje mezi hodnotami a_0 pro hodnotu bodu mřížky 0 a a_1 pro hodnotu bodu mřížky 1:

$$f(x) = a_0 + \text{smoothstep}(x) \times (a_1 - a_0) \text{ pro } 0 \leq x \leq 1$$

Příklad rovnice hladké interpolace (smoothstep):

$$f(t) = 6t^5 - 15t^4 + 10t^3$$

Kde t je normovaná vzdálenost bodu mezi mřížkovými body.

1.3.2 Simplexní šum

Později K. Perlin, upravil původní rovnici výpočtu šumu do podoby simplexní, mezi jehož hlavní přednosti patří nižší výpočetní složitost a nižší potřeba pro násobení. Simplexní šum využívá pro interpolování simplexu (n dimenzionální trojúhelníky), na rozdíl od klasických šumových funkcí, které využívají čtverce, a tedy i jeho složitost se snížila pouze na potřebu $n + 1$ rohů k získání hledaného bodu. Dále nemá žádné spatřitelné směrové artefakty, jež by mohli způsobit nerealističnost. Samotný proces algoritmu probíhá ve čtyřech krocích: vyzkoušení souřadnic, jednoduché dělení, vybrání gradientu a shrnutí jádra. [7]

1.3.3 Fraktální metody

Využitím prostých šumových funkcí je možné dosáhnout skoro realistického povrchu, a to už při použití šumu pouze na jedné frekvenci. Hory a údolí stejně tak pláně, mají na různých měřítcích, různé velikosti detailu a škály amplitud. Úplného zdůraznění těchto detailů je možné dosáhnout opakovaným aplikováním funkcí šumu v jiných oktávách, na stejnou výškovou mapu. Tedy aplikováním šumu o jiné frekvenci a škále na stejné mřížce. Tyto techniky jsou obecně označovány jako fraktální algoritmy. Matematicky vyjádřeno:

$$\text{noise}(f) + \frac{1}{2}\text{noise}(2f) + \frac{1}{4}\text{noise}(4f) \dots$$

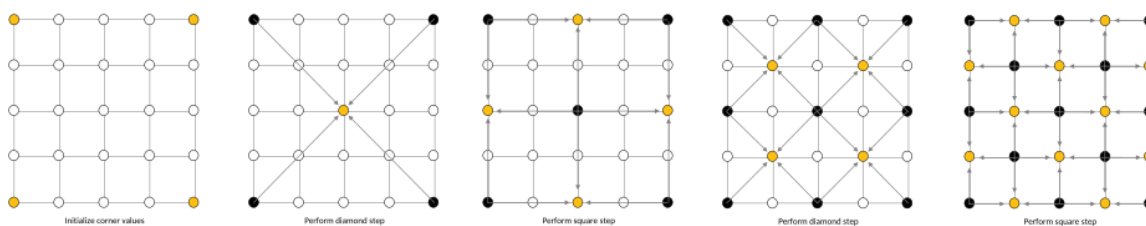
Kde *noise* je vybraný šum, zlomek definuje změnu frekvence a současně násobení určuje vstupu danou škálu. Tento postup lze sice použít i na ostatní typy algoritmů, avšak v některých případech je redundantní, a tedy jeho užití ztrácí smysl.

1.4 Diamond square algoritmus

Tento algoritmus předpokládá dosazení náhodné hodnoty (výšky) do rohů výchozí čtvercové mřížky o rozměrech $(2^n + 1) \times (2^n + 1)$ a následného rekurzního opakování dvou hlavních kroků.

V prvním, tzv. diamantovém kroku, dochází k vyznačení krajních bodů čtverce, což umožní určení jeho středu. Výška středového bodu je průměr výšek krajních bodů s přidáním náhodného posunu. To zajistí zachování variability. [6]

Následující čtvercový krok, je založený na stejném principu, avšak probíhá na vertikální a horizontální ose. Pro každý diamant v mřížce nastavíme střed a provedeme opět výpočet průměru okolních hodnot.[5]



Obrázek 1 Postup diamond square algoritmu krok po kroku

(zdroj: https://en.wikipedia.org/wiki/Diamond-square_algorithm)

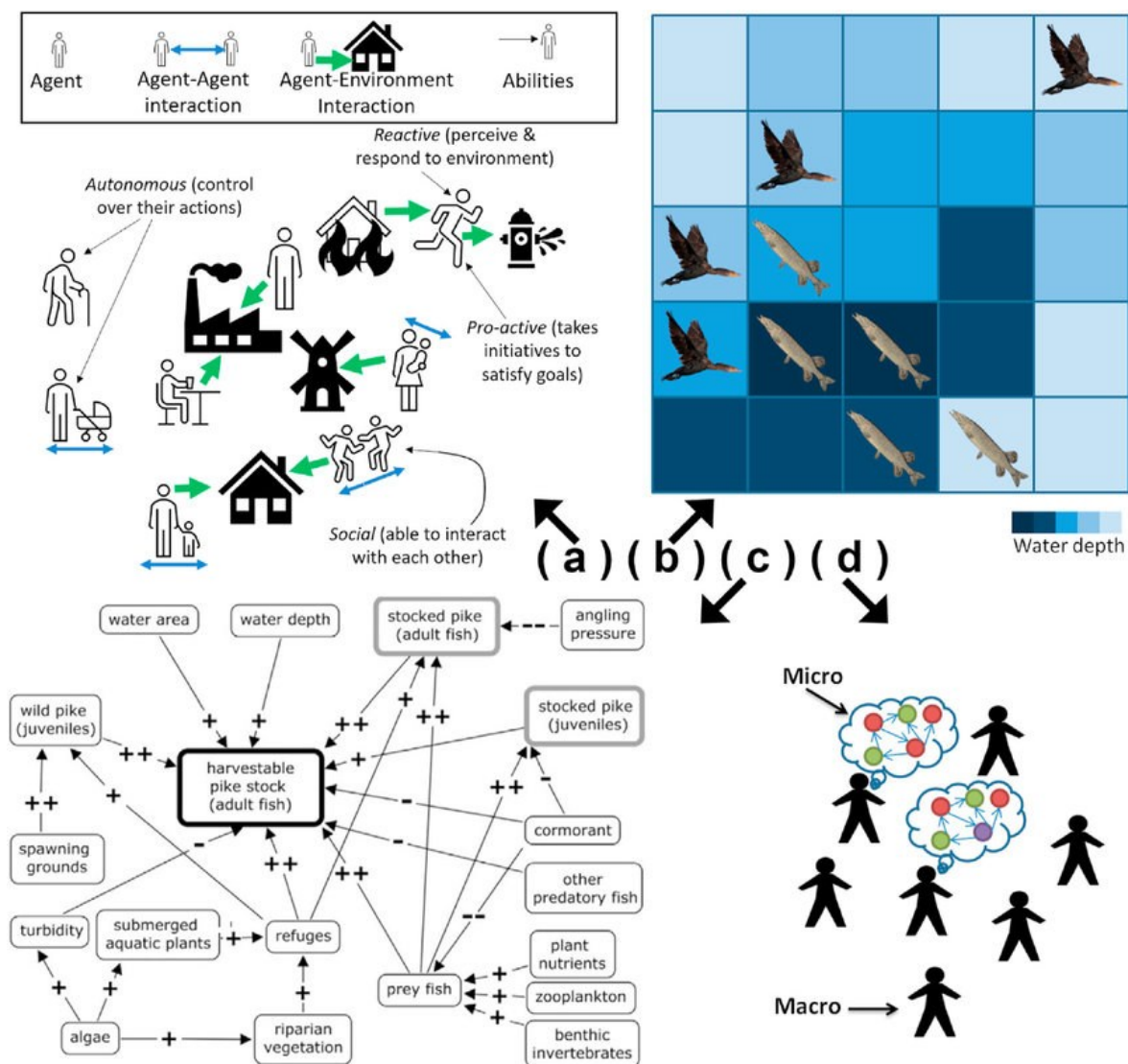
1.5 Algoritmy na bázi agentů

Slabým místem výše zmíněných přístupů ke generaci herního světa, je takřka nulová kontrolovatelnost jednotlivých procesů. Řízenou kontrolu nad procesy generace oproti fraktálním a dalším metodám, umožňují softwaroví agenti, se schopností PCG (Procedural Content Generation).

Zvolený počet agentů a jejich rozmanitosti, jsou schopni ovlivnit generaci terénu. Současně designér (uživatel) neztrácí schopnost kontrolovat jak, či jak moc, budou mít akce agentů vliv na navrhovaný terén s dopadem na konečný výsledek. V rámci generování krajiny jsou nejběžněji užívány tyto kategorie agentů:

- agenti hydraulické eroze simulují vliv vody na krajinu,
- agenti termální eroze vyhlazují hrany a tvoří jemnější terén,
- agenti random nebo drunkard's walk pomáhají při vytváření přírodních chodeb, jeskynních systémů a další.

Například přední videoherní vývojáři Doran a Parberry aplikovali tento přístup generování pomocí celé řady agentů, jež na sebe navazovali, a tak utvořili dalo by se říct, celý algoritmus této generace.



Obrázek 2 Obecné charakteristiky modelu založeného na agentech

2 GENEROVÁNÍ VEGETACE

Terén bez detailních prvků by byl v realistické hře či simulaci považován za strohý a málo uvěřitelný. Realistické ztvárnění herního světa vyžaduje komplexní přístup k implementaci vegetačních prvků. Rozmanitost tvarů a druhů, která je v reálném světě všudypřítomná nesmí narušit realistický pohled na generovanou krajinu. [8] Z toho důvodu se očekává přítomnost takové vegetace, která odpovídá herní scénérii. Stejně jako v případě generování terénu jsou i zde využívány různé algoritmy a šumy pro získání pohledu, který se nejvíce přibližuje skutečnosti. Celková generace se dělí do tří základních kroků, generace biomů, vegetace a jednotlivých rostlin.

2.1 Algoritmy pro generaci biomů

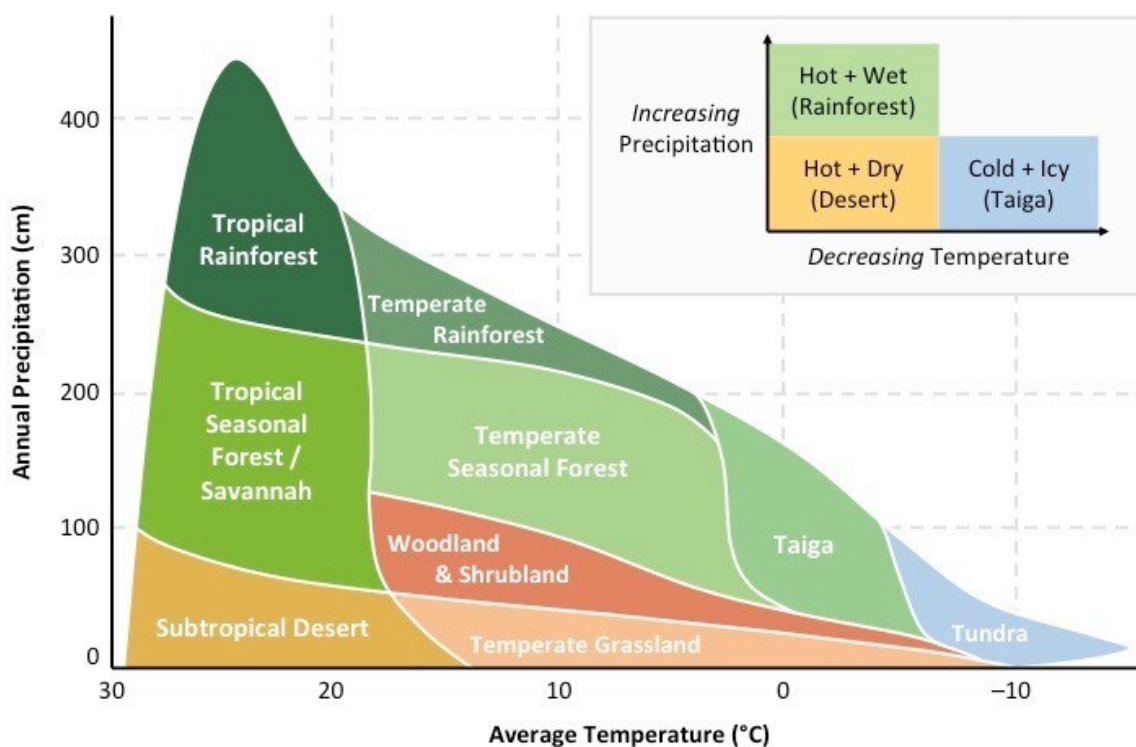
Biomy představují v kontextu rozdělení herního světa "zóny" s unikátními vlastnostmi, které mohou hráči poskytovat různé zážitky nebo výzvy. Biomy mají zásadní vliv na vegetaci v dané oblasti. Prezentované algoritmy zkoumají principy identifikace a charakterizace ekologických systémů specifických přírodních oblastí. Pozornost bude zaměřena na metody identifikace klimatických, geografických a ekologických parametrů, které určují vegetační pokryv. [8]

2.1.1 Whittakerův diagram

Pro posouzení vlivu podnebí na vegetační pokryv, je využíván Whittakerův diagram, jako nástroj používaný v ekologii a biogeografii, k vizualizaci vztahů mezi různými ekosystémy, nebo vegetačními typy, na základě klimatických faktorů, zejména teploty a srážek. Tento diagram je pojmenován po britském ekologovi Robertu Whittakerovi, který v 50. letech 20. století navrhl způsob, jak zobrazit tyto vztahy.

Whittakerův diagram je obvykle graf, kde:

- horizontální osa představuje průměrné roční srážky (od nízkých po vysoké),
- vertikální osa reprezentuje průměrnou roční teplotu (od chladného po teplé klima).



Obrázek 3 Whittakerův diagram rozložení zón na základě teploty a srážek

(zdroj: <https://old-ib.bioninja.com.au/options/option-c-ecology-and-conser/c2-communities-and-ecosyste/ecosystem-analysis.html>)

Na základě těchto dvou faktorů, se na diagramu zobrazují různé druhy vegetace a ekosystémů (např. tropické deštné lesy, savany, pouště, lesy mírného pásu atd.). Každý ekosystém nebo vegetační zóna má svou charakteristickou kombinaci teploty a srážek, jenž ovlivňuje druhy vegetace, které mohou v daném prostředí růst. Pro PCG můžeme k určení těchto hodnot využít výšku a pozici na mapě každého bodu.

2.1.2 Voronoiovův diagram

V procedurálním generování organických textur jsou s oblibou využívány, pro modelování 3D geometrických vzorů metodou tříštění či lámání, Voronoiovy diagramy. V matematice je Voronoiovův diagram rozdělením roviny, na jednotlivé oblasti, blízké každému z daného souboru objektů. Může být klasifikován také jako mozaika. V nejjednodušším případě, jsou tyto objekty konečnou množinou bodů v rovině, nazývaná semena, místa nebo generátory.

Rozdělením roviny na nesouvislé oblasti zvané buňky, z nichž každá obsahuje právě jedno semeno, tak aby všechny body v buňce byly blíže semenu této buňky než jakémukoli jinému semenu. Množina různých bodů v rovině je $P = \{p_1, p_2, \dots, p_n\}$. Rovina je dále

rozdělena na n buněk tak, že každá buňka obsahuje právě jedno semeno. Libovolný bod (x, y) je v buňce odpovídající místu p_i se souřadnicemi (x_{pi}, y_{pi}) tehdy a jen tehdy jestliže:

$$\sqrt{(x - x_{pi})^2 + (y - y_{pi})^2} < \sqrt{(x - x_{pj})^2 + (y - y_{pj})^2}$$

pro všechny p_j pro které platí $j \neq i, 1 \leq j, i \leq n$

To znamená Euklidovská vzdálenost od (x, y) k jakémukoli jinému místu je větší, než vzdálenost (x, y) k p_i . Tedy hranice takto definovaných buněk budou složeny z přímých čar a segmentů, tvořících konvexní polygony a budou definovány kolmými osami segmentů, spojujících každou dvojici míst. Koncept Voronoiových diagramů lze zobecnit na n rozměrů a na libovolně definovanou metriku vzdálenosti. Kromě toho mohou být jako místa použita obecná geometrická primitiva, jako jsou úsečky nebo křivky.

2.1.3 Worley noise

Worleyho šum je rozšířením Voronoiova diagramu a jeho využitím je dosaženo plynulých přechodů mezi regiony. Pokud se použije Worleyho šum jako vstup do algoritmu používajícího Voronoioův diagram, pak jsou výsledkem polygony, které jsou pro účely použití v grafických strukturách mnohem realističtější.

Princip Worleyho šumu spočívá v generování náhodně umístěných bodů v prostoru a následném výpočtu hodnot na základě vzdálenosti od těchto bodů. Je to typ šumu, který vytváří charakteristické vzory podobné "buňkám", kde každá oblast mezi těmito náhodně umístěnými body, má svou vlastní hodnotu, která závisí na vzdálenosti od nejbližších bodů.

Generování probíhá v několika krocích:

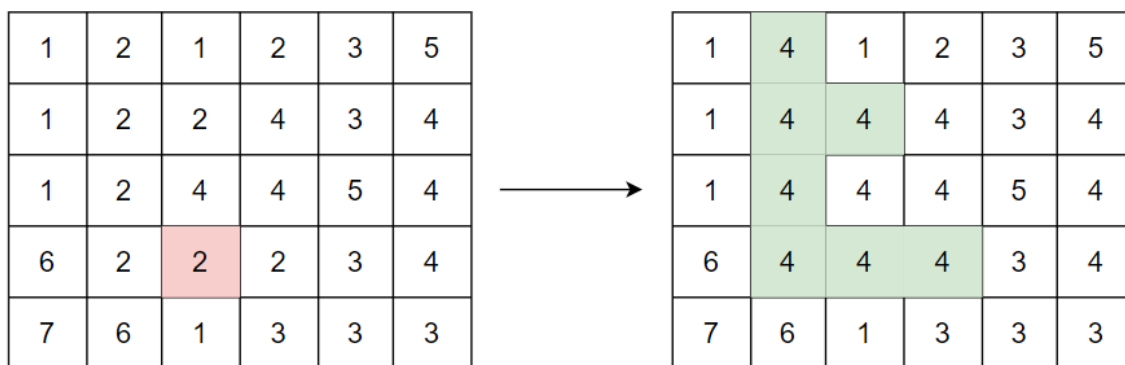
- ve zvoleném prostoru se rozmístí náhodné body. Tyto jsou fixní a slouží jako referenční, pro výpočet hodnoty šumu v okolních oblastech,
- pro každý bod, například pixel na obrazovce, nebo bod na povrchu objektu, se spočítá vzdálenost k nejbližšímu náhodně umístěnému bodu,
- následně dojde k výpočtu hodnoty šumu pro daný bod, obvykle na základě vzdálenosti k nejbližšímu bodu, či průměrem vzdáleností k několika nejbližším bodům. Výsledné vzdálenosti jsou pak normalizovány, nebo modifikovány, například přidáním další funkce, která mění hodnotu v závislosti na vzdálenosti, s cílem vytvořit texturní efekt.

2.1.4 Flood fill/Region growing

Další problematikou při tvorbě biomů, je vyplnění prázdných prostor v plochách a strukturách. Příklad nástroje pro jejich vyplnění je Flood fill, nazývaný též seed fill algoritmus, tedy zaplavovací algoritmus. Ve vícerozměrném poli určuje a mění oblast připojenou k danému uzlu s daným odpovídajícím atributem. Stává se tak nástrojem pro vyplnění spojených, podobně zbarvených oblastí jinou barvou. [9]

Varianta zaplavovacího algoritmu, nazývaná hraniční výplň, je definována jako oblast připojená k danému uzlu, která nemá konkrétní atribut, ale používá stejné algoritmy.

Tradiční algoritmus Flood-fill používá tři parametry: počáteční uzel, cílovou barvu a náhradní barvu. Algoritmus hledá všechny uzly v poli, které jsou spojeny s počátečním uzlem cestou cílové barvy a mění je na náhradní barvu. Pro hraniční výplň pak bude místo cílové barvy dodána hraniční barva.



Obrázek 4 Flood fill vyplňování do čtyř stran

(zdroj: https://www.naukri.com/code360/problem-details/flood-fill-algorithm_1089687)

Příklad: obrázek o velikosti 5×6 buněk, souřadnice počátečního červeného pixelu (4,3), má hodnotu 2. Nahrazením počátečního a všech přímo sousedících pixelů se stejnou hodnotou daného pixelu na hodnotu 4, získáme tak nový obrázek zobrazen vpravo. Upravené pixely jsou nově zvýrazněny zelenou barvou.

Další zaplavovací algoritmus sloužící pro vyplnění prostor je Region growing (RG) algoritmus. Jeho předností je efektivní přístup k segmentaci obrazu. RG algoritmy se liší v závislosti na kritériích použitých k rozhodnutí, zda má být pixel zahrnut do oblasti, nebo ne, dle typu konektivity použité k určení sousedů a dle strategii použité k návštěvě sousedních pixelů.

Základní přístup algoritmu pro rozšiřování oblasti vychází od počáteční oblasti uvnitř objektu, který má být segmentován. Pixely sousedící s touto oblastí jsou vyhodnoceny a následně je stanoveno, zda budou také zahrnuty jako součást objektu. Pokud ano, jsou přidány do oblasti a proces pokračuje do vyčerpání všech relevantních pixelů, které by měly být přidávány do dané oblasti.

2.2 Algoritmy pro tvorbu vegetace

Správné rozmístění rostlin zvyšuje estetickou hodnotu scény, a přispívá nejen k její věrohodnosti, ale také k její ekologické konzistenci.

2.2.1 Poisson disk Sampling

Poisson Disk Sampling algoritmus se běžně používá pro generování sady bodů, které jsou rozmístěny náhodně, přesto rovnoměrně v prostoru. Tyto body, následně při rozmisťování vegetace, představují jednotlivé rostliny. Poissonovo rozdělení zabezpečí, že mezi všemi body, bude dodržena také minimální vzdálenost. Dalším parametrem v algoritmu je disk, který při generování vychází ze skutečnosti, že každý bod bude vzdálen od ostatních, alespoň v takové vzdálenosti, aby se disky určitého poloměru nemohly překrývat. [10]

Fáze algoritmu:

- inicializace – volbou rádiusu disku odpovídající minimální vzdálenosti mezi body, se vytvoří mřížková struktura o velikosti buňky $r/\sqrt{2}$ (případně větší) pro efektivní kontrolu okolí bodu,
- generování bodů – pro každý vygenerovaný inicializační bod platí náhodný směr a náhodná vzdálenost, v prstencové oblasti kolem inicializačního bodu, přičemž platí, že náhodná vzdálenost je v rozmezí r do $2r$ v okruhu posledního bodu. Zároveň musí nově vygenerovaný bod být vzdálen minimálně r od všech dalších existujících bodů. V případě, že není splněna podmínka, bod je odhozen a operace se opakuje, až do momentu úspěšného přidání nových bodů,
- ukončení – po úspěšném přidání všech možných bodů, nebo po dosažení maximálního počtu pokusů o umístění bodu, je algoritmus ukončen.

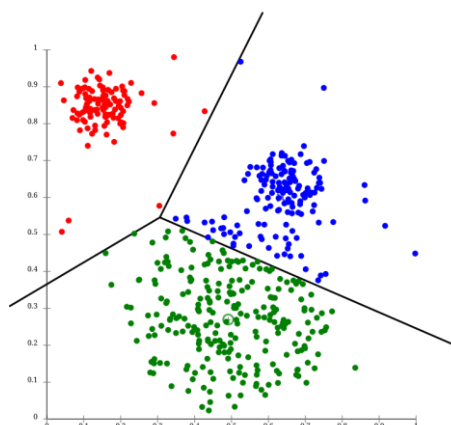
Nevýhodou Poisson Disk Sampling algoritmu je početní a časová náročnost, zejména je-li požadovaná malá vzdálenost bodů, naopak v oblastech s omezeními, vyžadujících rovnoměrné, náhodné rozdělení, je velmi přínosný.

2.2.2 Clustering

Clustering nebo také Cluster analysis, jsou metody, jejichž úkolem je seskupování množiny objektů takovým způsobem, že objekty ve stejné skupině jsou si navzájem více podobné. Clustering lze formulovat jako více cílový optimalizační problém. Vhodný shlukující algoritmus a nastavení parametrů závisí na individuální sadě dat a zamýšleném použití výsledků.

Connectivity-based clustering (hierarchical clustering) - je založeno na základní myšlence, vzájemného spojování s blízkými objekty než s objekty vzdálenějšími. Tyto algoritmy spojují „objekty“ a vytvářejí „shluky“ na základě jejich vzdálenosti.

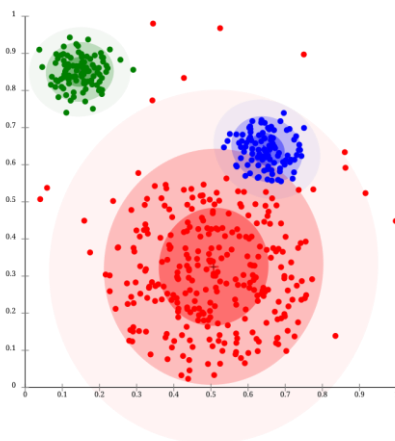
Centroid-based clustering – v shlukování založeném na centroidech, je každý shluk reprezentován centrálním vektorem. Optimalizační problém spočívá ve vyhledání středů pevně daných shluků a přiřazení objektů k nejbližšímu středu shluku tak, aby se minimalizovaly čtvercové vzdálenosti od shluku – neboli k-means clustering.



Obrázek 5 k-means clustering rozděluje data do Voronoiových buněk

(zdroj: https://en.wikipedia.org/wiki/Cluster_analysis)

Model-based clustering – shlukující rámec je založený na statistickém modelu. Tento přístup modeluje data pocházející ze směsi rozdělení pravděpodobnosti. Aby bylo možné získat pevné shlukování, jsou objekty často přiřazeny ke Gaussově distribuci.



Obrázek 6 Gaussovo modelování shluků.

(zdroj: https://en.wikipedia.org/wiki/Cluster_analysis)

2.3 Algoritmy pro generaci jednotlivých rostlin

Zatímco předchozí části se zabývaly výběrem a rozmístěním vegetace v rámci biomů, tato kapitola se zaměřuje na detailní modelování jednotlivých rostlin, zejména stromů a jejich větvení. Generování jednotlivých rostlin vyžaduje pokročilejší přístup než pouhé umístění 3D modelu – cílem je dosáhnout realistických, variabilních a biologicky uvěřitelných struktur, které obohacují celkový vzhled herního prostředí.

2.3.1 Modely na bázi gramatiky

Modely založené na gramatice pro procesní generování zahrnují použití formálních gramatik k algoritmickému vytváření obsahu. Tyto modely používají sadu pravidel (gramatiky), která definují, jak lze symboly kombinovat nebo transformovat za účelem generování struktur, jako je text, mapy, úrovně nebo dokonce celé virtuální světy. Cílem je napodobit způsob, jakým jsou jazyky a struktury vytvářeny pomocí řady syntaktických pravidel, díky čemuž jsou vysoce flexibilní a schopné produkovat širokou škálu obsahu z malé sady základních pravidel. [2]

Pravidla definují, jak lze prvky, často symboly, nahradit, nebo rozšířit. Jednoduché pravidlo může například znít: $A \rightarrow "B C"$, což znamená "A" může být nahrazeno sekvencí "B C". Mezi základní pravidla se řadí symboly, proces generování struktur, rozšíření pravidla, rekurzivní expanze a randomizace.

- symboly mohou být neterminální, nebo terminální a jsou vyžadovány při tvorbě tvarů. V gramatice jsou terminální symboly koncovými body (např. skutečná slova v jazyce, nebo předměty v herním světě) a neterminální symboly, tedy zástupné symboly, které je třeba nahradit více symboly,
- proces generování struktur – začíná startovacím symbolem neboli nonterminálem. Například při generování věty, počáteční symbol může být slovo „Sentence“,
- rozšíření pravidla – Počáteční symbol je rozšířen pomocí gramatických pravidel. Pokud je například pravidlo věta $\rightarrow$ Podstatné jméno_Fráze_Sloveso_Fráze, pak je rekurzivně možné použití pravidla pro vytvoření další struktury,
- rekurzivní expanze vychází z předchozí iterace, neterminální symboly jsou nahrazeny jinými neterminálními nebo terminálními symboly. To umožňuje generování složitých struktur z jednoduchých pravidel,
- randomizace umožňuje rozšířit rozmanitost, často obsahují náhodné volby. Pravidlo může například říkat „Sloveso“ $\rightarrow$ „spustit“ | "skok" | "swing", což znamená, že sloveso může být náhodně vybráno ze seznamu možných sloves.

Systémy založené na gramatice umožňují vysoký stupeň flexibility a rozmanitosti obsahu, který vytvářejí. Vývojáři mohou ovládat úroveň náhodnosti, nebo struktury jemným doladěním gramatických pravidel. Jakmile je gramatický systém nastaven, může generovat prakticky nekonečný počet možných výstupů. Navrhnout efektivní a robustní gramatický systém může být obtížné, zejména pro rozsáhlý nebo vysoce dynamický obsah. Zajistit, aby generovaný obsah byl logicky, nebo esteticky soudržný, může být složité, protože jednoduchá pravidla nemusí vést ke smysluplným výsledkům. [2]

2.3.2 Lindenmayerův systém (L systém)

L-systém jako formální metoda pro popis růstu rostlin a dalších biologických struktur, je dnes v počítačové grafice využíván také pro generování fraktálů a dalších komplexních tvarů.

L-systém se skládá ze tří základních složek:

- symbolový jazyk obsahuje základní symboly, které budou nahrazovány podle pravidel. Například to mohou být znaky jako F , $+$, $-$, atd. Symboly mohou představovat různé akce nebo stavy, například F může znamenat „posuň se vpřed a nakresli čáru“, zatímco $+$ a $-$ mohou znamenat otáčení v určitém směru,
- počáteční řetězec (Axiom) což je počáteční stav systému, který se postupně mění pomocí pravidel. Axiom je posloupnost symbolů, která definuje počáteční stav,
- pravidla (produkční pravidla) definují, jakým způsobem budou symboly nahrazovány. Každé pravidlo říká, jak nahradit jeden symbol nebo posloupnost symbolů, jiným symbolem nebo posloupností.

Příklad jednoduchého L-systému:

Symbolový jazyk $F, +, -$

Počáteční řetězec F

Pravidlo $F \rightarrow F + F - F - F + F$

Tento systém říká:

Každý symbol F je nahrazen posloupností $F + F - F - F + F$, což může reprezentovat růst struktury. Symboly $+$ a $-$ představují otáčení, například $+$ znamená otočení o určitou úroveň doprava a $-$ otočení doleva. Pokud bychom začali s axiomem F , aplikovali bychom pravidlo na každý symbol v řetězci pro daný počet iterací.

Například:

Po 1. iteraci (Axiom: F):

$$F \rightarrow F + F - F - F + F$$

Po 2. iteraci:

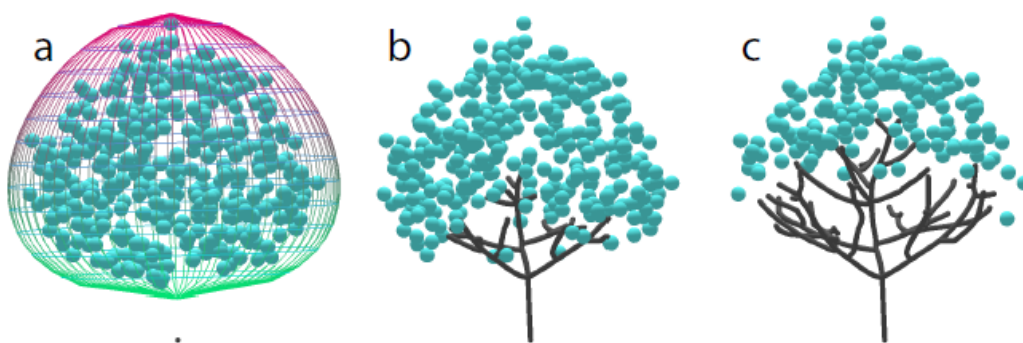
$$F + F - F - F + F \rightarrow F + F - F - F + F + F + F - F - F + F - F + F - F + F$$

A takto pokračujeme iteraci po iteraci až do naplnění stanoveného maxima.

2.3.3 Space colonization

Pro generování realistických stromů existuje mnoho algoritmů, také Space colonization algoritmus byl navržený primárně pro modelování vzorů architektury stromů, žilnatění listů a větvení. V současnosti je pro svojí vekou flexibilitu, založenou na principu soupeření o prostor mezi jednotlivými body, využíván také pro simulace přirozeného rozvoje civilizací, růstu měst a mnoho dalších.

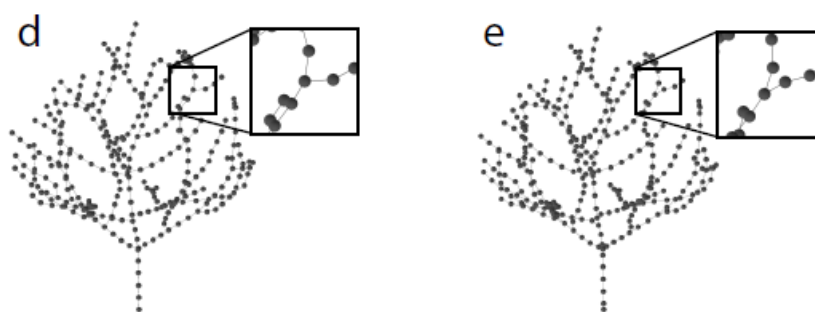
Na obrázku 7a jsou znázorněny jednotlivé fáze algoritmu. Vstupní trojrozměrná obálka koruny stromu je nejprve osazena sadou přitažlivých bodů (a), které signalizují dostupnost volného prostoru pro růst. Pokud k nim není z větve dosaženo, jsou z obálky odstraněny. Rozmístění přitažlivých bodů je prováděno rovnoměrně, aby byla zajištěna požadovaná hustota větví. Kostra stromu je následně generována iterativním postupem od počátečního uzlu, přičemž se nevyužívá kmen a rovnou se vytvářejí větve. Na obrázcích 7b a 7c je pak zachyceno postupné formování kostry stromu. [11]



Obrázek 7 a) Obálka s přitažlivými body a počátečním uzlem stromu; b, c) Generování kostry stromu.

<https://algorithmicbotany.org/papers/colonization.egwnp2007.large.pdf>

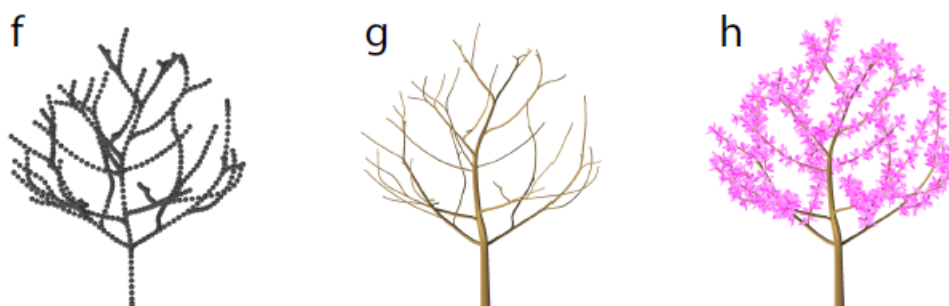
Proces je ukončen v okamžiku, kdy jsou odstraněny všechny přitažlivé body, kdy již nejsou žádné uzly v okruhu vlivu zbývajících přitažlivých bodů, nebo kdy je dosažen stanovený počet iterací. Uzly kostry mohou být dále redukovány, čímž se snižuje množství dat reprezentujících geometrii stromu. Přesunutím každého uzlu paralelně na polovinu cesty směrem k jeho bazálnímu sousedu a zmenšením úhlů větvení (viz obrázky 8d a 8e) může být výrazně ovlivněn výsledný vzhled stromu. [11]



Obrázek 8 d) Redukce uzlů; e) Přemístění uzlů, úprava úhlů větvení.

(<https://algorithmicbotany.org/papers/colonization.egwnp2007.large.pdf>)

Rozšířením křivkového dělení na větvené struktury původní či redukované kostry, je dosaženo přirozeného zakřivení spojů mezi body (9f). Následně jsou prováděny dokončovací kroky, při nichž je geometrie stromu modelována pomocí zobecněných válců vedených po ose kostry (9g). Výpočet předpokládá, že všechna zakončení větví mají stejný počáteční poloměr a směrem k základně stromu mohou být následně bazipetálně doplněny také další detaily, například květy či listy (9h).



Obrázek 9 f) Křivkové dělení; g) Konstrukce zobecněných válců; h) Doplnění detailů.

(<https://algorithmicbotany.org/papers/colonization.egwnp2007.large.pdf>)

3 GENERACE MĚST A CEST

Procedurální generování měst a cest je technika používaná k vytváření automatických stavebních struktur. Města jsou charakterizována složitými vazbami. Cesty a rozmístění jednotlivých budov musí být navrženy tak, aby dávaly smysl, s akcentem na historická, terénní a realistická rozmístění. Při aplikaci na budovy a silnice v herním designu nebo simulacích, umožňuje procedurálního generování, tvorbu dynamického, různorodého a rozsáhlého prostředí, bez nutnosti ručního zpracování jednotlivých detailů. [12]

3.1 Algoritmy pro generaci budov

3.1.1 Cellular automata

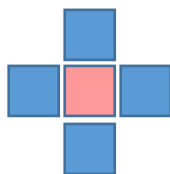
Cellular automato – Celulární automat (CA) je výpočetní model, užívaný v informatice, fyzice a bioinformatice, jako model pro výpočet růstu a vývoje určitých jevů. Model se skládá z n-dimenzionální mřížky, množiny stavů a množiny pravidel vývoje. Nejčastěji používané mřížky jsou jednorozměrné (vektory) nebo dvourozměrné. [13]

Většina modelů obsahuje pět základních rysů:

- diskrétní mřížka buněk se systémově skládá z jedno, dvou, případně trojrozměrné mřížky buněk,
- stejnorodost stanoví, že všechny buňky jsou rovnocenné,
- diskrétní stavy definují, že každá buňka může dosáhnout jednoho stavu z konečné diskrétní množiny stavů,
- lokální interakce stanoví, že každá buňka interaguje pouze s těmi buňkami, které spadají do jejího okolí,
- paralelismus v každém diskrétním kroku výpočtu mění stavy buněk současně. Každá buňka upravuje svůj stav podle přechodového pravidla s ohledem na stavy buněk ve svém okolí.

U CA výpočetních modelů pracujících s dvourozměrnou mřížkou buněk, se setkáváme nejčastěji se dvěma druhy okolí.

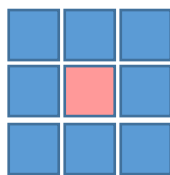
Von Neumannovo okolí zahrnuje čtyři buňky, které sousedí s centrální buňkou ve vertikálním a horizontálním směru (nahore, dole, vlevo, vpravo).



Obrázek 10 Okolí buňky CA – von Neumannovo okolí.

(zdroj vlastní)

Mooreovo okolí naopak zahrnuje osm buněk, tedy všechny buňky, které bezprostředně obklopují centrální buňku, včetně diagonálních sousedů.



Obrázek 11 Okolí buňky – Moorovo okolí

(zdroj vlastní)

Celulární automata jsou efektivním nástrojem pro modelování prostorových procesů, včetně generování měst a jejich rozvoje. Městské oblasti se mohou vyvíjet na základě jednoduchých pravidel, která určují změnu stavu jednotlivých buněk v prostoru. Každá buňka v CA může reprezentovat konkrétní část městské oblasti, ulici, blok, nebo například městskou čtvrť, přičemž jejich stav se následně může v čase, dle určených pravidel, dále měnit. [14]

Proces generování měst za použití CA:

- městský prostor je reprezentován obvykle 2D mřížkou, v níž mohou být buňkám přiřazovány různé stavy (neosídlený prostor, zástavby, komerční zóny, ulice, zeleň). Interakce probíhá mezi sousedními buňkami, což umožní dynamickou změnu v průběhu času,
- vývoj měst je spojen s přechodem mezi stavy, přičemž každá buňka může přecházet mezi různými stavy na základě počtu sousedů v určitých stavech. Například, buňka označená jako „prázdná parcela“ může být převedena na stav „zástavba“, pokud je obklopena dostatečným počtem sousedů ve stavu „zástavba“. Růst a rozšiřování města je umožňováno tam, kde jsou nové zóny vytvářeny podle dostupnosti prostředí. Celulární automata tak umožní napodobovat vzory, odpovídající skutečným městským strukturám,

- celulární automata lze využít k simulaci růstu města nejen v průběhu času, ale i podle geografických, či socio-ekonomických faktorů. CA umožňuje simulovat a optimalizovat také rozložení městské infrastruktury, například silnice, železnice, komerční zóny, vodní toky města atd.

Výhodou použití Celulárních automat pro generování měst je jejich jednoduchost, efektivita a flexibilita. CA jsou relativně jednoduché na implementaci, přesto mohou generovat velmi složité a realistické výsledky.

3.1.2 Agent-based model (ABM)

Další výpočtový algoritmus, je založen na bázi simulací a interakcí autonomních agentů. Vlastnosti systémů vyšší úrovně vycházejí z interakcí subsystémů nižší úrovně, ve snaze vytvořit a předpovědět výskyt složitějších jevů a struktur. Metoda založená na agentech pracuje nad rastrovou mřížkou. Uvnitř se pohybují agenti, stavějící každý svou část města, dle specifikované sady pravidel chování. Obecně jde o pravidla definující lokální charakter vytvářeného obsahu, na rozdíl například od přepisovacích pravidel L-systémů, postihuje globální rysy. [15]

Agenti jsou považováni za základní, klíčové jednotky modelu. Může jimi být zastoupena různorodá entita. Interakcemi mezi agenty je umožněna komunikace, vzájemné ovlivňování jejich rozhodování a chování. Pravidla jejich chování mohou být nastavena jednoduše, agent z určité oblasti je pouze přesunut do oblasti jiné, nebo složitější, kdy jsou pravidla založena na strojovém učení nebo adaptivním chování. Interakce mezi nimi mohou vést k emergentnímu chování na úrovni celého systému. To znamená, že složité vzorce, nebo chování mohou být vytvářeny bez nutnosti centrálního řízení nebo složitého plánování.

Například „developeři“, agenti pro stavby budov, jsou tvořeni dvojího typu. Komerční developeři preferují pro stavění budov oblasti v těsné blízkosti silnic, naproti tomu rezidentní developeři se těmito oblastem vyhýbají. Agenty obou typů je usilováno o vzájemné shlukování za předpokladu, že bude zamezeno velkým rozdílům v hustotě zalidnění. Postupují po silnicích a hledají volná políčka, v jejich blízkosti, vhodná pro navržení na nové stavební parcely, seskupením více políček. [15]

3.1.3 Wave function collapse

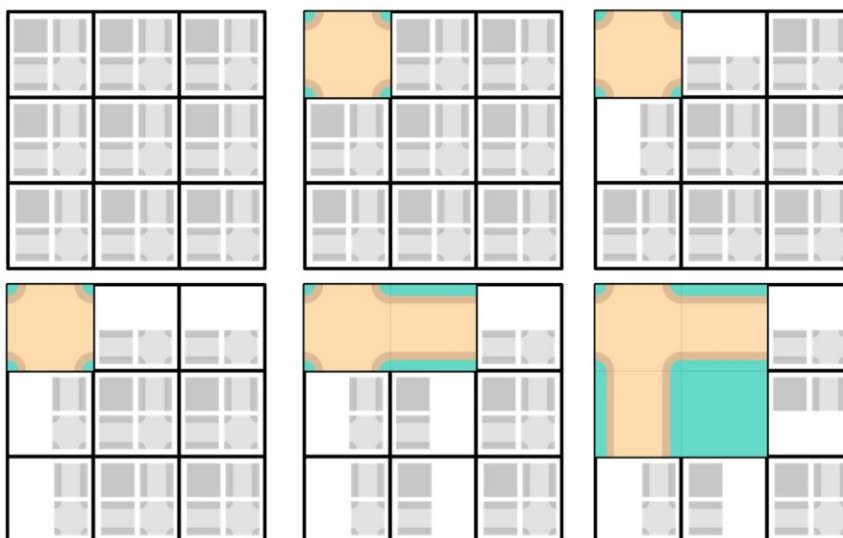
V algoritmu Wave function collapse (WFC) je plocha rozdělena mřížkou na buňky. Cílem je vyplnit mřížku buňkami tak, aby se sousední dlaždice navzájem spojily. Primárně zvolená buňka určuje pravidla pro výběr dalších, odlišných, přesto navazujících buněk, za předpokladu zabránění nelogického spojování. Každá buňka musí být kompatibilní s těmi, které jsou v přímém sousedství, tedy primárně zvolená buňka rozhoduje o tom, které dlaždice mohou být umístěny a kde jsou omezení. Připojená buňka se stává opět proměnnou pro výběr další dlaždice.

Algoritmus pracuje s malými vzory neboli "dlaždicemi", které jsou získané z nějakého referenčního vzoru, například textury nebo mapy. Dále algoritmus v sousedních buňkách zvolí texturu tak, aby odpovídala již vyplněné buňce a aby byl vytvořen předpoklad pro šíření vlny. Algoritmus pokračuje v tomto procesu, dokud není celý prostor (například mapa nebo textura) vyplněn konkrétními dlaždicemi. Pokud se proces dostane do stavu, kdy není možné najít kompatibilní dlaždici pro nějaký bod, algoritmus musí zpětně upravit předchozí rozhodnutí.



Obrázek 12 Vzorky k vyplnění mřížky

(zdroj: <https://www.boristhebrave.com/2020/04/13/wave-function-collapse-explained/>)



Obrázek 13 Princip algoritmu WFC

(zdroj: <https://www.boristhebrave.com/2020/04/13/wave-function-collapse-explained/>)

3.2 Algoritmy pro generaci dopravní infrastruktury

Generování silničních sítí pomocí procedurálních algoritmů umožňuje tvorbu efektivní a často rozmanité dopravní infrastruktury, přičemž vychází z předem definovaných pravidel, s cílem zohledňovat faktory jako terén, typ zástavby, hustotu obyvatelstva, přirozené překážky, optimalizovat rozmístění dopravních tras, a to i z hlediska efektivity a estetiky. [16]

3.2.1 Dijkstrův algoritmus

Dnes již klasickým, přesto stále jedním z nejpoužívanějších algoritmů pro návrh dopravních tras v počítačových simulacích a modelech, je Dijkstrův algoritmus. Jeho činnost je založena na postupném rozšiřování dosažitelných uzlů, přičemž každému uzlu je přiřazována nejkratší známá vzdálenost od počátečního bodu. Klíčovým prvkem je využívání prioritní fronty, kde uzly s nejnižší dosud nalezenou vzdáleností jsou zpracovávány jako první.

Algoritmus je spuštěn nastavením vzdálenosti počátečního uzlu na nulu a všech ostatních uzlů na nekonečno. Následně je opakovaně vybírán uzel s nejnižší aktuální hodnotou vzdálenosti a jsou aktualizovány vzdálenosti jeho sousedů na základě hodnot hrany, která je spojuje. Tento proces pokračuje, dokud nejsou všechny uzly zpracovány, nebo dokud není nalezena cesta ke konečnému cíli. V případě, že jsou v grafu obsaženy záporné hodnoty hran, pak již užití Dijkstrova algoritmu není vhodné, proto pro tyto případy je Dijkstrův algoritmus nahrazen např. Bellman-Fordůvým algoritmem.

V kontextu generování cest mezi městy, například v simulátorech, je umožněno využití Dijkstrova algoritmu v kombinaci s procedurálně generovaným terénem (například s využitím Simplexního šumu). Terénní data jsou integrována tak, že jsou každému uzlu nebo hraně přiřazovány specifické náklady, jež odpovídají obtížnosti průchodu daným úsekem terénu. Díky tomu je dosaženo nalezení nejen nejkratší cesty, ale jsou také respektovány specifické podmínky simulovaného prostředí.

3.2.2 Algoritmus A*

Za jednu z nejefektivnějších metod pro hledání nejkratších cest v grafových strukturách a při generování cest mezi městy, je odbornou veřejností považován a široce doporučován algoritmus A*. Algoritmus je založen na kombinaci systematického vyhodnocování uzlů

s využitím heuristiky, která je nastavena tak, aby byl odhadnut zbývající náklad na cestu k cílovému uzlu.

Při tomto procesu je využívána otevřená množina, do které jsou postupně zařazovány kandidátní uzly, a uzavřená množina, do níž jsou přesouvány uzly po jejich zpracování. Díky této metodě je zaručeno, že je nalezena cesta, jež minimalizuje celkové náklady, pokud je heuristika navržena konzistentně.

V prostředí, kde je terén generován pomocí procedurálních metod, jako je Simplexní šum, mohou být náklady na pohyb integrovány do výpočtu heuristiky. Tak je umožněno, aby byl výsledek výpočtu ovlivněn skutečnými podmínkami terénu, což vede k realistickému modelování cest mezi městy. Výsledné trasy jsou tak nejen nejkratší, ale i přizpůsobeny specifikům simulovaného prostředí.

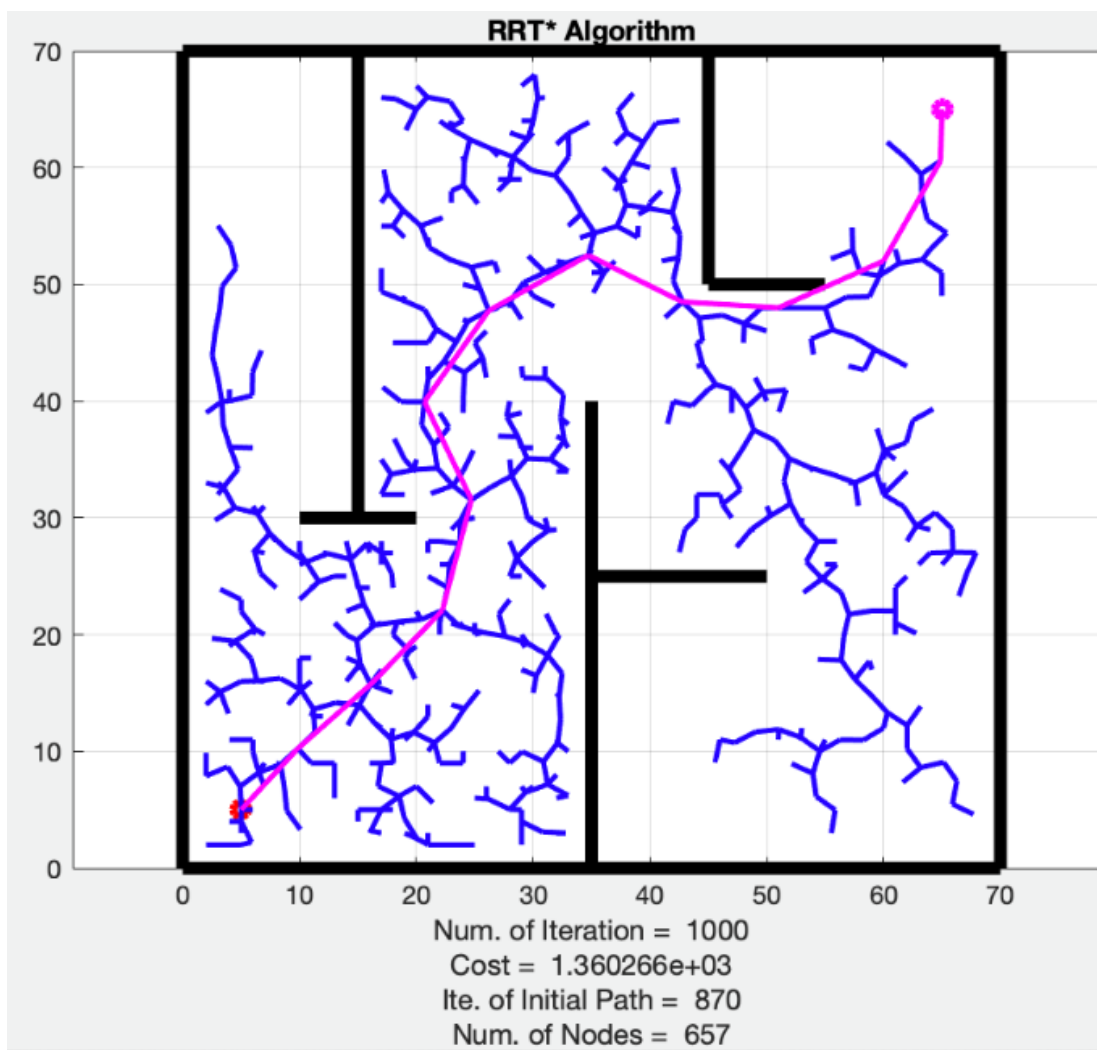
3.2.3 Algoritmus RRT (Rapidly exploring Random Tree)

RRT je algoritmus využíván pro průzkum prostoru a vyhledávání cest v prostředích, kde jsou podmínky často dynamické a složité. Principem jeho činnosti je náhodné rozšiřování stromu uzlů, do prostoru je periodicky vkládán nový bod, a nejbližší existující uzel je propojen s tímto novým bodem. Takto vzniká síť cest, která pokrývá velkou část prostoru a umožňuje rychlé nalezení cesty, od výchozího bodu k cíli.

V praxi vytvořením stromu, do kterého je nejprve zařazen pouze počáteční uzel představující výchozí bod, se následně v každé iteraci generuje náhodný bod uvnitř definovaného prostoru, což zajišťuje systematické prozkoumání celého prostředí. Pro každý takto vzniklý bod je směrem k náhodně vybranému bodu, s využitím předem definované maximální krokové délky, vyhledán nejbližší existující uzel ve stromu, od kterého bude provedeno rozšíření. Tato omezená vzdálenost zaručuje, že nově vznikající uzly jsou umístěny v dosažitelných intervalech a zároveň ověřuje, zda cesta mezi původním uzlem a novým bodem není zablokována překážkami. Pokud je zvolená cesta průchozí, nový uzel je přidán do stromu.

Po každém přidání se kontroluje, zda nový uzel není umístěn v dostatečné blízkosti cílového bodu. Pokud je tato podmínka splněna, vyhledávání se ukončí a výsledná cesta mezi výchozím bodem a cílem je považována za nalezenou. V opačném případě se celý proces od generování náhodného bodu po rozšíření stromu opakuje, dokud není dosaženo cíle, nebo dokud není překročen předem stanovený počet iterací.

Schopnost adaptovat se na měnící se podmínky, efektivní prozkoumání rozsáhlého prostoru a rychlou reakci na případné změny či překážky v prostředí, činí RRT algoritmus vhodným nástrojem pro aplikace v robotice, simulacích i herním designu. Algoritmus se neřídí striktně deterministickými pravidly, a proto není vyloučeno, že nalezená cesta nebude absolutně nejkratší. Přesto bývá často považován za dostačující a efektivní pro situace, kdy je kladen důraz na rychlost a flexibilitu. V prostředí, kde tradiční algoritmy mohou selhávat kvůli vysoké složitosti prostředí, a kde je potřeba rychle reagovat na neustále se měnící podmínky, je pro své vlastnosti využíván právě RRT algoritmus.



Obrázek 14 Hledání optimální cesty za pomoci RRT

(zdroj: https://www.researchgate.net/figure/Performance-of-RRT-Algorithm-1-st-Case_fig4_366866319)

II PRAKTICKÁ ČÁST

4 VÝVOJOVÉ PROSTŘEDÍ

Pro vývoj nástroje bylo zvoleno vývojové prostředí Unity ve verzi 2022.3.11f1, a to především z důvodu jeho vhodnosti pro kompletní implementaci nástroje pro procedurální generování herního světa a podpory dalších vývojových kroků, jako je export dat a celé mapy do cílové herní aplikace. Významným faktorem při volbě tohoto prostředí byla skutečnost, že samotný tankový simulátor UTW je vytvořen v prostředí Unity, což zajišťuje vyšší kompatibilitu a zjednodušuje integraci vytvořeného nástroje do simulátoru. [17]

Programový kód byl vytvářen v integrovaném vývojovém prostředí JetBrains Rider 2023.3.6, které bylo zvoleno pro svou uživatelskou přívětivost a předchozí zkušenosti s jeho používáním. K implementaci byl použit programovací jazyk C#, jenž představuje primární jazyk podporovaný prostředím Unity.

4.1 Knihovny

Za účelem usnadnění vývoje byly zvoleny dvě knihovny, které spolehlivě podporují implementaci a využití vybraných algoritmů. První zvolená knihovna FastNoiseLite umožňuje generování širokého spektra typů šumu, včetně klasického Perlinova, Simplexního i Worleyho šumu. Další knihovna PoissonDiskSampling implementuje algoritmus Poissonova vzorkování, využívaný například pro rovnoměrné rozmístění bodů v prostoru bez vzniku shluků.

Z hlediska licencování jsou obě tyto knihovny šířeny pod licencí MIT (Massachusetts Institute of Technology), která umožňuje jejich volné použití, úpravy i distribuci, což je plně v souladu s požadavky tohoto projektu.

4.2 Unity assety

V prostředí Unity se pojem asset (česky „prostředek“ či „prvek“) vztahuje na jakýkoli soubor nebo datovou strukturu, která je součástí projektu a slouží k tvorbě herního obsahu. Mezi typické příklady assetů patří 3D modely, textury, materiály, zvukové soubory, animace, skripty, prefaby, scény a další komponenty, které dohromady tvoří výslednou aplikaci, nebo hru.

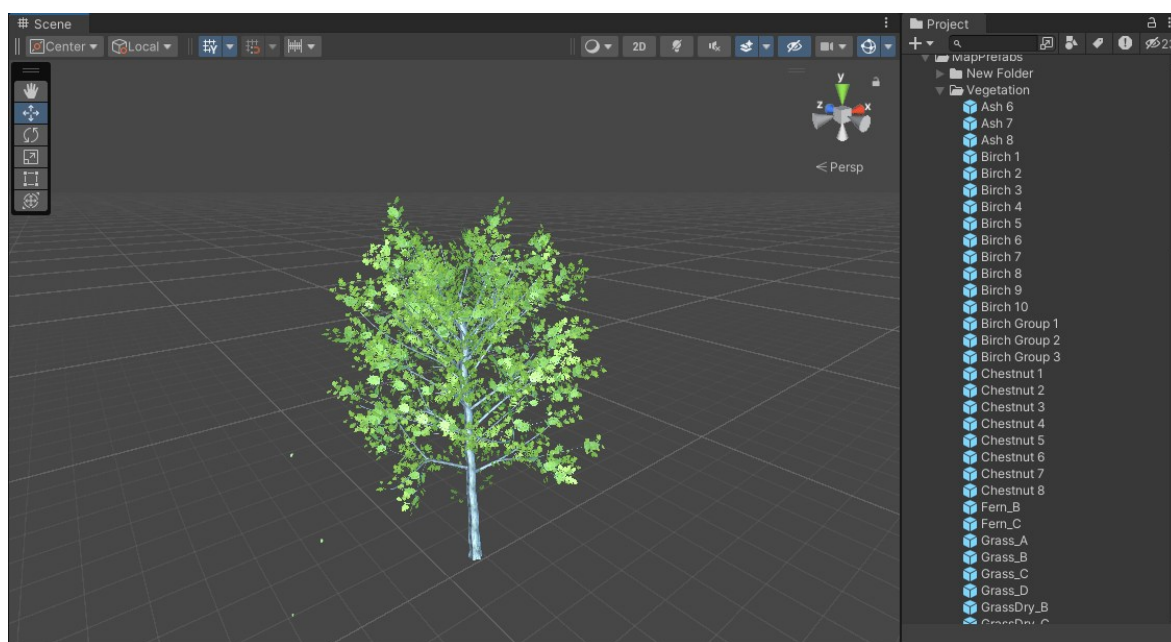
K usnadnění implementace a jako výchozí zdroj použitých herních objektů jako stromy, keře nebo materiály byly do projektu naimportovány tyto balíčky assetů:

- Terrain Sample Asset Pack,
- European Forests – Realistic Trees,
- Demo City By Versatile Studio.

Tyto balíčky obsahují základní assety potřebné pro testování a demonstraci funkčnosti nástroje. V případě potřeby rozšíření funkcionality, nebo vizuální stránky projektu si však uživatel musí další assety nainportovat samostatně do příslušných složek projektu.

Všechny uvedené balíčky jsou dostupné zdarma a jejich použití se řídí podmínkami služby Unity Asset Store Terms of Service and EULA.

Assety jsou importovány z Unity asset store prostřednictvím Package manageru a následně stáhnuty do projektu.



Obrázek 15 ukázka stromu Ash 8 z balíčku European Forests – Realistic Trees ve scéně

(Zdroj: vlastní)

Každý asset má editovatelné parametry, které může uživatel dle potřeby dále upravovat, například pozici, rotaci a škálu, materiál, model atd.

5 POROVNÁNÍ ZKOUMANÝCH ALGORITMŮ

Zvolením vhodného algoritmu, který se bude řídit předem stanovenými parametry, je možné dosáhnout efektivního generování herního světa, a to na optimální úrovni a v souladu s očekáváním uživatele.

5.1 Stanovení parametrů

Porovnání jednotlivých algoritmů procedurální generace, v praktické části projektu, respektuje jednotlivé typy obsahu – terén, biomy a vegetace – shodně s kategoriemi řešenými již v teoretické části. Hodnocení relevantních algoritmů se provádí podle čtyř klíčových kritérií vyjádřených slovním hodnocením, v rozmezí velmi nízké až velmi vysoké:

- realističnost výstupu – srovnává do jaké míry výsledek generace připomíná struktury reálného světa, od hrubých a geometrických útvarů (velmi nízké) až po plynulé a realisticky vypadající útvary (velmi vysoké),
- časová náročnost – měří jak moc, daný algoritmus, svou implementací zatíží, a tedy zpomalí proces generace (měla by být co nejnižší). Čas generování v jednotkách či nízkých desítkách vteřin je vyhodnocen jako velmi nízký, naproti tomu v minutách až desítkách minut jako velmi vysoký,
- procesní složitost – sleduje matematickou a implementační obtížnost. Kritérium popisuje obtížnost implementace modulu algoritmu v nástroji. Obdobně jako časová náročnost by měla procesní složitost nabývat co nejmenších hodnot,
- vhodnost pro simulátor – vyhodnocuje schopnosti algoritmu, generovat útvary podporující bezproblémový pohyb tanků, dle očekávání uživatele. Terén s ostrými přechody tvoří pro pohyb tanku nežádoucí překážky, a proto je hodnocen jako velmi nízký, naopak plynulý spojitý terén je hodnocen jako velmi vysoký.

5.1.1 Výběr algoritmu pro generování terénu

Pro generování terénu (výškové mapy krajiny) byly uvažovány následující algoritmy: Interpolační algoritmy bilineární a bikubické, gradientní Perlinův, Simplexní a fraktální šum (kombinace více oktáv Perlinova či Simplexního šumu), algoritmus Diamond-Square (midpoint displacement) a simulace eroze a dalších algoritmů na bázi agentů. Každý z těchto přístupů má odlišné vlastnosti z hlediska vytvářených tvarů krajiny, nároků na výpočet, času pro generování a složitosti realizace.

Tabulka 1 Srovnání algoritmů pro generování terénu

Algoritmus	Realističnost výstupu	Časová náročnost	Procesní složitost	Vhodnost pro simulátor
Bilineární	Velmi nízká	Nízká	Velmi nízká	Nízká
Bikubický	Nízká	Nízká	Nízká	Nízká
Perlinův šum	Střední	Nízká	Střední	Střední
Simplexní šum	Vysoká	Velmi nízká	Střední	Vysoká
Fraktální šum	Vysoká	Nízká	Střední až vyšší	Vysoká
Diamond Square	Střední	Velmi nízká	Nízká	Střední
Na bázi Agentů	Velmi vysoká	Vysoká	Vysoká	Vysoká

(Zdroj: Vlastní)

Z výše uvedené tabulky vyplývá, že nejlepších výsledků z hlediska efektivity i kvality dosahují algoritmy založené na šumu. Ačkoliv je algoritmus Diamond-Square velmi rychlý a poskytuje poměrně realistické výstupy, nedosahuje úrovně Perlinova či Simplexního šumu. Algoritmy využívající agentní přístup sice generují nejrealističtější terény, avšak jejich výpočetní náročnost při tvorbě rozsáhlých map představuje značnou zátěž pro systém, což je z pohledu této implementace nežádoucí. Z pohledu kompromisu mezi realistickým vzhledem a časovou efektivitou se jako nejvhodnější jeví právě šumové algoritmy, zejména jejich fraktální kombinace. Přestože implementace fraktálního šumu může být technicky náročnější, jeho výhody v podobě detailnosti a vizuální věrohodnosti převyšují případné nevýhody.

5.1.2 Výběr algoritmu pro generování biomů

Pro generování biomů, tedy samostatných oblastí herní mapy, byly uvažovány tyto algoritmy: Whittakerův diagram, Voronoiův diagram, Worleyho šum, Flood fill, Region growing algoritmus a Wave function collapse. Ačkoli výstupem všech těchto algoritmů bude technicky vzato rozdělení herní mapy na určité oblasti, ne všechny současně umožňují realistické zobrazení a zároveň dostatečnou míru uživatelské podpory, v rozsahu potřeb a očekávání tohoto projektu.

Tabulka 2 Srovnání algoritmů pro generování biomů

Algoritmus	Realističnost výstupu	Časová náročnost	Procesní složitost	Vhodnost pro simulátor
Whittakerův diagram	Vysoká	Velmi nízká	Nízká	Vysoká
Voronoiův diagram	Střední	Nízká	Střední	Střední
Worleyho šum	Střední	Nízká	Střední	Střední
Flood fill	Střední	Velmi nízká	Nízká	Nízká
Region Growing	Střední	Nízká	Nízká	Střední
Wave function collapse	Vysoká	Vysoká	Vysoká	Vysoká

(Zdroj: Vlastní)

Mezi hlavní kandidáty pro generaci biomů byly, jak lze z tabulky odvodit Whittakerův diagram, Flood fill algoritmus, Voronoiův diagram a Worleyho šum. I když se může na první pohled zdát Whittakerův diagram jako obecně nejlepší volba dle daných kritérií, v případě tohoto projektu tomu tak není. Pravidla, ze kterých whittakerovo rozdělení vychází, by mohla být příliš omezující a determinující pro generování biomu, čemuž se chceme vyhnout. Algoritmus Flood fill šíří své biomy plošně, což není v tomto případě vhodné. Navíc napojení jeho výstupů na vyšší logiku, například generace vegetace, měst apod. je obtížné. Voronoiův diagram je výborným řešením, které nabízí kontrolovaný počet biomů s důrazem na jejich náhodné a rovnoměrné rozmístění v prostředí, avšak může zobrazovat buněčné obrazce.

Z výše uvedeného šetření vyplývá, že neoptimálnější volbou pro tento projekt je Worleyho šum. Ten kombinuje rozdělení mapy ověřenou metodou Voronoiova diagramu s šumovou funkcí tak, aby se co nejvíce odstranily artefakty obrazců, a zároveň zůstala uchována dostatečná rychlost a realističnost generace.

5.1.3 Výběr algoritmu pro generování vegetace

Generování vegetace se v této části zaměřuje na rozmístění individuálních objektů (stromů, keřů, tráv) v rámci terénu a jejich biomů, nikoliv na jejich vnitřní strukturu. Porovnány byly následující algoritmy: Poisson disk sampling, Clustering, Algoritmy na bázi simulace růstu a celulární automata. Každý z výše uvedených algoritmů přistupuje k řešení problémů jiným způsobem.

Tabulka 3 Srovnání algoritmů pro generování vegetace

Algoritmus	Realističnost výstupu	Časová náročnost	Procesní složitost	Vhodnost pro simulátor
Poisson Disk Sampling	Středně	Velmi nízká	Střední	Střední
Clustering	Vysoká	Nízká	Nízká	Vysoká
Simulace růstu	Velmi vysoká	Vysoká	Vysoká	Vysoká
Celulární automata	Střední až vyšší	Střední	Střední	Vysoká

(Zdroj: Vlastní)

Generování vegetace samotné, závisí na biomech a jejich navržení, důležitá jsou také pravidla, určující, kde se smí a nesmí rostliny nacházet. Pro vysokou časovou a procesní náročnost, byly vyřazeny algoritmy Simulace růstu a Celulárních automat. Poisson disk sampling je velmi rychlý algoritmus a požadovanou oblast pokryje správnou vegetací z množiny prvků také v realistických vzdálenostech. Bohužel pokud se jedná o rozsáhlejší vegetační struktury, jako lesy a hustší porosty, bude rozmístění všech rostlin hodně rozptýlené, a proto i méně realistické.

Clustering má také nízkou časovou náročnost, jeho nevýhodou je však přílišné rozdělení a náhodné rozmístění vegetace uvnitř clusteru. Proto bylo spojení těchto dvou přístupů, kdy Clustery určí místa pro rozmístění a následně Poisson disk sampling zaručí rovnoměrné pokrytí vhodnou vegetací, vyhodnoceno jako neoptimálnější řešení.

6 IMPLEMENTACE NÁSTROJE V PROSTŘEDÍ UNITY

Na základě teoretické přípravy, nových zjištění a s využitím zvolených nástrojů a algoritmů, je možné realizovat programovou část projektu. Uvedené moduly výsledného nástroje jsou naprogramovány v prostředí JetBrains Rider v jazyce C#.

6.1 Vytvoření terénu mapy

Základní terén mapy je nejdůležitějším prvkem celé generace, jelikož udává rozlohu herní mapy a její výšková maxima a minima. Do této sekce můžeme zařadit i rozdělení mapy na biomy nebo regiony, které nejen že upravují generaci prvků terénu, ale i vegetace, měst, AI (Artificial intelligence) atd.

6.1.1 Mapa biomů

Před vytvářením výškové mapy je zapotřebí rozhodnout v jakém poměru bude daný biom ovlivňovat každý z bodů mapy. K tomuto napomáhá skript BiomeMapGenerator.cs generací mapy vah biomů. Jedná se o jednoduché trojrozměrné pole, jež nám určuje přesné souřadnice bodu a poměrnou váhu každého biomu v něm. Součet všech vah biomů musí být vždy roven 1 a nesmí tento práh přesáhnout.

```
float[, ,] weightMap = new float[height, width, biomeCount];
```

Modul nejprve vytvoří seznam s náhodnými centry biomů. Poté projde dvěma cykly celé pole dostupných bodů mapy v ose X a Z. Pro každý bod vypočítá jeho příslušnost k jednotlivým biomům.

```
for (int biomeIndex = 0; biomeIndex < biomeCount; biomeIndex++)
{
    Vector2Int centerPos = biomeCenterPositions[biomeIndex];

    float deltaX = x - centerPos.x;
    float deltaZ = z - centerPos.y;
    float euclideanDistance = Mathf.Sqrt(deltaX * deltaX + deltaZ *
        deltaZ);

    float distortionValue = distortionNoise.GetNoise(x, z);
    float distortedDistance = euclideanDistance + (distortionValue *
        distortionAmplitude);
```

```
float weight = Mathf.Exp(-(distortedDistance * distortedDistance)
/ (2f * gaussianSigma * gaussianSigma));
biomeWeights[biomeIndex] = weight;
totalWeightSum += weight;
}
float distortionAmplitude = 30f;
float gaussianSigma = 50f;
```

Výsledná vzdálenost bodu od centra biomu vznikne součtem euklidovské vzdálenosti s násobkem zkresleného Worleyho šumu. Ten vzniká násobením šumu proměnnou `distortionAmplitude`. Její hodnota byla určena postupným testováním vstupů do nalezení nejvhodnějšího výsledku. Proces pokračuje výpočtem přibližné váhy příslušnosti k biomu za pomoci nedokonalé Gaussovy funkce. Posléze se dopočítá totální váha pro všechny biomy kvůli následné normalizaci.

```
for (int biomeIndex = 0; biomeIndex < biomeCount; biomeIndex++)
{
weightMap[z, x, biomeIndex] = biomeWeights[biomeIndex] /
totalWeightSum;
}
```

Jako poslední krok se provede normalizace vah všech biomů v bodě tak, aby byla splněna podmínka, že jejich součet nesmí přesáhnout hodnotu 1. Následné generování a vyhlazení přechodů mezi biomy by v jiném případě generovalo nerelevantní hodnoty.

6.1.2 Výšková mapa

Výšková mapa slouží k vytvoření 3D struktury terénu. Za její generaci odpovídá skript `HeightMapGenerator.cs`. Skládá se ze dvou hlavních cyklů. První cyklus projde všechny známé biomy a vytvoří jejich výškové mapy za pomoci fraktálního šumu, které následně uloží do pole výškových map. Druhý cyklus kombinuje výškové mapy z vytvořeného pole a na základě vah biomů upraví hodnotu celkové výšky v daném bodě. Výsledkem je 2D výšková mapa terénu s hladkými přechody mezi biomy.

Jelikož generátor šumu `FastNoiseLite` vrací hodnoty v rozsahu $[-1, 1]$, je nutné tyto výstupy normalizovat do intervalu $[0, 1]$ pomocí lineární transformace. Tím je zajištěno, že výsledné hodnoty výšky jsou vždy nezáporné. Současná implementace výškové mapy tedy neumožňuje výšky pod hladinou nula ($y = 0$), což omezuje možnost modelování jezer, moří nebo podzemních prostor.

6.1.3 Utvoření finálních výšek biomů

Po vygenerování výškové mapy přichází na řadu modul BiomeHeightModifier.cs, který upravuje výšky na základě parametrů jednotlivých biomů. Každý biom může mít definovaný vlastní výškový multiplikátor (heightMultiplier), který například způsobí, že horské oblasti budou celkově výrazně vyvýšené oproti rovinám.

Algoritmus pracuje s váhovou mapou biomů a pro každý bod vypočte vážený průměr multiplikátorů podle zastoupení biomů. Zároveň se zohledňuje ostrost přechodů mezi oblastmi biomů – čím prudší přechod, o to menší bude nutná korekce výšky, čímž se docílí přirozených hranic mezi zónami biomů.

```
float finalMultiplier = (totalWeight > 0f) ? weightedMultiplier /
    totalWeight : 1f;

float gradientMagnitude = ComputeBiomeWeightGradient(biomeWeightMap,
    x, z, biomeCount, width, height);
float blendFactor = Mathf.Clamp01(1f - gradientMagnitude * 1.5f);
blendFactor = Mathf.SmoothStep(0f, 1f, blendFactor);

float baseHeight = heightMap[z, x];
float modifiedHeight = baseHeight * finalMultiplier;

float maxDelta = 0.2f * baseHeight;
float delta = Mathf.Abs(modifiedHeight - baseHeight);
if (delta > maxDelta)
{
    modifiedHeight = baseHeight + Mathf.Sign(modifiedHeight -
        baseHeight) * maxDelta;
}

heightMap[z, x] = Mathf.Lerp(baseHeight, modifiedHeight, blendFactor *
    0.7f);
```

Dodatečně se při výpočtu ošetřuje také maximální přípustná změna výšky, což zabráňuje náhlým výškovým skokům, které by mohly působit nerealisticky. Metoda ComputeBiomeWeightGradient vypočítá hodnotu gradientní váhy všech biomů v okolí každého bodu. Tento gradient vyjadřuje míru změny váhy jednotlivých biomů v daném bodě. Algoritmus porovnává rozdíl váhy daného biomu s jeho čtyřmi sousedy a vypočítá průměr těchto rozdílů. Nakonec je vybrána maximální hodnota přes všechny biomy, čímž se získá celková „ostrost“ přechodu. Pokud je gradient nízký (biomová mapa

se mění plynule), úprava výšky je výraznější. V případě prudkých přechodů je efekt výškové modifikace potlačen, aby nevznikaly viditelné skoky ve výšce.

Výsledkem je tak hladce modulovaná výšková mapa, která odpovídá charakteru jednotlivých biomů, ale zároveň zachovává vizuální kontinuitu a hladkost v místech, kde se biomy střídají.

6.1.4 Vytvoření mřížky mapy

Po vytvoření výškových a biomových dat, je potřeba tato data převést do trojrozměrné reprezentace. Tato operace je realizována pomocí metody `GenerateTerrainMesh.cs`. Z důvodu lepší kontroly nad výstupy procedurální generace, bývá mesh neboli mřížka mapy ve vlastních systémech generována programově.

Mesh tvoří základní strukturální prvek každé herní mapy, neboť definuje nejen samotnou topografii, ale i geometrickou přesnost, na které dále závisí další systémy – například vegetace, rozmístění objektů, nebo navigační AI.

Při programovém generování meshe je nejdůležitější správně definovat a aktualizovat:

- vrcholy mřížky a jejich trojúhelníky které spojují vrcholy do ploch,
- namapovat UV (textury) souřadnice textur pro správné aplikování materiálů,
- přepočty normálů, díky kterým bude fungovat osvětlení a stínování správně.

Bez těchto kroků by výsledný mesh nebyl vizuálně ani funkčně použitelný.

6.1.5 Další terénní úpravy

Za účelem zvýšení realismu generovaného terénu byly po vytvoření výškové mapy provedeny dvě erozní simulace: tepelná a hydraulická. Obě techniky iterativně upravují výšková data a napodobují přirozené procesy, které formují krajinu v reálném světě.

Tepelná eroze simuluje sesuvy půdy ve strmých svazích. Pokud je rozdíl výšek mezi dvěma sousedními body větší než zadaná prahová hodnota (tzv. talus), část materiálu se přesune z vyššího bodu do nižšího. Tento mechanismus snižuje ostré vrcholy a vytváří plynulejší profil terénu.

```

if (diff > talus)
{
    float amountToMove = (diff - talus) * 0.5f;
    deltaMap[z][x] -= amountToMove;
    deltaMap[nz][nx] += amountToMove;
}

```

Hydraulická eroze pracuje se simulací srážek, odtoku vody a sedimentace. Voda, která na povrch dopadne, se šíří do nižších oblastí, přenáší část povrchu jako sediment a po částečném odpaření ukládá zbylý materiál na nová místa. Tento typ eroze je velmi účinný pro tvorbu přirozených útvarů jako jsou říční koryta, doliny či delty.

```

for (int i = 0; i < 4; i++)
{
    if (flow[i] > 0)
    {
        int nx = x + offsets[i][0];
        int nz = z + offsets[i][1];
        float amount = (flow[i] / totalDelta) * waterMap[z][x] * 0.5f;
        waterMap[z][x] -= amount;
        waterMap[nz][nx] += amount;
        float sediment = amount * 0.05f;
        sedimentMap[z][x] -= sediment;
        sedimentMap[nz][nx] += sediment;
    }
}

```

Na oba procesy je aplikováno více iterací, čímž dochází k postupnému „modelování“ terénu do hladších a věrohodnějších tvarů.

6.1.6 Finalizace mapy

Po dokončení výpočtů výškové mapy, mapy biomů a generování meshe, je vytvořen nový herní objekt (GameObject), který slouží jako vizuální reprezentace výsledného terénu ve scéně. Objekt je vložen do hlavního kontejneru mapy a je mu případně přiřazen definovaný tag pro snadnější správu.

Tomuto objektu jsou přidány základní komponenty:

- MeshFilter – základní kontejner pro vygenerovaný 3D mesh,
- MeshRenderer – vykresluje daný mesh do scény,

- MeshCollider – obstarává fyzikální interakci ostatních objektů s meshem.

Vizuální stránku terénu určuje přiřazený materiál ze zvoleného dominantního biomu. Pokud není k dispozici, použije se výchozí materiál s Unity Standard shaderem. Tento přístup umožňuje vygenerovat plnohodnotný terén k zobrazení ve scéně, včetně užitých materiálů.

6.2 Tvorba vegetace na mapě

Po vytvoření celého terénu mapy, je možné navázat další kroky generování herního obsahu. Jedním z nich je vytvoření vegetace v oblastech, kde je to vhodné. Pro dosažení lepší kontroly herních vlastností jednotlivých rostlin, byl zvolen způsob generace z prefabů. Tedy každá jednotlivá rostlina může obsahovat mimo hlavní komponentu rostliny i „dětské“ komponenty, jež budou dále chování ve hře ovlivňovat. Například MeshCollider pro interakci s dalšími herními objekty. Celé generování vegetace obstarává komponenta VegetationGenerator.cs.

6.2.1 Příprava generace vegetace

Nežli se začnou jednotlivé herní objekty rostlin generovat, VegetationGenerator určí dominantní biom pro každý bod mapy. Poté načte a předpřipraví vegetační prefaby každého biomu do listu prefabů. V posledním kroku projde všechny dostupné biomy, ověří, zda a jaký druh vegetace se může v daném biomu generovat a zavolá příslušné metody pro obsluhu generování daných rostlin.

6.2.2 Vymezení ploch generace

V cyklech generace stromů a keřů je volána metoda GenerateClusteredPoints, která pomáhá s rozhodováním o finálním pokrytí biomu vegetací. Protože ne vždy uživatel chce, aby celý biom byl pokryt vegetací, může si zvolit procento pokrytí, a tedy ovlivnit, jak moc bude biom vegetací zabrán. Tato metoda využívá dvou výše zmíněných přístupů Poisson disk samplingu a Clusteringu pro rozdělení objektů na herní mapě.

```
int maxClusterCount = 20;

int clusterCount = Mathf.Clamp(Mathf.RoundToInt(Mathf.Lerp(1f,
    maxClusterCount, clusterDensity)), 1, maxClusterCount);

float totalTargetArea = coverage * width * height;
float clusterRadius = Mathf.Sqrt(totalTargetArea / (clusterCount *
    Mathf.PI));
```

```
float adjustedCoverage = Mathf.Clamp01(coverage * 1.25f);
```

Pokud je pokrytí mapy velké a Clustering by se nevyplatil, aplikuje se pouze sampling. V opačném případě jsou vytvořena centra shluků, v nichž se aplikuje sampling pro rovnoměrné pokrytí oblasti shluku. Nejprve se vypočte celkový počet shluků a jejich pokrytí tak, aby odpovídal co nejvíce nastavení od uživatele. Dále se vygenerují shluky, nad kterými je aplikován náhodný offset, filtrovaný výběr bodů a šum. Takové oblasti mají více organický vzhled a nepřipomínají klasické tvary, jež generuje samplig.

```
for (int c = 0; c < clusterCount; c++)
{
    int cx, cz;
    int attempts = 0;
    do
    {
        cx = Random.Range(0, width);
        cz = Random.Range(0, height);
        attempts++;
    } while (!isInBiome(cx, cz) && attempts < 100);

    if (attempts >= 100) continue;

    float jitterX = Random.Range(-0.5f, 0.5f) * clusterRadius * 0.2f;
    float jitterZ = Random.Range(-0.5f, 0.5f) * clusterRadius * 0.2f;
    Vector2 center = new Vector2(cx + jitterX, cz + jitterZ);

    List<Vector2> cluster = FastPoissonDiskSampling.Sampling(
        center - Vector2.one * clusterRadius,
        center + Vector2.one * clusterRadius,
        minDist * 0.8f);

    foreach (var p in cluster)
    {
        Vector2 local = p - center;
        local.x *= Random.Range(0.8f, 1.2f);
        local.y *= Random.Range(0.8f, 1.2f);

        float nx = p.x / width;
        float nz = p.y / height;
```

```
float noiseValue = Mathf.PerlinNoise(nx * 5f, nz * 5f);
    if (local.magnitude > clusterRadius || noiseValue < 0.4f)
continue;

    if (p.x >= 0 && p.x < width && p.y >= 0 && p.y < height &&
        isInBiome(p.x, p.y) && Random.value <= adjustedCoverage)
    {
        allPoints.Add(p);
    }
}
```

6.2.3 Generování vegetace

Po naplnění seznamu záznamů v `instancesToPlace`, složených z pozic ve scéně a prefabů jednotlivých rostlin, se vytvoří objekty ve scéně. Tyto se náhodně natočí a současně je upravena jejich velikost. Každý z nově vygenerovaných objektů je přiřazen společnému rodiči. Výsledkem je realistické a rozmanité rozmístění vegetace.

```
foreach (var (position, prefab) in instancesToPlace)
{
    GameObject obj = (GameObject)PrefabUtility.InstantiatePrefab(prefab);
    obj.transform.position = position;
    obj.transform.SetParent(vegetationParent.transform);
    float rotY = Random.Range(0f, 360f);
    obj.transform.rotation = Quaternion.Euler(0, rotY, 0);
    float scaleFactor = Random.Range(0.9f, 1.1f);
    obj.transform.localScale = Vector3.one * scaleFactor;
}
```

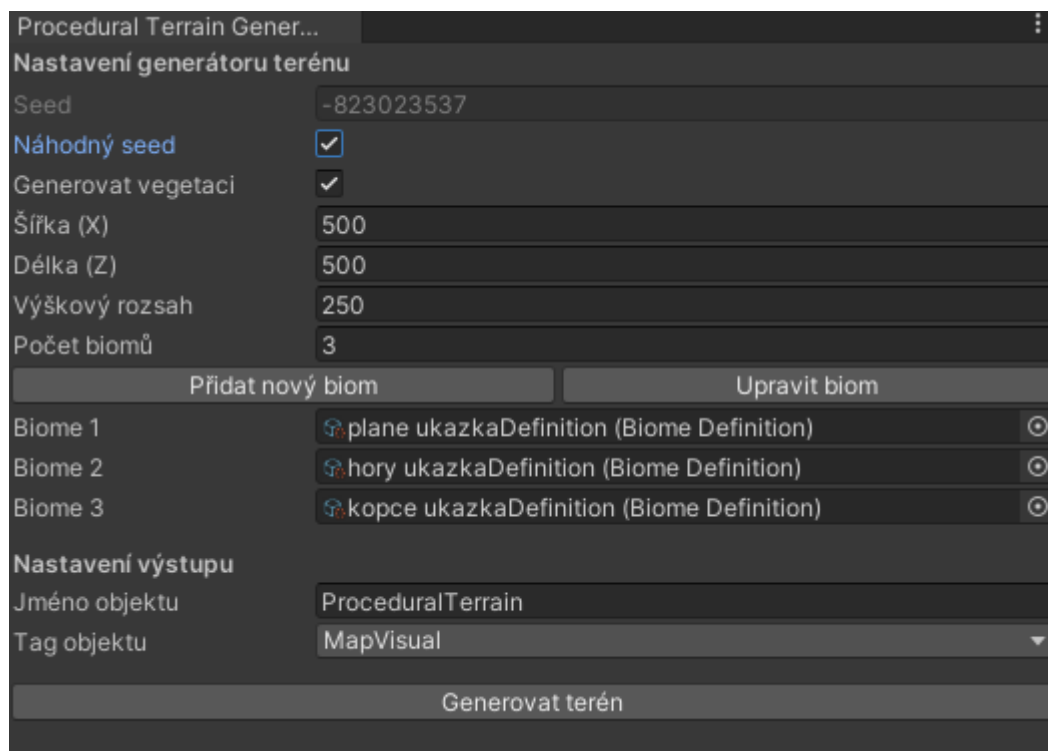
7 POPIS NÁSTROJE

Cílem této práce bylo vytvoření nástroje pro řízené procedurální generování herního světa. Aplikační část byla navržena tak, aby byla schopna uživateli nabídnout versatilní nástroj, s co nejvyšší možností ovlivnit procedurální generaci herního světa s uživatelsky přívětivým prostředím.

7.1 Hlavní nastavení nástroje

Design hlavního ovládacího panelu nástroje je navržen tak, aby byl uživateli přehledný a intuitivní. Okno Nastavení generátoru terénu má k dispozici tyto základní ovládací prvky:

- pole seed – obsah nastaví primární seed, dle kterého se řídí algoritmy generace. Přijímá jakékoli znaky jako vstupní hodnoty. Stejný seed bude generovat pokaždé stejný výsledek,
- zaškrťovací pole Náhodný seed – uživatel má možnost nechat generovat náhodný seed. Tím se zaručí originální výsledek v každé jedné generaci,
- zaškrťovací pole Generovat vegetaci – zpřístupní jednotlivým biomům zahájit generování vegetace,
- pole Šířka a Délka – udává velikost výsledné mapy. Terén ani mesh není škálován, takže velikost bude odpovídat jednotkám herního světa. Jako vstupní hodnoty přijímá výhradně celá čísla,
- pole Výškový rozsah – definuje maximální výšku terénu. Výškový rozsah je v rozmezí od 0 po hodnotu nastavenou v poli. Přijímá pouze celá čísla,
- pole Počet biomů – vymezí počet biomů do kterého bude mapa rozdělena. Minimální počet biomů pro spuštění generace je 1. Přijímá výhradně celočíselné hodnoty,
- tlačítko Přidat nový biom – otevře okno s nabídkou přidání nového biomu do databáze biomů a automaticky ho vloží do pole vybraných biomů,
- tlačítko Upravit biom – otevře okno pro úpravu již existujících biomů,
- pole Jméno – pojmenuje výsledný herní objekt,
- rozbalovací seznam Tag – nastaví volitelné označení výsledného herního objektu.



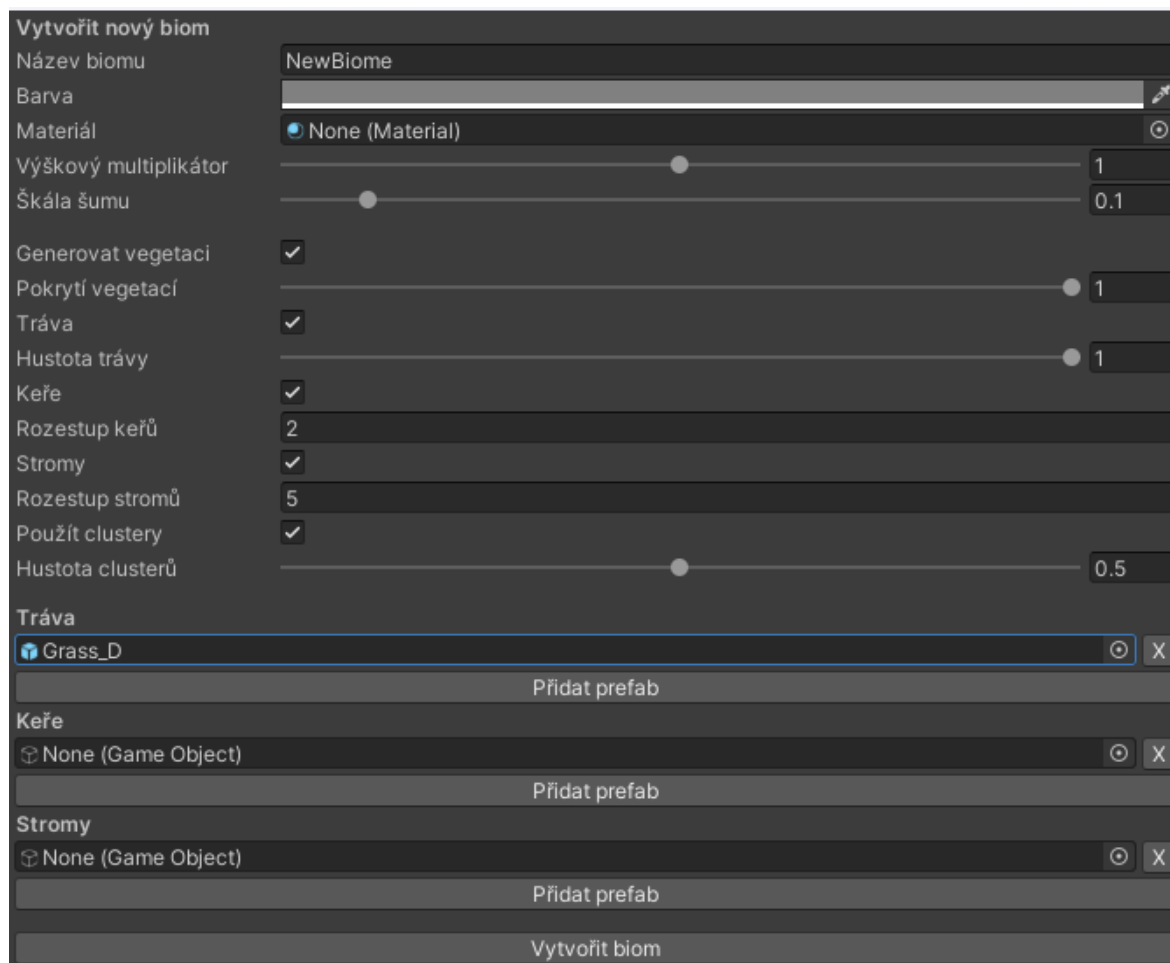
Obrázek 16 Nastavení hlavních parametrů generace

(Zdroj: vlastní)

7.2 Přidání či úprava biomu

Pro efektivní správu biotů, jsou navržena tlačítka, jež dovolí uživateli měnit či vytvářet záznamy o biomech bez nutnosti procházet adresářovou strukturu. Defaultně se všechny záznamy o biomech generují ve složce Assets/Biotmes/\$jméno_biomu\$.

- pole Název biotu – určí jméno celého biotu i odpovídající složky tak, aby se v adresářové struktuře dala snadno najít. Musí být vždy vyplněno,
- výběr barvy Barva – nastaví reprezentaci barvy biotu při generaci meshe,
- výběr assetu Materiál – nastavení biotu materiál,
- posuvník Výškový multiplikátor – určuje míru následné výškové úpravy po dokončení generace. Pohybuje se v rozsahu hodnot 0 až 2, kdy nula potlačí jakékoli výškové změny a 2 je naopak zdvojnásobí,
- posuvník Škála šumu – rozhoduje o velikosti terénních detailů, může nabývat hodnot 0 až 1, kdy šum s hodnotou 0 vytváří uniformní plochy, skoro neměnné, na druhé straně 1 vytváří velmi detailní až chaotické prvky,
- zaškrtnuté pole Generovat Vegetaci – rozhoduje o generaci vegetace v biotu.



Obrázek 17 Možná nastavení při vytváření biomu

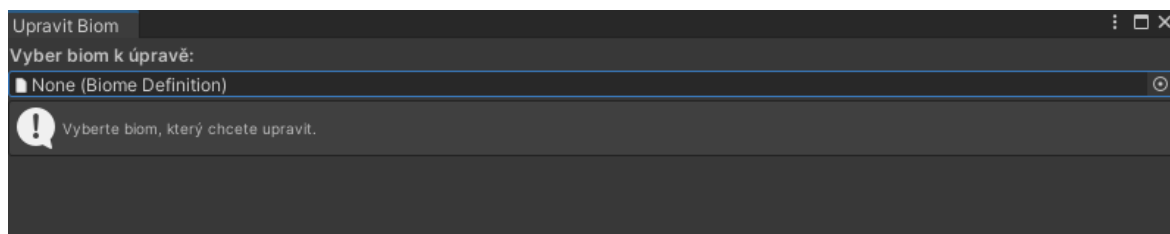
(Zdroj: vlastní)

Po zaškrtnutí tlačítka povolení generace vegetace se ukáže rozšíření okna s následujícími možnostmi:

- posuvník Pokrytí vegetací – nastavuje měřítko procentuálního pokrytí biomu vegetací v rozmezí hodnot od 0 po 1. Pokud bude vybrána hodnota větší než 0.95, biom bude vegetací souvisle pokryt, při hodnotě pod 0.95 se mohou generovat shluky simulující přírodní útvary, jako lesy nebo houští,
- zaškrťovací pole Tráva – rozhoduje, zda se budou nebo nebudou generovat travní porosty v biomu,
- posuvník Hustota trávy – v hodnotách od 0 po 1 určuje hustotu travního porostu v celém biomu. Tráva netvoří clustery,
- zaškrťovací pole Keře – rozhoduje, zda se budou nebo nebudou generovat keře v biomu,

- pole Rozestup keřů – vymezuje minimální rozestupy mezi keři v biomu. Nabývá pouze celočíselných nezáporných hodnot,
- zaškrťovací pole Stromy – rozhoduje, zda se budou nebo nebudou generovat stromy v biomu,
- pole Rozestup stromů – vymezuje minimální rozestupy mezi stromy v biomu. Nabývá pouze celočíselných nezáporných hodnot,
- zaškrťovací pole Použít clustery – rozhoduje, zda bude vegetace v biomu rozdělena do clusterů či nikoli,
- posuvník Hustota clusterů – nabývá hodnot v rozmezí 0 až 1. Se snižující se hodnotou klesá počet clusterů vygenerovaných na mapě, při hodnotě nula vznikne pouze jeden cluster,
- tlačítko Přidat prefab – vytvoří záznam v seznamu pro daný typ vegetace. K vytvořenému záznamu lze přiřadit prefab z dostupných prefabů v projektu, nebo ho smazat,
- tlačítko Vytvořit biom – uloží nově nastavené hodnoty, vytvoří potřebné složky biomu a uloží jeho záznam.

Úprava biomů probíhá obdobným způsobem jako jejich vytváření. Uživatel má možnost vybrat konkrétní biom, který chce editovat, a následně upravit veškeré parametry potřebné pro další generaci terénu. Po provedení požadovaných změn lze tyto úpravy uložit, případně zvolený biom zcela odstranit. Pokud si uživatel nepřeje provedené změny zachovat, může editor jednoduše zavřít, nebo přejít k úpravě jiného biomu, čímž dojde k jejich zrušení.

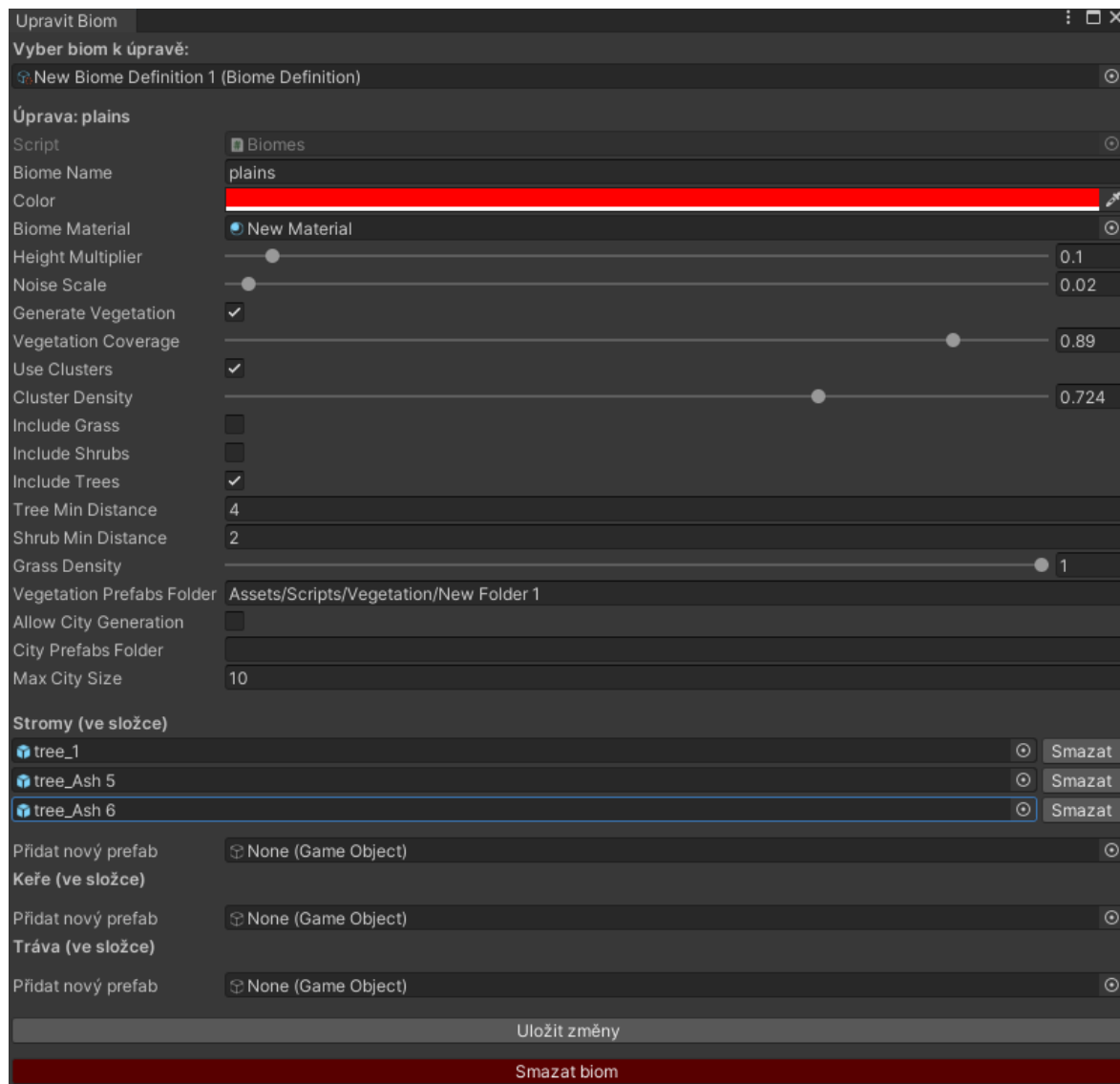


Obrázek 18 Úvodní okno úpravy biomů před vybráním biomu pro editaci

(Zdroj: vlastní)

Při úpravě biomu má uživatel k dispozici opět možnost přidat či odebrat prefab vegetace ze složky, avšak po výběru nového typu vegetace je nezbytné potvrdit přidání

prefabu do složky. V opačném případě by systém neobstaral správné uložení a pojmenování prefabu do adekvátní složky v adresářové struktuře. Tento přístup zachovává jednoduchost exportu biomů v případě, že by je chtěl uživatel dále sdílet.



Obrázek 19 Editační okno vybraného biomu

(Zdroj: vlastní)

7.3 Uživatel

Nástroj jako takový obsahuje základní prefaby vegetace pro použití ve složce Assets/MapPrefabs/Vegetation. Pro zachování konvence a správného fungování nástroje, je kladen důraz na ukládání veškerých budoucích uživatelsky přidaných prefabů do složky, nebo jejich podsložek.

Předností minimálních uživatelských omezení je dosaženo maximální míry variability a efektu uživatelských vstupů v nástroji, tedy uživatel sám do maximální míry ovlivňuje vygenerovaný produkt. Pokud uživatel není s výsledkem spokojen, může generaci opakovat, či vlastnoručně provést dodatečné změny.

7.4 Ukázka použití nástroje

V úvodní fázi uživatel definuje základní představu o podobě výsledné mapy a jejím obsahu. Tato představa slouží jako vodítko při nastavování parametrů v nástroji pro řízenou generaci herního světa.

Samotný nástroj je spouštěn prostřednictvím vývojového prostředí Unity s využitím rozšíření UTW SDK. V hlavní nabídce aplikace (horní lišta) uživatel zvolí položku UTW a následně klikne na možnost Create Project. V nově otevřeném okně je třeba vybrat typ projektu – v tomto případě MAP.

V následujícím kroku je nutné vytvořit a zvolit cílovou složku, do které bude projekt uložen. Po jejím výběru pomocí tlačítka Select Folder dojde k vytvoření projektu a načtení hlavního rozhraní Unity.

Po úspěšném načtení se zobrazí standardní pracovní prostředí Unity, doplněné o specifické panely nástroje UTW. Mezi uživatelsky preferované panely patří:

- Scéna – náhled aktuálního stavu herního světa,
- Nastavení generátoru terénu – slouží k definici parametrů generace,
- Hierarchie scény – zobrazuje strukturu všech objektů ve scéně,
- Náhled objektu – vizualizace vybraných prvků,
- Adresářová struktura a projektový panel – pro práci s daty a soubory projektu.

Klíčové parametry generace terénu se nastavují právě v panelu generátoru terénu, který představuje hlavní nástroj pro ovlivnění výsledné podoby virtuální krajiny.

Procedural Terrain Gener...

Nastavení generátoru terénu

Seed: -823023537

Náhodný seed: ☒

Generovat vegetaci: ☒

Šířka (X): 500

Délka (Z): 500

Výškový rozsah: 250

Počet biomů: 3

Přidat nový biom Upravit biom

Biome 1: plane ukazkaDefinition (Biome Definition)

Biome 2: hory ukazkaDefinition (Biome Definition)

Biome 3: kopce ukazkaDefinition (Biome Definition)

Nastavení výstupu

Jméno objektu: ProceduralTerrain

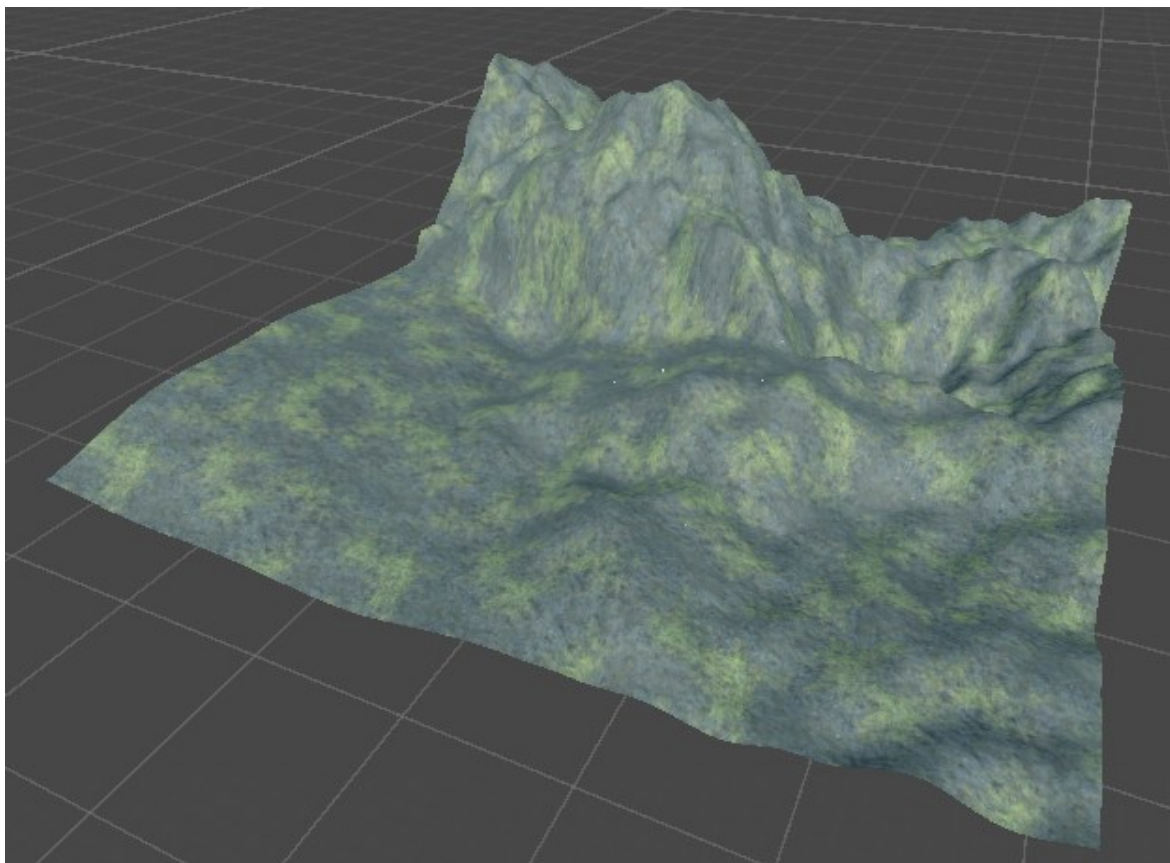
Tag objektu: MapVisual

Generovat terén

Obrázek 20 Hodnoty nastavené pro ukázkovou simulaci

(Zdroj: vlastní)

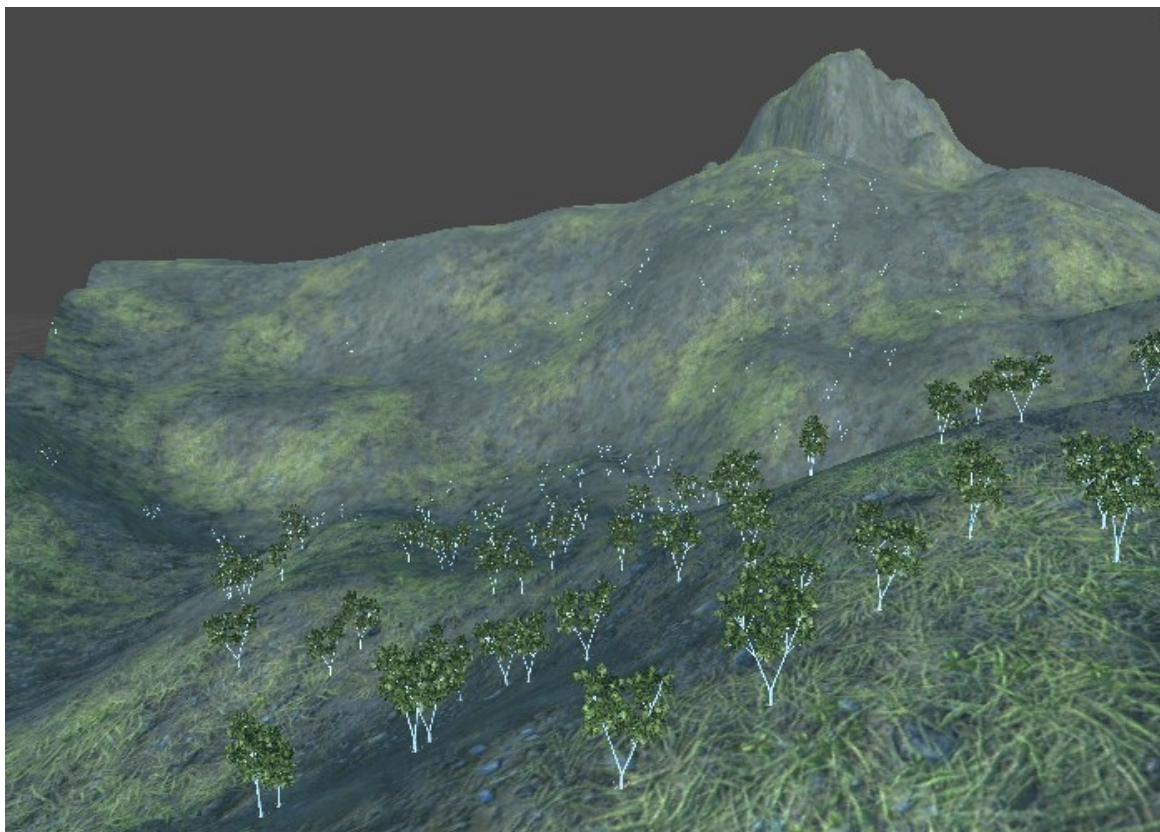
V uvedeném příkladu je zvolen náhodně generovaný seed, dále je umožněno náhodné generování vegetace, šířka a délka mapy bude 500 jednotek, maximální výška mapy 250 jednotek. Jsou vybrány tři předem připravené biomy. Biom „plane ukazka“ svým nastavením simuluje rozložení přirozených planinných oblastí, „hory ukazka“ a „kopce ukazka“ tvoří hornatější útvary. Jméno a tag byly vyplněny pro lepší navigaci v Hierarchii scény.



Obrázek 21 Výsledek ukázkové generace

(Zdroj vlastní)

Z výsledků je patrné, že se terén úspěšně rozdělil do tří odlišných zón, přičemž každá z nich generovala vlastní výškovou mapu. Za účelem lepší vizualizace generování vegetace byla u biomu typu „kopce ukazka“ upravena hodnota parametru pokrytí vegetací na 0,6 a zároveň bylo aktivováno seskupování (clustering). Výsledný stav je znázorněn na obrázku 29.



Obrázek 22 Clustering a pokrytí mapy vegetací v praxi

(Zdroj: vlastní)

Po provedení této úpravy je terén vizuálně realistický a připravený k exportu. V panelu Open Project lze uložit vygenerovaný terén společně s jeho nastavením. Pomocí tlačítka Generate AssetBundles se následně vytvoří soubor typu AssetBundle určený pro export mapy.

ZÁVĚR

Cílem této bakalářské práce bylo navrhnout a implementovat nástroj pro řízené procedurální generování herního světa, s důrazem na jeho využití v simulátoru UTW.

V teoretické části byly analyzovány dostupné algoritmy procedurálního generování a rozděleny podle jejich nejvhodnějšího zaměření. Pro praktickou část byly porovnány možnosti využití těchto algoritmů, se zaměřením na konkrétní oblast, tedy implementace terénu a vegetace.

Jako nejefektivnější se při testování ukázalo rozdělení mapy do jednotlivých biomů, které umožnilo jednoduché řízení generace a zároveň nabídlo uživatelsky přívětivé rozhraní. Tento přístup se rovněž osvědčil při rozšiřování o další moduly. Významnou součástí implementace byla správná koordinace a komunikace mezi jednotlivými moduly generace, což vyžadovalo značné množství času, iterací a optimalizací. Přesto se podařilo dosáhnout plně funkčního systému, který generuje realistické výstupy a je připraven k dalšímu rozšiřování.

Tvorba vlastního kódu pro implementaci algoritmů by byla neefektivní a časově náročná, proto bylo využito specializovaných knihoven FastNoiseLite a PoissonDiskSampling, které jsou již optimalizované a ověřené praxí. Tento krok výrazně přispěl k urychlení vývoje v jeho pozdějších fázích.

Za jednu z významných výhod prostředí Unity lze označit jeho uživatelskou přívětivost a flexibilitu při tvorbě vlastních editorových panelů. Tato vlastnost se osvědčila například při návrhu a úpravě biomů, jelikož mimo jiné umožňuje vytvářet přizpůsobené rozhraní podle potřeb projektu.

V této fázi vývoje nástroj dosáhl stanoveného cíle, schopnosti generovat realistické krajiny. Rychlý vývoj současného videoherního průmyslu, posouvá standardy v oblasti vizuální věrohodnosti. S ohledem na tuto skutečnost by bylo možné dále rozšířit projekt také implementací procedurální generace měst a cest, čímž by se zvýšila celková úroveň realismu generovaného prostředí.

Takové rozšíření by překročilo zadaný rozsah této práce, nicméně zůstává vhodným podnětem pro navazující diplomovou práci, kde by mohlo dojít k integraci dalších systémů generace s ohledem na urbanistické principy, herní logiku a interaktivitu uživatele. Budoucí

vývoj tak může rozšířit současný nástroj na komplexní generátor plně funkčního herního světa.

SEZNAM POUŽITÉ LITERATURY

- [1] ADAMS, Tarn a SHORT, Tanya X. (ed.). *Procedural generation in game design*. Boca raton: CRC Press, [2017]. ISBN 978-1-4987-9919-5.
- [2] SHAKER, Noor; TOGELIUS, Julian; NELSON, Mark J.; SHAKER, Noor; TOGELIUS, Julian et al. Fractals, noise and agents with applications to landscapes. Online. In: *Procedural Content Generation in Games*. Computational Synthesis and Creative Systems. Cham: Springer International Publishing, 2016, s. 57-72. ISBN 978-3-319-42714-0. Dostupné z: https://doi.org/10.1007/978-3-319-42716-4_4. [cit. 2025-02-24].
- [3] SILVA, Rafael C. E; FACHADA, Nuno; DE ANDRADE, Diogo a CODICES, Nelio. Procedural Generation of 3D Maps With Snappable Meshes. Online. *IEEE Access*. 2022, roč. 10, s. 43093-43111. ISSN 2169-3536. Dostupné z: <https://doi.org/10.1109/ACCESS.2022.3168832>. [cit. 2025-05-25].
- [4] FOLEY, James D. *Computer graphics: principles and practice*. 2nd ed. Boston: Addison-Wesley, 1996. ISBN 02-018-4840-6
- [5] SHIFFMAN, Daniel. *The nature of code: simulating natural systems with processing*. [Mountain View]: Daniel Shiffman, [2012]. ISBN 978-0-9859308-0-6.
- [6] LENGYEL, Eric. *Mathematics for 3D game programming and computer graphics*. Third edition. Boston, MA: Course Technology, Cengage Learning, [2012]. ISBN 978-1-4354-5886-4.
- [7] BARVÍŘ, Marek. SADA PROCEDURÁLNĚ GENEROVANÝCH EFEKTŮ. Online, Bakalářská práce. Brno: Vysoké učení technické v Brně, 2018. Dostupné také z: <https://dspace.vut.cz/items/e1305b1b-26d2-42ae-861c-28222434c5e2/full>.
- [8] Hendrikx, M., Meijer, S., Van Der Velden, J., & Iosup, A. (2013). Procedural content generation for games. *ACM Transactions on Multimedia Computing Communications and Applications*, 9(1), 1–22. <https://doi.org/10.1145/2422956.2422957>
- [9] LAW, Goerge. Quantitative Comparison of Flood Fill and Modified Flood Fill Algorithms. Online. In: *International Journal of Computer Theory and Engineering*. Vol. 5. 2013, s. 6. Dostupné z: <https://www.ijcte.org/papers/738-T012.pdf>. [cit. 2025-05-25].

- [10] COOK, Robert L. a Tony DEROSE. Hierarchical Poisson disk sampling distributions. In: Graphics Interface '92: Proceedings of the 18th Annual Conference on Computer Graphics Interface. Toronto: Canadian Human-Computer Communications Society, 1992, s. 213–220.
- [11] RUNJE, Tomasz, Adam RUNJE, and Bedrich BENES. *Modeling Trees with a Space Colonization Algorithm*. In: Eurographics Workshop on Natural Phenomena (EGWNP). 2007. Dostupné z: <https://algorithmicbotany.org/papers/colonization.egwnp2007.large.pdf>
- [12] KUTÁLEK, Jan. *Procedural Generation of Videogame Environments*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická, 2021. Dostupné také z: <https://dcgi.fel.cvut.cz/wp-content/wpallimport-dist/theses/pdf/theses-2021-kutalja1-thesis.pdf>.
- [13] ILACHINSKI, Andrew. *Cellular automata: a discrete universe*. Singapore: World Scientific, 2001. ISBN 98-123-8183-X.
- [14] EBERT, D.S., F.K. MUSGRAVE, D. PEACHEY, K. PERLIN a S. WORLEY. Texturing & Modeling: A Procedural Approach. In: DRETTAKIS, G. a R. SCOPIGNO, eds. Eurographics 2010 – State of the Art Reports. Berlin: Springer, 2010. ISBN 978-3-642-14034-1. DOI: 10.1007/978-3-642-14034-1_1
- [15] FRODL, Vojtěch. *Generování územních plánů vesnic do herního prostředí*. Online, Diplomová práce. Brno: Masarykova univerzita Fakulta informatiky, 2018. Dostupné také z: <https://is.muni.cz/th/cvc2j/generovani-uzemnich-planu-digital.pdf>.
- [16] JORMEDAL, Martin. *Procedural generation of road networks using L-systems*. Online, Bakalářská práce. Linköping: Univerzita Linköping, 2013. Dostupné také z: <https://liu.diva-portal.org/smash/get/diva2:1467574/FULLTEXT01.pdf>.
- [17] NYSTROM, Robert. *Game programming patterns*. [London]: Genever Benning, [2014]. ISBN 978-0-9905829-0-8.

SEZNAM OBRÁZKŮ

Obrázek 1 Postup diamond square algoritmu krok po kroku	17
Obrázek 2 Obecné charakteristiky modelu založeného na agentech	18
Obrázek 3 Whittakerův diagram rozložení zón na základě teploty a srážek	20
Obrázek 4 Flood fill vyplňování do čtyř stran	22
Obrázek 5 k-means clustering rozděluje data do Voronoiových buněk	24
Obrázek 6 Gaussovo modelování shluků.....	25
Obrázek 7 a) Obálka s přitažlivými body a počátečním uzlem stromu; b, c) Generování kostry stromu.	28
Obrázek 8 d) Redukce uzlů; e) Přemístění uzlů, úprava úhlů větvení.....	29
Obrázek 9 f) Křivkové dělení; g) Konstrukce zobecněných válců; h) Doplnění detailů.....	29
Obrázek 10 Okolí buňky CA – von Neumannovo okolí.	31
Obrázek 11 Okolí buňky – Moorovo okolí.....	31
Obrázek 12 Vzorčky k vyplnění mřížky.....	33
Obrázek 13 Princip algoritmu WFC	33
Obrázek 14 Hledání optimální cesty za pomoci RRT	36
Obrázek 15 ukázka stromu Ash 8 z balíčku European Forests – Realistic Trees ve scéně .	39
Obrázek 16 Nastavení hlavních parametrů generace.....	53
Obrázek 17 Možná nastavení při vytváření biomu	54
Obrázek 18 Úvodní okno úpravy biomů před vybráním biomu pro editaci	55
Obrázek 19 Editační okno vybraného biomu	56
Obrázek 20 Hodnoty nastavené pro ukázkovou simulaci.....	58
Obrázek 21 Výsledek ukázkové generace	59
Obrázek 22 Clustering a pokrytí mapy vegetací v praxi	60

SEZNAM TABULEK

Tabulka 1 Srovnání algoritmů pro generování terénu	41
Tabulka 2 Srovnání algoritmů pro generování biomů	42
Tabulka 3 Srovnání algoritmů pro generování vegetace	43

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

NPC Non playable character, nehráčská postava hry

GPS Global Positioning Systém, systém pro zjištění přesných souřadnic ve světě

PCG Procedural content in game, procedurálně generovaný herní obsah

AI Artificial intelligence, umělá inteligence, ve spojení s herním světem se většinou jedná o NPC postavy, které mají předem nastavené chování

STAG Informační systém UTB

CGI Computer generated imagery, počítačově generovaný obraz

MIT Massachusetts Institute of Technology, Massachusettský technologický institut

SEZNAM PŘÍLOH

Příloha P I: Unity projekt ve formátu .zip

PŘÍLOHA P I: UNITY PROJEKT VE FORMÁTU .ZIP

Zkompilovaná verze aplikace UTW SDK zabalena do zip archivu. Pro spuštění je potřeba rozbalit zip soubor do libovolného adresáře a poté ho spustit skrze Unity Hub. Obsahuje:

- Unity package programu UTW-SDK.unitypackage
- Adresář zdrojových souborů programu UTW-SDK_source