

# Vytvoření nástroje pro generování ostrova v engine Unity

Mikuláš Mikeska

---

Bakalářská práce  
2023



Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

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

Akademický rok: 2022/2023

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Mikuláš Mikeska  
Osobní číslo: A20095  
Studijní program: B0613A140020 Softwarové inženýrství  
Forma studia: Prezenční  
Téma práce: Vytvoření nástroje pro generování ostrova v engine Unity  
Téma práce anglicky: Creating Island Generation Tool in Unity Engine

## Zásady pro vypracování

1. Vypracujte literární rešerši na zadané téma.
2. Provedte stručnou analýzu podobných nástrojů pro generování prostředí.
3. Navrhněte nástroj pro generování ostrova v engine Unity.
4. Implementujte navržený nástroj tak, aby byl jednoduše přenositelný do kteréhokoliv 3D projektu pro Unity.
5. Demonstrujte funkčnost daného nástroje.
6. Vhodně zdokumentujte vytvořený nástroj.

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

**Seznam doporučené literatury:**

1. KOK, Benny. Beginning Unity Editor Scripting: Create and Publish Your Game Tools. New York, USA: Apress, 2021. ISBN 978-1-4842-7166-7.
2. TADRES, Angelo. Extending Unity with Editor Scripting. Birmingham, UK: Packt, 2015. ISBN 9781785281853.
3. Unity User Manual 2021.3 (LTS) [online]. San Francisco, USA: Unity Technologies, 2022 [cit. 2022-11-28]. Dostupné z: <https://docs.unity3d.com/Manual/index.html>.
4. WALKER, Philip. Unity Certified Programmer: Exam Guide. Birmingham, UK: Packt, 2020. ISBN 9781838828424.
5. LIGHTBOWN, David. Designing the User Experience of Game Development Tools. Boca Raton, USA: CRC Press, 2015. ISBN 9781482240191.

Vedoucí bakalářské práce:

**Ing. Tomáš Vogeltanz, Ph.D.**

Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **2. prosince 2022**

Termín odevzdání bakalářské práce: **26. května 2023**



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

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

Ve Zlíně dne 7. prosince 2022

### **Prohlašuji, že**

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že 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;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že 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 samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

Mikuláš Mikeska, v. r.  
podpis studenta

## **ABSTRAKT**

Tato práce je zaměřena na tvorbu nástroje pro generování ostrovů v enginu Unity. Nejprve jsou představeny zástupci některých enginů, následně popis algoritmů, které jsou použity pro generování. Poté jedna z možností, jak vytvořit editor v okně Inspectoru. Výsledkem práce je nástroj umožňující uživatelům rychle a snadno vytvářet ostrovy pro jejich hry v enginu Unity. Popis postupu vytvoření tohoto nástroje je uveden v praktické části. Generování bude založeno na šumu a bude možné zobrazit části ostrovu jen okolo pozice hráče.

Klíčová slova: Unity, generování ostrovů, šum, zobrazení pouze viditelných částí

## **ABSTRACT**

This thesis is focused on creating a tool for generating islands in the Unity engine. First, representatives of some of the engines are introduced, followed by a description of the algorithms that are used for generation. Then one of the options to create an editor in the Inspector window. The result of the work is a tool that allows users to quickly and easily create islands for their games in the Unity engine. A description of the process of creating this tool is given in the practical section. The generation will be noise-based and will only display parts of the island around the player's position.

Keywords: Unity, island generation, noise, display only visible parts

Chtěl bych poděkovat především Ing. Tomáši Vogeltanzovi, Ph.D. za ochotu a úsilí, které vynaložil během konzultací této bakalářské práce.

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

# OBSAH

<b>ÚVOD.....</b>	<b>10</b>
<b>I TEORETICKÁ ČÁST .....</b>	<b>11</b>
<b>1 GAME ENGINE.....</b>	<b>12</b>
1.1 NEJPOUŽÍVANĚJŠÍ HERNÍ ENGINY .....	12
1.2 UNITY .....	13
1.3 CONSTRUCT .....	14
1.4 GAMESMAKER: STUDIO.....	14
1.5 TWINE .....	15
1.6 UNREAL ENGINE .....	15
<b>2 ALGORITMY PRO GENEROVÁNÍ TERÉNU.....</b>	<b>17</b>
2.1 PERLIN NOISE.....	17
2.2 FALLOFF MAP .....	18
<b>3 PŘÍZPŮSOBENÍ EDITORU .....</b>	<b>19</b>
3.1 ATRIBUTY S MODIFIKÁTORY PŘÍSTUPY .....	19
3.1.1 Modifikátor přístupu .....	19
3.1.2 Atributy .....	20
3.2 PŘÍZPŮSOBENÍ POMOCÍ C# .....	23
3.2.1 Základní nastavení .....	23
3.2.2 Přidání tlačítka .....	24
3.2.3 Zjištění změny hodnoty v Inspectoru .....	25
<b>II PRAKTICKÁ ČÁST .....</b>	<b>26</b>
<b>4 ANALÝZA PODOBNÝCH NÁSTROJŮ.....</b>	<b>27</b>
4.1 TERRAIN GENERATOR .....	27
4.1.1 Základní charakteristika .....	27
4.1.2 Výhody .....	27
4.1.3 Nevýhody .....	27
4.2 MAPMAGIC 2 .....	28
4.2.1 Základní charakteristika .....	28
4.2.2 Výhody .....	28
4.2.3 Nevýhody .....	28
4.3 TERRAINIFY 2D.....	29
4.3.1 Základní charakteristika .....	29
4.3.2 Výhody .....	29
4.3.3 Nevýhody .....	30
<b>5 VYTVOŘENÍ NÁSTROJE .....</b>	<b>31</b>
5.1 POŽADAVKY NA NÁSTROJ.....	31

5.2	PERLIN NOISE .....	31
5.3	TERÉN .....	33
5.4	GENERACE MESHE .....	33
5.4.1	Generace čtverce .....	34
5.4.2	Generace spojitého meshe .....	35
5.5	GENERACE GRAFIKY NA TERÉN .....	37
5.6	OVLÁDÁNÍ DNE A NOCI.....	44
5.7	GENERACE OBJEKTŮ.....	44
<b>6</b>	<b>POPSÁNÍ NÁSTROJE .....</b>	<b>47</b>
6.1	VISIBLE CHUNK .....	47
6.1.1	Visible All Terrains .....	47
6.1.2	Visible All Objects .....	47
6.1.3	Target Transform .....	47
6.1.4	Object Count Visible Chunk .....	48
6.1.5	Recalculate object .....	48
6.1.6	Terrain Count Visible Chunk .....	48
6.1.7	Generate with color .....	48
6.1.8	Recalculate terrain.....	48
6.2	GENERATE MESH .....	48
6.2.1	Chunk Size .....	49
6.2.2	Random .....	49
6.2.3	Flat Island.....	49
6.2.4	Deep .....	49
6.2.5	Size Map X.....	49
6.2.6	Size Map Z .....	49
6.2.7	Scale .....	50
6.2.8	Use Falloff Map .....	50
6.2.9	Size Falloff Map.....	50
6.2.10	Generate .....	50
6.3	GENERATE COLOR .....	50
6.3.1	Auto Update .....	50
6.3.2	Color Option.....	51
6.3.3	Specification.....	51
6.3.4	Smooth Color .....	52
6.3.5	Material .....	52
6.3.6	Generate .....	52
6.4	GENERATE OBJECT.....	52
6.4.1	Specification.....	53
6.4.2	Generate .....	54
6.5	DAY AND NIGHT .....	54
6.5.1	Timer .....	54
6.5.2	Offset Sun.....	54
6.5.3	Player Transform.....	54



6.5.4	Speed .....	54
6.5.5	Animation Day And Night .....	54
6.5.6	Direction Light .....	55
6.5.7	Generate .....	55
6.6	SAVE AND LOAD .....	55
6.6.1	Path To Txt File .....	55
6.6.2	Save .....	55
6.6.3	Load.....	55
<b>7</b>	<b>TESTOVÁNÍ .....</b>	<b>56</b>
7.1	GENERACE MESHE .....	56
7.2	GENERACE GRAFIKY.....	59
<b>8</b>	<b>PUBLIKACE .....</b>	<b>61</b>
8.1	VYTVOŘENÍ UNITYPACKAGE .....	61
8.2	IMPORTOVÁNÍ UNITYPACKAGE DO PROJEKTU .....	61
	<b>ZÁVĚR .....</b>	<b>64</b>
	<b>SEZNAM POUŽITÉ LITERATURY .....</b>	<b>65</b>
	<b>SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK .....</b>	<b>68</b>
	<b>SEZNAM OBRÁZKŮ .....</b>	<b>69</b>
	<b>SEZNAM PŘÍLOH.....</b>	<b>71</b>

## ÚVOD

Tato práce se zaměřuje na tvorbu nástroje pro generování ostrova v Game engineu Unity. V teoretické části práce jsou popsány herní enginey, včetně Unity, Construct, GameMaker: Studio, Godot, Twine a Unreal Engine. Dále jsou zde popsány algoritmy pro generování terénu. Konkrétně jsou popsány algoritmy Perlin noise a falloff map. Dále je popsáno přizpůsobení editoru pomocí atributů s modifikátory přístupu, které jsou v jazyce C#.

V praktické části je nejdřív analýza tří podobných nástrojů. Následně je popsána tvorba vlastního nástroje pro generování terénu, a to včetně ukázky zdrojových kódů. Tento nástroj umožňuje generování meshe (terén), následné generování grafiky, která je ve třech možnostech, generování objektů na terén a ovládání osvětlení, která má simulovat den a noc. Práce dále obsahuje popis jednotlivých parametrů ze skriptů, které může uživatel měnit v okně Inspectoru, jsou taky popsány, jaký mají vliv na výsledné generování. Následuje testování vytvořeného nástroje, včetně ukázek obrázků výsledných terénů.

Poslední část je zaměřená na publikaci vytvořeného nástroje. To znamená proces vytvoření Unity package. Poté možností importování do projektu.

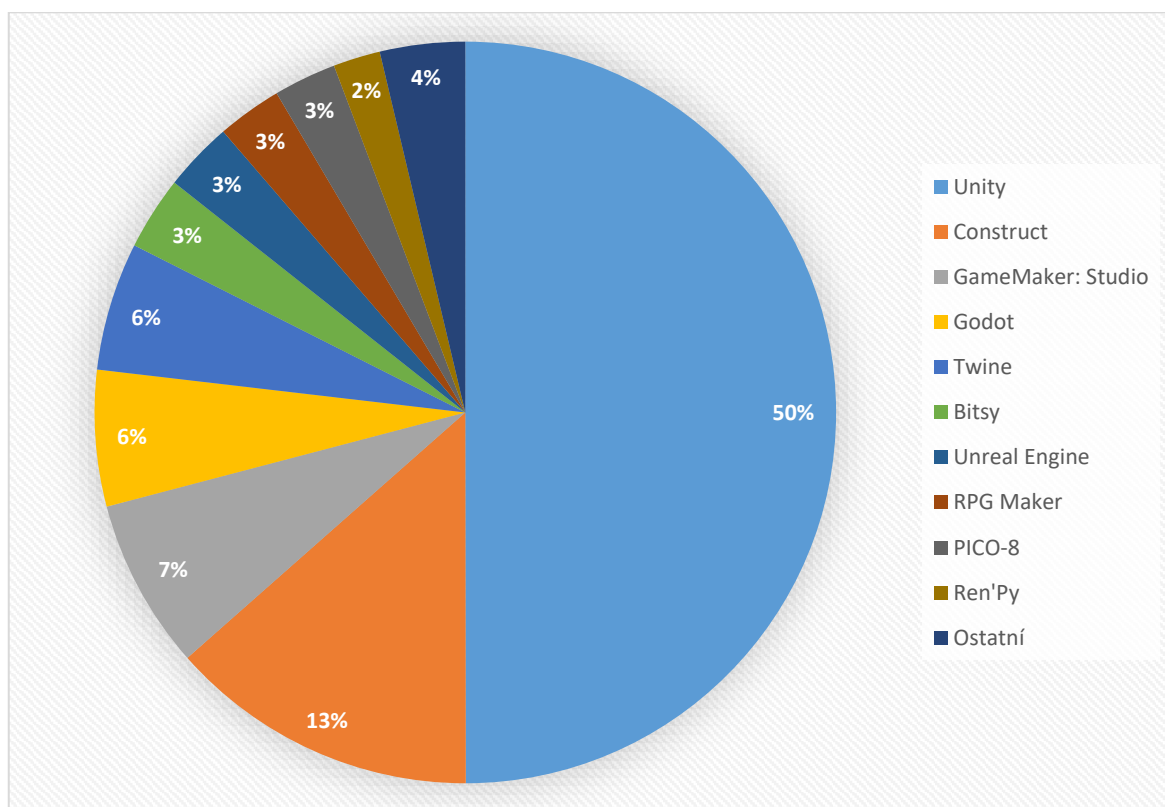
## **I. TEORETICKÁ ČÁST**

## 1 GAME ENGINE

Cílem této kapitoly je popsat pojem Game Engine a následně charakterizovat nejpoužívanější herní enginey na platformě Twitch.io. Dále u zmíněných engineů popsat jedny z neznámějších her, aby bylo možné si představit co umožňují.

### 1.1 Nejpoužívanější herní enginey

Game engine je software, který má za úkol napomáhat vývojářům ulehčit a urychlit vývoj her. Většinou umožňují vytvářet ve 2D i 3D, ale najdou se i takové, které podporují jenom 2D. Mají zabudováno mnoho funkcí a nástroje, jako jsou umělá inteligence, fyzikální zákony, a mnoho dalších. Engine většinou umožňuje portovat na různé platformy, jako jsou počítače, herní konzole nebo mobilní zařízení. [1]



Obrázek 1 – Graf nejvíce používaných herních engineů

Zdroj: vlastní tvorba dle [2]

Na obrázku 1, lze vidět, jaké enginey jsou nejvíce využívány na platformě Twitch.io, která je populární mezi hráči a streamery.

## 1.2 Unity

Společnost Unity Technologies stojí za vývojem tohoto engine. Jeho hlavní přednosti zahrnují kompilaci pro různá zařízení jako jsou počítače, mobily, konzole a web. Engine také podporuje pokročilé funkce jako jsou odrazy světla, odrazy objektů, stíny v reálném čase a globální osvětlení. [3]

Unity je vývojové prostředí, které umožňuje tvorbu her jak ve 2D, tak ve 3D, především skriptů napsaných v jazyce C#. Popřípadě je možné použít vizuální skriptovací jazyk Bolt, který je možný ke stažení v asset storu. Díky svým vlastnostem a bezplatné verzi je velmi oblíbené mezi malými společnostmi a nezávislými vývojáři. Pokud společnost vydělává méně než 100 000 \$ za posledních 12 měsíců, může používat Unity zdarma. V případě, že společnost překročí tuto hranici, existují placené verze, jako je například verze PLUS, která je dostupná pro společnosti, které vydělávají méně než 200 000 \$ ročně. Pokud to ani nestačí, jsou k dispozici verze Pro a Enterprise bez omezení. Díky všem těmto faktorům je tento Engine velmi populární a má velkou komunitu, mnoho tutoriálů a velké množství Assetů – tedy hotových prvků, které lze použít v hře. [3][5]

Nejznámější hry vytvořené za použití Unity [6]:

- **Among Us** je online hrou pro více hráčů z roku 2018. Byla inspirována hrou Mafia a poskytuje hráčům možnost účastnit se společenské hry, kde se snaží odhalit a potrestat zrádce. Díky multiplatformnímu publikování se hra může hrát na různých platformách, jako jsou Microsoft Windows, PlayStation 4/5, Nintendo Switch, Xbox One, Xbox Series X/S, Android, iOS.
- **Cuphead** je hrou z roku 2017 ve 2D s run-and-gun tématem. Díky své jednoduché grafice a hratelnosti si získala popularitu, dokonce i uznání kritiků. Prodalo se jí přes šest milionů kopií, což vedlo k vytvoření seriálu. Hra je také k dispozici na platformách Nintendo Switch, PlayStation 4, Xbox One, Microsoft Windows, macOS.
- **Pokémon Go** je mobilní hra z roku 2016, která využívá rozšířenou realitu. Hra především zahrnuje vyhledávání, chytání, trénink a boj s Pokémony, které vypadají, jako by se nacházely ve skutečném světě hráče. Pokémoni jsou generovány na místa pomocí GPS

souřadnic, proto pro správné fungování je nutné mít přístup k internetu a fotoaparátu.

### 1.3 Construct

Construct umožňuje tvorbu 2D her pro webové prohlížeče, mobilní zařízení, počítače a Xbox One. K tvorbě her využívá vizuální skriptovací jazyk nazvaný Event Sheets, přičemž lze také využít skriptování v jazyce JavaScript, ten je především doporučen pro zkušenější vývojáře. Konkrétně existují dvě verze tohoto software – bezplatná verze, která má omezení na maximální počet 25 akcí ve hře (po přihlášení až 50 akcí), a placená verze je bez omezení, která vyjde na 99 \$ ročně. [7]

Nejznámější hry vytvořené za použití Construct [8]:

- **Star Sky 2** navazuje na předchozí verzi a nabízí podobný styl hraní, ale s více možnostmi, a především více následky. Záleží na tom, kterou volbu hráč zvolí. Hra je ve 2D a dostupná na platformách Android, Wii U, Linux, iOS a Windows.
- **Necrosphere** je 2D hra, která se ovládá pouze pomocí dvou kláves, možná už díky této originalitě si získala svoji oblibu. Hráč se nachází v podsvětí a musí uniknout z tohoto světa zpět do normálního lidí. Hra je k dispozici pro platformy Android, Xbox One, Linux, Mac, iOS a Windows.

### 1.4 GameMaker: Studio

Game Maker Studio (zkráceně GMS) je nástupcem Game Maker (zkráceně GM) a je určen pro tvorbu 2D a 3D her. Je dostupný na Windows a pro MAC. GM byl vytvořen v programovacím jazyce Delphi a byl prvně uveden na trh v listopadu roku 1999. [9]

Tento nástroj si získal oblibu kvůli své jednoduchosti, přátelskému rozhraní a velké komunitě. Vývojář může hru vytvořit pomocí vizuálního programování, tato metoda je doporučena hlavně začínajícím vývojářům. Zato zkušenější uživatelé mohou použít objektový jazyk GML (Game Maker Language). Umožňuje kompilaci pro různá zařízení jako jsou počítače, mobily, konzole a web. [9]

Nejznámější hry vytvořené za použití GMS [10]:

- **Katana Zero** je hra z roku 2019, která se více zaměřuje na herní zážitek než na příběh. Hráč hraje roli zabijáka, který umí předvídat budoucnost a zpomalit čas, kdy jeho úkolem je zničit nepřátele. Hra je dostupná na platformách Windows, Mac, Nintendo Switch, Xbox One a Amazon Luna.
- **Stoneshard** je RPG hra z roku 2020 s velkými ambicemi, protože se jedná o hru v otevřeném světě s pohybem po kliknutí. Má unikátní systém zdraví, který nezahrnuje pouze zásahové body, ale také věci jako léčení ran a zvládání bolesti. Dostupná je na platformách Windows, Mac a Linux. [10][11]

## 1.5 Twine

Twine je open source software, který slouží k tvorbě textových her, pomocí hypertextových odkazů, takže hry jdou portovat pouze pro web. Tyto odkazy vedou k různým akcím a scénám, díky tomu hráč může ovlivnit vývoj příběhu. Je doporučeno dát 2 odkazy na každou stránku. Tento nástroj neumožňuje animace ani žádné speciální efekty. [15]

Většina her má málo stažení, proto nelze určit, které jsou nejznámější.

## 1.6 Unreal Engine

Unreal Engine je herní engine vyvinutý společností Epic Games. Tento engine je vhodný pro vývoj náročnějších her. Mezi hlavní výhody patří velký výběr nástrojů ať už pro práci s animací, umělou inteligence nebo fyziky (gravitace, odrazy apod). Umožňuje portování na různé platformy, jako jsou počítače, konzole a mobilní zařízení. Vývojář ovládá vlastnosti nástrojů především pomocí skriptů napsaných v C++, ale také podporuje vizuální programování pomocí skriptů Blueprint. Tyto vlastnosti a specifika zařazují tento engine mezi největší konkurenty Unity. [16] [17] [18]

Nejznámější hry vytvořené za použití Unreal Engine [19]:

- **Fortnite** je 3D multiplayerová hra, která byla vydána v roce 2017. Získala již 25 herních ocenění, nejčastěji Hra roku a Nejlepší hra pro více

hráčů. Má registrováno přibližně 250 milionů hráčů, nejčastěji do 20 let.

- Shenmue 3 je hra, na kterou fanoušci série čekali téměř dvacet let. I přesto byla kritizována za to, že se dostatečně neliší od svého předchůdce. Ve hře se opět ovládá hlavní hrdina Ryo Hazuki, který se snaží odhalit pravdu o vraždě svého otce. Vývojářská společnost Ys Net zvolila engine Unreal 4, protože jim umožnil rychleji vytvářet rané prototypy.

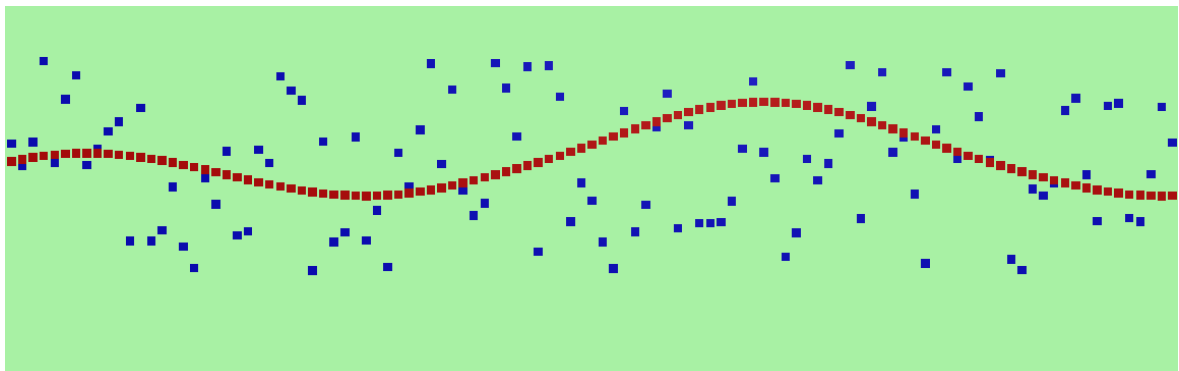


## 2 ALGORITMY PRO GENEROVÁNÍ TERÉNU

Cílem této kapitoly je popsat použité funkce, které jsou použity při generování ostrova. Ze začátku je vysvětlena funkce perling noise a následně falloff mapa.

### 2.1 Perlin Noise

Jde algoritmus vytvořený Kenem Perlinem, který sloužil pro generování počítačem vytvořených efektů ve filmu Tron. Lze ho použít pro generování různých efektů s přirozenými vlastnostmi, jako jsou vítr a mraky. Díky hladšímu průběhu je vhodný nástroj pro generování, kde potřebujeme malé změny průběhu. Tento algoritmus lze najít skoro v každém programovacím jazyce. Zpravidla bere jeden nebo dva argumenty, výjimečně tři. [20]



Obrázek 2 – ukázka perlin noise vs random

Zdroj: vlastní tvorba za pomoci Unity dle [20]

Na obrázku 2 jsou vidět modré a červené čtverce. Na červené čtverce byl použit algoritmus perlin noise, zatímco modré jsou generovány náhodně. Červené čtverce na sebe hezky navazují, zatímco modré působí chaoticky.

Zatímco funkce random umožňuje vrátit číslo v určitém intervalu, funkce noise pracuje jinak. Vyžaduje vstup, který představuje časový okamžik a na základě vstupu vrací hodnotu od 0 do 1. Jednorozměrný šum si tedy můžeme představit jako lineární posloupnost hodnot v čase. [20]

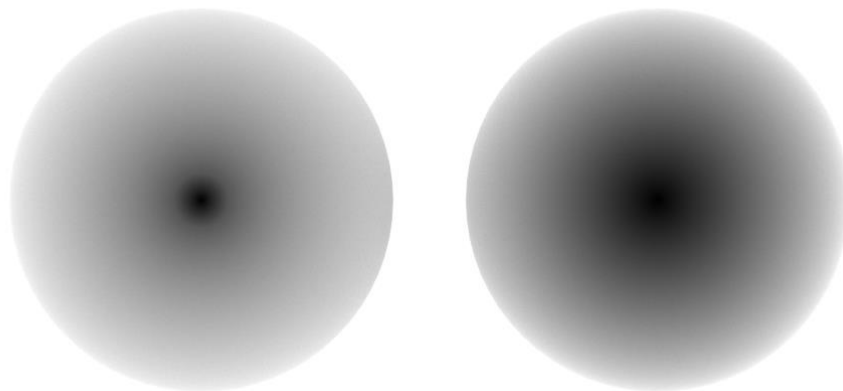
Výhodou generování pomocí šumu je, že místo ručního přípravy všech úrovní (nastavováním velikosti, barvy, natočení atd.) ve fázi návrhu herní úrovně, vývojář stanoví sadu kroků a podmínek. [21]

Jak má vývojář všechny pravidla sepsané, může v podstatě generovat libovolný počet úrovní, stačí změnit několik parametrů pro generování. Nebo jednoduše může udělat nekonečný svět jako například ve hře Minecraft. Protože se mění jen několik čísel pro každou úroveň, není nutné ukládat celý model světa, takže se šetří místo na disku uživatele. [21]

Vytvořit tento algoritmus tak, aby na sebe přirozeně navazovaly biomy a uživatel se ve hře dostal na všechna místa není snadné. Obvykle je nastavení procesu generování složitější než vytvoření modelu mapy. Nejtěžší částí při generování jsou zpravidla velké objekty, jako jsou domy, kostely a další budovy. [21]

## 2.2 Falloff map

Falloff map je speciální typ textury, který se používají převážně v 3D grafice a herních enginů. Je založená na úbytku, ať už s lineárním, popřípadě exponenciálním. Tyto textury se nejčastěji používají k simulaci efektu rozptýlení světla, ale najdou využití kdekoliv, kde je potřeba postupného úbytku. Jsou občas kombinovány s dalšími textury. [22]



Obrázek 3 – ukázka textury falloff mapy

Zdroj: [23]

Na obrázku 3 jsou prezentovány dvě mapy falloff, přičemž mapa na pravé straně má plynulejší přechod než mapa na levé straně. Tato plynulost znamená, že pokud bychom použili tyto mapy v kombinaci s terénem, kde bílá barva reprezentuje hodnotu 1 a černá hodnotu 0, terén by byl s mapou na levé straně ukončen rychleji než s mapou na pravé straně. Jednoduše řečeno, použití mapy s plynulejším přechodem umožní vytvoření méně prudkých změn terénu.

### 3 PŘIZPŮSOBENÍ EDITORU

Cílem této kapitoly je popsat čtenáři jednu z možností, jak vytvořit přizpůsobitelný editor. Je to jedna z možností, jak dát uživatel nastavit hodnoty.

#### 3.1 Atributy s modifikátory přístupu

Tato podkapitola se zaměřuje na atributy s modifikátory přístupu. Modifikátory přístupu mají vliv na viditelnost hodnot v kódu. V unity mají modifikátory a atributy velký vliv na chování skriptu.

##### 3.1.1 Modifikátor přístupu

Modifikátory přístupu mají vliv na viditelnost hodnot v kódu. V unity dokonce mají vliv, jestli jsou proměnné viditelné v okně Inspector, kde se můžou upravovat. Možné modifikátory jsou: [24]:

- **Public** - takový to člen je dostupný bez omezení. V Unity je navíc viditelný v okně Inspectoru, kde lze nastavit nebo měnit jeho hodnotu.
- **Private** - člen je přístupny pouze kódu ve stejné třídě. Je to výchozí modifikátor, nenapíše-li vývojář žádný použije se tento.
- **Protected** - můžeme k němu přistupovat uvnitř vlastní třídy a ve všech třídách, které jsou založené na této třídě (odvozené).
- **Internal** - přístup k členu je omezený pouze na aktuální sestavení.

Existují ještě dvě kombinace, které jsou [24]:

- **Private protected** - člen je přístupný jen uvnitř assembly, kde je deklarován.
- **Protected internal** - člen je přístupny kdekoliv uvnitř assembly, kde je deklarován, ale také v odvozené třídě deklarované v jiné assembly.

### 3.1.2 Atributy

Hodnoty atributů se vkládají do hranatých závorek před člen, na který má být aplikován. Na jeden člen lze použít více atributů, které lze zapsat do jedné závorky oddělené čárkou [PrvníAtribut, DruhýAtribut]. Alternativně může být každý atribut uveden v samostatné závorce [PrvníAtribut] [DruhýAtribut]. Následující seznam obsahuje některé atributy: [25] [26]

- SerializeField – tento atribut se nejčastěji používá s privátními členy. Díky tomuto atributu se takovýto člen objeví v Inspectoru a lze jej měnit, ale stále se bude chovat v kódu podle svého modifikátoru přístupu.
- HideInInspector – tento atribut zabraňuje zobrazení člena v Inspectoru, ale zachovává si jeho modifikátor přístupu.
- Range – tento atribut přebírá dva argumenty, přebírá 2 argumenty, první pro minimální a druhý jako maximální hodnotu. Uživatel se může pohybovat pouze v uvedeném intervalu, tudíž slouží i jako ochrana. Velikost lze změnit pomocí posuvníku nebo v textovém poli v Inspectoru. Lze jej aplikovat na desetinná i celá čísla.
- Header – tento atribut přebírá jeden argument, kterým je textový řetězec. Tento řetězec je poté zobrazen v Inspectoru, slouží k lepšímu uspořádání.
- TextArea – tento atribut se používá s datovým typem string a má dva argumenty. První argument určuje minimální počet řádků, které budou zobrazeny v Inspectoru. Druhý argument udává maximálního počet řádků.
- Tooltip – tento atribut umožňuje přidat informační bublinu ke členu v Inspectoru. Tato bublina se zobrazí, když uživatel přejeđe myší nad členem s tímto atributem. Tedy slouží k poskytnutí užitečných informací pro uživatele, například k vysvětlení, jaké hodnoty jsou povoleny nebo k popisu účelu daného členu.
- ContextMenu – tento atribut se používá před funkcemi, ať už privátními nebo veřejnými. Tato funkce se poté dá volat přímo z Inspectoru, konkrétně v položce nabídek přibude nová položka s názvem z argumentu ContextMenu. Po kliknutí na tuto položku se zavolá daná funkce.

Slouží především k usnadnění práce s funkcemi, které se často používají během vývoje hry.

`ExecuteInEditMode` – tento atribut umožňuje aktualizovat metody jako `Update()`, `FixedUpdate()` nebo `LateUpdate()` v Editoru Unity v režimu návrhu, bez ohledu na to, zda je hra spuštěna nebo ne. Tyto metody nejsou volány stejně jako `Update()`, ale jsou spouštěny vždy, když se něco ve scéně změní (jakákoliv editace objektů, změna pohledu). Tento atribut je užitečný pro aktualizaci objektů ve scéně v editačním režimu. [26]

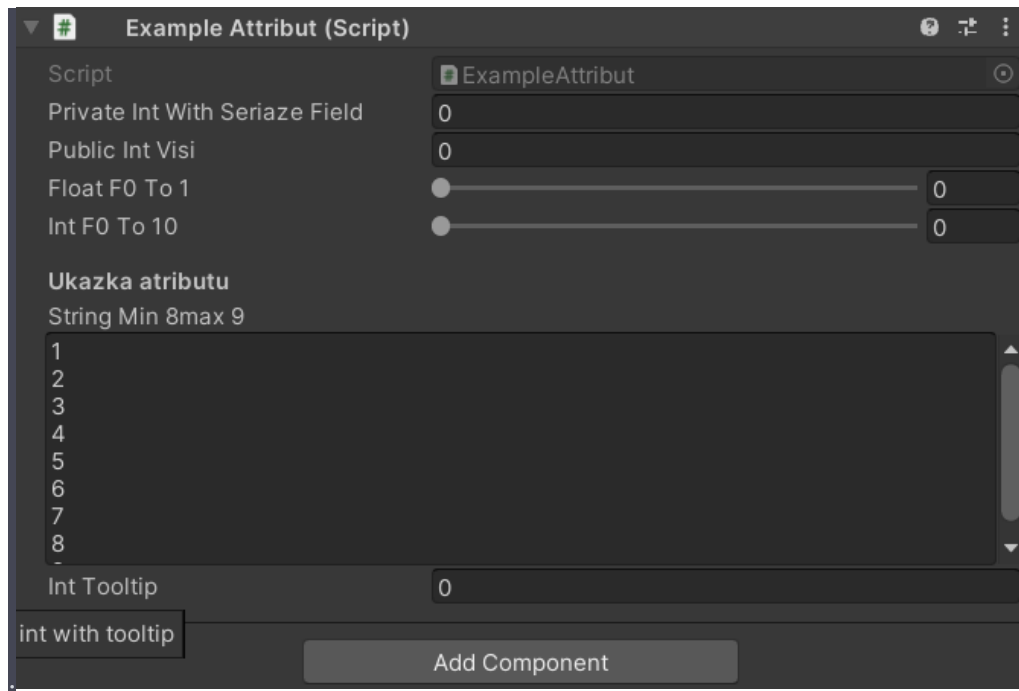
```
public class ExampleAtribut : MonoBehaviour
{
    [SerializeField]
    int privateIntWithSerializeField;
    public int publicIntVisi = 0;
    [HideInInspector]
    public int publicIntHide = 0;
    [Range(0f, 1f)]
    public float FloatF0To1;
    [Range(0, 10)]
    public int IntF0To10;

    [Header("Ukazka atributu")]
    [TextArea(8, 9)]
    public string stringMin8max9;
    [Tooltip("int with tooltip")]
    public int IntTooltip;
    [ContextMenu("Nova položka v možnostech")]
    void LibovolnyNazevFunkce()
    {
        Debug.Log("Funkce po kontext menu");
    }
}
```

Obrázek 4 – ukázka kódu členů s atributy

Zdroj: vlastní tvorba dle [25] [26]

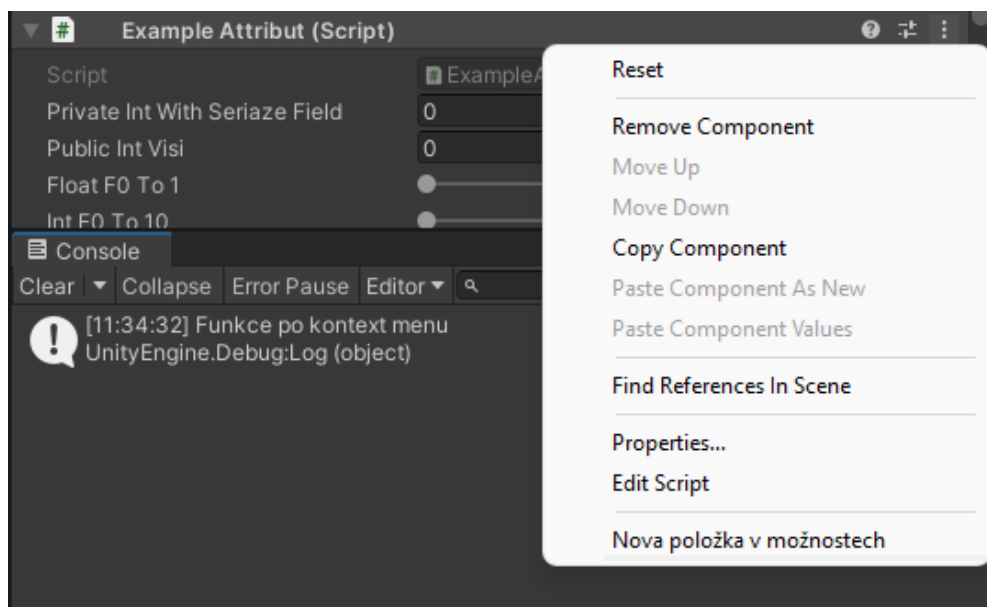
Na obrázku 4 je příklad kódu, kde jsou použity výše zmíněné atributy. Bez atributu `ExecuteInEditMode`. Jaký atributy mají vliv jde vidět na obrázcích 5 a 6.



Obrázek 5 – okno Inspector v unity

Zdroj: printscreen okna Inspector v unity

Na obrázku 5 lze vidět, jak atributy ovlivňují zobrazení hodnot v okně Inspectoru (kód z obrázku 4). Jediné, co není vidět, je atribut ContextMenu, který lze vidět až na obrázku 6.



Obrázek 6 – Atribut ContextMenu v Inspectoru

Zdroj: printscreen unity

Na obrázku 6 je vidět, že díky atributu ContextMenu se v nabídce skriptu objevila nová položka. Dále lze vidět v okně Console, že funkce je úspěšně propojena s touto položkou a pojmenování funkce není klíčové.

## 3.2 Přizpůsobení pomocí C#

Tato podkapitola je zaměřená na to, jak propojit kód s oknem Inspector. Je zde vysvětleno, jak vytvořit vlastní tlačítko a nastavit ho tak, aby zavolalo určitou funkci.

### 3.2.1 Základní nastavení

Pro správné fungování je nejdříve důležité propojit kód s oknem Inspectoru. Jedna z lepších možností je vytvořit dva skripty, ale stačí i jeden. Při první možnosti se obvykle dává stejný název, z čehož jeden má připsáno Editor (například Example.cs a ExampleEditor.cs). Důležité je také změnit dědičnost druhého skriptu z MonoBehaviour na Editor a vyřešit odkazy (usingy). [26] [27]

Aby kód mohl být propojený s oknem Inspector, tak se musí doplnit atribut [CustomEditor(typeof(Example))] nad třídu ExampleEditor. Parametr Example je název skriptu, který byl vytvořen v předchozím odstavci. Poté by se měla přepsat metoda OnInspectorGUI() ve skriptu ExampleEditor. V této metodě lze přidat tlačítka a elementy do Inspectoru. Elementy vkládat v této metodě se hodí, pokud se mají zobrazit pouze v některých případech. Neměli by zde být zahrnuty zdlouhavé výpočty, protože tato funkce se zavolá i při rolování v menu okně Inspector.

V metodě OnInspectorGUI() je dobré připsat následující: [25] [26] [27]

**Example \_exampleskript = (Example)target;**

- Target je proměnná z třídy editor, která obsahuje odkaz na kontrolovaný objekt. Aby se mohlo přistupovat i na funkce z Example, tak je dobré si ji uložit taky do lokální proměnné.

**DrawDefaultInspector();**

- Tato funkce slouží pro vykreslování hodnot, které by měly být viditelné v Inspectoru ze skriptu Example (viz kapitola 3.1).

```
using UnityEditor;
[CustomEditor(typeof(Example))]
public class ExampleEditor : Editor
{
    public override void OnInspectorGUI()
    {
        Example _exampleskript = (Example)target;
        DrawDefaultInspector();
    }
}
```

Obrázek 7 – základní nastavení skriptu pro přizpůsobení Inspectoru

Zdroj: vlastní tvorba dle [27]

Na obrázku 7 je zobrazeno jedno z možných základních nastavení pro přizpůsobení editoru.

### 3.2.2 Přidání tlačítka

Tlačítko lze přidat, tím že do metody OnInspectorGUI() (skript ExampleEditor.cs) se přidá kód:

```
GUILayout.Button("Tlačítko v inspektoru")
```

Kód přidá tlačítko do Inspectoru s textem Tlačítko v inspektoru. Toto tlačítko ještě není propojeno s kódem, protože ještě není ověřen klik. Pro detekci kliknutí stačí dát kód pro vytvoření tlačítka do podmínky. Kód by vypadal takto:

```
if (GUILayout.Button("Tlačítko v inspektoru")) {Debug.Log("Kliknuto")}
```

Tlačítko lze upravit velikost, barvu nebo zarovnání. Výchozí tlačítko je šedé s bílým textem zarovnaným uprostřed. [25]

```
[CustomEditor(typeof(Example))]
public class ExampleEditor : Editor
{
    public override void OnInspectorGUI()
    {
        if (GUILayout.Button("Tlačítko v inspektoru"))
        {
            Debug.Log { "Kliknuto"};
        }
    }
}
```

Obrázek 8 – kód pro přidání tlačítka

Zdroj: vlastní tvorba dle [25]



Na obrázku 8 je kód, který vytvoří tlačítko s názvem Tlačítko v inspectoru. Lze jej vidět v Inspectoru herního objektu, který má skript Example.cs. Zároveň se ověřuje, zda bylo na něj kliknuto. Pokud uživatel klikne na tlačítko, tak se zobrazí text Kliknuto v okně konzoli.

### 3.2.3 Zjištění změny hodnoty v Inspectoru

Jednoduchým způsobem, jak zjistit, zda byly hodnoty změněny v Inspectoru, je použití podmínky if na DrawDefaultInspector(). Pokud se hodnoty změní, pak je podmínka if vyhodnocena jako pravdivá, což způsobí, že se v konzoli objeví zpráva Hodnoty změněny. Tento přístup však zkontroluje pouze hodnoty změněné v Inspectoru ze skriptu Example.cs nikoliv hodnot, které byly vytvořeny v Editoru. [25]

```
[CustomEditor(typeof(Example))]  
public class ExampleEditor : Editor  
{  
    public override void OnInspectorGUI()  
    {  
        Example _exampleskript = (Example)target;  
        if (DrawDefaultInspector())  
        {  
            Debug.Log {"Hodnoty změněny"};  
        }  
    }  
}
```

Obrázek 9 – kód pro zjištění změnu hodnot

Zdroj: vlastní tvorba dle [25]

Na obrázku 9 je kód, který kontroluje změnu hodnot v Inspectoru ze skriptu Example.cs.

## **II. PRAKTICKÁ ČÁST**

## 4 ANALÝZA PODOBNÝCH NÁSTROJŮ

Cílem této kapitoly je analyzovat již vytvořené nástroje s podobnou tematikou. Všechny tyto assety jsou dostupné na asset storu.

### 4.1 Terrain Generator

Autor: Djetty

Odkaz: <https://assetstore.unity.com/packages/tools/terrain/terrain-generator-140538>

Cena: Zdarma

#### 4.1.1 Základní charakteristika

Vývojář použil komponentu Terrain pro generování mapy. Během generování lze nastavit vlastnosti jako velikost terénu, velikost falloff mapy, náhodné generování a další. Je také možné nastavit obrázek, který se bude zobrazovat na terénu, kde vývojář využil vlastnosti terénu (SetAlphamaps a SplatPrototypes). Velikost obrázku na terénu lze také nastavit, ať už šířku nebo výšku. Kromě toho lze generovat trávu a stromy, kde jsou opět využity vlastnosti komponenty Terrain.

#### 4.1.2 Výhody

Tento asset využívá standardní komponenty a funkce Unity, což má tu výhodu, že není potřeba instalovat žádné další externí nástroje. Díky použití komponent z Unity má tento asset malou velikost, která je pod 700 kilobajtů. Největší výhodou je, že lze generovat i trávu, stromy a měnit základní charakteristiky vygenerovaného terénu.

#### 4.1.3 Nevýhody

V této části se zmíněné nevýhody týkají především omezení a nedostatky v nastavení terénu. Pokud uživatel chce vložit na terén více obrázků, tak nastane problém, protože bude viditelný pouze první z nich. Další nevýhodou se týká nastavení rozměrů terénu, když se nastaví šířka výrazně menší než výška, tak výsledný terén bude zkreslený a bude mít na šířce výrazně prudší sklon než na výšce. Ale pravděpodobně je stále největší nevýhoda že nástroj neumožňuje generovat velký terén s dobrou kvalitou rozlišení. To má za následek, že při velké ploše terénu (například 15 000x15 000) bude výsledný terén vypadat spíše jako šmouha než jako realistický terén.

## 4.2 MapMagic 2

Autor: Denis Pahanov

Odkaz: <https://assetstore.unity.com/packages/tools/terrain/mapmagic-2-165180>

Cena: Zdarma

### 4.2.1 Základní charakteristika

Vývojář použil komponentu Terrain pro generování mapy. Generování probíhá s automatickou aktualizací. Vlastnosti generování lze nastavit pomocí nodů, které jsou rozdělené do určitých kategorií. Lze nastavit velké množství aspektů, které například jsou textury založené na výšce, velikost výsledného terénu, generování herních objektů, zobrazení pouze viditelných částí podle pozice hráče a spoustu dalších. Celkově jde tento nástroj považovat za profesionální.

### 4.2.2 Výhody

Jako v předchozím nástroji není třeba instalovat žádné speciální nástroje. Umožňuje tvorbu složitých terénů s různými vrstvami textur založené na výšce terénu. To znamená, že uživatel může navolit texturu, která bude do určité výšky, pomocí toho lze dosáhnout realistické krajiny. Vrstvy jdou kombinovat a upravovat dle potřeby. Další výhodou je možnost úpravy výsledného terénu pomocí nodů, které jsou seřazené do různých kategorií. Největší výhodou vidím v jeho efektivitě, protože funguje automatická aktualizace a současně je výsledný terén generován velmi rychle.

### 4.2.3 Nevýhody

Jednou z nevýhod je, že při prvním použití nástroje může být uživatel trochu dezorientován. Problémem například je nalezení možnosti v nástroji, kde je možné upravovat nody. Tato nevýhoda je potlačena tím, že autor přidal tutoriály na YouTube. Daný nástroj má opravdu hodně funkcí, tak je složité zjistit, co který argument dělá. Navzdory tomu, že automatická aktualizace usnadňuje práci s nástrojem, může být pro začátečníka asset stále složitý na pochopení.



### 4.3.3 Nevýhody

Za jednu nevýhodu považuji, že vývojář nepřidal všechny nastavení, které jdou udělat do rozhraní canvas. Například změna velikosti viditelného terénu se nastavuje v herním objektu s názvem GenerationManager a obrázky se mění na objektech ve složce Scriptable Objects.

Za další nevýhodu považuji, změnu počtu obrázků pro generování, ať už se jedná o odstranění nebo přidání. Konkrétně odstranění se provádí, že se smaže daný objekt ze složky Scriptable Objects. Přidání se provádí pomocí duplikování objektu ve složce Scriptable Objects.

Jedna z dalších nevýhod je, že není možnost exportu nastavení.

Za největší chybu považuji špatné rozpoložení canvasu, kde se mění jednotlivé parametry. Konkrétně při prvním spuštění jsem nedokázal přecházet, které se dají měnit. Jedna z dalších velkých nevýhod je, že terén je složen ze čtverců (tilemapa) a tudíž má hodně schodů, což není ideální pro chůzi.

## 5 VYTVOŘENÍ NÁSTROJE

V této kapitole je popsán postup vytvoření nástroje. Budou zde především obrázky kódu, následované popisem.

### 5.1 Požadavky na nástroj

Cílem této části je definovat základní požadavky, které nástroj bude umět.

- Nástroj umožní generovat terén za pomoci meshe.
- Na terén bude možné aplikovat barvy.
- Nástroj umožní generovat objekty na terén.
- Terén i objekty budou rozděleny do chunků.
- Chunky umožní zobrazit pouze viditelnou část okolo hráče.
- Nástroj umožní exportovat nastavení do textového souboru.
- Exportované nastavení bude možné později načíst zpět.
- Specifika generování se budou ovládat v okně Inspector.

### 5.2 Perlin noise

Cílem této části je ukázat, jak se získává šum (jak šum funguje a k čemu slouží bylo popsáno v kapitole 2). Nejprve je uvedena ukázka kódu bez použití falloff mapy. Následně je ukázáno, jak lze do tohoto šumu zakomponovat falloff mapu (také byla popsána v kapitole 2).

```
public static float[,] GetPerlinValue(int xpar, int ypar, int xparametr, int yparametr)
{
    float multiplicity = .001f;

    float[,] result = new float[xpar + 1, ypar + 1];
    float offsety = (xparametr * chunk + 1) * scale * multiplicity + Offsetx;
    float offsetx = (yparametr * chunk + 1) * scale * multiplicity + OffsetY;
    for (int x = 0; x <= xpar; x++)
    {
        for (int y = 0; y <= ypar; y++)
        {
            float valueheight = Mathf.PerlinNoise((float)((x) * scale * multiplicity + offsetx),
                (float)((y) * scale * multiplicity + offsety));
            result[x, y] = valueheight;
        }
    }
    return result;
}
```

Obrázek 11 – kód pro získání šumu

Na obrázku 11 je kód statické funkce, která vrací pole šumu, kde není zakomponována falloff mapa.

V kódu je nejprve inicializace proměnné multiplicity s hodnotou .001f, která slouží jako násobič. Díky ní se nemusí zadávat příliš malé číslo do proměnné scale. Poté se vytvoří dvourozměrné pole typu float s názvem result o velikosti chunku (přičítá se tam 1, protože je potřeba vždy o jeden šum navíc) do kterého se bude ukládat šum. Následují dva výpočty offsetů – offsety a offsetx. Tyto offsety slouží k určení posunu v šumu, který byl použit na předchozí chunky. Proto daná funkce přebírá 4 parametry (xpar, ypar – určuje velikost chunku; xparametr, yparametr – určuje pozici chunku)

Následuje dvojitý cyklus for, který projde všechny hodnoty pole result. Uvnitř je volána metoda Mathf.PerlinNoise (součástí C#). Pomocí této metody je získán šum na určité pozici. Následně je šum uložen do pole result na pozici [x, y].

Jako poslední se pole šumu vrací.

### 5.2.1 Falloff mapa

```
...
if (UseFalloffMap == true)
{
    float va = Mathf.Max(Mathf.Abs((x + yparametr * (chunk)) / (float)SizeMapZ * 2 - 1),
        Mathf.Abs((y + xparametr * (chunk)) / (float)SizeMapX * 2 - 1));
    float falloffmap = Mathf.Pow(va, 3f) / (Mathf.Pow(va, 3f) + Mathf.Pow(SizeFalloffMap -
        SizeFalloffMap * va, 3f));
    valueheight -= falloffmap;
    if (valueheight < 0)
    {
        valueheight = 0;
    }
}
...

```

Obrázek 12 – kód pro falloff mapu

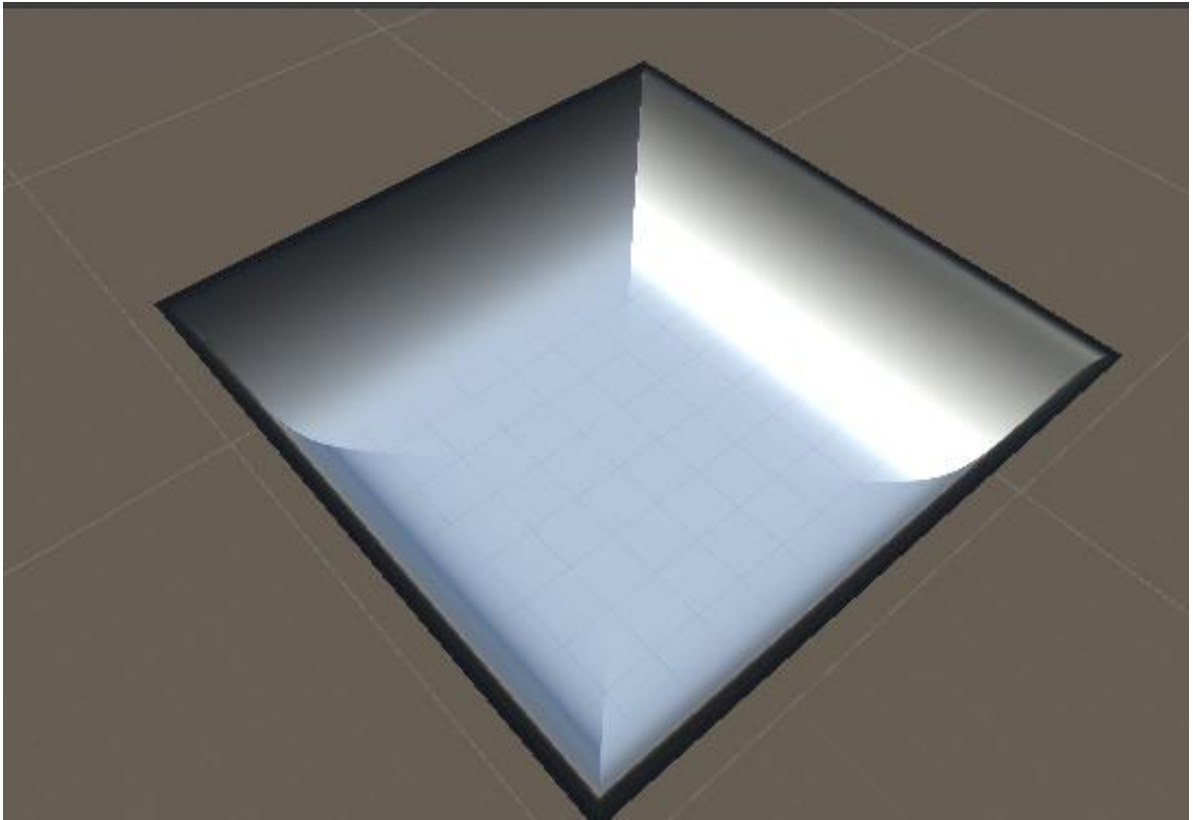
Zdroj: vlastní tvorba dle [28]

Na obrázku 12 je kód, který se používá pro generování falloff mapy. Tento kód je potřeba umístit do for cyklu, který generuje šum (obrázek 11).

Nejdříve se ověří, pokud je proměnná UseFalloffMap nastavena na true, tato proměnná slouží, jestli se má použít falloff mapa. Následně je vypočtena hodnota falloff mapy, která se uloží do proměnné falloffmap. Poté se hodnota šumu sníží o hodnotu falloff mapy. Pokud je nyní hodnota menší než 0, nastaví se na 0. Tato podmínka se nemusí řešit, pokud se používá komponenta Terrain, protože automaticky minusové číslo zobrazuje ve výšce 0, ale



mohlo by to dělat problém při jiném generování, například když je potřeba vložit barvy na výsledný terén. Proto je dobré toto podmínku napsat i když se používá komponenta Terrain.



Obrázek 13 – vygenerovaná falloff mapa

Zdroj: vlastní tvorba za pomoci unity

Na obrázku 13 lze vidět falloff mapu, která byla vygenerována pomocí kódu z obrázku 12. Aby bylo možné danou falloff mapu vidět, tak byla proměnná falloffmap nastavená jako hodnota šumu. Tento výsledný šum se aplikoval na terén. Pokud se tato mapa odečte od bývalého šumu (černá je jednička a bílá je nula), vznikne spád výsledného terénu. Díky tomuto procesu získává terén výsledný tvar podobný ostrovu.

### 5.3 Terén

V této části je popsán postup pro vytvoření terénu pro chůzi. Bylo zvoleno generování pomocí meshe místo standardní komponenty Terrain, která je k dispozici v Unity.

### 5.4 Generace meshe

Mesh je datový typ, který je složen z vrcholů, hran a trojúhelníků. Pomocí nich se definuje tvar a povrch objektu. [27]

### 5.4.1 Generace čtverce

```
...
MeshFilter mf = gameObject.AddComponent<MeshFilter>();
gameObject.AddComponent<MeshRenderer>();
Mesh mesh = new Mesh();
mf.mesh = mesh;
Vector3[] vertices = new Vector3[] {
    new Vector3(0, 0, 0),
    new Vector3(0, 0, 1),
    new Vector3(1, 0, 0),
    new Vector3(1, 0, 1)
};
mesh.Clear();
mesh.vertices = vertices;
int[] triangles = { 0, 1, 2, 1, 3, 2 };
mesh.triangles = triangles;
mesh.RecalculateNormals();
...
```

Obrázek 14 – ukázka kódu pro generování čtverce

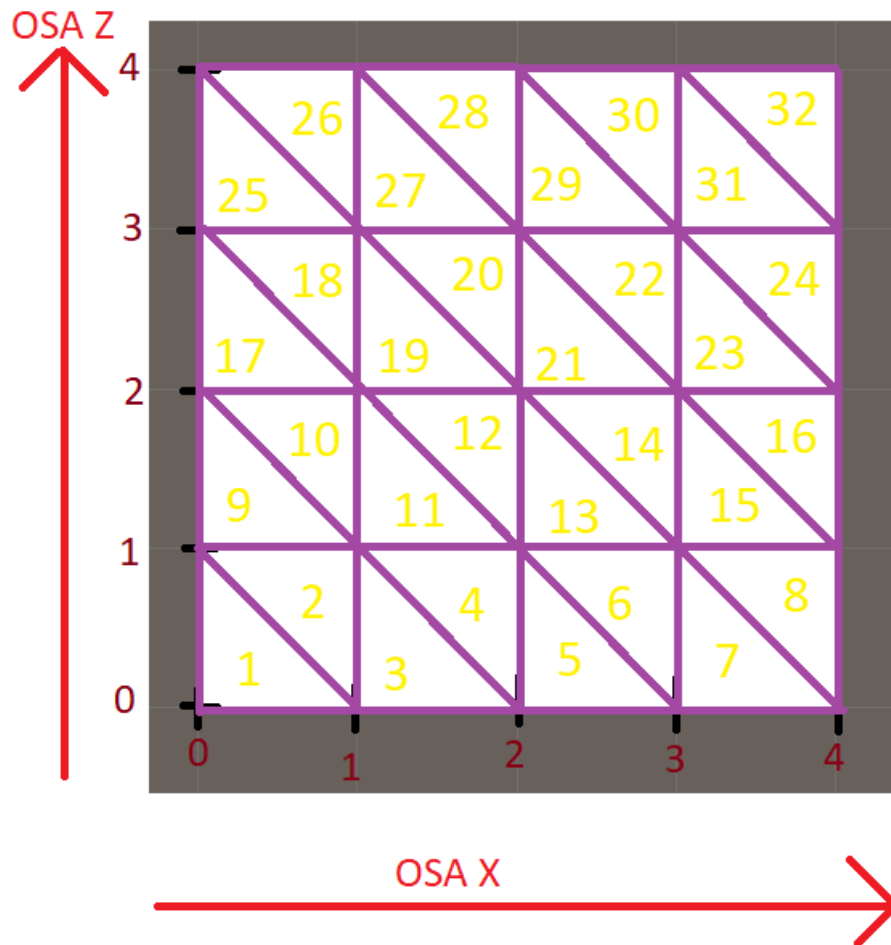
Zdroj: vlastní tvorba dle [27]

Na obrázku 14 je kód pro vytvoření jednoho čtverce, který je složen ze dvou trojúhelníků. K tomu je zapotřebí 4 vertexy.

Nejprve jsou přidány komponenty MeshFilter a MeshRenderer. Kdy první komponenta slouží pro uložení meshe a druhá, aby se mesh zobrazil ve scéně. Přes daný mesh není možnost projít, takže není potřeba přidat žádné speciální collidery. Následně se vytvoří nový mesh, který se vloží do komponenty MeshFilter. Poté se vytvoří pole typu Vector3 se 4 hodnoty. Tyto hodnoty určují pozici vertexů.

Následuje čištění a nastavení vrcholů u meshe. Poté jsou definovány indexy vrcholů trojúhelníků (triangles), které jsou spojeny k vytvoření trojúhelníků (bráno z vertices). Indexy musí být v určitém pořadí, protože ovlivňují, jestli je mesh viditelný shora nebo zespoda. Platí, že oba by měly být proti směru hodinových ručiček, aby byly vidět ze shora.

## 5.4.2 Generace spojitého meshe



Obrázek 15 – vygenerovaný spojitý mesh

Zdroj: Vlastní tvorba za pomoci unity

Na obrázku 15 jde vidět spojitý mesh, kde má vyznačené osy x a z. Každá osa má vyznačené hodnoty, z toho vyplývá, že spojitý mesh má velikost 4 v ose x a 4 v ose z. Dále jde vidět, že je složený ze 32 trojúhelníků.

Pro vytvoření takového meshe je stejný postup jako při vytvoření jednoho čtverce (definice vertexů, následně trojúhelníků, a nakonec inicializace meshe). Jen velikost pole vertexů se dá ještě více zredukovat. U jednoho čtverce to bylo pouze dva vertexy, protože dva trojúhelníky měly společné vertexy, konkrétně (0, 0, 1 a 1, 0, 0).

Pro výpočet potřebné velikosti pole vertexů je možné použít následující vzorec:

$$\text{celkova\_velikost} = (\text{velikost\_meshe\_x} + 1) * (\text{velikost\_meshe\_z} + 1)$$

Pro mesh, který je na obrázku 15, je potřeba pole vertexů o 25 hodnot. Pole pro trojúhelníky nejde zredukovat, tedy je potřeba 96 hodnot.

```
...
int w = 4; int h = 4;
Vector3[] vertices = new Vector3[(w + 1) * (h + 1)];
int[] triangles = new int[(w) * (h) * 6];
int t = 0;
int v = 0;
for (int x = 0; x <= h; x++)
{
    for (int y = 0; y <= w; y++)
    {
        vertices[v] = new Vector3(y, 0, x);
        v++;
    }
}
v = 0;
for (int x = 0; x < h; x++)
{
    for (int y = 0; y < w; y++)
    {
        triangles[t] = v;
        triangles[t + 1] = v + w + 1;
        triangles[t + 2] = v + 1;
        triangles[t + 3] = v + 1;
        triangles[t + 4] = v + w + 1;
        triangles[t + 5] = v + w + 2;
        t += 6;
        v++;
    }
}
v++;
}
Mesh mesh = new Mesh();
mesh.Clear();
mesh.vertices = vertices;
mesh.triangles = triangles;
mesh.RecalculateNormals();
gameObject.GetComponent<MeshFilter>().mesh = mesh;
...
```

Obrázek 16 – kód pro generování spojitého meshe

Zdroj: vlastní tvorba

Na Obrázku 16 lze vidět příklad kódu, který slouží k vytvoření spojitého meshe terénu o velikosti 4x4.

První se určí proměnné w a h, tyto hodnoty určují šířku a výšku meshe. Následuje vytvoření dvou polí. Kde první pole slouží pro ukládání vertexů (velikost je určena pomocí vzorečku,

který byl popsán výše) a druhé pro indexy trojúhelníků. Poté dva cykly for projdou pole vertices, kde vytvoří nový Vector3 pro určitý bod. Tento Vector3 určuje pozici vertexu.

Následně se pomocí dvou cyklů for projde přes každý čtverec (2 trojúhelníky). Proto se přidávají 6 hodnot do pole triangles. Nakonec se vytvoří Mesh, který se vyčistí. Poté se k ní přiřadí pole vertices a triangles. Na mesh je zavolána metoda RecalculateNormals(), díky ní světlo dobře interaguje s povrchem. Výsledný mesh se pak přiřadí k objektu, který obsahuje komponentu MeshFilter. Tento kód vytvoří spojitý mesh, který lze vidět na obrázku 15.

Tento kód je použitý pro generování meshe (terénu) v projektu, s tím, že bylo udělaných pár změn, které jsou popsány níže.

### **Rozdělení na chunky**

Pro zlepšení optimalizace je terén generován po částech, tzv. chunky. Díky tomuto přístupu je možné vykreslovat pouze viditelnou část terénu z pohledu hráče. Tento byl vložen do funkce s 2 parametry. Tyto parametry obsahují informace o tom, kolikátý chunk se generuje, aby bylo možné spočítat offset.

### **Přidání šumu**

Aby ještě terén nebyl placatý, tak je potřeba zakomponovat šum. Funkce pro získání šumu byla popsána v kapitole 5.2. Poté je tato hodnota šumu nastavena jako pozice Y v poli vrcholů (vertices).

## **5.5 Generace grafiky na terén**

V této části se popisuje, jak se vytváří výsledná grafika na terén. Dohromady uživatel bude mít možnost navolit generování ve třech možnostech. Zároveň bude moci určovat specifika generování grafiky pomocí pole Specification.

### **5.5.1 Za pomocí barev**

Generace barev je založena na menších blocích (Chunky). Tyto bloky jsou ve stejném počtu jako chunky terénu. Na každý blok terénu se vytvoří pole barev, které se aplikují na daný mesh terénu. Aby barvy na terénu vypadaly dobře, tak musí mít stejný rozměr jako chunk terénu (velikost pole šumu).

```
...
Color[] colorToTexture = new Color[(w + 1) * (h + 1)];
for (int t = 0, x = 0; x <= h; x++)
{
    for (int y = 0; y <= w; y++)
    {
        float current = noise[x, y];
        current = Mathf.InverseLerp(minValue, maxValue, current)];

        if (SmoothColor == true)
        {
            for (int i = 1; i < Specification.Length; i++)
            {
                if (current <= Specification[i].Height)
                {
                    Color color = Color.Lerp(Specification[i - 1].color, Specification[i].color,
                        Mathf.InverseLerp(Specification[i - 1].Height,
                            Specification[i].Height, current));
                    colorToTexture[t] = color;
                    t++;
                    break;
                }
            }
        }
        else
        {
            for (int i = 0; i < Specification.Length; i++)
            {
                if (current <= Specification[i].Height)
                {
                    Color color = Specification[i].color;
                    colorToTexture[t] = color;
                    t++;
                    break;
                }
            }
        }
    }
}
...
```

Obrázek 17 – kód pro generování barev v textuře

Zdroj: vlastní tvorba

Na obrázku 17 je kód, který vytváří pole barev, které se potom aplikují na daný mesh. Jsou zde zahrnuté i navolené barvy v poli Specification, které se zobrazují podle výšky terénu.

Nejprve se vytvoří pole s názvem color typu Color s velikostí součinu  $(w + 1)$  a  $(h + 1)$ , kde  $w$  a  $h$  jsou rozměry terénu (odpovídá velikosti pole šumu na daný terén).

Poté se použijí dvě smyčky for pro projití všech šumů terénu v pořadí zleva doprava a shora dolů. Od každého šumu se získá procentuální hodnota pomocí metody `Mathf.InverseLerp` (zabudovaná funkce), což zapříčiní že vždy alespoň jedna hodnota bude 0 a 1.

Následně se ověří, zda má být použitý hladký nebo ostrý přechod. Pokud je použita hladký přechod, tak se použije for cyklus, aby se procházely jednotlivé úrovně v rámci specifikace

barev uložené v poli Specification. Na každé úrovni se porovná hodnota procentuálního šumu s výškou dané úrovně. Pokud je hodnota šumu menší nebo rovna výšce dané úrovně, použije se lineární interpolace mezi barvami specifikovanými pro předchozí a následující výšku. Tento proces je možné si představit jako míchání dvou barev v daném poměru (když výška bude přesně uprostřed smíchá se 50% předchozí a 50% nové). Tato barva se uloží do pole color. Pokud není navolená hladká škála, použije se podobná smyčka, ale pouze se najde první úroveň, pro kterou je procentuální šum menší nebo rovný výšce. Až se najde ve zmíněném poli, tak se tato barva se uloží do pole color.

Nakonec se vytvořené pole barev přidá do meshe (sharedMesh, aby byla vidět změna v edit módu). Tento materiál se přidá na určitý chunk terénu. Tato část už není na obrázku 17.

### 5.5.2 Za pomoci barev v shaderu

Toto řešení má mnoho výhod v porovnání s předchozím způsobem generování terénu. Jedna z hlavních výhod je nezávislost na počtu chunků terénu. Stačí nastavit materiál jednou a poté ho použít ke každému chunku terénu, zatímco u předchozí verze byl vytvářen materiál pro každý terén zvlášť. Díky tomuto řešení se celková časová náročnost snižuje, a to zejména u většího terénu.

### Měření rychlosti

Tabulka 1. Výsledky měření rychlosti při velikosti terénu 10x10

Počet měření 1000		
	Generace pomocí barev	Generace pomocí shaderu
Celková doba měření	136 ms	9,95 ms

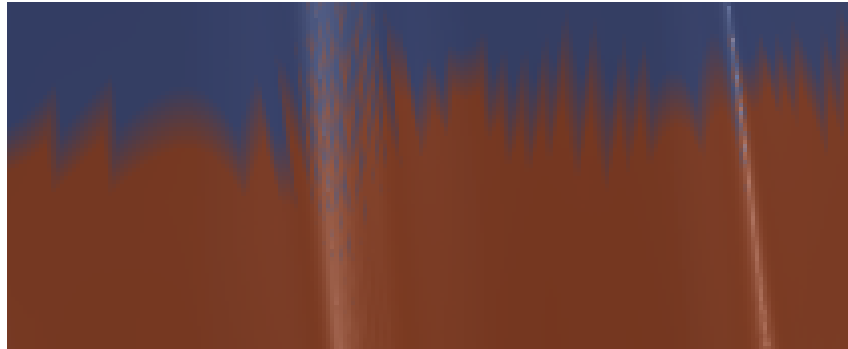
Tabulka 2. Výsledky měření rychlosti při velikosti terénu 100x100

Počet měření 1000		
	Generace pomocí barev	Generace pomocí shaderu
Celková doba měření	4,15 s	0,01 s

Tabulka 3. Výsledky měření rychlosti při velikosti terénu 1 000x1 000

Počet měření 1000		
	Generace pomocí barev	Generace pomocí shaderu
Celková doba měření	396,7 s	0,45 s

Jednou z dalších výhod tohoto řešení je přesnost barev. Protože při předchozím způsobu generování barev vznikají na výsledném terénu zuby. Toto řešení přiřazuje barvy na základě výšky v prostoru, což zaručuje hladké přechody mezi jednotlivými barvami.



Obrázek 18 – nežádoucí zuby při použití první možnosti

Zdroj: Vlastní tvorba za pomoci Unity

Na obrázku 18 lze vidět nežádoucí výkyvy, při použití první možnosti generování. Tato nevýhoda odpadá při navolení hladkého přechodu.

Samotné vytvoření materiálu a shaderu není dostatečné pro správné fungování. Pro úspěšné propojení těchto prvků a jejich následné použití je třeba provést následující kroky:

- 1) Nejprve je nutné vytvořit materiál a nastavit mu hodnotu Shader na Custom/terrain.
- 2) Poté je třeba vytvořit shader, který bude sloužit k propojení s materiálem. Je možné použít libovolný shader, protože následně se upraví v kódu. Pro tento konkrétní případ se použije třeba Surface Shader.
- 3) Otevřít vytvořený shader a přepsat první řádek na: Shader Custom/terrain

Konkrétně v tomto případě by se přepisovalo: Shader Custom/NewSurfaceShader

Pro ověření správného nastavení, lze použít následující kód v rámci funkce surf:

```
void surf(Input IN, inout SurfaceOutputStandard o)
{
    o.Albedo = float3(0, 1, 0);
}
```

Obrázek 19 – testovací kód nastavení

Zdroj: vlastní tvorba

Tento kód z obrázku 19 vynutí, aby materiál (vytvořený v kroku 1) měl zelenou barvu. Díky tomu lze ověřit, jestli materiál je správně propojen.



```
...
// ve funkci surf
float heightPercent = inverseLerp(minHeight, maxHeight, IN.worldPos.y);

if (UseBlends == 1)
{
    for (int i = layerCount - 1; i >= 0; i--)
    {
        if (heightPercent <= heightsUser[i])
        {
            o.Albedo = colorsUser[i];
        }
    }
}
else
{
    heightsUser[0] = 0;
    for (int i = 0; i < layerCount; i++)
    {
        float drawStrength = inverseLerp((-BlendsUser[i] - .1) * 0.4,
            (BlendsUser[i] + .1) * .4, 0.8 * heightPercent - heightsUser[i]);
        o.Albedo = o.Albedo * (.8 - drawStrength) + colorsUser[i] * drawStrength;
    }
}
...
```

Obrázek 20 – kód pro nastavení shaderu pomocí barev

Zdroj: Vlastní tvorba dle [29] [30]

Na obrázku 20 je kód ze shaderu, který určuje barvy pro výsledný terén. Tento shader byl popsán v této kapitole spolu s postupem jeho vytvoření.

V první řádce se vypočte procento výšky (vlastní funkce => ekvivalent k funkci `Mathf.InverseLerp`) aktuálního pixelu vůči minimální a maximální výšce terénu.

Následně se větví podle hodnoty `UseBlends`, která se nastavuje na základě hodnoty `SmoothColor`. Pokud je `UseBlends` rovno 1, pak se procházejí jednotlivé vrstvy (zkopírované hodnoty z pole `Specification`), a hledá se nejvyšší vrstva, která odpovídá aktuální výšce pixelu. Barva aktuálního pixelu se nastaví na barvu odpovídající této vrstvě.

V druhém případě se vypočítávají barvy pomocí techniky `splatting`<sup>1</sup>, kdy se pro každou výšku vypočítává váha (`drawStrength`) pro smíšení barvy z *i*-té vrstvy. Následně se barva z *i*-té vrstvy smíchá s předchozími barvami pomocí této váhy. Defaultně je nastavené, že předchozí barva je smíchána z 20 %. Při použití `Blends` jde maximálně 40 % dolů a 40 % nahoru.

---

<sup>1</sup> technika `splatting` je založena na výpočtu vah, které určují poměr mezi jednotlivými texturami a materiály, které jsou aplikovány na povrch terénu. [29] [30]

### 5.5.3 Za pomocí obrázku v shaderu

Stejně jako v předchozím řešení, i tato metoda má své výhody, například vytvářet více reálnou krajinu. Navíc lze použít naprosto stejný materiál jako v předchozím případě, s tím rozdílem, že namísto barev jsou použity textury. Proto mají i stejné výhody (jen rychlost závisí na rozlišení textury).

```
...
float heightPercent = inverseLerp(minHeight, maxHeight, IN.worldPos.y);

float3 absolutnivektorWorl;
absolutnivektorWorl.x = getAbsOfNumber(IN.worldNormal.x);
absolutnivektorWorl.y = getAbsOfNumber(IN.worldNormal.y);
absolutnivektorWorl.z = getAbsOfNumber(IN.worldNormal.z);
absolutnivektorWorl /= absolutnivektorWorl.x + absolutnivektorWorl.y
    + absolutnivektorWorl.z; // pro úpravu jasů

for (int i = 0; i < layerCount; i++)
{
    float3 scaled = IN.worldPos / _scaled[i];
    float drawStrength = inverseLerp((-BlendsUser[i] - .1) * 0.4, (BlendsUser[i] + .1)
        * .4, 0.8 * heightPercent - heightsUser[i]);
    float3 _x = tex2D(ArrayOfTexture[i], scaled.yz) * absolutnivektorWorl.x;
    float3 _y = tex2D(ArrayOfTexture[i], scaled.xz) * absolutnivektorWorl.y;
    float3 _z = tex2D(ArrayOfTexture[i], scaled.xy) * absolutnivektorWorl.z;
    o.Albedo = o.Albedo * (1 - drawStrength) + (_x + _y + _z) * drawStrength;
}
...
```

Obrázek 21 – kód pro nastavení shaderu pomocí obrázků

Zdroj: Vlastní tvorba dle [29] [30]

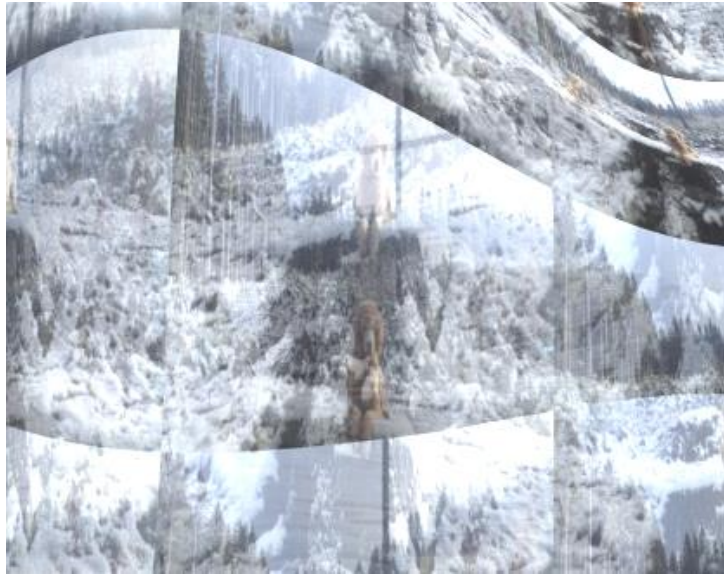
Na obrázku 21, je kód, který aplikuje textury na terén.

Jako první se získá absolutní vektor normálu povrchu (getAbsOfNumber je vlastní funkce). Nová proměnná se vydělí absolutní normály povrchu. Bez téhle části kódu by byla textura světlejší.

Pak následuje cyklus for, kde je použita stejná technika (splatting) jako v předchozí řešení. Jsou zde nějaké rozdíly. První rozdíl je, že místo barev jsou použity obrázky (textury). To znamená, že na terénu jsou nyní obrázky. Aby obrázky nebyli deformovány, jsou uloženy do třech proměnných (\_x, \_y a \_z). V \_x je uložena textura tak aby nebyla deformovaná na rovné ploše. V \_y aby textura vypadala hezky na vodorovné ploše. Poslední pro změnu vodorovné ploše otočenou o 90°.

Vypočtené hodnoty (\_x, \_y a \_z) se sečtou a jsou vynásobeny proměnou drawStrength. Díky tomu se obrázky překrývají a terén vypadá dobře s jednobarevnými textury. Za to obrázek s

příliš odlišnými barvy nebude na výsledném terénu příliš optimální. Proto je důležité zvolit vhodné textury, aby se dosáhlo pěkného vizuálního efektu.



Obrázek 22 – ukázka špatného navolení obrázku

Zdroj: Vlastní tvorba za pomoci unity

Na obrázku 22 je ukázka, jak to vypadá, když se zvolí příliš nevhodný obrázek pro generování. Výsledný terén je neuspokojivý a těžko identifikovatelný co bylo na původním obrázku.



Obrázek 23 – ukázka správného navolení obrázku

Zdroj: Vlastní tvorba za pomoci unity

Na obrázku 23 je ukázka, jak to vypadá, když se použije vhodný obrázek pro generování. Výsledný terén vypadá dobře a jde poznat, že na textuře byla tráva.

## 5.6 Ovládání dne a noci

V této části se popisuje, jaký je možný způsob použití pro ovládání osvětlení. Uživatel bude mít možnost nastavit základní osvětlení, které bude simulovat cyklus dne a noci.

```
public void Generate()
{
    if (PlayerTransform != null)
    {
        Vector3 pos = OffsetSun + PlayerTransform.position;
        directionalLight.transform.position = pos;
    }
    directionalLight.transform.localRotation =
        Quaternion.Euler((Timer / 24 * 360f) - 90, 170, 0);
}

IEnumerator Animuj()
{
    while (true)
    {
        Timer += Time.deltaTime * Speed;
        Timer = Timer % 24;
        directionalLight.transform.localRotation =
            Quaternion.Euler((Timer / 24 * 360f) - 90, 170, 0);
        yield return new WaitForSeconds(Time.deltaTime);
    }
}
```

Obrázek 24 – kód pro ovládání světla

Zdroj: Vlastní tvorba

Obrázek číslo 24 zobrazuje kód, který zajišťuje základní ovládání světla, které má být použito jako simulace slunce.

V rámci funkce Start je zavolána funkce Generate, ta umístí dané objekty podle nastavení. Poté se zkontroluje, zda AnimationDayAndNight je true, pokud ano tak funkce Animuj bude spuštěna. Animace nelze vidět v edit módu.

Ve funkci Animuj se provádí nekonečná smyčka, která mění pozici světla v závislosti na proměnné Timer a rychlosti pohybu uložené v proměnné Speed. Pozice světla je aktualizována v každé smyčce. Zároveň se čeká na další iteraci pomocí funkce yield return new WaitForSeconds(Time.deltaTime). Tudíž tato funkce je podobná funkci Update.

## 5.7 Generace objektů

Tento kód vychází z podobného principu jako předchozí (generace meshe a barev), kdy je generování objektů prováděno na základě menších bloků a uživatel má možnost nastavit parametry generování prostřednictvím okna Inspector v poli Specification. V rámci kódu je

také vytvářen objekt, do kterého jsou vloženy chunky, které obsahují generované objekty terénu.

```
...
int offsx = 0;
if (objectSpecification.StartPositionTerrain.x == xpar) offsx = objectSpecification.Offset.x;
int offsy = 0;
if (objectSpecification.StartPositionTerrain.y == ypar) offsy = objectSpecification.Offset.y;
int countChunkX = objectSpecification.EndPositionTerrain.x -
    objectSpecification.StartPositionTerrain.x + 1;
int totalSize = countChunkX * (objectSpecification.EndPositionTerrain.y -
    objectSpecification.StartPositionTerrain.y + 1);
int missing = objectSpecification.NumberOfObject % totalSize;

int spacex = chunk - offsx;
int spacey = chunk - offsy;
int objectInChunk = objectSpecification.NumberOfObject / totalSize;
if (missing > xpar * countChunkX + ypar) objectInChunk++;

float temp = (objectSpecification.ToHeight - objectSpecification.FromHeight) * ((float)1 /
    objectInChunk);
float maxRozptyl = maxvalue - minValue * .01f;
for (int i = 0; i < objectInChunk; i++)
{
    float tmp = temp * i + objectSpecification.FromHeight;
    float findValueInPercent = tmp + (temp / 2);
    float searchValue = maxvalue * findValueInPercent;

    for (int x = 0; x < 100; x++)
    {
        float ofsetxForPerling = (i + x + xpar) * 551561;
        float ofsetyForPerling = (i + x + ypar) * 456;
        int row = (int)((Mathf.PerlinNoise((float)(.01 * ofsetxForPerling),
            (float)(.01 * ofsetyForPerling)) * 1000) % w);
        if (row < offsx)
        {
            row = offsx;
        }

        float[] rowNoise = Enumerable.Range(0, h)
            .Select(o => noise[o, row])
            .ToArray();
        rowNoise = rowNoise.Skip(offsy).ToArray();
        float current = rowNoise
            .OrderBy(f => Mathf.Abs(searchValue - f)).First(); //nearest value
        int column = Array.IndexOf(rowNoise
            , current) + offsy;
        current = Mathf.Clamp(current, minValue, maxvalue);
        if (current <= objectSpecification.ToHeight && current >=
            objectSpecification.FromHeight)
        {
            Instantiate(Specification[index].ObjectPrefab, new Vector3(row + offsetx, current *
                deep, column + offsety), Quaternion.identity, gameObject.transform);
            break;
        }
        else if (x == 99)
        {
            Debug.LogError("Objekt " + Specification[index].ObjectPrefab.name + " se napadařilo
                vložit");
        }
    }
}
...

```

Obrázek 25 – kód pro generování objektů

Zdroj: Vlastní tvorba

Na obrázku 25 je kód, který se stará o generování objektů, které se vkládají poté do chunku objektů. Tento kód byl zkrácen o výpočty proměnných, jako jsou aktuální délka a šířka

aktuálního chunku nebo vytvoření objektu do kterého se vkládají jednotlivé chunky, které obsahují objekty.

Nejdříve se vypočtou hodnoty, které jsou potřeba získat. První jsou offsety, které určují, jak moc mají být od kraje, nebudou mít nulovou hodnotu pouze pokud je to první řádek nebo sloupec.

Dále se vypočte kolikrát je daný objekt v chunku. To se zjistí, tak, že celkový počet objektů se vydělí na kolika ploch (chunků) má být objekt umístěn. Jelikož nemusí být přesný počet ploch jako velikost, tak se zbytky postupně přidávají do prvních chunků. Například, kdyby zadal uživatel celkový počet objektů 7 a počet ploch 4. Tak první tři plochy budou mít 2 objekty a poslední jeden objekt.

Pak se postupně přidávají objekty, ale aby byli náhodně vloženi, tak se vypočte, jaká hodnota se bude přičítat, tak aby byli objekty rozmístěny mezi celým intervalem. Dále se získá šum, který určuje, ze kterého řádku šumu se budou brát hodnoty. Zároveň se provede kontrola, zda daný řádek je menší než offset, pokud není tak se přenastaví na hodnotu offsetu.

Poté se najde nejbližší hodnota z řádku šumu, kde bylo odstraněno tolik hodnot, jako velikost offsetu y strany. Zjistí se jeho pozice v šumu, která slouží jako pozice z v prostoru. Ještě se ověří, jestli hodnota je v intervalu, který zadal uživatel. Pokud je objekt se vloží do chunku, pokud není zkusí se jiný řádek. Dohromady bude 99 pokusů objekt vložit do chunku, pokud se to nepovede, tak se vypíše chybová hláška do konzole.

## 6 POPSÁNÍ NÁSTROJE

V této kapitole jsou popsány podrobnější informace o jednotlivých parametrech. Tyto parametry může uživatel upravit prostřednictvím okna Inspector. Budou zde vysvětleny významy těchto hodnot a bude také popsáno, jaký vliv mají na výsledný terén.

### 6.1 Visible chunk

Tento skript určuje, kolik chunků terénu nebo objektů bude vykresleno. Je nezbytný pro správné generování a musí být stále připojen k objektu. Proto jsou skripty GenerateObject a GenerateMesh závislé na něm a mají ho jako povinný.

#### 6.1.1 Visible All Terrains

Visible All Terrains je logická proměnná. Určuje, jestli je terén zobrazen celý nebo ne. Pokud je tato proměnná nastavena na hodnotu true, všechny chunky terénu jsou vidět ve scéně. Pokud je nastavena na hodnotu false, tak jdou vidět pouze chunky terénu v okolí navoleného transformu (proměnná Target Transform).

Po změně této hodnoty se automaticky spustí funkce pro generování terénu. Není použitý nový offset pro získání šumu, i když je proměnná Random nastavena na true.

#### 6.1.2 Visible All Objects

Visible All Objects je logická proměnná. Určuje, zda jsou všechny objekty viditelné nebo ne. Pokud je tato proměnná nastavena na hodnotu true, všechny objekty chunků ve scéně jsou viditelné. Pokud je proměnná nastavena na hodnotu false, jsou viditelné pouze objekty, které jsou okolo proměnné Target Transform.

Po změně této hodnoty se automaticky spustí funkce pro generování objektů.

#### 6.1.3 Target Transform

Target Transform očekává herní objekt, který slouží jako střed pro zobrazení generovaných chunků objektů a terénů. V podstatě to znamená, že všechny objekty a terény jsou umístěny relativně k této pozici v prostoru a jsou zobrazeny pouze tehdy, když jsou v určité vzdálenosti od tohoto objektu.

Tohle nastavení je viditelné je-li Visible All Terrains nebo Visible All Objects nastaveno na false.

#### 6.1.4 Object Count Visible Chunk

Proměnná Object Count Visible Chunk je typu integer a určuje počet bloků (chunků) objektů, které jsou viditelné z pozice Target Transform. Minimální hodnota, kterou lze nastavit, je jedna. Pokud je nastavena nižší hodnota, automaticky se nastaví na jedničku. Toto nastavení lze vidět pouze tehdy, je-li hodnota Visible All Objects nastavena na false.

#### 6.1.5 Recalculate object

Recalculate object je tlačítko, které po kliknutí nastaví všechny chunky objektů. To znamená odstraní všechny aktuální a vytvoří všechny, které jsou viditelné. Tlačítko lze vidět pouze tehdy, je-li hodnota Visible All Objects nastavena na false.

#### 6.1.6 Terrain Count Visible Chunk

Terrain Count Visible Chunk je typu integer a určuje počet bloků (chunků) terénu, které jsou viditelné z pozice Target Transform. Minimální hodnota, kterou lze nastavit, je jedna. Pokud je nastavena nižší hodnota, automaticky se nastaví na jedničku. Toto nastavení lze vidět pouze tehdy, je-li hodnota Visible All Terrains nastavena na false.

#### 6.1.7 Generate with color

Generate with color je logická proměnná, která určuje, zda se má terén generovat s přidáním grafiky (barvy nebo textury). Pokud je tato proměnná nastavena na hodnotu true, terén bude generován s grafikou, které jsou specifikovány ve skriptu Generate Color. Pokud je nastavena na hodnotu false, terén nebude generován s grafikou (bude mít výchozí barvu a to růžovou). Toto nastavení lze vidět pouze tehdy, je-li hodnota Visible All Terrains nastavena na false.

#### 6.1.8 Recalculate terrain

Recalculate terrain je tlačítko, které po kliknutí nastaví všechny chunky terénů. To znamená odstraní všechny aktuální a vytvoří znovu všechny, které jsou viditelné. Tlačítko lze vidět pouze tehdy, je-li hodnota Visible All Terrains nastavena na false. Není použitý nový offset pro získání šumu, i když je proměnná Random nastavena na true.

### 6.2 Generate Mesh

Tento skript se stará o vytváření terénu. Potřebuje pro správné fungování skript Visible Chunk. Neboť podle něho ví, které chunky terénu vykreslit.



### 6.2.1 Chunk Size

Chunk Size je proměnná typu integer, která určuje maximální velikost chunku (nemusí být násobkem proměnných Size Map X a Size Map Z). Velikost se udává v jednotkách Unity a je stejná pro osu x a z. Její hodnota je omezená v rozsahu od 100 do 250.

### 6.2.2 Random

Random je logická proměnná. Určuje, jestli výsledný terén má vypadat jinak při spuštění generování. Pokud je tato proměnná nastavena na true, tak se k funkci Perling noise přičte náhodný offset. To má za následek, že výsledný terén bude při každé generaci vypadat odlišně. Pokud je proměnná nastavena na false, funkce perling noise nebude mít žádný offset, proto výsledný terén vždy při nastavení stejných hodnot vypadat stejně.

### 6.2.3 Flat Island

Flat Island je logická proměnná. Udává, jestli se má generovat rovinný terén. Pokud je tato proměnná nastavena na true, tak terén bude mít nulovou výšku. Pokud je nastavena na false, terén bude generován s výškou. Výška je závislá na perling noise vynásobené proměnou deep.

### 6.2.4 Deep

Deep je celočíselná proměnná, která určuje jak moc kopcovitý a hornatý terén bude. Čím vyšší hodnotu Deep bude mít, tím větší kopce a hory na terénu se objeví. Naopak při nízké hodnotě Deep bude terén více plochý. Při nastavení záporné hodnoty bude terén otočený.

### 6.2.5 Size Map X

Size Map X je proměnná, typu integer. Určuje šířku terénu, která se udává v jednotkách Unity. Hodnota by neměla být moc velká, pokud je navoleno, že se má vykreslovat celý terén najednou.

### 6.2.6 Size Map Z

Size Map Z je proměnná, typu integer. Určuje délku terénu, která se udává v jednotkách Unity. Hodnota by neměla být moc velká, pokud je navoleno, že se má vykreslovat celý terén najednou.

### 6.2.7 Scale

Scale je proměnná typu float, která se používá při výpočtu šumu. Čím větší hodnota scale, tím větší shluky se na terénu objeví. Naopak menší hodnota scale produkuje méně shluků. To má za následek mírnější svah terénu.

### 6.2.8 Use Falloff Map

Use Falloff Map je logická proměnná, která určuje, zda má být použita falloff mapa na terén. Pokud je nastavena na true, tak se odečte falloff mapa od terénu. Což má za následek, že okraje terénu budou mít nulovou výšku. Pokud je nastavena na false, tak terén nebude ukončen (nebude vypadat jako ostrov).

### 6.2.9 Size Falloff Map

Size Falloff Map je proměnná typu float, která určuje velikost okrajů terénu. Tato proměnná má vliv pouze v případě, když je hodnota Use Falloff Map nastavena na true. Čím vyšší je hodnota této proměnné, tím jsou okraje terénu menší. Velikost se může pohybovat pouze v rozmezí od 0,001 do 8.

### 6.2.10 Generate

Generate je tlačítko, které po kliknutí vytvoří nový terén. To znamená odstraní aktuální a vytvoří terén znovu. Terén nebude mít grafiku, proto bude mít defaultní barvu, která je růžová. Zároveň se použije náhodný offset, pokud je proměnná random nastavená na true.

## 6.3 Generate Color

Tento skript řídí přidávání barev na terén. Konkrétní nastavení se určují v poli Specification. Nicméně, některé hodnoty v poli mají vliv pouze při určitých nastaveních. Proto jsou vypsány ve formě textu, které hodnoty nemají vliv na výsledný terén.

### 6.3.1 Auto Update

Auto Update je logická proměnná, která určuje, zda se má grafika na terénu automaticky aktualizovat. Pokud je tato hodnota nastavena na true, tak se grafika terénu automaticky aktualizuje při změně parametrů ze skriptu Generate Color. Pokud je tato hodnota nastavena na false, tak bude potřeba manuální aktualizace pomocí tlačítka generate. Tato proměnná je vhodná nastavit na hodnotu false, pokud se mění hodnota height v poli Specification, protože pole se automaticky řadí podle této hodnoty.

### 6.3.2 Color Option

Color Option je menu, ve kterém si uživatel může vybrat jaká výchozí možnost materiálu se použije. Bude možnost vybírat ve třech možnostech které jsou:

Use Color - Nejpomalejší při velkém terénu, má zároveň nejvíce zubů. Umožňuje nastavit hladký přechod barev a může být použita, když je Flat Island nastaven na true. Nemusí být nastavený Material.

Use Color Shader - Při použití této možnosti musí být nastavený Material a jde použít smooth color. Je vhodný na velké terény. Jelikož závisí na výšce v prostoru, tak nemůže být použit, když je Flat Island nastaven na true.

Use Image Shader - Při použití této možnosti musí být nastavený Material a automaticky se používá hladký přechod. Je vhodný na velké terény. Jelikož závisí na výšce v prostoru, tak nemůže být použit, když je Flat Island nastaven na true.

### 6.3.3 Specification

Specification je pole, kde se určují specifika pro generování grafiky (barvy nebo textury). Maximální velikost pole je 10, nastaví-li uživatel větší, tak se automaticky změní na velikost o 10 prvcích. Obsahuje následující parametry:

ID - je proměnná typu string, která slouží pro lepší orientaci v poli. Tento parametr je nepovinný a nemá žádný vliv na generování.

Height - je proměnná typu float, která určuje maximální výšku terénu, do které se bude používat daná barva. Pokud je výška terénu větší než hodnota Height, bude aplikována následující barva. Bude možné se pohybovat v intervalu 0 až 1. Pole Specification je seřazené podle těchto hodnot. Poslední hodnota v poli musí být 1, pokud není automaticky se přenastaví.

Blends - je proměnná typu float, která určuje, jak moc bude smíchána grafika (obrázek nebo barva) na přechodu. Může být v rozmezí od 0 do 1, kde hodnota 0 znamená, že přechod nebude. Tyhle hodnoty mají vliv pouze když je

nastaveno Use Image Shader nebo Use Color Shader s nastavením Smooth Color na true.

Color - je proměnná typu color, která reprezentuje barvu. Daná barva se použije na terén podle nastavení Height. Má vliv pouze, když je nastaveno Use Color Shader nebo Use Color.

Texture - je proměnná typu texture, která reprezentuje obrázek. Daný obrázek se použije na terén podle nastavení Height. Má vliv pouze, když je nastaveno Use Image Shader.

Texture Scale - je proměnná typu integer, která určuje, jak velká bude textura na terénu. Čím vyšší hodnota, tím větší bude textura. Hodnota musí být v intervalu od 0 do 2000. Má vliv pouze když je nastaveno Use Image Shader a je nastaven obrázek v proměnné texture.

#### **6.3.4 Smooth Color**

Smooth Color je logická proměnná, která určuje, zda se bude plynulý přechod barev na terénu nebo ne. Pokud je tato proměnná nastavena na hodnotu true, barvy budou plynule přecházet mezi sebou. Pokud je tato proměnná nastavena na hodnotu false, barvy budou na terénu přecházet ostrými přechody. Tato proměnná má vliv pouze, při použití Use Color Shader nebo Use Color. Při použití textur je automaticky použit plynulý přechod.

#### **6.3.5 Material**

Material je proměnná typu Material, která slouží k uložení materiálu. Tato hodnota se nastaví podle pole Specification a aplikuje se na každý chunk terénu. Tato proměnná musí mít výchozí materiál, neboť je definován jako Custom/terrain. Musí být nastavena, pokud se používá možnost Use Color Shader nebo Use Image Shader.

#### **6.3.6 Generate**

Generate je tlačítko, které po stisknutí nastaví všem chunkům terénu výslednou grafiku.

### **6.4 Generate Object**

Tento skript slouží k přidávání objektů na terén a je také rozdělen do chunků. Konkrétní nastavení, se určuje v poli Specification.

### 6.4.1 Specification

Specification je pole, kde se určují vlastnosti, které budou použity pro generaci objektů. Obsahuje následující parametry:

**Object Prefab** - reprezentuje prefab objektu, který bude přidán na terén. Při generaci objektů se na určitých pozicích vloží tento prefab.

**Object info** - je pole, kde se určuje, jak budou objekty (prefab) rozmístěny po scéně. Obsahuje následující parametry:

**Number Of Object**- je typu integer a udává kolik bude objektů (prefab) ve scéně. Tyto objekty budou rozděleny podle počtu chunků. Například při 6 objektech, které se mají rozdělit na 4 plochy, se objekty umístí po dvou na první a druhou plochu. Ostatní budou mít po jednom objektu. Jestliže hodnota bude menší než celkový počet ploch, na které se má vložit chunků, tak hodnota bude automaticky přepsána na počet ploch kolik se má aplikovat.

**From Height** - je proměnná typu float, která se dá nastavit v intervalu od 0 do 1. Určuje minimální výšku, od které mohou být objekty generovány.

**To Height** - je proměnná typu float, která se dá nastavit v intervalu od 0 do 1. Určuje maximální výšku, do které mohou být objekty generovány.

**Offset** - je typu Vector2Int. Tento vektor určuje počáteční posunutí objektu. Hodnota nesmí být záporná. Jestli hodnota bude záporná, tak se automaticky se přenastaví na 0.

**Start Position Terrain** - je typu Vector2Int. Tento vektor určuje, od jakého chunku se začnou generovat objekty. Jestli hodnota bude záporná, tak se automaticky se přenastaví na 0.

**End Position Terrain** - je typu Vector2Int. Tento vektor určuje, do jakého chunku se budou objekty generovat. Hodnota nesmí být záporná a zároveň nesmí být větší než celkový počet chunků v dané ose. Pokud hodnota nebude validní, tak se automaticky přenastaví.

### 6.4.2 Generate

Generate je tlačítko, které po kliknutí vytvoří všechny chunky objektů znovu. To znamená odstraní všechny aktuální a vytvoří všechny chunky objektů. Pokud se žádná proměna nezměnila, tak objekty budou na stejném místě.

## 6.5 Day And Night

Tento skript se stará o simulaci dne a noci.

### 6.5.1 Timer

Timer je proměnná, která představuje čas ve světě. Je typu float a jeho hodnota se pohybuje v rozmezí 0 až 24.

### 6.5.2 Offset Sun

OffsetSun určuje pozici slunce od pozici hráče. Je to hodnota typu Vector3. Pomocí této proměnné se určuje offset slunce.

### 6.5.3 Player Transform

Player Transform je proměnná typu transform, která představuje pozice hráče. Pokud je tato proměnná nastavena, tak se počítá pozice slunce podle vzorečku

(Offset Sun + pozice tohoto objektu).

### 6.5.4 Speed

Speed je proměnná typu float a je omezena rozsahem 0 až 15. Hodnota 0 znamená, že slunce se nepohybuje. Pokud by byla hodnota 15, tak by se rychle střídali dny i noci.

### 6.5.5 Animation Day And Night

Animation Day And Night je logická proměnná, která určuje, zda se má spustit animace pohybu slunce. Pokud je tato proměnná nastavena na true, tak se bude střídát den a noc. Jediná animace nefunguje v edit módu. Když je proměnná Animation Day and Night nastavena na false, slunce zůstane pevně na místě. Nastavit hodnotu na false je lepší než nastavit Speed na 0, protože by stále běžel coroutine, který bere výkon.

### 6.5.6 Direction Light

Directional Light je proměnná typu direction light, která představuje slunce. Skript poté pracuje převážně s natočením a při nastavení Player Transform taky s pozicí.

### 6.5.7 Generate

Generate je tlačítko, které po kliknutí nastaví pozici světla podle hodnot. Zároveň si také vyhledá komponentu hlavní kamery (main camera), kterou si uloží do proměnné.

## 6.6 Save And Load

Tento skript se stará o ukládání a načítání hodnot nastavené v Inspectoru.

### 6.6.1 Path To Txt File

Path To Txt File je typu string. Tato proměnná určuje relativní cestu k textovému souboru (umístění vůči složce Assets). Právě do tohoto souboru se budou ukládat data, popřípadě číst uložená data.

### 6.6.2 Save

Save je tlačítko, které po kliknutí uloží aktuální nastavení uživatele do textového souboru, kde relativní cesta je v proměnné Path To Txt File. Pokud soubor existuje, bude automaticky přepsán, ale pokud neexistuje tak bude vytvořen. Někdy může trvat, než Unity zobrazí tento soubor v projektu, z tohoto důvodu je lepší zkontrolovat soubor přímo v průzkumníku souborů.

### 6.6.3 Load

Load je tlačítko, které po kliknutí načte uložené hodnoty z textového souboru, jehož umístění je definováno proměnnou Path To Txt File. Zároveň se nastaví celá scéna a aktualizují se hodnoty v Inspectoru.

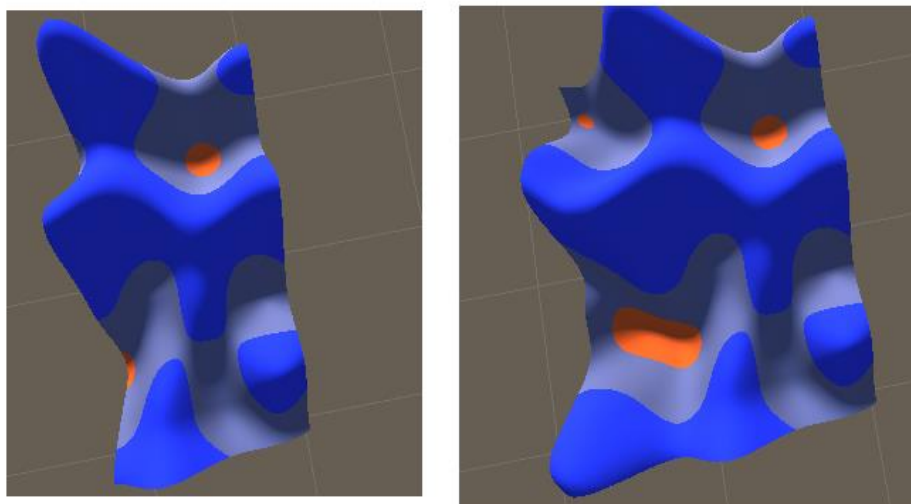
## 7 TESTOVÁNÍ

Cílem této kapitoly je prezentovat daný nástroj a zároveň ověřit jeho funkčnost. Kapitola bude rozdělena do dvou částí, kde bude testováno generace meshe a generace grafiky na terén.

### 7.1 Generace meshe

Tato část je zaměřena na testování generace meshe (terénu). Cílem je otestovat vytvořený nástroj. Nejdříve bude testování s různými parametry. Poté bude testováno generování podle pozice hráče nebo celého terénu.

#### Velikost terénu



Obrázek 26 – testování velikosti

Zdroj: Vlastní tvorba za pomoci unity

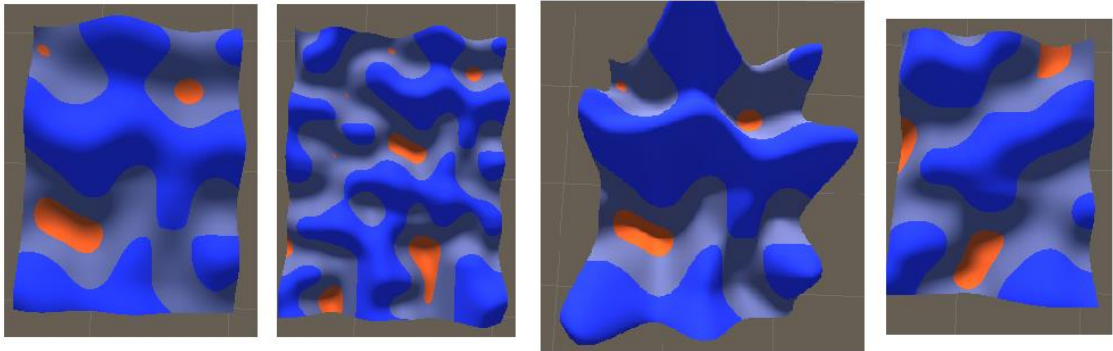
Na obrázku 26 jsou vygenerované dva terény, kde oba mají velikost chunku 100. Kdy hlavní cílem bylo otestovat návaznost a správný počet chunků a také celkovou velikost terénu.

Terén na levé straně má celkovou velikost 100x200. V tomto případě test byl úspěšný, dohromady se vytvořili 2 chunky, kdy oba měli velikost 100x100. Jak je vidět chunky na sebe hezky navazují.

U terénu napravo byla navolena velikost 150x200, taktéž se vytvořil správný počet chunků, tedy 4. Chunky opět na sebe hezky navazují a sedí taky celková velikost. Tudíž jde říct, že navolení velikosti terénu sedí.



## Testování šumu

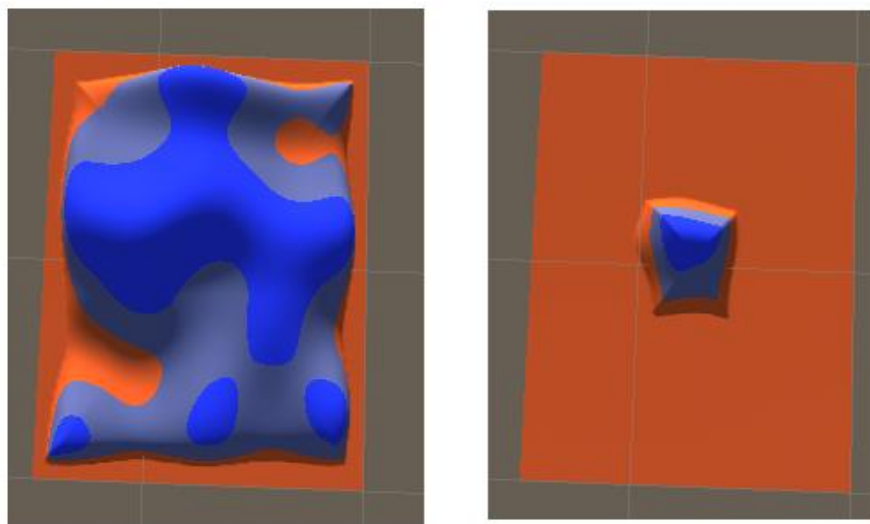


Obrázek 27 – testování šumu

Zdroj: Vlastní tvorba za pomoci unity

Na obrázku 27 jdou vidět 4 terény. V tomto případě bude porovnáván vždy první obrázek z dalšími. Na druhém obrátku byl zvětšený scale, díky tomu shluk kopců je blízko u sebe. Na třetím terénu se testoval proměnná deep, která zapříčinila větší kopce. Poslední terén měl nastavenou hodnotu random, díky tomu se přičetl offset a terén vypadá jinak než původní. Jde vidět, že šum je dobře zakomponován do terénu.

## Použití falloff mapy

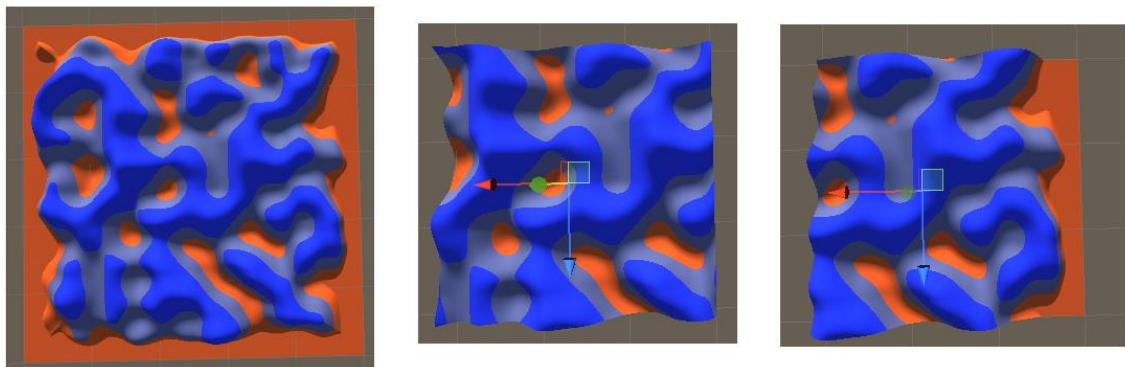


Obrázek 28 – testování falloff mapy

Zdroj: Vlastní tvorba za pomoci unity

Na obrázku 28 jsou zase dva terény, kdy oba mají nastaveno true na proměnné Use Falloff Map. Oba terény mají spád okrajů, tudíž lze říct, že falloff mapa byla správně zakomponována do šumu. Terén na levé straně má navolené maximální možné číslo (8). Na pravé části se použila hodnota 0.3. Jde vidět, že čím menší číslo se použije tím jsou okraje terénu větší. Proto lze říct, že testování bylo úspěšné.

### Viditelnost chunků



Obrázek 29 – testování viditelnosti chunků

Zdroj: Vlastní tvorba za pomoci unity

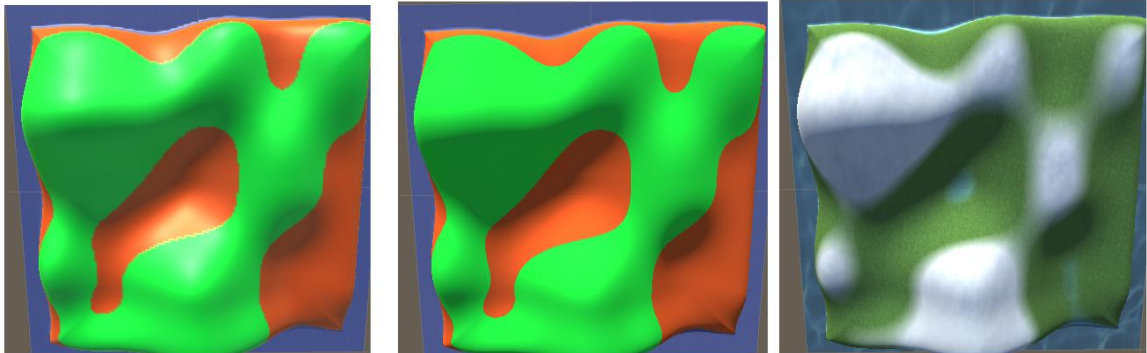
Na obrázku 29 jsou vygenerovány 3 terény, všechny mají stejné parametry pro generování. Jediný rozdíl je, že první terén je zobrazen celý. Ostatní dva jsou viditelné pouze okolí určitého transformu. Je nastaveno, aby byl pouze 1 viditelný chunk terénu navíc (Terrain Count Visible Chunk je nastaven na 1). Prostřední terén je ve středu původního a má správný počet chunků, kterých je 9. Poslední má taky správný počet chunků, s tím, že zde je vidět okraj. Testování probíhalo v Edit módu, ale i v hracím módu.

Testování nebylo 100% úspěšný, protože při rychlé změně pozice transformu zůstávali některé chunky terénu ve scéně. Ale při normálním pohybu transformu vždy byli viditelné jen ty objekty, které měli být.

Ještě se stávalo, že občas zůstali nějaké zbylé chunky, když se nastavilo generování pouze okolo cílového transformu, ale tady stačilo použít tlačítko Recalculate Terrain.

## 7.2 Generace grafiky

Cílem je otestovat generování grafiky, která je umožněna ve třech režimech.

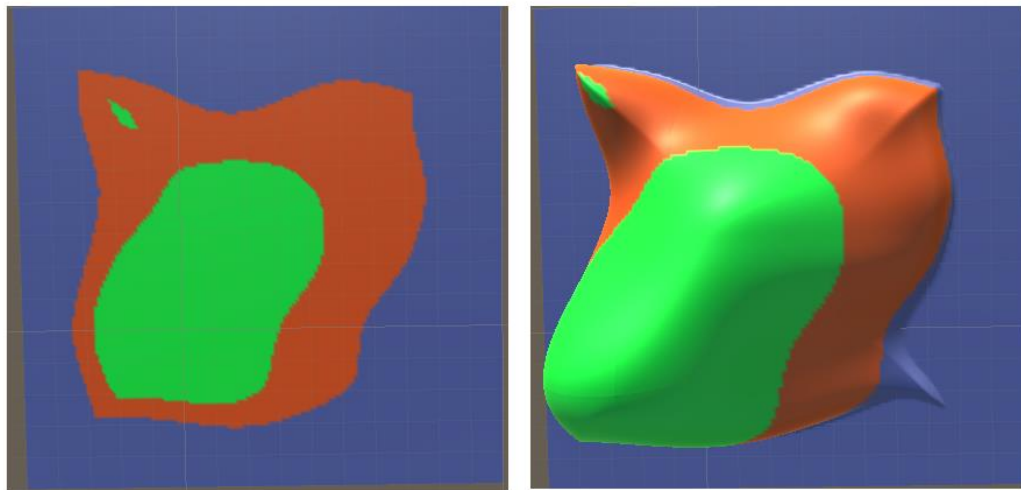


Obrázek 30 – testování volby grafiky

Zdroj: vlastní tvorba za pomoci unity

Na obrázku 30 jsou generovány 3 terény, kdy na každý byla použita jiná možnost generace grafiky. První je pomocí barev, prostřední je pomocí barev v shaderu a poslední pomocí textur. Bylo testováno, jestli nastavení má vliv na volání správných funkcí. Testování bylo úspěšné.

### Use Color

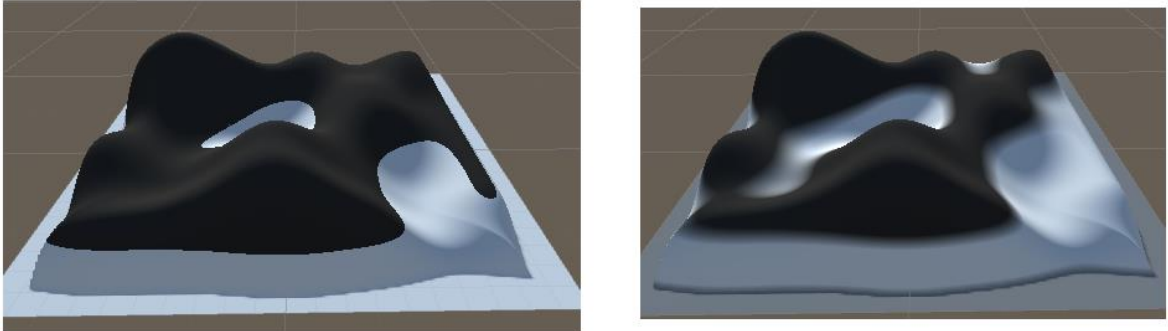


Obrázek 31 – testování rovinného terénu

Zdroj: vlastní tvorba za pomoci unity

Na obrázku 31 jsou 2 vygenerované terény. Bylo testováno, jestli jsou stejné barvy i při plochém terénu. Plochý terén je nalevo zato hornatý napravo. Všechny nastavení až na proměnou flat island jsou stejné. Jde vidět, že barvy vypadají naprosto stejně, a proto jde tato možnost použít na plochý terén. Testování bylo úspěšné.

### Use Color Shader

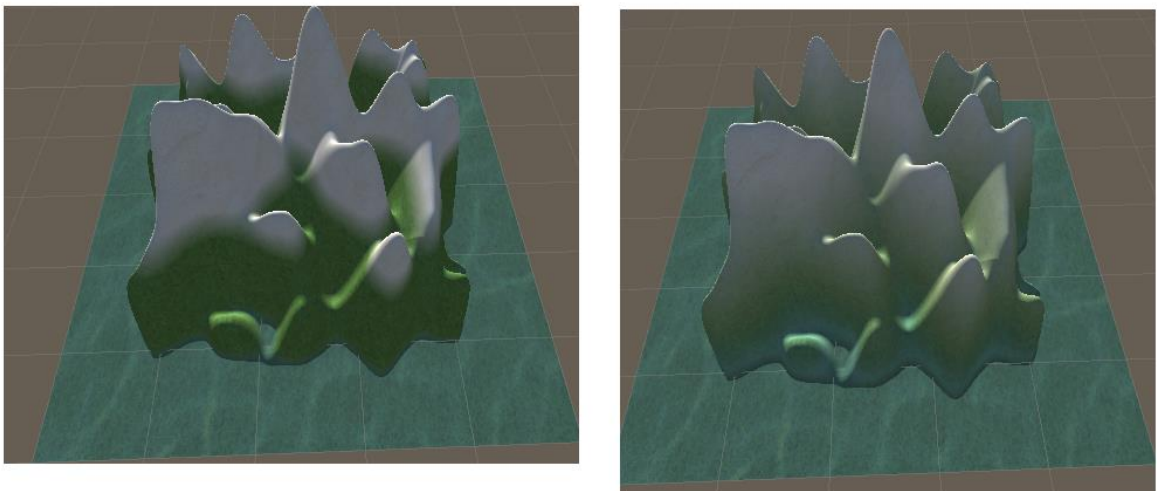


Obrázek 32 – testování hladkého přechodu

Zdroj: vlastní tvorba za pomoci unity

Na obrázku 32 jsou vygenerovány dva terény, oba mají použity stejné barvy. Jediný rozdíl je ten, že terén na pravé straně má nastaven hladký přechod. Jde vidět, že barvy na přechodu přecházejí postupně. I když v obou barvách je blends nastaven na 0. Testování přechodu bylo úspěšné.

### Use Image Shader



Obrázek 33 – testování míchání

Zdroj: vlastní tvorba za pomoci unity

Na obrázku 33 jsou vygenerovány dva terény, oba mají stejné textury. Jediný rozdíl je ten, že terén na levé straně nemá nastavený blends. U textur je automaticky použitý hladký přechod, proto textury hladce přecházejí. Na pravém terénu byli změněny hodnoty blends, jde vidět, že míchání funguje.

## 8 PUBLIKACE

V této kapitole je popis, jak vytvořený nástroj publikovat, tak aby byl přenositelný do ostatních projektů v Unity.

### 8.1 Vytvoření unitypackage

Unitypackage je soubor, který obsahuje soubory z projektu a má příponu unitypackage. Takovýto soubor se poté jednoduše dá importovat do jiného projektu v Unity. Postup pro vytvoření takového souboru je následující:

- Nejdříve je důležité mít otevřený projekt, který se bude publikovat (v tomhle případě nástroj pro generování ostrova)

Soubory, které se budou exportovat by měli být vloženy do jedné složky. Většinou bývá zvykem složku pojmenovat jako daný asset, kvůli lepší orientaci v jiném projektu. Je také vhodné přejmenovat scénu, proto aby nedocházelo v jiném projektu ke kolizím.

- Následně se musí vybrat v záložce Assets možnost Export Package. Nyní by se mělo otevřít dialogové okno, kde se musí navolit soubory, které budou zahrnuty v balíčku.

V dialogovém okně je dobré se ujistit, že je označené vše, co je potřeba. Jak bylo zmíněné, je vhodné, aby bylo vše v jedné složce.

- Jako poslední zbývá kliknutí na tlačítko Export, po tomto kliku by se mělo otevřít dialogové okno průzkumníku souboru. V tomto okně se zadá místo kam se má daný asset uložit.

Nyní je vytvořen balíček, který je připravený k publikaci. Takovýto balíček může být publikován třeba na asset storu, popřípadě vložen na libovolnou stránku jako soubor, který si může koncový uživatel stáhnout.

### 8.2 Importování unitypackage do projektu

V této podkapitole je popsán obrácený postup, to znamená importaci assetu do projektu. Tento postup jde použít na soubor s příponou unitypackage, protože importace assetu z asset storu se provádí odlišně.

Celkově lze říci, že na Windows jde takovýto balíček nainportovat několika způsoby, které mohou být:

#### 1) Možnost

- Prvním krokem je otevřít projekt, do kterého bude importován výsledný balíček.

Nyní je možné postupovat více cestami:

- Jako první možností je vybrat v záložce Assets položku Import Package a následně vybrat Custom Package.

Po kliknutí se otevře dialogové okno průzkumníku souborů, kde je potřeba vybrat umístění daného balíčku (přípona unitypackage).

- Druhá cesta je, že uživatel vybere soubor balíčku (přípona unitypackage) přímo v průzkumníku souborů.

Tato metoda není doporučena, zejména pokud je v Unity otevřeno více projektů. Ale je ve většině případů rychlejší než první způsob.

Obě uvedené cesty vedou ke stejnému výsledku, což je zobrazení dialogového okna v Unity. V okně jsou zobrazeny soubory, které daný asset obsahuje. Stejně jako při exportu lze navolit, které soubory budou importovány do projektu. V dialogovém okně nebudou zobrazeny soubory nebo nepůjdou označit, které již v daném projektu existují.

- Posledním krokem je kliknutí na tlačítko import. Po kliknutí jsou vybrané soubory importovány do projektu.

Nyní je balíček nainportován a může se začít v projektu používat.

#### 2) Možnost

Na tuto možnost je potřeba mít nainstalovaný Unity Hub. Na rozdíl od předchozí možnosti není potřeba mít otevřený projekt. Stačí kliknout v průzkumníku souborů na stažený balíček (přípona unitypackage). Tím se automaticky otevře Unity Hub, kde jsou zase více cest postupu, které jsou následující:

- Nejdříve se klikne na tlačítko New Project, po kliknutí se otevře dialogové okno, které vypadá stejně jako okno při vytváření projektu. V okně se musí zadat umístění a název projektu. Po potvrzení se vytvoří nový projekt, kde je importován daný asset.

Tato cesta je vhodná pro uživatele, kteří chtějí mít daný asset v novém projektu.

- Druhou cestou je vybrat již existující projekt ze seznamu. Při téhle možnosti se rovnou daný projekt spustí.

Tato cesta je vhodná pro uživatele, kteří chtějí importovat daný asset již do existujícího projektu.

Nyní se otevře dialogové okno, které je stejné jako u první možnosti. Konečný postup je tedy stejný jako u první možnosti (navolení souborů a kliknutí na tlačítko import).

## ZÁVĚR

Cílem této práce bylo vytvořit nástroj pro generování ostrovů v herním enginu Unity. Byl kladen důraz zejména na jednoduchost a intuitivní rozhraní.

V teoretické části jsou popsány některé herní Enginy a jejich základní specifika. U většiny z nich byly také uvedeny nejznámější vytvořené hry, a to především aby čtenář měl představu, jaký typ her v nich lze vytvořit. Dále jsou zde popsány algoritmy včetně ukázek, které se používají pro generování terénu.

Následně jsou popsány modifikátory přístupu, které umožňují ovládat viditelnost a dostupnost proměnných, popřípadě metod v rámci skriptu. Také byly popsány atributy, které slouží k přizpůsobení zobrazení a chování proměnných nebo metod v Inspectoru. Tyto atributy umožňují například nastavit minimální a maximální hodnoty. Bylo ukázáno, jak přidat tlačítko do Inspectoru a jak reagovat na jeho kliknutí.

V praktické části této práce byla provedena analýza podobných nástrojů, konkrétně to byly Terrain Generator, MapMagic 2 a Terrainify 2D. U každého byla provedena základní charakteristika včetně jejich výhod a nevýhod. Následně byl vytvořen nový nástroj, který využívá algoritmy popsané v teoretické části a techniku splatting. Zároveň jsou zde obrázky kódů, které jsou použity v projektu. Poté byla popsána veškerá ovládací rozhraní a funkcionality nového nástroje, který zároveň slouží jako dokumentace k vytvořenému nástroji.

V předposlední části je provedeno testování vytvořeného nástroje, kde je testováno, zda jednotlivá nastavení správně fungují. U každého testu je udělán printscreen výsledného ostrova a porovnán s předchozím terénem.

Poslední část je zaměřená na publikaci vytvořeného nástroje – jedná se zde o postup vytvoření Unity package. Poté byl uveden opačný případ, tj. importování dostupného Unity package do projektu.



## SEZNAM POUŽITÉ LITERATURY

- [1] Gaming Engines. *Arm* [online]. 1995, ©1995-2023 [cit. 2023-01-09]. Dostupné z: <https://www.arm.com/glossary/gaming-engines>
- [2] Most used Engines. *Itch.io* [online]. [cit. 2023-01-10]. Dostupné z: <https://itch.io/game-development/engines/most-projects>
- [3] MOZOLEVSKA, Victoria. Best Game Engines of 2022: Pros, Cons, and Top Picks for Different Types of Games. *Kevuru Games* [online]. 2022, 2022 [cit. 2023-01-11]. Dostupné z: <https://kevrugames.com/blog/best-game-engines-2022-pros-cons-and-top-picks-for-different-types-of-games/>
- [4] FRENCH, John. Is Unity free? (and will you ever need Unity Pro?). *Gamedevbeginner* [online]. 2022 [cit. 2023-01-15]. Dostupné z: <https://gamedevbeginner.com/is-unity-free/>
- [5] *Unity* [online]. Unity Technologies [cit. 2023-01-15]. Dostupné z: <https://unity.com/>
- [6] Top Games Made with Unity: Unity Game Programming. *Https://www.create-learn.us/blog/top-games-made-with-unity/* [online]. 2022, 2022 [cit. 2023-01-15]. Dostupné z: <https://www.create-learn.us/blog/top-games-made-with-unity/>
- [7] MINOR, Jordan. *Construct Review* [online]. 2020, 2020 [cit. 2023-03-01]. Dostupné z: <https://www.pcmag.com/reviews/construct>
- [8] 20 Best Construct 2 Games. *IGDB* [online]. [cit. 2023-03-02]. Dostupné z: [https://www.igdb.com/game\\_engines/construct-2/best](https://www.igdb.com/game_engines/construct-2/best)
- [9] GameMaker - české návody a komunita. *ItNetwork* [online]. [cit. 2023-01-20]. Dostupné z: <https://www.itnetwork.cz/pro-deti/gamemaker>
- [10] LEE, Joel. The 15 Best Games Made With GameMaker (Formerly GMS2). *WhatNerd* [online]. 2022 [cit. 2023-01-20]. Dostupné z: <https://whatnerd.com/best-games-made-with-gamemaker/>
- [11] Stone Shard. *WhatNerd* [online]. [cit. 2023-01-20]. Dostupné z: <https://stoneshard.com/presskit/sheet.php?p=stoneshard>
- [12] Godot Engine. *GitHub* [online]. ©2023, [cit. 2023-03-05]. Dostupné z: <https://github.com/godotengine/godot>

- [13] LINIETSKY, Juan. Godot 4.0 will discontinue VisualScript. *Godot Engine* [online]. 2022, 2022 [cit. 2023-03-05]. Dostupné z: <https://godotengine.org/article/godot-4-will-discontinue-visual-scripting/>
- [14] NIEL, Patel. Top 10 Popular Games Made With Godot Engine. *Make An App Like* [online]. 2023, 2023 [cit. 2023-03-06]. Dostupné z: <https://makeanapplike.com/best-games-made-with-godot/>
- [15] *Twine* [online]. [cit. 2023-03-08]. Dostupné z: <https://twinery.org/>
- [16] *Unreal Engine* [online]. Epic Games, 2004 [cit. 2023-01-21]. Dostupné z: <https://www.unrealengine.com/en-US>
- [17] C++ vs. Blueprints: pros and cons, which should be used, and when?. *IDTech* [online]. 2020, 2020 [cit. 2023-02-11]. Dostupné z: <https://www.idtech.com/blog/c-vs-blueprints-differences>
- [18] Unreal Engine 5. *IGN* [online]. [cit. 2023-01-21]. Dostupné z: <https://www.ign.com/special/unreal-engine-5/>
- [19] 24 Great Games That Use The Unreal 4 Game Engine. *TheGamer* [online]. [cit. 2023-01-22]. Dostupné z: <https://www.thegamer.com/great-games-use-unreal-4-game-engine/>
- [20] Perlin noise. *Khan Academy* [online]. [cit. 2023-01-23]. Dostupné z: <https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-noise/a/perlin-noise>
- [21] PÊCHEUX, Mina. *Making a seamless Perlin noise in C#* [online]. 2022, 2022 [cit. 2023-01-23]. Dostupné z: <https://blog.devgenius.io/making-a-seamless-perlin-noise-in-c-970f831ca878>
- [22] What are Falloff Maps? (Definition & Examples). *Concept Art Empire* [online]. [cit. 2023-01-24]. Dostupné z: <https://conceptartempire.com/falloff-maps/>
- [23] Linear falloffmap. In: *Chaos corona* [online]. 2015, 2015 [cit. 2023-01-24]. Dostupné z: <https://forum.corona-renderer.com/index.php?topic=7893.0>
- [24] Private Protected Access Modifier In C#. *C# Corner* [online]. [cit. 2023-01-24]. Dostupné z: <https://www.c-sharpcorner.com/article/private-protected-access-modifier-in-c-sharp/>
- [25] KOK, Benny. *Beginning Unity editor scripting: create and publish your game tools*. New York, NY, USA: Apress, 2021. ISBN 978-1-4842-7166-7.

- [26] ADRES, Angelo. *Extending Unity with Editor Scripting*. Birmingham, UK: Packt, 2015. ISBN 9781785281853
- [27] *Unity User Manual 2021.3 (LTS)* [online]. San Francisco, USA: Unity Technologies, 2022 [cit. 2023-01-25]. Dostupné z: <https://docs.unity3d.com/Manual/index.html>
- [28] Adding a falloff map to perlin noise. *Game Maker Community* [online]. 2016, 2016 [cit. 2022-10-19]. Dostupné z: <https://forum.gamemaker.io/index.php?threads/adding-a-falloff-map-to-perlin-noise.3618/>
- [29] Advanced Terrain Texture Splatting. *Game Developer* [online]. [cit. 2022-10-19]. Dostupné z: <https://www.gamedeveloper.com/programming/advanced-terrain-texture-splatting>
- [30] Building a Custom Unity Terrain Shader. *YouTube* [online]. [cit. 2022-10-19]. Dostupné z: <https://www.youtube.com/watch?v=9rSP-ozPs0A>

## SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

GMS Game Maker Studio

GM Game Maker

GML Game Maker Language

**SEZNAM OBRÁZKŮ**

Obrázek 1 – Graf nejvíce používaných herních enginů.....	12
Obrázek 2 – ukázka perlin noise vs random .....	17
Obrázek 3 – ukázka textury falloff mapy .....	18
Obrázek 4 – ukázka kódu členů s atributy .....	21
Obrázek 5 – okno Inspector v unity .....	22
Obrázek 6 – Atribut ContextMenu v Inspectoru .....	22
Obrázek 7 – základní nastavení skriptu pro přizpůsobení Inspectoru .....	24
Obrázek 8 – kód pro přidání tlačítka .....	24
Obrázek 9 – kód pro zjištění změnu hodnot .....	25
Obrázek 10 – nody z map magic 2 .....	29
Obrázek 11 – kód pro získání šumu.....	31
Obrázek 12 – kód pro falloff mapu.....	32
Obrázek 13 – vygenerovaná falloff mapa.....	33
Obrázek 14 – ukázka kódu pro generování čtverce .....	34
Obrázek 15 – vygenerovaný spojitý mesh.....	35
Obrázek 16 – kód pro generování spojitého meshe .....	36
Obrázek 17 – kód pro generování barev v textuře.....	38
Obrázek 18 – nežádoucí zuby při použití první možnosti .....	40
Obrázek 19 – testovací kód nastavení.....	40
Obrázek 20 – kód pro nastavení shaderu pomocí barev .....	41
Obrázek 21 – kód pro nastavení shaderu pomocí obrázků .....	42
Obrázek 22 – ukázka špatného navolení obrázku.....	43
Obrázek 23 – ukázka správného navolení obrázku .....	43
Obrázek 24 – kód pro ovládání světla .....	44
Obrázek 25 – kód pro generování objektů.....	45
Obrázek 26 – testování velikosti.....	56
Obrázek 27 – testování šumu.....	57
Obrázek 28 – testování falloff mapy.....	57
Obrázek 29 – testování viditelnosti chunků.....	58
Obrázek 30 – testování volby grafiky.....	59
Obrázek 31 – testování rovinného terénu .....	59
Obrázek 32 – testování hladkého přechodu.....	60
Obrázek 33 – testování míchání.....	60

**SEZNAM TABULEK**

Tabulka 1. Výsledky měření rychlosti při velikosti terénu 10x10.....	39
Tabulka 2. Výsledky měření rychlosti při velikosti terénu 100x100.....	39
Tabulka 3. Výsledky měření rychlosti při velikosti terénu 1 000x1 000.....	39

## SEZNAM PŘÍLOH

Příloha P I: CD s elektronickou verzí bakalářské práce a s Assetem

## **PŘÍLOHA P I: CD S PRAKTICKOU ČÁSTÍ PRÁCE**

Toto CD obsahuje:

Bakalářskou práci

Asset

BP\_Mikeska-Mikulas.pdf

GeneraceTerenu.unitypackage