

Vývoj Unity assetu pro automatické generování bludišť

Bc. Mikuláš Mikeska

Diplomová práce
2025



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2024/2025

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Bc. Mikuláš Mikeska
Osobní číslo: A23320
Studijní program: N0613A140022 Informační technologie
Specializace: Softwarové inženýrství
Forma studia: Prezenční
Téma práce: Vývoj Unity assetu pro automatické generování bludišť
Téma práce anglicky: Development of a Unity Asset for Automatic Maze Generation

Zásady pro vypracování

- Vypracujte literární rešerši na zadané téma.
- Provedte stručnou analýzu dostupných nástrojů a algoritmů pro generování bludišť v Unity.
- Navrhněte Unity asset pro automatické generování bludišť.
- Implementujte navržený asset tak, aby byl snadno použitelný a přenositelný do jakéhokoliv 2D nebo 3D projektu v Unity.
- Demonstrujte funkčnost vytvořeného nástroje.
- Vhodně zdokumentujte vytvořený nástroj.

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

Seznam doporučené literatury:

1. BUCK, Jamis. *Mazes for programmers: code your own twisty little passages*. P1.0-July. Pragmatic bookshelf. Dallas, Texas: Pragmatic Programmers, [2015]. ISBN 978-1-68050-055-4.
2. Unity User Manual 2022.3 (LTS) [online]. San Francisco, USA: Unity Technologies, 2022 [cit. 2024-06-10]. Dostupné z: <https://docs.unity3d.com/Manual/index.html>.
3. SMED, Jouni a HAKONEN, Harri. *Algorithms and networking for computer games*. Chichester: John Wiley, c2006. ISBN 0-470-01812-7.
4. KOK, Benny. *Beginning Unity Editor Scripting: Create and Publish Your Game Tools*. New York, USA: Apress, 2021. ISBN 978-1-4842-7166-7.

Vedoucí diplomové práce: **Ing. Tomáš Vogeltanz, Ph.D.**
Ústav počítačových a komunikačních systémů

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



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

L.S.

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

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA DIPLOMOVÉ PRÁCE

Beru na vědomí, že

- odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení diplomové práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – diplomovou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- pokud je výstupem diplomové 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 diplomové práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Cílem této diplomové práce je návrh a implementace nástroje pro generování bludišť v herním enginu Unity, který umožňuje tvorbu různých typů bludišť pro počítačové hry. Nástroj podporuje generování bludišť s více patry a obsahuje editační režim pro manuální úpravy zdí a také minimapu pro orientaci hráče v prostředí. Teoretická část se věnuje herním enginům Unity, Unreal Engine a Godot, popisuje jejich rozdíly. Dále objasňuje rozdíl mezi labyrintem a bludištěm a popisuje vlastnosti perfektního bludiště. Součástí této části je také analýza různých algoritmů pro generování bludišť, včetně jejich výhod a nevýhod. Praktická část se zaměřuje na vývoj nástroje v Unity, testování výkonu algoritmů a porovnání jejich efektivity. Nástroj umožňuje generování bludišť náhodně nebo na základě obrázku, podporuje čtvercové i hexagonové mřížky, kontrolu propojenosti buněk a ověření řešitelnosti bludiště. Kromě toho obsahuje vlastní shader a využívá dynamické zobrazení bludiště pomocí chunků.

Klíčová slova: generování bludišť, Unity, algoritmy, editační režim

ABSTRACT

The aim of this thesis is to design and implement a maze generation tool within the Unity game engine that enables the creation of various types of mazes for computer games. The tool supports multi-level maze generation, includes an editing mode for manual wall adjustments, and also features a minimap to help the player navigate the environment. The theoretical part focuses on game engines such as Unity, Unreal Engine, and Godot, and describes their differences. It also explains the distinction between a labyrinth and a maze, and outlines the characteristics of a perfect maze. This section includes an analysis of various maze generation algorithms, along with their advantages and disadvantages. The practical part is dedicated to the development of the tool in Unity, including performance testing of the algorithms and comparison of their efficiency. The tool supports random maze generation or generation based on an image, and it accommodates both square and hexagonal grids. It also includes connectivity checks and solvability validation to ensure that the generated maze can always be completed. Additionally, the tool features a custom shader and utilizes dynamic maze rendering using chunks.

Keywords: maze generation, Unity, algorithms, edit mode

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

OBSAH

ÚVOD.....	13
I TEORETICKÁ ČÁST	14
1 HERNÍ ENGINE	15
1.1 UNITY	15
1.2 UNREAL ENGINE	16
1.3 GODOT	17
2 ZÁKLADNÍ POJMY	18
2.1 LABYRINT	18
2.2 BLUDIŠTĚ	19
3 VLASTNOSTI PERFEKTNÍHO BLUDIŠTĚ	20
3.1 UNIKÁTNÍ CESTA MEZI DVĚMA BODY	20
3.2 ABSENCE CYKLŮ	20
3.3 PROPOJENOST VŠECH OBLASTÍ	20
4 ALGORITMY PRO GENEROVÁNÍ BLUDIŠŤ	21
4.1 NÁHODNÉ GENEROVÁNÍ	21
4.2 BINARY TREE	21
4.3 SIDEWINDER.....	22
4.4 RECURSIVE BACKTRACKER.....	23
4.5 KRUSKALŮV ALGORITMUS	24
4.6 HUNT AND KILL	24
4.7 UPRAVENÝ HUNT AND KILL.....	25
5 PŘÍZPŮSOBNÍ EDITORU V UNITY.....	27
5.1 MODIFIKÁTORY PŘÍSTUPU V UNITY.....	27
5.1.1 Přehled přístupových úrovní	27
5.2 ATRIBUTY V UNITY A JEJICH VYUŽITÍ.....	27
5.2.1 Nejčastější atributy používané v Unity	28
5.3 SERIALIZACE TŘÍD A STRUKTUR V UNITY	30
6 VYTVÁŘENÍ VLASTNÍHO ATRIBUTU V UNITY	31
6.1 VYTVOŘENÍ VLASTNÍHO ATRIBUTU	31
6.2 VYTVOŘENÍ PROPERTYDRAWER PRO VLASTNÍ ATRIBUT	32
6.2.1 Další možnosti v metodě OnGUI ve třídě PropertyDrawer	32
7 DEFINICE ASSEMBLY	36
7.1 EDITOR ASSEMBLY	37

II	PRAKTICKÁ ČÁST	38
8	ANALÝZA PODOBNÝCH NÁSTROJŮ	39
8.1	MAZEGEN	39
8.2	MAZE GENERATOR	40
9	MĚŘENÍ ČASŮ GENEROVACÍCH ALGORITMŮ PRO BLUDIŠTĚ	41
10	VYTVOŘENÍ NÁSTROJE PRO GENEROVÁNÍ BLUDIŠTĚ	47
10.1	POŽADAVKY NA NÁSTROJ	47
10.2	VLASTNÍ ATRIBUTY	47
10.2.1	VisibleIf	48
10.2.2	LabeledVector2	48
10.2.3	Required	49
10.3	GENEROVÁNÍ BLUDIŠTĚ	50
10.4	KONTROLA PROPOJENOSTI BUNĚK	51
10.4.1	Definice směrů u hexagonové mřížky	51
10.5	GENEROVÁNÍ ZDÍ	52
10.5.1	Abstraktní třída pro generování	53
10.5.2	Čtvercové bludiště	55
10.5.3	Hexagonové bludiště	58
10.6	MAZE COLOR	59
10.6.1	Vytvoření Custom shaderu	59
10.6.2	Nastavení hodnot v shaderu za pomoci C#	60
10.6.3	Kód v shaderu pro čtvercové bludiště	61
10.6.4	Kód v shaderu pro hexagonové bludiště	62
10.7	MAZE VISIBLE CHUNK	64
10.8	CHUNK RENDERER	65
10.8.1	SquareChunkRenderer	65
10.8.2	HexagonChunkRenderer	65
10.9	MINIMAPA	66
10.9.1	Zobrazení buněk	66
10.9.2	Asynchronní vykreslování	66
10.10	EDITAČNÍ REŽIM	67
10.10.1	Generování editačního prostředí	67
10.10.2	Interakce s objekty	67
10.11	KONTROLA ŘEŠITELNOSTI	68
10.11.1	Zjištění cesty	68
10.11.2	Výsledné renderování	69
10.12	VYTVOŘENÍ ASSEMBLY DEFINICÍ	69
11	POPIS NÁSTROJE	71

11.1	MAZE SCALER.....	71
11.2	MAZE VISIBLE CHUNK	71
11.3	MAZE GENERATOR MANAGER	72
11.4	MAZE COLOR	73
11.5	SOLVABLE CHECKER.....	74
11.6	MINIMAP	74
12	TESTOVÁNÍ ASSETU	76
13	TESTOVÁNÍ VÝKONU ASSETU	82
13.1	PRVNÍ OBRÁZEK	82
13.2	DRUHÝ OBRÁZEK	83
13.3	TŘETÍ OBRÁZEK.....	84
14	EXPORT A IMPORT UNITY ASSETU.....	86
14.1	VYTVOŘENÍ BALÍČKU UNITYPACKAGE	86
14.2	IMPORT BALÍČKU UNITYPACKAGE DO PROJEKTU	86
14.3	PUBLIKOVÁNÍ ASSETU NA UNITY ASSET STORE	87
	ZÁVĚR	89
	SEZNAM POUŽITÉ LITERATURY.....	91
	SEZNAM OBRÁZKŮ	94
	SEZNAM TABULEK.....	96
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	97
	SEZNAM PŘÍLOH.....	98

ÚVOD

Cílem této diplomové práce je navrhnout a implementovat nástroj pro generování bludišť v herním enginu Unity, jenž usnadní tvorbu různých typů bludišť pro použití v počítačových hrách. Mezi hlavní vlastnosti nástroje patří podpora generování vícepatrových bludišť, což umožňuje vytvářet složitější struktury, a integrovaný editační režim, jenž dává uživatelům možnost manuálně upravovat zdi. Práce je rozdělena na dvě hlavní části: teoretickou a praktickou.

Teoretická část se zaměřuje na analýzu herních enginů, jako jsou Unity, Unreal Engine a Godot. Dále jsou vysvětleny rozdíly mezi labyrintem a bludištěm, a specifikované vlastnosti perfektního bludiště – jedinečná cesta mezi dvěma body, absence cyklů a propojenost všech oblastí. Následuje přehled různých algoritmů pro generování bludišť, přičemž se zaměřujeme na jejich výhody a nevýhody. Mezi popisované metody patří náhodné generování, Binary Tree, Sidewinder, Recursive Backtracker, Kruskalův algoritmus, Hunt and Kill a jeho modifikovaná verze.

Praktická část se věnuje návrhu a implementaci nástroje v Unity. Nejdříve je provedena analýza stávajících řešení, jako jsou MazeGen a Maze Generator, a poté je popsán samotný proces vývoje nástroje. Důležitou součástí práce je testování výkonu jednotlivých algoritmů a jejich vzájemné porovnání z hlediska efektivity. Měřeny jsou časy generování různých typů bludišť, přičemž je použita čtvercová mřížka bez zakázaných oblastí.

Nástroj umožňuje generování bludišť několika způsoby. První metodou je náhodné rozmístění buněk s 50 % pravděpodobností obsazení. Druhá metoda využívá obrázek, kde oblasti s alfa kanálem 0 označují prázdné buňky. Velikost obrázku lze měnit přímo v editoru, přičemž pod ním je zobrazená výsledná velikost bludiště. Bludiště může být generováno ve formě čtvercové nebo hexagonové mřížky. Po vygenerování se provádí kontrola propojenosti buněk, aby bylo zajištěno, že bludiště tvoří souvislou oblast.

Další funkcí nástroje je kontrola řešitelnosti, která ověřuje existenci průchozí cesty mezi definovanými body. Generování bludiště probíhá pomocí chunků, což umožňuje dynamické zobrazení pouze těch částí, které jsou v okolí hráče. Součástí nástroje je také minimapa, která zobrazuje pozici hráče a vizualizuje navštívené buňky. Pro její plynulé vykreslování je využito asynchronní generování pomocí coroutine. Mezi další funkcionality patří editační režim, který umožňuje uživatelům upravovat jednotlivé zdi.

I TEORETICKÁ ČÁST

1 HERNÍ ENGINE

Herní engine je specializovaný softwarový framework, který slouží pro vývoj videoher. Obsahuje knihovny a podpůrné programy, které usnadňují práci s grafikou, fyzikou, zvukem, umělou inteligencí a síťovou komunikací. [1]

Základní komponenty herního enginu jsou:

- **Rendering Engine** – Zajišťuje vykreslování 2D a 3D grafiky. Pomáhá vytvářet iluzi pohybu, hloubky a osvětlení. Mezi technologie, které se používají, patří OpenGL, DirectX a Vulkan.
- **Physics Engine** – Simuluje fyzikální chování objektů, kolize, gravitaci a interakce mezi nimi. Příklady: PhysX, Havok, Bullet.
- **Audio Engine** – Spravuje zvukové efekty a hudbu, synchronizuje je s herními událostmi a aplikuje různé zvukové efekty. Příklady: FMOD, Wwise, OpenAL.
- **Umělá inteligence (AI)** – Řídí chování NPC a dalších herních prvků. Umožňuje realistické reakce na akce hráče. Příklady: stavové automaty, stromové chování, neuronové sítě.
- **Networking** – Podporuje multiplayer, synchronizuje stav hry mezi hráči a spravuje síťovou komunikaci. Příklady: RakNet, Photon, enet.

Herní enginy mají významný vliv na herní průmysl a určují hranice možností při vývoji moderních her. [1]

1.1 Unity

Unity je multiplatformní herní engine určený pro vývoj 2D a 3D aplikací. Je založen na komponentovém systému, kde objekty ve scéně (GameObjects) získávají funkcionalitu prostřednictvím připojených komponent, jako jsou Transform, Rigidbody nebo MeshRenderer. Programování probíhá v jazyce C#, ale Unity umožňuje integraci jiných jazyků, jako například Rust nebo Python. Unity Editor umožňuje vizuální úpravy scén, import assetů a ladění kódu. Engine rovněž podporuje real-time multiplayer, síťové funkce a integraci s externími API. [2]

Jednou z největších výhod Unity je jeho široká multiplatformní podpora. Umožňuje vývoj her a aplikací pro Windows, macOS, Linux, iOS, Android, PlayStation, Xbox, Nintendo Switch, WebGL, VR/AR zařízení a Smart TV. Další výhodou je rozsáhlá komunita

vývojářů a přístup k Unity Asset Store, který nabízí širokou škálu předpřipravených assetů a nástrojů. Unity také podporuje vizuální skriptování prostřednictvím nástrojů jako Playmaker nebo Bolt, což usnadňuje tvorbu her i bez znalosti programování. Engine nabízí výkonné renderovací technologie, například HDRP (High Definition Render Pipeline), URP (Universal Render Pipeline) a Shader Graph. K dispozici jsou rovněž nástroje pro optimalizaci výkonu, jako multithreading, LOD (Level of Detail) a occlusion culling. Unity zahrnuje také analytické a monetizační funkce, například in-app nákupy, reklamy a předplatné. [2]

Mezi nevýhody Unity patří omezení některých funkcí na specifických platformách. Engine může mít také výkonové limity při složitějších fyzikálních simulacích, což může být problém u náročnějších projektů. U komerčního využití je nutné počítat s možnými licenčními poplatky. [2]

1.2 Unreal Engine

Unreal Engine (UE) je výkonný herní engine vyvinutý společností Epic Games. Využívá se pro vývoj videoher, simulací a vizualizací. Nabízí nástroje pro tvorbu 3D prostředí, animací, úpravu zvuku a profesionální renderování grafiky. Unreal Engine je zdarma, ale pokud komerční projekt vygeneruje více než 1 milion USD ročně, Epic Games si vezme 5 % zisku nad tuto částku. Produkty distribuované přes Epic Games Store jsou tohoto poplatku osvobozeny.[3]

Hlavní programovací jazyk je C++, což zajišťuje stabilní a efektivní správu paměti, ale může být náročnější pro začátečníky. Alternativně je k dispozici Blueprint, vizuální skriptovací systém, který umožňuje tvorbu herní logiky bez psaní kódu. Unreal Engine podporuje také Python pro automatizaci a některé skriptovací úlohy. [3] [4]

Mezi hlavní konkurenty Unreal Engine patří Unity, které je díky podpoře jazyka C# a přístupnějšímu rozhraní vhodnější pro začátečníky. Unreal Engine se vyniká především pokročilým 3D renderingem a je preferovanou volbou pro vývoj AAA her. Naopak Unity se často využívá pro mobilní hry, 2D projekty a AR/VR aplikace. [2] [3]

Unreal Engine Marketplace umožňuje nákup a prodej assetů, jako jsou 3D modely, animace, zvuky a kód. Epic Games si z prodeje bere 12 % provizi, takže tvůrci si ponechají 88 % zisku. Unreal Engine je zdarma dostupný pro školy a univerzity. [3]

1.3 Godot

Godot je open-source herní engine, který podporuje vývoj 2D a 3D her. Je známý svou flexibilitou, multiplatformní podporou a přístupností pro vývojáře všech úrovní. Godot obsahuje integrovaný editor s nástroji pro tvorbu herních scén, animací a skriptování. Podporuje vlastní skriptovací jazyk GDScript, který je podobný Pythonu, a také C# a C++. Engine je zcela zdarma a nevyžaduje žádné licenční poplatky. [5]

Mezi hlavní výhody Godotu patří jeho open-source povaha, která umožňuje vývojářům přizpůsobit engine jejich potřebám. Podporuje širokou škálu platforem, včetně Windows, macOS, Linux, iOS, Android, HTML5 a dalších. Godot je známý svým intuitivním a přehledným uživatelským rozhraním. Dále nabízí vestavěné nástroje pro animace, skriptování a správu scén. [5]

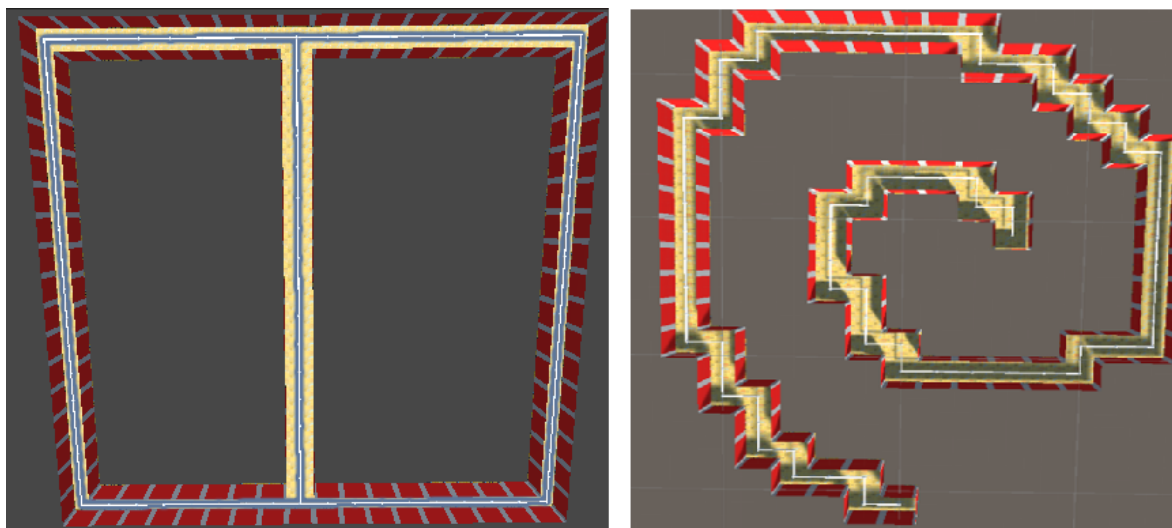
Nevýhodou Godotu je menší komunita ve srovnání s Unity nebo Unreal Engine, což může znamenat omezenou dostupnost zdrojů a tutoriálů. Ačkoli Godot podporuje 3D hry, jeho nástroje a funkce nejsou na takové úrovni jako u některých jiných enginů, což může být omezením pro složitější projekty. [6]

2 ZÁKLADNÍ POJMY

Tato kapitola se zaměří na objasnění základních pojmů, zejména na rozdíl mezi labyrintem a bludištěm.

2.1 Labyrint

Labyrint je struktura s jedinou cestou vedoucí od startu k cíli, která neumožňuje zabloudit. Existují však i specifické typy labyrintů, v nichž se cesta na začátku rozdvíjí, ale všechny větve nakonec směřují ke stejnému cíli. Toto rozvětvení může vzniknout buď hned na začátku, nebo v průběhu cesty, avšak vždy se musí v cíli spojit. [7]



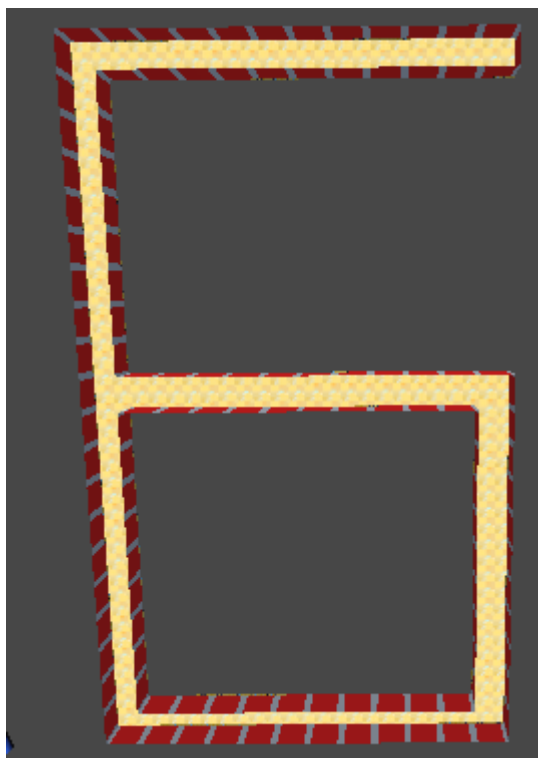
Obrázek 1 – ukázka labyrintů

Zdroj: vlastní tvorba za pomoci Unity

Na obrázku 1 jsou zobrazeny dvě struktury, které odpovídají definici labyrintu. Levá struktura obsahuje tři paralelní cesty vedoucí k tomu samému cíli, aniž by nabízela slepé uličky nebo možnost zabloudit. Pravá struktura má spirálovitý tvar, kde jediná možná cesta postupně směřuje ke středu. V obou případech neexistují odbočky vedoucí do nesprávných směrů, což znamená, že každý se nevyhnutelně dostane k cíli bez ohledu na to, kam se vydá. Díky těmto vlastnostem jde o labyrinty, nikoli bludiště. U levé struktury to samozřejmě platí, pokud uvažujeme start, kde se protínají tři cesty, a cíl, kde se protínají další tři cesty. Kdyby byl start nebo cíl zvolen jinde, jednalo by se o bludiště.

2.2 Bludiště

Bludiště je hlavolam tvořený větvenými cestami, přičemž alespoň jedna odbočka nevede zpět k cíli. Za bludiště se považuje i situace, kdy se cesta rozdvojí a později se opět spojuje, ale ne na samotném konci. Důležité je, že alespoň jednou je možné zabloudit. [7]



Obrázek 2 – ukázka bludiště

Zdroj: vlastní tvorba za pomoci Unity

Na obrázku 2 je zobrazeno jednoduché bludiště za předpokladu, že nevolíme start vpravo nahoře a cíl někde ve spodní polovině. Toto je považováno za bludiště, protože existuje alespoň jedno rozhodnutí, které nevede k dobrému řešení. I když je pravděpodobnost zabloudění malá, v tomto bludišti je stále možné se ztratit.

3 VLASTNOSTI PERFEKTNÍHO BLUDIŠTĚ

Tato kapitola se zaměří na vlastnosti, které by měla mít správně tzv „perfektně vygenerované bludiště“.

3.1 Unikátní cesta mezi dvěma body

V jakémkoli perfektním bludišti existuje mezi libovolnými dvěma body právě jedna unikátní cesta. To znamená, že při pohybu mezi dvěma zvolenými místy nelze volit mezi více trasami. [8]

3.2 Absence cyklů

Perfektní bludiště neobsahuje kruhové smyčky. Pokud by v něm existovaly cykly, umožnilo by to více než jedno možné řešení pro dosažení konkrétního cíle, čímž by byla porušena jeho základní struktura. [8] [9]

3.3 Propojenost všech oblastí

Každá buňka v perfektním bludišti je propojena se zbytkem struktury. Neexistují žádné izolované části, které by byly odříznuty od ostatních. Každý prvek bludiště tak tvoří součást jedné souvislé sítě průchodů. [8]

4 ALGORITMY PRO GENEROVÁNÍ BLUDIŠŤ

Tato kapitola se zaměří na některé algoritmy pro generování bludišť a jejich analýzu.

4.1 Náhodné generování

Algoritmus pro náhodné generování bludiště probíhá ve dvou základních fázích. Nejprve se vytvoří mřížka požadované velikosti, která představuje základní strukturu bludiště. Poté se jednotlivým buňkám této mřížky náhodně přiřazují hodnoty – u každé buňky se rozhoduje, zda bude představovat zeď, nebo zůstane průchozí. Tento proces řídí generátor náhodných čísel, který na základě stanovené pravděpodobnosti určuje, zda buňka bude obsahovat zeď. Pokud vygenerované číslo překročí stanovený práh, buňka zůstane průchozí, jinak se stane zdí. Nakonec je nutné zajistit, aby okraje bludiště byly tvořeny pevnými zdmi.

Alternativním přístupem je nejprve celé bludiště vyplnit zdmi a následně některé z nich odstranit. I zde je důležité správně nastavit okraje, aby zůstala zachována konzistence a bludiště bylo uzavřené. [8] [10]

Výhody náhodného generování bludiště

- Jednoduchá implementace – rozhodování o každé buňce je založeno pouze na generování náhodného čísla.
- Rychlost a efektivita výpočtu.
- Možnost využití při testování algoritmů pro hledání cest. [8] [9]

Nevýhody náhodného generování bludiště

- Bludiště nemusí být vždy řešitelné, protože mezi startovním a cílovým bodem nemusí existovat spojitá cesta.
- Výsledná struktura může být chaotická a obtížně čitelná.
- Mohou se vyskytovat nepřírozené vzory, jako příliš velké otevřené prostory nebo shluky zdí. [8] [9]

4.2 Binary Tree

Algoritmus Binary Tree je jednoduchý a efektivní způsob generování bludišť, který probíhá ve dvou hlavních fázích. Nejprve se vytvoří pravidelná mřížka, kde každá buňka je obklopena zdmi. Tento krok stanoví základní strukturu bludiště, na které se následně vytvoří

průchody. Ve druhé fázi algoritmus pro každou buňku rozhoduje, zda se vytvoří průchod severním nebo východním směrem. Toto rozhodnutí je založeno na generování náhodného čísla v rozsahu 0 až 1. Pokud je vygenerované číslo menší než 0,5, odstraní se severní zeď; pokud je větší nebo rovno 0,5, odstraní se východní zeď.

Specifická pravidla pro buňky na okrajích:

- Buňky na severním okraji vždy odstraní východní zeď.
- Buňky na východním okraji odstraní severní zeď.
- Buňka v severovýchodním rohu zůstává nezměněna. [11]

Výhody algoritmu Binary Tree:

- Snadná implementace a nízké paměťové nároky.
- Rychlost – má lineární časovou složitost $O(n)$.
- Vytváří perfektní bludiště.
- Možnost postupného generování, což je výhodné například pro hry, které mají velké mapy. [11] [13]

Nevýhody algoritmu Binary Tree:

- Algoritmus má bias – tendenci vytvářet dlouhé, nepřerušované chodby podél severní a východní hrany.
- Struktura bludiště je do jisté míry předvídatelná.
- Nelze použít pro tvorbu nepravidelných nebo specificky tvarovaných bludišť. [11] [13]

4.3 Sidewinder

Sidewinder algoritmus postupuje řádek po řádku zprava doleva. Vytváří „běhy“ – sekvence propojených buněk. Každá buňka je propojena s pravým sousedem (pokud není na pravém okraji). V náhodném okamžiku se vybere jedna buňka z běhu a propojí se směrem nahoru (pokud to není první řada). Po propojení směrem nahoru se běh resetuje a začíná nový. [11]

Výhody Sidewinder algoritmu

- Dobře vyvážené cesty.

- Rychlost – lineární časová složitost $O(n)$.
- Vytváří perfektní bludiště.
- Možnost postupného generování. [11] [13]

Nevýhody Sidewinder algoritmu

- Nerovnoměrné rozložení chodeb – bludiště obsahuje více horizontálních než vertikálních cest.
- Cesty podél severní hrany mohou být příliš jednoduché.
- Omezená flexibilita při generování specifických struktur. [11] [13]

4.4 Recursive Backtracker

Recursive Backtracker je algoritmus pro generování bludišť, který využívá metodu hloubkového prohledávání (Depth-First Search). Algoritmus začíná v náhodně vybrané buňce, kterou označí jako navštívenou, a pokračuje tím, že postupně vytváří cestu mezi buňkami. Vybere náhodného souseda, který ještě nebyl navštíven, a odstraní stěnu mezi aktuální buňkou a tímto sousedem. Poté se přesune na souseda a celý proces opakuje.

Pokud algoritmus narazí na buňku, která nemá žádného neprozkoumaného souseda (tedy „slepou uličku“), vrátí se zpět do předchozí buňky, která má ještě neprozkoumané sousedy. Tento proces pokračuje, dokud má v zásobníku nějaké buňky. [11]

Výhody Recursive Backtracker

- Generuje perfektní bludiště.
- Lze jej snadno modifikovat, např. přidáním hmotnosti k cestám.
- Umožňuje definování zakázaných oblastí. [11] [14]

Nevýhody Recursive Backtracker

- Vysoké paměťové nároky kvůli rekurzivnímu zásobníku.
- Vytváří dlouhé cesty a velké slepé uličky.
- Pomalejší než jednodušší alternativy. [14]

4.5 Kruskalův algoritmus

Kruskalův algoritmus je založen na hledání minimální kostry (Minimum Spanning Tree, MST) v grafu. Tento algoritmus postupně přidává stěny (nebo hrany) mezi buňkami, dokud není celé bludiště propojené. Algoritmus začíná tím, že každá buňka je samostatná komponenta (takzvaný ostrov). Všechny stěny mezi buňkami jsou uloženy do seznamu, který je následně náhodně zamíchán. Poté se pro každou náhodně vybranou stěnu zkontroluje, zda propojuje dvě různé komponenty – tedy buňky, které ještě nejsou propojené. Pokud ano, stěna se odstraní a buňky se spojí do jedné komponenty. Tento proces pokračuje, dokud všechny buňky nejsou propojené. [11]

Výhody Kruskalova algoritmu

- Rovnoměrně rozložené chodby.
- Perfektní bludiště bez předvídatelných vzorů.
- Vhodné pro generování v reálném čase. [11]

Nevýhody Kruskalova algoritmu

- Vyšší paměťové nároky.
- Náročná správa seznamu hran. [11]

4.6 Hunt and Kill

Hunt and Kill algoritmus začíná výběrem výchozí buňky, obvykle v jednom z rohů, a jejím postupným prozkoumáváním. Pokračuje výběrem sousedních neprozkoumaných buněk, dokud nenarazí na slepou uličku. Jakmile se dostane na konec cesty, ukončí aktuální postup a přechází do fáze hledání (hunt). V této fázi začne procházet od výchozí buňky, dokud nenajde neprozkoumanou buňku, která má alespoň jednoho souseda již zahrnutého do bludiště (navštívená buňky). Jakmile takovou buňku najde, propojí ji s prozkoumanou částí a algoritmus se vrací do fáze prozkoumávání (kill), pokračuje ve výběru sousedních buněk a znovu postupuje, dokud nenarazí na slepou uličku.

Tento proces se opakuje, dokud nejsou prozkoumány všechny buňky a bludiště není dokončeno. Hunt and Kill se podobá algoritmu Recursive Backtracker, ale liší se způsobem řešení slepých uliček – místo návratu k poslednímu rozhodovacímu bodu (jako u Recursive Backtracker) aktivně vyhledává novou neprozkoumanou buňku. Díky tomu Hunt and Kill

obvykle vytváří kratší chodby než Recursive Backtracker, ale o něco delší než Kruskalův algoritmus. [11]

Výhody Hunt and Kill algoritmu:

- Plná kontrola nad tvorbou bludiště.
- Generuje perfektní bludiště.
- Ideální cesty a slepé uličky.
- Možnost definovat zakázané oblasti. [15]

Nevýhody Hunt and Kill algoritmu:

- Pomalejší než některé jiné metody. [15]

4.7 Upravený Hunt and Kill

Upravený Hunt and Kill algoritmus generuje bludiště postupným propojením buněk v mřížce a odstraňováním zdí mezi nimi. Začíná v levém dolním rohu mřížky, kde označí první buňku jako navštívenou. Poté vybírá sousední neprozkoumané buňky a propojuje je s již navštívenými odstraněním zdí. Tento proces pokračuje, dokud algoritmus nenarazí na slepou uličku, poté přechází do fáze Hunt.

Ve fázi Hunt algoritmus prohledává dosud neprozkoumanou oblast a hledá buňky, na které by mohl navázat. Přitom se vyhýbá buňkám, které už byly označeny jako navštívené (tzn. neprohledává již prozkoumané sloupce a řádky). Pokud narazí na nenavštívenou buňku, která však nemá žádného souseda vhodného k napojení, začne prozkoumávat celé bludiště, dokud nenajde vhodnou buňku pro propojení. Jakmile takovou buňku najde, vrací se zpět do fáze Kill.

Pokud ve fázi Kill algoritmus opět narazí na slepou uličku, vrací se k předchozí nepropojené buňce a zkouší ji napojit. Pokud je to možné, pokračuje ve fázi Kill. Tento proces se opakuje, dokud nenajde vhodné spojení s již navštívenými buňkami.

Algoritmus končí, jakmile ve fázi Hunt nezůstávají žádné nenavštívené buňky.

Díky tomu, že algoritmus neprochází celé bludiště při každém hledání propojení, je rychlejší než tradiční Hunt and Kill. Celé bludiště je prohledáváno pouze tehdy, pokud algoritmus narazí na buňku, kterou nelze napojit na již navštívené buňky.

Výhody upraveného Hunt and Kill

- Nízká paměťová náročnost.
- Rychlejší než klasická verze.
- Možnost zakázat oblasti.
- Generuje perfektní bludiště

Nevýhody upraveného Hunt and Kill

- Složitější na pochopení.
- Oproti klasické verzi, pokud jsou dva a více ostrovů tento algoritmus může uvíznout na ostrově, protože se nikdy nenapojí. Proto je důležité zkontrolovat, zda jsou buňky tvořeny jako jeden ostrov.

5 PŘÍZPŮSOBENÍ EDITORU V UNITY

Tato kapitola se zabývá možnostmi úpravy editoru Unity pomocí přizpůsobitelných prvků.

5.1 Modifikátory přístupu v Unity

Modifikátory přístupu určují, kdo a odkud může s proměnnou nebo metodou pracovat. Unity umožňuje propojení těchto modifikátorů s editorem, čímž ovlivňuje viditelnost hodnot v Inspectoru.

5.1.1 Přehled přístupových úrovní

V Unity jsou k dispozici následující modifikátory přístupu:

- **public** – Proměnná nebo metoda je přístupná z jakéhokoliv jiného skriptu a zároveň viditelná v Inspectoru, kde lze její hodnotu snadno měnit.
- **private** – Přístup je omezen pouze na třídu, ve které je proměnná deklarována. Pokud není uveden jiný modifikátor, automaticky se nastaví **private**. V Inspectoru se soukromé proměnné standardně nezobrazují.
- **protected** – Umožňuje přístup uvnitř třídy a jejích děděných potomků.
- **internal** – Proměnná nebo metoda je přístupná pouze v rámci stejného sestavení (assembly).

Kromě základních úrovní existují i jejich kombinace:

- **private protected** – Přístupná pouze v rámci stejného assembly a jen pro dědické třídy.
- **protected internal** – Viditelná uvnitř stejného assembly, ale přístupná i pro dědické třídy mimo něj. [16] [17]

5.2 Atributy v Unity a jejich využití

Atributy umožňují rozšířit vlastnosti proměnných a metod v Unity, ovlivňují jejich zobrazování v editoru a přidávají dodatečné funkce. Zapisují se v hranatých závorkách před deklarovaným prvkem. Pokud se použije více atributů, lze je oddělit čárkou [Atribut1, Atribut2] nebo zapsat samostatně [Atribut1] [Atribut2]. [18] [19]

5.2.1 Nejčastější atributy používané v Unity

SerializeField

Umožňuje zobrazit soukromou proměnnou v Inspectoru, aniž by bylo nutné měnit její přístupový modifikátor na public. Proměnná musí být serializovatelná, což znamená, že musí patřit mezi následující typy:

- základní datové typy (int, float, bool, string),
- některé struktury (Vector3, Quaternion, Color, Rect, Bounds, AnimationCurve, Gradient, Matrix4x4),
- pole a List<T> (pokud T je serializovatelný),
- vlastní třídy nebo struktury,
- odkazy na Unity objekty (GameObject, Transform, MonoBehaviour, Material, Texture, Sprite).

Unity nserializuje vlastnosti (get/set).

HideInInspector

Skryje proměnnou v Inspectoru, i když je deklarována jako public nebo má atribut [SerializeField]. Proměnná tedy není v editoru viditelná ani upravitelná.

Range(min, max)

Omezuje hodnotu proměnné v Inspectoru na zadaný rozsah. Funguje pro celočíselné i desetinné hodnoty a zobrazí posuvník pro nastavení hodnoty.

Header(„Název sekce“)

Přidává nadpis nad proměnnou nebo skupinu proměnných v Inspectoru. Pomáhá organizovat proměnné do logických bloků, což usnadňuje jejich správu v komplexních skriptech.

TextArea(min, max)

Používá se pro víceřádková textová pole. Definuje minimální a maximální počet řádků, které budou v Inspectoru viditelné. Užitečné pro delší texty jako popisy, poznámky nebo JSON data.

Tooltip(„Vysvětlení“)

Přidává krátký textový popis, který se zobrazí při najetí myši na proměnnou v Inspectoru. Pomáhá s orientací v kódu a usnadňuje pochopení účelu proměnné.

ContextMenu(„Název v menu“)

Umožňuje spouštět metody přímo z Inspectoru přes kontextové menu. Užitečné pro testování funkcí nebo aktivaci metod bez nutnosti spustit hru.

ExecuteInEditMode

Umožňuje provádět určité metody (např. Update() nebo FixedUpdate()) už během režimu editace, tedy bez nutnosti spustit hru. Vhodné pro akce jako úprava pozic objektů nebo aktualizace parametrů přímo v editoru. [18] [19] [20] [29]

```
[HideInInspector]
public int hiddenVariable;
[Range(0, 10)]
public float speed;
[Header("Movement Settings")]
public float speed;
[Header("Rotation Settings")]
public float rotationSpeed;
[TextArea(3, 10)]
public string description;
[Tooltip("The speed of the object movement")]
public float speed;
[ContextMenu("Reset Position")]
void ResetPosition()
{
    transform.position = Vector3.zero;
}
[ExecuteInEditMode]
void Update()
{
    transform.Rotate(Vector3.up * Time.deltaTime);
}
```

Obrázek 3 – Kód ukázka atributů

Zdroj: vlastní tvorba dle [18] [19] [20]

Na obrázku 3 je ukázka kódu, který ilustruje využití těchto atributů. Atribut HideInInspector skryje proměnnou hiddenVariable, přestože je veřejná. Atribut Range(0, 10) přidává posuvník pro proměnnou speed, který omezuje hodnoty na rozsah mezi 0 a 10. Pomocí atributů Header(„Movement Settings“) a Header(„Rotation Settings“) jsou

proměnné jako speed a rotationSpeed organizovány do sekcí v editoru. Atribut TextArea(3, 10) nastavuje viditelnost 3 až 10 řádků pro proměnnou description, zatímco Tooltip(„The speed of the object movement“) poskytuje nápovědu pro proměnnou speed. Atribut ContextMenu(„Reset Position“) umožňuje resetování pozice objektu metodou ResetPosition(). Nakonec ExecuteInEditMode umožňuje objektu rotovat i během editace.

5.3 Serializace tříd a struktur v Unity

V Unity lze serializovat vlastní datové typy, což umožňuje jejich zobrazení a úpravu v Inspectoru. K tomu slouží atribut [Serializable], který označuje, že daný objekt lze převést na datový formát vhodný pro uložení nebo zobrazení.

Serializace tříd

Pokud má být instance vlastní třídy viditelná v Inspectoru, musí se před její definicí přidat atribut [Serializable]. Tento atribut zajistí serializaci všech veřejných proměnných a polí třídy, což umožní jejich zobrazení a editaci v editoru Unity. [21]

Po přidání tohoto atributu lze tuto třídu použít v jiném skriptu, kde budou její proměnné přístupné v Inspectoru. Například pokud definujeme třídu s proměnnými characterName, health a attackPower, zobrazí se v editoru a lze je tam upravit. [21]

Serializace struktur

Stejným způsobem lze serializovat i struktury, které se od tříd liší tím, že se chovají jako hodnotové typy. Zatímco třídy uchovávají v zásobníku pouze referenci a data jsou uložena na haldě, struktury je ukládají přímo v zásobníku, což může pozitivně ovlivnit výkon a způsob jejich využití.

Například struktura EnemyAttributes bude fungovat obdobně jako třída, ale její data budou přímo uložena v paměti na zásobníku. [21] [22]

6 VYTVÁŘENÍ VLASTNÍHO ATRIBUTU V UNITY

Unity poskytuje možnost vytvářet vlastní atributy, které mohou usnadnit správu a vizualizaci dat v editoru. Díky této možnosti mohou vývojáři výrazně vylepšit uživatelské rozhraní editoru a zpříjemnit práci s daty přímo v Unity.

Vlastní atributy umožňují:

- Změnit způsob, jakým se proměnné zobrazují v Inspectoru – například zvýraznit důležité hodnoty, upravit jejich barvu nebo vytvořit vlastní rozložení.
- Omezit a validovat vstupy – zajistit, že uživatel zadává hodnoty pouze v požadovaném rozsahu nebo specifickém formátu.
- Rozšířit funkcionalitu editoru – například přidáním tlačítek nebo interaktivních prvků, které usnadní práci se skripty a komponentami [23] [24] [25]

6.1 Vytvoření vlastního atributu

V Unity lze vlastní atributy vytvářet definováním nové třídy, která dědí z `PropertyAttribute`. Tento přístup umožňuje přizpůsobit vzhled a chování proměnných, na které se atribut aplikuje.

Například následující kód ukazuje, jak vytvořit atribut `HighlightAttribute`, který zvýrazní proměnnou v Inspectoru pomocí definovaných RGB hodnot:

```
public class HighlightAttribute : PropertyAttribute
{
    public Color color;

    public HighlightAttribute(float r, float g, float b)
    {
        color = new Color(r, g, b);
    }
}
```

Obrázek 4 – Ukázka kódu vlastního atributu pro zvýraznění

Zdroj: vlastní tvorba dle [23] [24] [25]

Na obrázku 4 je kód, ve kterém konstruktor atributu přijímá tři hodnoty (R, G, B), jež určují barvu zvýraznění. Tato uložená hodnota barvy se následně využívá v `PropertyDraweru` k vizuální úpravě zobrazení proměnné v Inspectoru.

6.2 Vytvoření PropertyDrawer pro vlastní atribut

Aby se mohl atribut správně vykreslit v Unity Inspectoru, je nutné vytvořit třídu, která dědí z PropertyDrawer. Tato třída definuje, jak se proměnná zobrazí v editoru, a umožňuje její plnou vizuální přizpůsobitelnost.

```
[CustomPropertyDrawer(typeof(HighlightAttribute))]  
public class HighlightDrawer : PropertyDrawer  
{  
    public override void OnGUI(Rect position, SerializedProperty property, GUIContent  
label)  
    {  
        HighlightAttribute highlight = (HighlightAttribute)attribute;  
        Color prevColor = GUI.color;  
  
        GUI.color = highlight.color;  
        EditorGUI.PropertyField(position, property, label);  
        GUI.color = prevColor;  
    }  
  
    public override float GetPropertyHeight(SerializedProperty property, GUIContent  
label)  
    {  
        return EditorGUI.GetPropertyHeight(property, label) * 2;  
    }  
}
```

Obrázek 5 – Ukázka kódu PropertyDrawer pro zvýraznění

Zdroj: vlastní tvorba dle [23] [24] [25]

Na obrázku 5 je kód, který ilustruje způsob, jakým PropertyDrawer pracuje se zadaným atributem. Nejprve se změní barva pozadí, čímž se vizuálně odliší zvýrazněná proměnná. Poté se vykreslí samotné pole v Inspectoru s novým nastavením. Po dokončení vykreslování se barva pozadí obnoví na výchozí hodnotu, aby další prvky v editoru nebyly ovlivněny touto změnou. Kromě toho přepisuje metodu GetPropertyHeight, která určuje výšku, jakou by měl mít prvek v Inspectoru. V tomto případě je výška nastavena na dvojnásobek standardní hodnoty, což znamená, že prvek zabere v Inspectoru dvojnásobný prostor.

6.2.1 Další možnosti v metodě OnGUI ve třídě PropertyDrawer

Metoda OnGUI je zásadní pro definování vzhledu a interakce s jednotlivými proměnnými. Kromě změny pozadí, jak bylo ukázáno výše, lze také přidávat různé vstupy, jako jsou posuvníky, textová pole nebo tlačítka.

Změna barvy pozadí

Pomocí OnGUI lze přizpůsobit pozadí pro každý prvek v Inspectoru a tím zvýraznit důležité hodnoty nebo poskytnout vizuální feedback při změně hodnoty proměnné. Tato možnost je užitečná při validaci vstupů nebo při signalizování uživatelských chyb. [17] [18] [19]

```
public override void OnGUI(Rect position, SerializedProperty property, GUIContent label)
{
    GUI.color = Color.red; |
    EditorGUI.PropertyField(position, property, label);
}
```

Obrázek 6 – Kód pro změnu zvýraznění prvku v Inspectoru

Zdroj: vlastní tvorba dle [23] [24] [25]

Na obrázku 6 je kód, který definuje metodu OnGUI pro vlastní PropertyDrawer v Unity, která přizpůsobuje vzhled vlastností v Inspectoru. Nejprve nastaví červenou barvu (GUI.color = Color.red;), čímž ovlivní vzhled následujících vykreslovacích prvků. Poté vykreslí standardní vstupní pole (EditorGUI.PropertyField(position, property, label;)), které umožňuje uživateli upravit hodnotu proměnné.

Vykreslování vlastních vstupních polí

PropertyDrawer umožňuje do editoru přidat vlastní ovládací prvky, jako jsou posuvníky, textová pole nebo interaktivní tlačítka. Tento přístup výrazně usnadňuje manipulaci s hodnotami, zejména u komplexnějších datových typů, jako jsou struktury nebo seznamy. [23] [24] [25]

```
public override void OnGUI(Rect position, SerializedProperty property, GUIContent label)
{
    float value = property.floatValue;
    value = EditorGUI.Slider(position, label, value, 0f, 50f);
    property.floatValue = value;
}
```

Obrázek 7 – Kód pro přidání posuvníku v Inspectoru

Zdroj: vlastní tvorba dle [23] [24] [25]

Na obrázku 7 je kód, který přizpůsobuje vykreslení vlastnosti v Inspectoru pomocí vlastního PropertyDraweru, ale nevykresluje standardní vstupní pole (EditorGUI.PropertyField), takže se původní hodnota proměnné v základním formátu vůbec nezobrazí. Místo toho získává aktuální hodnotu proměnné (property.floatValue), poté ji

upravuje pomocí `EditorGUI.Slider`, což znamená, že na proměnnou aplikuje pouze posuvník s rozsahem od 0 do 50. Nakonec se aktualizovaná hodnota uloží zpět do vlastnosti (`property.floatValue = value;`), což zajistí, že změna bude správně serializována. Díky tomu lze hodnotu upravovat pomocí posuvníku přímo v Inspectoru.

Zobrazení textů nebo popisků

Další užitečnou funkcí je možnost přidat k proměnným popisky nebo tooltipsy, které poskytují uživateli dodatečné informace. [23] [24] [25]

```
public override void OnGUI(Rect position, SerializedProperty property, GUIContent label)
{
    EditorGUI.LabelField(position, "Toto je popis");
}
```

Obrázek 8 – Kód pro přidání popisu v Inspectoru

Zdroj: vlastní tvorba dle [17] [18] [19]

Na obrázku 8 je kód, který přizpůsobuje vykreslení vlastnosti v Inspectoru pomocí vlastního `PropertyDraweru`, ale nezobrazuje původní hodnotu proměnné. Místo toho využívá `EditorGUI.LabelField`, který jednoduše zobrazí statický text „Toto je popis“ na dané pozici. Tento kód tedy slouží pouze k vykreslení popisku a neponechává žádný prostor pro interakci. V Inspectoru nejde zjistit, jaký byl předchozí prvek, protože původní prvek je zcela nahrazen textovým popisem.

Kombinování více prvků

Kombinování několika vizuálních úprav v rámci jednoho `PropertyDraweru` může být efektivní při vytváření složitějšího uživatelského rozhraní. Například je možné vykreslit hodnotu proměnné, přidat textový popis a tlačítko pro další akce. Je však důležité zajistit, aby se prvky nepřekrývaly. [24] [25]

```
...  
  
    public override void OnGUI(Rect position, SerializedProperty property, GUIContent  
label)  
    {  
        float propertyHeight = EditorGUI.GetPropertyHeight(property, label);  
  
        Rect propertyRect = new Rect(position.x, position.y, position.width,  
propertyHeight);  
        EditorGUI.PropertyField(propertyRect, property, label);  
  
        Rect labelRect = new Rect(position.x, position.y + propertyHeight + 5,  
position.width, 20);  
        EditorGUI.LabelField(labelRect, "Tento prvek je modifikován.");  
  
        Rect buttonRect = new Rect(position.x, position.y + propertyHeight + 25,  
position.width, 20);  
        if (GUI.Button(buttonRect, "Klikni"))  
        {  
            Debug.Log("Tlačítko bylo stisknuto!");  
        }  
    }  
  
    public override float GetPropertyHeight(SerializedProperty property, GUIContent  
label)  
    {  
        float propertyHeight = EditorGUI.GetPropertyHeight(property, label);  
        return propertyHeight * 4;  
    }  
  
...
```

Obrázek 9 – Ukázka kódu pro přidání více prvků v Inspectoru

Zdroj: vlastní tvorba dle [24] [25]

Na obrázku 9 je kód, který upravuje vykreslení vlastnosti v Inspectoru pomocí vlastního PropertyDraweru a přidává k ní další prvky. Nejprve se pomocí EditorGUI.GetPropertyHeight zjistí výška standardního vstupního pole, aby se správně určila jeho pozice. Poté se vlastnost vykreslí běžným způsobem pomocí EditorGUI.PropertyField. Pod ní se přidá popisek „Tento prvek je modifikován.“, který je umístěn o několik pixelů níže. Nakonec se přidá tlačítko s nápisem „Klikni“, které po stisknutí vypíše do konzole zprávu „Tlačítko bylo stisknuto!“. Metoda GetPropertyHeight vrací čtyřnásobek výšky vlastnosti, což zajišťuje dostatek místa pro všechny přidané prvky v Inspectoru.

7 DEFINICE ASSEMBLY

V Unity představuje assembly definice (.asmdef) nástroj, který umožňuje organizovat kód do samostatných kompilovaných bloků. Každé assembly je samostatnou jednotkou kódu, která se zkompile do jedné dynamické knihovny (DLL). Tento přístup k organizaci kódu je užitečný ve větších projektech, kde pomáhá zrychlit kompilaci, zvýšit modularitu a usnadnit správu závislostí mezi různými částmi aplikace.

Assembly definice vznikají pro specifické složky v projektu, což určuje, jaké soubory budou součástí daného assembly. Například lze vytvořit samostatná assembly pro herní mechaniky, nástroje pro vývojáře nebo pro systémy uživatelského rozhraní.

Výhody použití assembly definice

Jedním z hlavních přínosů assembly definic je rychlejší kompilace. Unity kompiluje pouze kód, který se změnil, a to v rámci definovaných assembly. Změny v jednom assembly neovlivní kompilaci ostatních assembly, což zrychluje celý proces. [26]

Další výhodou je modularita. Rozdělení projektu na různé moduly s vlastní logikou a nezávislými závislostmi podporuje čistou architekturu a lepší oddělení zodpovědností. Různé části kódu, jako jsou UI, herní mechaniky nebo síťová komunikace, mohou být spravovány a testovány samostatně. [26] [27]

S assembly definicemi je také snadnější správa závislostí. Lze efektivně definovat vztahy mezi assembly a kontrolovat, jaké knihovny a skripty jsou v různých částech projektu použity. [26]

Dalším přínosem je lepší organizace kódu. Díky assembly definicím je projekt přehlednější a lépe strukturovaný. Snadno lze označit části projektu jako samostatné moduly nebo balíčky, což je výhodné při týmové práci nebo při růstu projektu. [27]

Nevýhody použití Assembly Definition

Navzdory výhodám může být konfigurace assembly definic složitější. Nesprávné nastavení může vést k problémům, jako jsou chyby při kompilaci nebo špatné odkazování mezi assembly. [20]

Dalším potenciálním problémem jsou nesprávně nastavené závislosti. Pokud kód jednoho assembly odkazuje na jiné, které zároveň odkazuje zpět, vzniká zacyklení, což znemožní kompilaci. [20]

Použití assembly definice v konkrétních scénářích

Assembly definice se vyplatí využít v rozsáhlých projektech, kde se kód často mění, protože mohou výrazně zkrátit dobu kompilace. Jsou také užitečné při týmové spolupráci, kde pomáhají s řízením závislostí mezi moduly.

Další situací, kde se jejich použití hodí, je modularizace kódu. Pokud je potřeba rozdělit projekt na logické celky, které lze nezávisle testovat a udržovat, assembly definice výrazně přispívají k lepší organizaci a modularitě kódu. [20] [26]

7.1 Editor Assembly

V Unity je důležité určit, zda bude assembly určeno pro editor, nebo zda se bude součástí finálního buildu aplikace. Editor Assembly znamená, že dané assembly bude dostupné pouze v Unity editoru a nebude zahrnuto do produkční verze hry.

Jak definovat Editor Assembly

Pokud je assembly určeno pouze pro editor, musí být umístěno do složky Editor. Při vytváření nebo úpravě souboru .asmdef je třeba v Unity Inspectoru zaškrtnout volbu „Editor“ pod položkou „Include Platforms“ (pokud tato možnost není dostupná, je nutné upravit soubor .asmdef a přidat „Editor“ do sekce Include Platforms). [20] [27]

Kdy použít Editor Assembly

Editor Assembly je vhodné použít u skriptů, které mají fungovat pouze v editoru. Typickým příkladem jsou nástroje pro tvorbu editorových oken, skripty pro vizualizaci dat. Tento přístup zajistí, že tyto skripty nebudou zahrnuty do finální produkční verze hry, což zlepší výkon a sníží velikost výsledného buildu. [20]

Kdy Editor Assembly nepoužívat

Pokud je kód určen pro produkční verzi hry, neměl by být definován jako Editor Assembly. Takový kód by měl být součástí běžného assembly, které bude zahrnuto do finálního buildu hry. [20]

II PRAKTICKÁ ČÁST

8 ANALÝZA PODOBNÝCH NÁSTROJŮ

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

8.1 MazeGen

- Autor: Goldor
- Odkaz: <https://assetstore.unity.com/packages/tools/level-design/mazegen-226572>
- Cena: Zdarma

Základní charakteristika

MazeGen v Inspectoru nabízí jednoduché nastavení velikosti generovaného bludiště, přičemž každá buňka má tvar čtverce. Nástroj je postaven na Scriptable Objectu, což umožňuje generování bludiště v oddělených vláknech. Díky tomu je možné pokračovat v práci, i když probíhá výpočet samotného bludiště.

Výhody

Mezi hlavní výhody Mazegenu patří parametr Seed, který zajišťuje konzistenci výsledků při generování bludiště. Dalšími výhodami jsou vysoká rychlost generování a možnost nastavit různé parametry, jako je velikost bludiště, Seed a velikost jednotlivých chunků. Nástroj je dobře dokumentován na GitHubu. Všechny buňky v generovaném bludišti jsou navštívitelné, což zaručuje, že bludiště bude plně průchodné. Implementovaný pohyb kamery dále zjednodušuje navigaci při testování.

Nevýhody

Mezi hlavní nevýhody patří, že generované bludiště nemá vlastnosti perfektního bludiště. Kromě toho je nutné vybrat 7 prefabů pro vytvoření bludiště. Výsledné bludiště často postrádá okraje, což může způsobit, že hráč může spadnout mimo něj.

8.2 Maze Generator

- Autor: styanton
- Odkaz: <https://assetstore.unity.com/packages/tools/modeling/maze-generator-38689>
- Cena: Zdarma

Základní charakteristika

Maze Generator umožňuje generování bludiště pomocí různých algoritmů a úpravu jeho struktury prostřednictvím parametrů, jako jsou rozměry, mezery mezi buňkami, nebo přidání pilířů a cílového objektu. Implementována je základní logika pro pohyb hráče a sbírání mincí.

Výhody

Maze Generator podporuje více algoritmů, což umožňuje uživatelům vybrat z různých metod generování bludišť, jako jsou rekurzivní zpětný průchod, stromové metody nebo dělení. Všechny tyto algoritmy generují vlastnosti perfektního bludiště. Díky přizpůsobitelnosti je možné nastavit rozměry bludiště, šířku a výšku buněk, přidávat mezery nebo používat vlastní prefaby. Generace poskytuje možnost použití vlastního Seedu pro zajištění stejného výsledného bludiště při opakovaném generování. Navíc skript automaticky kromě generování dat bludiště také umístí herní objekt do scény, konkrétně mince.

Nevýhody

Generování větších bludišť může být výkonnostně náročné. Dalším omezením je nemožnost přesně určit počet mincí, které se ve scéně objeví.

9 MĚŘENÍ ČASŮ GENEROVACÍCH ALGORITMŮ PRO BLUDIŠTĚ

Cílem této kapitoly je změřit a porovnat rychlost jednotlivých algoritmů popsaných v kapitole 4 při generování bludiště.

Měření byla provedena na zařízení Lenovo Legion 5 (notebook) s následující hardwarovou konfigurací:

- Procesor: Intel Core i5-10300H, 4 jádra, 2.50 GHz
- Operační paměť: 16 GB DDR4 2933 MT/s (2× SODIMM)
- Grafická karta: NVIDIA GeForce RTX 2060
- Úložiště: SSD TS1TMTE110S
- Operační systém: Windows 11, verze 24H2

Pro účely reprodukovatelnosti výsledků jsou zdrojové kódy algoritmů přiloženy jako soubor `AlgorithmusExample.unitypackage` v příloze této práce.

Tabulka 1 – Výsledky při 10 měření a velikosti 10x10

Algoritmus	Průměrný čas (ms)	Minimální čas (ms)	Maximální čas (ms)
Random	0,0435	0,0069	0,3676
Binary Tree	0,02311	0,0012	0,2073
Sidewinder	0,05858	0,0064	0,5246
Recursive Backtrack	0,23882	0,062	1,798
Kruskal	0,32434	0,0593	1,897
Hunt & Kill	0,28839	0,0463	2,4369
Upravený Hunt & Kill	0,26017	0,0632	1,9694

Zdroj: vlastní tvorba za pomoci Unity

Tabulka 1 ukazuje výsledky měření pro bludiště o velikosti 10×10. Nejrychlejším algoritmem byl Binary Tree, který dosáhl nejnižšího průměrného i minimálního času. Hned

za ním následoval Random, který měl o něco vyšší hodnoty, ale stále vykazoval velmi rychlou generaci bludiště.

Naopak, algoritmy Recursive Backtrack, Kruskal a Hunt & Kill se ukázaly jako pomalejší, přičemž Hunt & Kill a jeho upravená verze měly vyšší maximální časy než ostatní metody.

Tabulka 2 – Výsledky při 10 měření a velikosti 20x20

Algoritmus	Průměrný čas (ms)	Minimální čas (ms)	Maximální čas (ms)
Random	0,03101	0,027	0,0433
Binary Tree	0,00676	0,0061	0,0082
Sidewinder	0,05045	0,0456	0,0576
Recursive Backtrack	0,25426	0,2416	0,2816
Kruskal	1,14984	0,3969	2,2154
Hunt & Kill	0,38028	0,3662	0,3997
Upravený Hunt & Kill	0,27212	0,2518	0,3089

Zdroj: vlastní tvorba za pomoci Unity

Tabulka 2 prezentuje výsledky měření pro větší bludiště 20×20.

Nejrychlejším algoritmem zůstává Binary Tree, který vykazuje nejnižší výpočetní časy.

Algoritmy Recursive Backtrack, Kruskal a Hunt & Kill mají delší výpočetní časy. Mezi těmito metodami vykazuje Kruskal nejvyšší maximální čas.

Upravená verze Hunt & Kill je rychlejší než původní varianta, ale stále není tak efektivní jako Binary Tree nebo Random.

Tabulka 3 – Výsledky při 10 měření a velikosti 50x50

Algoritmus	Průměrný čas (ms)	Minimální čas (ms)	Maximální čas (ms)
Random	0,48043	0,1644	3,2361
Binary Tree	0,02433	0,019	0,0339
Sidewinder	0,19804	0,1567	0,2175
Recursive Backtrack	2,51146	1,4907	7,026
Kruskal	5,19776	2,1203	8,5202
Hunt & Kill	3,25276	2,4246	6,91
Upravený Hunt & Kill	2,66743	1,583	6,4972

Zdroj: vlastní tvorba za pomoci Unity

Tabulka 3 ukazuje výsledky pro velikost bludiště 50×50. S rostoucí velikostí bludiště je patrné, že všechny algoritmy vyžadují delší dobu výpočtu.

Nejrychlejším algoritmem stále zůstává Binary Tree, který poskytuje velmi nízké časy. Sidewinder si také zachovává relativně stabilní výkon, ale jeho časy jsou téměř desetkrát vyšší než u Binary Tree.

Naopak, algoritmy Recursive Backtrack, Kruskal a Hunt & Kill se ukazují jako výrazně pomalejší. Kruskal zde již dosahuje průměrného času nad 5 ms a jeho maximální čas se blíží 9 ms. Hunt & Kill a jeho upravená verze mají vysoké maximální hodnoty.

Tabulka 4 – Výsledky při 10 měření a velikosti 100x100

Algoritmus	Průměrný čas (ms)	Minimální čas (ms)	Maximální čas (ms)
Binary Tree	0,43359	0,0776	3,1518
Sidewinder	0,93222	0,6318	1,1708
Random	0,72777	0,6661	0,941
Recursive Backtrack	8,29561	6,0472	12,7329

Upravený Hunt & Kill	9,426539	6,3566	15,7445
Kruskal	29,9302	14,1187	47,1395
Hunt & Kill	31,51138	26,8801	43,9605

Zdroj: vlastní tvorba za pomoci Unity

Tabulka 4 ukazuje výrazný nárůst výpočetních časů u všech algoritmů při velikosti bludiště 100×100.

Algoritmus Binary Tree si stále udržuje dobrý výkon s relativně nízkým průměrným časem. Sidewinder má podobně stabilní výkon, ale je přibližně dvakrát pomalejší než Binary Tree.

U složitějších algoritmů jako Recursive Backtrack, Kruskal a Hunt & Kill dochází k výraznému prodloužení výpočetních časů. Kruskal dosahuje hodnot přes 47 ms, což je značný rozdíl oproti Binary Tree. Hunt & Kill je dokonce ještě pomalejší a jeho maximální časy přesahují 43 ms.

Tabulka 5 – Výsledky při 10 měření a velikosti 200x200

Algoritmus	Průměrný čas (ms)	Minimální čas (ms)	Maximální čas (ms)
Random	3,52477	2,646	6,2714
Binary Tree	0,99779	0,2955	3,5622
Sidewinder	2,5309	2,4889	2,6853
Recursive Backtrack	32,68461	27,7506	42,9072
Kruskal	133,8412	66,3492	233,1104
Hunt & Kill	378,1632	370,7074	384,5074
Upravený Hunt & Kill	33,01399	28,7633	37,2529

Zdroj: vlastní tvorba za pomoci Unity

Tabulka 5 ukazuje výrazné zpomalení složitějších algoritmů. Binary Tree zůstává nejrychlejší, Sidewinder je mírně pomalejší, ale stabilní. Naopak Recursive Backtrack,

Kruskal a Hunt & Kill se výrazně zpomalují – Kruskal dosahuje 233 ms, Hunt & Kill až 378 ms. Upravená verze Hunt & Kill je o něco rychlejší.

Tabulka 6 – Výsledky při 10 měření a velikosti 300x300

Algoritmus	Průměrný čas (ms)	Minimální čas (ms)	Maximální čas (ms)
Random	6,61097	5,8868	9,1857
Binary Tree	1,44857	0,6354	3,7074
Sidewinder	6,51925	5,6175	9,6921
Recursive Backtrack	68,18134	63,786	75,9563
Kruskal	407,5775	243,7956	567,3527
Hunt & Kill	1 850,016	1 814,286	1 911,553
Upravený Hunt & Kill	67,50076	63,5719	73,0902

Zdroj: vlastní tvorba za pomoci Unity

Tabulka 6 jasně ukazuje, jak s rostoucí velikostí bludiště roste rozdíl mezi efektivními a neefektivními algoritmy. Binary Tree zůstává rychlý, s maximálními časy pod 4 ms. Sidewinder dosahuje průměrného času kolem 6 ms. Recursive Backtrack, Kruskal a Hunt & Kill, které jsou výrazně pomalejší. Kruskal má průměrný čas 407 ms, zatímco Hunt & Kill dosahuje velkých hodnot 1850 ms. Upravená verze Hunt & Kill si vede o dost lépe.

Tabulka 7 – Výsledky při 10 měření a velikosti 500x500

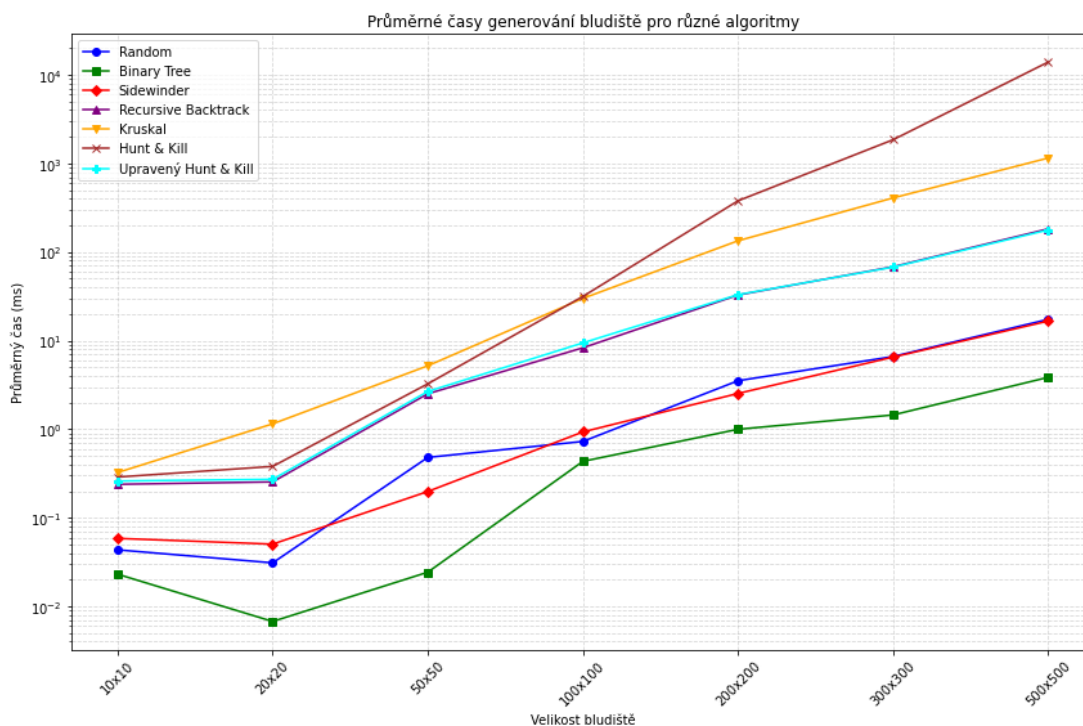
Algoritmus	Průměrný čas (ms)	Minimální čas (ms)	Maximální čas (ms)
Random	17,38925	16,3192	20,5248
Binary Tree	3,85252	1,7673	8,7509
Sidewinder	16,60841	15,7306	19,7386
Recursive Backtrack	182,5211	170,2584	222,3173
Kruskal	1 150,284	662,9161	1 592,946

Hunt & Kill	13 927,43	13 822,36	1 4035,89
Upravený Hunt & Kill	177,4643	171,486	183,8151

Zdroj: vlastní tvorba za pomoci Unity

Tabulka 7 ukazuje extrémní rozdíly mezi jednotlivými algoritmy při největší velikosti bludiště 500×500.

Nejrychlejší algoritmy, Binary Tree a Sidewinder, stále poskytují relativně nízké výpočetní časy. Naopak, Recursive Backtrack a Kruskal mají výrazně vyšší časy, přičemž Kruskal dosahuje maximální hodnoty až 1592 ms. Nejhorší výkon má Hunt & Kill, který dosahuje průměrného času 13 927 ms, což znamená, že jeho použití pro velké bludiště je neefektivní. Upravená verze Hunt & Kill si vede o dost lépe.



Obrázek 10 – Graf průměrných hodnot pro generování bludiště

Zdroj: vlastní tvorba za pomoci Pythonu a knihovny matplotlib

Na obrázku 10 lze vidět, který graf zobrazuje průměrné časy generování bludiště pomocí různých algoritmů pro různé velikosti bludiště (10×10 až 500×500). Výsledky ukazují rozdíly ve výpočetní náročnosti jednotlivých algoritmů, přičemž některé metody vykazují výrazně vyšší časy při větších rozměrech bludiště.

10 VYTVOŘENÍ NÁSTROJE PRO GENEROVÁNÍ BLUDIŠTĚ

Tato kapitola se zaměřuje na vývoj nástroje pro generování bludišť.

10.1 Požadavky na nástroj

Generování bludiště s více patry

- Nástroj umožní generovat bludiště až o 12 patrech, přičemž každé patro může mít jinou výšku a šířku.
- Bludiště může být tvořeno čtvercovými nebo hexagonálními buňkami.

Automatická kontrola propojenosti

- Po vygenerování bludiště systém ověří, zda jsou všechny buňky propojené.
- Zabráni vzniku izolovaných oblastí, které by mohly být nepřístupné.

Ověření řešitelnosti

- Nástroj zajistí, že bludiště obsahuje alespoň jednu platnou cestu z výchozího bodu do cíle (bod a cíl zadává uživatel).
- Implementace algoritmů pro vyhledávání cest.

Zobrazovací systém s omezeným výhledem

- Hráč uvidí pouze omezenou část bludiště odpovídající jeho aktuální pozici.

Generace na základě obrázku nebo zadaných rozměrů

- Alfa kanál v obrázku určuje prázdné buňky.
- Možnost upravovat velikost bludiště pomocí proměnné.

Editační režim

- Vývojář může interaktivně upravovat bludiště kliknutím na stěny – přidávat nebo odstraňovat překážky.
- Stěny, jejichž odstranění by mohlo vést k pádu hráče, zůstávají nezničitelné.

10.2 Vlastní atributy

Tento nástroj obsahuje tři vlastní atributy: VisibleIf, LabeledVector2 a Required.

10.2.1 VisibleIf

Tento atribut umožňuje skrýt nebo zobrazit určitou proměnnou v Inspectoru na základě hodnoty jiného parametru. V projektu se používá zejména pro zobrazení proměnných v rámci skriptu MazeVisibleChunk.

```
public override void OnGUI(Rect position, SerializedProperty property, GUIContent label)
{
    VisibleIfAttribute visibleIf = (VisibleIfAttribute)attribute;
    string conditionFieldName = visibleIf.conditionFieldName;
    bool compare = visibleIf.compare;

    SerializedProperty conditionField =
        property.serializedObject.FindProperty(conditionFieldName);

    if (conditionField != null && conditionField.propertyType == SerializedPropertyType.Boolean)
    {
        if (conditionField.boolValue == compare)
        {
            EditorGUI.PropertyField(position, property, label, true);
        }
    }
    else
    {
        Debug.LogWarning($"VisibleIf: Field '{conditionFieldName}' not found or is not a
boolean.");
        EditorGUI.PropertyField(position, property, label, true);
    }
}
```

Obrázek 11 – Kód pro VisibleIf v PropertyDrawer

Na obrázku 11 je kód, který definuje atribut VisibleIf v Inspectoru. Nejprve získá název kontrolované proměnné a očekávanou hodnotu, poté ji pomocí FindProperty vyhledá a ověří, zda je skutečně typu bool. Pokud podmínka odpovídá očekávané hodnotě, proměnná se zobrazí, jinak zůstane skrytá. Pokud však kontrolovaná proměnná neexistuje nebo není typu bool, zobrazí se varování v konzoli a vlastnost se vykreslí standardním způsobem.

10.2.2 LabeledVector2

Atribut LabeledVector2 rozšiřuje zobrazení dvourozměrného vektoru (Vector2) tím, že umožňuje přidat vlastní popisky k jednotlivým souřadnicím (X a Y). V daném assetu je použit u scaleru aby věděl uživatel, které osy v prostoru scaler ovlivňuje.

```
public override void OnGUI(Rect position, SerializedProperty property, GUIContent label)
{
    LabeledVector2Attribute splitVector2Attribute = (LabeledVector2Attribute)attribute;
    if (property.propertyType != SerializedPropertyType.Vector2)
    {
        EditorGUI.LabelField(position, label.text, "Use LabeledVector2Attribute with
Vector2.");
        return;
    }

    float lineHeight = EditorGUIUtility.singleLineHeight;
    float spacing = EditorGUIUtility.standardVerticalSpacing;

    float fieldWidth = (position.width - spacing) / 2;
    Rect labelXRect = new Rect(position.x, position.y, fieldWidth, lineHeight);
    Rect labelYRect = new Rect(position.x + fieldWidth + spacing, position.y, fieldWidth,
lineHeight);

    Rect fieldXRect = new Rect(position.x, position.y + lineHeight + spacing, fieldWidth,
lineHeight);
    Rect fieldYRect = new Rect(position.x + fieldWidth + spacing, position.y + lineHeight +
spacing, fieldWidth, lineHeight);

    EditorGUI.LabelField(labelXRect, splitVector2Attribute.XLabel);
    EditorGUI.LabelField(labelYRect, splitVector2Attribute.YLabel);

    property.vector2Value = new Vector2(
        EditorGUI.FloatField(fieldXRect, property.vector2Value.x),
        EditorGUI.FloatField(fieldYRect, property.vector2Value.y)
    );
}
```

Obrázek 12 – Kód pro LabeledVector2 v PropertyDrawer

Na obrázku 12 je kód, který definuje atribut LabeledVector2 v Inspectoru. Nejprve ověří, zda proměnná odpovídá očekávanému datovému typu, a pokud ne, zobrazí varování. Poté hodnoty rozdělí na dvě vstupní pole, ke kterým vykreslí odpovídající popisky. Uživatel tak může přímo upravovat hodnoty souřadnic X a Y v Inspectoru.

10.2.3 Required

Atribut Required slouží k označení proměnných, které musí být v Inspectoru vyplněné. V daném assetu se používá dvakrát, a to konkrétně na objekt Player transform a Player Camera.

```
public override void OnGUI(Rect position, SerializedProperty property, GUIContent label)
{
    bool isMissing = property.propertyType == SerializedPropertyType.ObjectReference
    && property.objectReferenceValue == null;

    if (isMissing)
    {
        var previousColor = GUI.color;
        GUI.color = Color.red;
        EditorGUI.PropertyField(position, property, label);
        GUI.color = previousColor;
        EditorGUI.HelpBox(position, $"{property.name} is required!", MessageType.Error);
    }
    else
    {
        EditorGUI.PropertyField(position, property, label);
    }
}
```

Obrázek 13 – Kód pro Required v PropertyDrawer

Na obrázku 13 je kód, který definuje vzhled atributu Required v Inspectoru. Nejprve ověřuje, zda proměnná je typu ObjectReference a zda není null. Pokud není hodnota nastavena, pole se podbarví červeně a pod ním se zobrazí chybová hláška informující uživatele, že daná hodnota je povinná. Tímto způsobem je zajištěno, že vývojář na žádný důležitý odkaz nezapomene.

Navíc je v projektu přítomen další skript, který doplňuje funkcionalitu atributu Required. Tento skript při přechodu do play módu zkontroluje, zda jsou všechny označené hodnoty správně vyplněné. Pokud detekuje prázdnou hodnotu, přechod do hracího režimu se zablokuje a v konzoli se vypíše přesné umístění objektu.

10.3 Generování bludiště

Existují dvě hlavní metody převodu bludiště na datovou reprezentaci:

1. Generování z obrázku – Alfa kanál obrázku určuje obsazenost buněk:
 - Průhledný pixel = prázdná buňka.
 - Neprůhledný pixel = buňka bludiště.
2. Náhodné generování – Každá buňka má 50% pravděpodobnost obsazenosti. Velikost se nastavuje individuálně pro každé patro v Inspectoru.

10.4 Kontrola propojenosti buněk

Po vygenerování je nutné ověřit, zda je možné se z libovolné buňky dostat do jiné. Kontrola propojenosti závisí na typu mřížky:

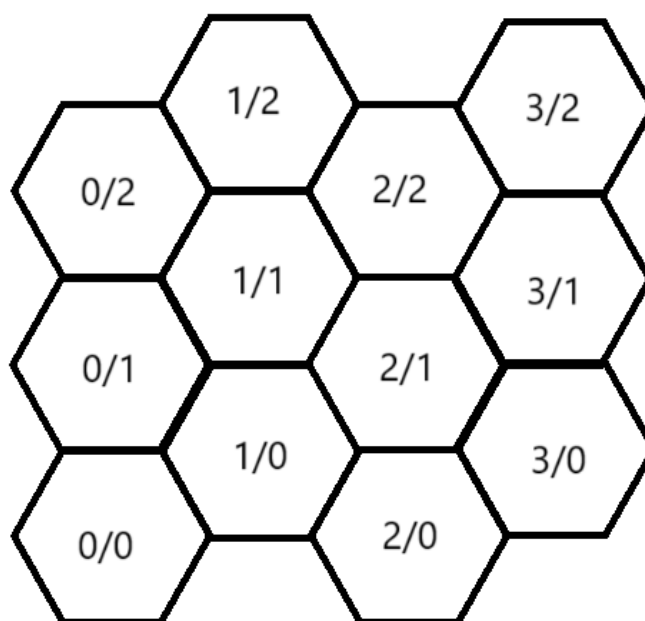
- Čtvercová mřížka – Každá buňka má čtyři sousedy.
- Hexagonová mřížka – Každá buňka má šest sousedů.

Proces:

1. Vybere se výchozí buňka.
2. Algoritmus Finder Component projde propojené buňky.
3. Pokud jsou všechny buňky propojené, bludiště je validní.
4. Pokud některé buňky nejsou propojené, algoritmus provede úpravy pro dosažení propojenosti.

10.4.1 Definice směrů u hexagonové mřížky

Hexagonová mřížka má 6 směrů propojení. Navíc se liší pravidla pro sudé a liché sloupce.



Obrázek 14 – Ukázka hexagonového bludiště

Zdroj: vlastní tvorba

Na obrázku 14 je zobrazeno hexagonové bludiště, které ukazuje strukturu polí tvořených šestiúhelníky.

Například

2/1 má sousedy

- 2/2 – směr (0, 1)
- 3/1 – směr (1, 0)
- 3/0 – směr (1, -1)
- 2/0 – směr (0, -1)
- 1/0 – směr (-1, -1)
- 1/1 – směr (-1, 0)

1/1 má sousedy:

- 1/2 – směr (0, 1)
- 2/2 – směr (1, 1)
- 2/1 – směr (1, 0)
- 1/0 – směr (0, -1)
- 0/1 – směr (-1, 0)
- 0/2 – směr (-1, 1)

10.5 Generování zdí

Tato podkapitola se zaměřuje na proces vytváření bludiště z jednotlivých buněk. Pro generování zdí byl vybrán upravený Hunt and Kill algoritmus, který byl popsán v kapitole 4. Výsledky měření jeho rychlosti lze nalézt v kapitole 9.

Struktura zdí v bludišti (DataOfMaze)

Bludiště je tvořeno následujícími datovými strukturami:

- floorsWalls – seznam uchovávající informace o podlahách jednotlivých buněk.
- listWallsHorizontal – seznam všech vodorovných zdí mezi buňkami.
- listWallsVertical – seznam všech svislých zdí mezi buňkami.
- sizeHorizontal – rozměry mřížky vodorovných zdí
- sizeVertical – rozměry mřížky svislých zdí
- sizeFloors – rozměry podlahových buněk

10.5.1 Abstraktní třída pro generování

Tato třída slouží jako základ pro generování bludišť a definuje hlavní metody, které se používají při jejich tvorbě.

```
public abstract class MazeAlgorithmBase
{
    protected List<WallsStateEnum[,]> mazeArray;
    internal List<DataOfMaze> floors = new List<DataOfMaze>();
    protected int currentRow;
    protected int currentColumn;
    protected int currentFloor;

    protected MazeAlgorithmBase(List<WallsStateEnum[,]> mazeArray)
    {
        this.mazeArray = mazeArray;
    }

    internal List<DataOfMaze> GetWallsList(bool removeWalls)
    {
        InitializeWalls();
        if (removeWalls) GenerateMaze();
        return floors;
    }

    internal abstract DataOfMaze CreateNewFloor(int width, int height);
    protected abstract List<bool> GetNeighbor(int floorIndex, int x, int y, WallsStateEnum
statusForSearch);
    protected abstract void RemoveWall(int direction, bool updateDirection = true);
    internal abstract void SetVisibilityWall(DataOfMaze floor, int x, int y, int width);
}
```

Obrázek 15 – Kód abstraktní třídy pro inicializaci zdí

Na obrázku 15 je kód abstraktní třídy `MazeAlgorithmBase`, která slouží jako základ pro implementaci generování bludiště pomocí upraveného algoritmu Hunt and Kill.

Jedním z atributů této třídy je `mazeArray` – seznam dvourozměrných polí typu `WallsStateEnum[,]`, který reprezentuje stav buněk v jednotlivých patrech bludiště. Dalším atributem je `floors` – seznam uchovávající informace o jednotlivých patrech.

Třída obsahuje metodu `GetWallsList(bool removeWalls)`, která inicializuje stěny bludiště. Pokud je parametr `removeWalls` nastaven na `true`, zavolá také metodu `GenerateMaze()`. Výsledkem této metody je seznam `floors`, obsahující data o bludišti.

Dále třída definuje několik abstraktních metod, které musí být implementovány v jejích potomcích. V tomto assetu jsou dva potomci pro hexagonové a čtvercové bludiště. Abstraktní metody jsou:

- `CreateNewFloor(int width, int height)` – vytváří nové patro bludiště.
- `GetNeighbor(int floorIndex, int x, int y, WallsStateEnum statusForSearch)` – hledá sousední buňky na základě určitého stavu stěn (nenavštívené a navštívené sousedy).

- RemoveWall(int direction, bool updateDirection = true) – odstraňuje stěnu v daném směru.
- SetVisibilityWall(DataOfMaze floor, int x, int y, int width) – nastavuje viditelnost všech stěn kolem určité buňky.

```
void GenerateMaze()  
{  
    bool needJoin = false;  
    for (int index = 0; index < mazeArray.Count; index++)  
    {  
        ProcessMazeFloor(index, ref needJoin);  
    }  
}  
  
void ProcessMazeFloor(int index, ref bool needJoin)  
{  
    var mazeFloor = mazeArray[index];  
    if (mazeFloor == null) return;  
  
    for (int y = 0; y < mazeFloor.GetLength(1); y++)  
    {  
        for (int x = 0; x < mazeFloor.GetLength(0); x++)  
        {  
            if (mazeFloor[x, y] == WallsStateEnum.NotVisited)  
            {  
                currentFloor = index;  
                currentColumn = y;  
                currentRow = x;  
  
                if (needJoin)  
                    JoinToVisitedCell();  
                else  
                    mazeArray[index][x, y] = WallsStateEnum.Visited;  
  
                RepeatUntilHaveUnvisitedNeighbor();  
                needJoin = true;  
            }  
        }  
    }  
}
```

Obrázek 16 – Zbytek kódu z abstraktní třídy pro inicializaci zdí

Na obrázku 16 je kód funkce GenerateMaze(), která generuje bludiště. Nejprve nastaví needJoin na false, což znamená, že nové cesty se zatím nespojují s existujícími. Poté prochází všechna patra v mazeArray a volá metodu ProcessMazeFloor().

Metoda `ProcessMazeFloor()` iteruje buňky patra. Pokud narazí na nenavštívenou buňku (`WallsStateEnum.NotVisited`), nastaví ji jako aktuální. Pokud je `needJoin = true`, propojí ji s existující cestou pomocí `JoinToVisitedCell()`, jinak ji pouze označí jako navštívenou. Nakonec `RepeatUntilHaveUnvisitedNeighbor()` pokračuje v generování, dokud jsou dostupné neprozkoumané buňky, a `needJoin` se nastaví na `true`, aby se další nalezená buňka připojila k cestě.

10.5.2 Čtvercové bludiště

Vytvoření patra

```
internal override DataOfMaze CreateNewFloor(int width, int height)
{
    return new DataOfMaze
    {
        floorsWalls = new WallVisibilityStatus[width * height],
        listWallsHorizontal = new WallVisibilityStatus[width * (height + 1)],
        listWallsVertical = new WallVisibilityStatus[(width + 1) * height],
        sizeHorizontal = new Vector2Int(width, height + 1),
        sizeVertical = new Vector2Int(width + 1, height),
        sizeFloors = new Vector2Int(width, height)
    };
}
```

Obrázek 17 – Kód pro přidání čtvercového patra

Na obrázku 17 je kód, který slouží k vytvoření nového patra bludiště. Metoda `CreateNewFloor` přijímá dva parametry – šířku (`width`) a výšku (`height`) patra. Vrací objekt `DataOfMaze`, který obsahuje informace o zdech a podlahách jednotlivých buněk.

Vytvoření stěn

```
internal override void SetVisibilityWall(DataOfMaze floor, int x, int y, int width)
{
    floor.listWallsHorizontal[floor.getIndex(x, y + 1, width)] =
    WallVisibilityStatus.VisibleInNormalMode; // up
    floor.listWallsVertical[floor.getIndex(x + 1, y, width + 1)] =
    WallVisibilityStatus.VisibleInNormalMode; // right
    floor.listWallsHorizontal[floor.getIndex(x, y, width)] =
    WallVisibilityStatus.VisibleInNormalMode; // down
    floor.listWallsVertical[floor.getIndex(x, y, width + 1)] =
    WallVisibilityStatus.VisibleInNormalMode; // left
    floor.floorsWalls[floor.getIndex(x, y, width)]
    = WallVisibilityStatus.VisibleInNormalMode; // floor
}
```

Obrázek 18 – Kód pro vytvoření všech stěn kolem buňky

Kód na obrázku 18 slouží k inicializaci všech zdí kolem buňky v bludišti. Metoda `SetVisibilityWall` nastavuje viditelnost všech čtyř stěn a podlahy pro konkrétní buňku na souřadnicích (`x, y`).

Získání sousedů

```
protected override List<bool> GetNeighbor(int floorIndex, int x, int y, WallsStateEnum
statusForSearch)
{
    var neighbors = new List<bool>();
    var currentFloor = mazeArray[floorIndex];
    if (currentFloor == null) return neighbors;

    // Check up
    neighbors.Add(y < (currentFloor.GetLength(1) - 1) && currentFloor[x, y + 1] ==
statusForSearch);

    // Check right
    neighbors.Add(x < (currentFloor.GetLength(0) - 1) && currentFloor[x + 1, y] ==
statusForSearch);

    // Check down
    neighbors.Add(y > 0 && currentFloor[x, y - 1] == statusForSearch);

    // Check left
    neighbors.Add(x > 0 && currentFloor[x - 1, y] == statusForSearch);

    // Check upper floor
    if (floorIndex < mazeArray.Count - 1)
    {
        var upperFloor = mazeArray[floorIndex + 1];
        neighbors.Add(x < upperFloor.GetLength(0) && y < upperFloor.GetLength(1) &&
            upperFloor[x, y] == statusForSearch);
    }
    else
    {
        neighbors.Add(false);
    }

    // Check lower floor
    if (floorIndex > 0)
    {
        var lowerFloor = mazeArray[floorIndex - 1];
        neighbors.Add(x < lowerFloor.GetLength(0) && y < lowerFloor.GetLength(1) &&
            lowerFloor[x, y] == statusForSearch);
    }
    else
    {
        neighbors.Add(false);
    }

    return neighbors;
}
```

Obrázek 19 – Kód pro získání sousedů čtvercového bludiště

Kód na obrázku 19 ukazuje metodu GetNeighbor(), která identifikuje sousední buňky v bludišti na základě jejich stavu definovaného v WallsStateEnum. Výsledkem metody je seznam hodnot true nebo false, kde true znamená, že daná sousední buňka odpovídá hledanému stavu.

Nejprve se metoda pokusí získat aktuální patro bludiště z pole mazeArray. Pokud je null, vrátí prázdný seznam sousedů. Následně zkontroluje čtyři hlavní směry – nahoru, vpravo, dolů a vlevo – a ověří, zda sousední buňka existuje a odpovídá požadovanému stavu.

Poté se metoda zaměřuje na horní a dolní patro. Pokud existuje horní patro, ověří, zda odpovídající buňka splňuje podmínku, a přidá ji do seznamu. Stejný postup se opakuje i pro dolní patro. Pokud patro neexistuje, přidá se hodnota false, čímž se indikuje, že soused v daném směru neexistuje.

Odstranění stěn

```
protected override void RemoveWall(int direction, bool updateDirection = true)
{
    int widthHorizontal = floors[currentFloor].sizeHorizontal.x;
    int widthVertical = floors[currentFloor].sizeVertical.x;
    int widthFloors = floors[currentFloor].sizeFloors.x;

    if (direction == 0) // up
    {
        var upIndex = floors[currentFloor].getIndex(currentRow, currentColumn + 1,
widthHorizontal);
        floors[currentFloor].listWallsHorizontal[upIndex] = WallVisibilityStatus.VisibleInEditMode;
        if (updateDirection) currentColumn++;
    }
    else if (direction == 1) // right
    {
        var rightIndex = floors[currentFloor].getIndex(currentRow + 1, currentColumn,
widthVertical);
        floors[currentFloor].listWallsVertical[rightIndex] =
WallVisibilityStatus.VisibleInEditMode;
        if (updateDirection) currentRow++;
    }
    else if (direction == 2) // down
    {
        var downIndex = floors[currentFloor].getIndex(currentRow, currentColumn, widthHorizontal);
        floors[currentFloor].listWallsHorizontal[downIndex] =
WallVisibilityStatus.VisibleInEditMode;
        if (updateDirection) currentColumn--;
    }
    else if (direction == 3) // left
    {
        var leftIndex = floors[currentFloor].getIndex(currentRow, currentColumn, widthVertical);
        floors[currentFloor].listWallsVertical[leftIndex] = WallVisibilityStatus.VisibleInEditMode;
        if (updateDirection) currentRow--;
    }
    else if (direction == 4) // upper floor
    {
        var upperFloorIndex = floors[currentFloor + 1].getIndex(currentRow, currentColumn,
floors[currentFloor + 1].sizeFloors.x);
        floors[currentFloor + 1].floorsWalls[upperFloorIndex] =
WallVisibilityStatus.VisibleInEditMode;
        if (updateDirection) currentFloor++;
    }
    else if (direction == 5) // lower floor
    {
        var lowerFloorIndex = floors[currentFloor].getIndex(currentRow, currentColumn,
widthFloors);
        floors[currentFloor].floorsWalls[lowerFloorIndex] = WallVisibilityStatus.VisibleInEditMode;
        if (updateDirection) currentFloor--;
    }
    mazeArray[currentFloor][currentRow, currentColumn] = WallsStateEnum.Visited;
}
```

Obrázek 20 – Kód pro odstranění stěny

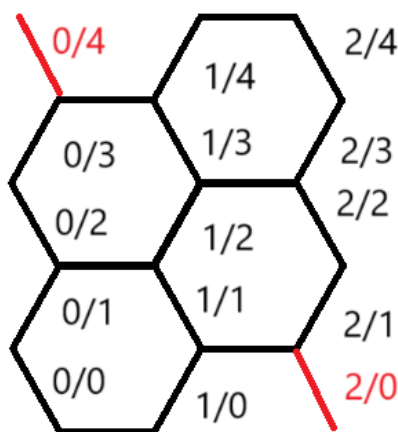
Na obrázku 20 je kód, který slouží k odstranění stěny mezi buňkami v bludišti. Metoda `RemoveWall` přijímá parametr `direction`, který určuje směr odstranění stěny (nahoru, dolů, vlevo, vpravo nebo mezi patry). Na základě tohoto směru se změní hodnota v příslušném poli na `VisibleInEditMode`, což znamená, že stěna byla odstraněna a umožňuje průchod.

10.5.3 Hexagonové bludiště

Hexagonové bludiště se vytváří podobně jako čtvercové, ale s několika rozdíly. Vertikální stěny jsou o něco větší, konkrétně:

- Velikost vertikálních zdí = šířka + 1; výška * 2 + 1;

Tento rozdíl způsobuje, že ne všechny vertikální stěny jsou využity. Pokud by bylo nutné využít všechny vertikální stěny, bylo by třeba změnit strukturu ukládání. V aktuální implementaci to však není nutné, protože na celé bludiště připadají dvě nepoužité stěny.



Obrázek 21 – Znázornění nevyužitých vertikálních stěn v hexagonovém bludišti

Zdroj: vlastní tvorba

Na obrázku 21 je hexagonové bludiště se zvýrazněnými stěnami, které nikdy nebudou využity. Pokud by bludiště mělo o jednu řadu více, přebývající stěna by se přesunula nahoru, konkrétně na index 3/4.

Díky tomu, jak jsou stěny rozmístěné, je nutné tuto skutečnost zohlednit při vytváření stěn kolem jednotlivých buněk, stejně jako při odstraňování. Hledání sousedů se musí provádět odlišným způsobem, což je podrobněji vysvětleno v kapitole 10.4.1.

10.6 Maze Color

Skript MazeColor slouží k nastavování materiálů a vizuálních parametrů bludiště v Unity. Uchovává materiály pro labyrint (Custom shader), výtah a editační režim a obsahuje také pole specifications.

10.6.1 Vytvoření Custom shaderu

V Unity lze vytvořit vlastní shader a propojit ho s materiálem, který pak lze přiřadit objektům ve scéně. Pro správné propojení je potřeba udělat následující kroky.

Nejprve je nutné vytvořit nový shader, který bude propojen s materiálem. Může se použít jakýkoli shader, protože se později upraví v kódu. Pro tento konkrétní případ se použije například Surface Shader.

Pak stačí otevřít vytvořený shader a upravit první řádek na: Shader Custom/MazeShader. tomto konkrétním případě by se řádek upravil ze „Shader Custom/NewSurfaceShader“ na „Shader Custom/MazeShader“.

Posledním krokem je vytvoření materiálu a nastavení hodnoty Shader na Custom/MazeShader.

Pro ověření správného nastavení lze použít následující kód v rámci funkce surf: [29]

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

Obrázek 22 – Testovací kód shaderu

Testovací kód z obrázku 22 nastaví výsledný materiál na modrou barvu, čímž lze ověřit správné propojení shaderu a materiálu.

10.6.2 Nastavení hodnot v shaderu za pomoci C#

```

...
void SetMaterial()
{
    if (labyrinthMaterial != null && specificadions.Length > 0)
    {
        labyrinthMaterial.SetFloatArray("_HeightFloorTexture", FillArray(specificadions.Select(o
=> o.heightFloorTexture).ToArray()));
        labyrinthMaterial.SetFloatArray("_Outline", FillArray(specificadions.Select(o =>
o.outline).ToArray()));
        labyrinthMaterial.SetFloatArray("_HeightAnotherColor", FillArray(specificadions.Select(o
=> o.heightWallColor).ToArray()));
        labyrinthMaterial.SetFloatArray("_CellSize", FillArray(specificadions.Select(o =>
o.scalerTexture).ToArray()));
        labyrinthMaterial.SetColorArray("_Color", FillArray(specificadions.Select(o =>
o.colorWall).ToArray()));
        labyrinthMaterial.SetColorArray("_ColorOutline", FillArray(specificadions.Select(o =>
o.colorOutline).ToArray()));

        labyrinthMaterial.SetFloat("_ScalerWidth", scaler.x);
        labyrinthMaterial.SetFloat("_ScalerHeight", scaler.y);

        labyrinthMaterial.SetInt("_Hex", hex ? 1 : 0);
        for (int i = 0; i < specificadions.Length; i++)
        {
            labyrinthMaterial.SetTexture("TexturesArray" + i, specificadions[i].mainTexture);
        }
    }
}
private T[] FillArray<T>(T[] inputArray, int targetLength = 12)
{
    T[] resultArray = new T[targetLength];
    for (int x = 0; x < inputArray.Length && x < targetLength; x++)
    {
        resultArray[x] = inputArray[x];
    }
    return resultArray;
}
...

```

Obrázek 23 – Kód pro nastavení proměnných v shaderu

Na obrázku 23 je kód, který nastavuje různé proměnné materiálu (custom shader). Nejprve kontroluje, zda je materiál a pole specifications správně inicializováno. Pokud ano, naplní několik vlastností materiálu, jako jsou texture výšky, obrysy, barvy a parametry škálování, přičemž hodnoty získává z pole specifications a předává je materiálu. Metoda FillArray zajišťuje, že každé pole hodnot předané materiálu má pevnou velikost (v tomto případě 12), přičemž pokud je pole kratší, zbytek se vyplní výchozími hodnotami. Tato pevná velikost pole je důležitá, protože shader očekává specifickou délku polí pro každou vlastnost.

Pevná velikost pole je důležitá, protože shaderový kód je navržen tak, aby zpracovával konzistentní počet hodnot. Pokud by velikost pole byla měněná, vedlo by to k chybám. Zejména to dělá problém, když se pole zvětšuje.

10.6.3 Kód v shaderu pro čtvercové bludiště

```

... // kód pro čtvercové bludiště
for (int i = maxlayer - 1; i >= 0; i--)
{
    float worldPosX = IN.worldPos.x + (0.5 * _ScalerWidth);
    float worldPosYMod = fmod(IN.worldPos.y + .5 * _ScalerHeight, _ScalerHeight);
    if (IN.worldPos.y + (1.5 * _ScalerHeight) >= heightsUser[i]
        && IN.worldPos.y + (1.5 * _ScalerHeight) - heightsUser[i] < _ScalerHeight)
    {
        if (worldPosYMod < _HeightFloorTexture[i] * _ScalerHeight)
        {
            float2 uvBase;

            if (abs(IN.worldNormal.y) > 0.8)
            {
                uvBase = IN.worldPos.xz / _CellSize[i];
            }
            else{
                uvBase = ( IN.worldPos.zy + IN.worldPos.xy) / _CellSize[i];
                uvBase.y *= 0.5;
            }

            fixed4 col = GetTexture(i, frac(uvBase));

            col.a *= 0.5;

            o.Albedo = col.rgb;
            o.Alpha = col.a;
            return;
        }

        if (fmod((worldPosX + (_Outline[i] / 2) * _ScalerWidth) , _ScalerWidth) <
            _Outline[i] * _ScalerWidth &&
            fmod(IN.worldPos.z + (_Outline[i] / 2) * _ScalerWidth, _ScalerWidth) <
            _Outline[i] * _ScalerWidth )
        {
            o.Albedo = _ColorOutline[i].rgb;
            o.Alpha = _ColorOutline[i].a;
            return;
        }
        if (worldPosYMod > _HeightAnotherColor[i] * _ScalerHeight)
        {
            o.Albedo = _ColorOutline[i].rgb;
            o.Alpha = _ColorOutline[i].a;
            return;
        }
        o.Albedo = _Color[i].rgb;
        o.Alpha = _Color[i].a;
        return;
    }
}
...

```

Obrázek 24 – Kód v shaderu pro čtvercové bludiště

Na obrázku 24 je kód shaderu, který se stará pro vykreslování čtvercového bludiště. Cyklus zpracovává jednotlivé vrstvy prostředí od nejvyšší k nejnižší. Na základě výšky se vybírá textura a barva.

Shader také pracuje s obrysy – pokud je pozice v blízkosti okrajů buněk, použije se speciální barva.

10.6.4 Kód v shaderu pro hexagonové bludiště

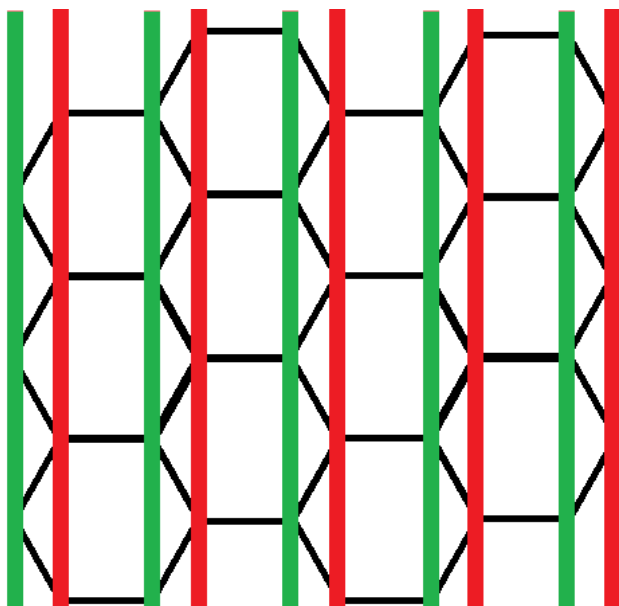
Shader pro hexagonové bludiště funguje podobně jako u čtvercového, ale liší se výpočtem obrysů a mapováním textur.

Obrysy

```
...
    if (fmod(offset + (1.25 * _ScalerWidth) + ((_Outline[i] * _ScalerWidth) / 3), 1.5 * _ScalerWidth) <
        (_Outline[i] * _ScalerWidth)) {
        o.Albedo = _ColorOutline[i].rgb;
        o.Alpha = _ColorOutline[i].a;
        return;
    }
    if (fmod(offset + (.25 * _ScalerWidth) + ((_Outline[i] * _ScalerWidth) / 1.5),
        1.5 * _ScalerWidth) < (_Outline[i] * _ScalerWidth)) {
        o.Albedo = _ColorOutline[i].rgb;
        o.Alpha = _ColorOutline[i].a;
        return;
    }
    if (worldPosYMod > _HeightAnotherColor[i] * _ScalerHeight)
    {
        o.Albedo = _ColorOutline[i].rgb;
        o.Alpha = _ColorOutline[i].a;
        return;
    }
    ...
```

Obrázek 25 – Kód výpočtu obrysů v hexagonovém bludišti

Na obrázku 25 je zobrazen kód pro výpočet obrysů u hexagonového bludiště. Zatímco u čtvercového bludiště stačila jedna podmínka pro kontrolu spoje a druhá pro překročení výšky, u hexagonového bludiště bylo nutné rozdělit kontrolu na dvě části spojů a zároveň třetí pro výšku.



Obrázek 26 – Barevné rozlišení podmínek pro výpočet obrysů hexagonového bludiště

Na obrázku 26 je zobrazeno hexagonové bludiště jako ukázka, jak by obrysy vypadaly, kdyby byly počítány na základě dvou podmínek pro kontrolu spojů (podmínka pro výšku není zohledněná). Každá podmínka je zobrazená jinou barvou. První podmínka, označená zelenou barvou, se vztahuje k prověřování první části spojů, zatímco druhá podmínka, zobrazená červenou barvou, slouží k určení druhé části spojů.

Mapování textury

Dále je nutné upravit způsob mapování textury. Pokud by se použilo stejné mapování jako u čtvercového bludiště, došlo by k deformaci textury na otočených stěnách, zejména k jejímu roztažení.

```
...
if (worldPosYMod < _HeightFloorTexture[i] * _ScalerHeight)
{
    float2 uvBase;
    if (abs(IN.worldNormal.y) > 0.8)
    {
        uvBase = IN.worldPos.xz / _CellSize[i];
    }
    else{
        if (abs(IN.worldNormal.x) < 0.8)
        {
            uvBase = IN.worldPos.yx / _CellSize[i];
            uvBase = RotateUV(uvBase, radians(90.0));
        }
        else if (abs(IN.worldNormal.x) > abs(IN.worldNormal.z))
        {
            uvBase = IN.worldPos.xy / _CellSize[i];
            uvBase.x *= 1.75;
        }
    }

    fixed4 col = GetTexture(i, frac(uvBase));

    col.a *= 0.5;

    o.Albedo = col.rgb;
    o.Alpha = col.a;
    return;
}
```

Obrázek 27 – Kód mapování textu u Hexagonového bludiště

Na obrázku 27 je kód, který zajišťuje správné mapování textury na hexagonové bludiště. Kód proto nejprve kontroluje, zda se daný bod nachází v oblasti podlahy, a pokud ano, pokračuje výpočtem souřadnic textury. Ty se určují na základě normály povrchu.

Pokud normála směřuje téměř svisle, používají se souřadnice xz, což je vhodné pro vodorovné plochy. U šikmých stěn je potřeba provést speciální úpravy – při malé hodnotě x se souřadnice počítají v systému yx a následně se otočí o 90 stupňů. V případech, kdy normála směřuje více do strany, se používá souřadnicový systém xy, přičemž osa x se násobí faktorem 1,75, aby se kompenzovalo zkreslení. Nakonec se získá správná textura pomocí `GetTexture()`, kde se poté přiřadí výslednému materiálu,

10.7 Maze Visible Chunk

Skript `MazeVisibleChunk` slouží k řízení viditelnosti částí bludiště na základě pozice hráče. Umožňuje dynamické generování viditelných chunků bludiště, čímž optimalizuje výkon a zlepšuje plynulost renderování.

```
[ExecuteInEditMode]
internal class MazeVisibleChunk : MonoBehaviour
{
    [SerializeField]
    internal bool VisibleAllMaze;
    [SerializeField, VisibleIf("VisibleAllMaze", false)]
    internal Transform transformPlayer;
    internal Vector3 InitialPosition { get; set; }
    [SerializeField, VisibleIf("VisibleAllMaze", false), Range(1, 100)]
    internal int VisibleTerrain = 1;
    [SerializeField, VisibleIf("VisibleAllMaze", false), Range(2, 10)]
    internal int VisibleNumberFloor = 2;

    private void Update()
    {
        if (transformPlayer == null) return;

        Vector3 currentPosition = transformPlayer.position;
        if (currentPosition != InitialPosition)
        {
            InitialPosition = currentPosition;
            if (!TryGetComponent<MazeGeneratorManager>(out var maze)) return;
            if (!VisibleAllMaze) maze.GenMazeWithPlayerPosition(currentPosition);
        }
    }
}
```

Obrázek 28 – Kód skriptu `MazeVisibleChunk`

Na obrázku 28 je kód, který definuje třídu `MazeVisibleChunk` v Unity. Tento skript se stará o dynamické zobrazování bludiště podle pozice hráče.

Metoda `Update()` pravidelně kontroluje hráčovu polohu a upravuje viditelnou část bludiště. Pokud není hráčský objekt přiřazen, metoda se ihned ukončí. Po získání aktuální pozice hráče ji skript porovná s uloženou hodnotou `InitialPosition`. Pokud se hráč pohnul,

InitialPosition se aktualizuje a zavolá se metoda GenMazeWithPlayerPosition(), která vygeneruje odpovídající viditelnou část bludiště.

Pokud je aktivní režim VisibleAllMaze, dynamické generování se vypne a zobrazí se celé bludiště.

10.8 Chunk Renderer

Tato kapitola se zabývá implementací renderovacích tříd SquareChunkRenderer a HexagonChunkRenderer, které slouží k vykreslování chunků bludiště.

10.8.1 SquareChunkRenderer

Třída SquareChunkRenderer je zodpovědná za vykreslování chunků, pokud buňky bludiště mají čtvercový tvar. Jejím hlavním úkolem je generovat a zobrazovat jednotlivé komponenty, jako jsou stěny, podlahy, stropy a výtahové plošiny. Data o těchto prvcích jsou získávána ze struktury MazeData.

Hlavní metodou třídy je RenderChunk, která vykresluje chunky na základě zadaných souřadnic X, Y a úrovně podlahy. Tato metoda také rozlišuje mezi standardními a editačními chunky.

Kroky vykreslování chunku:

- Vytvoření hlavních objektů: Pro každý chunk se vytváří hlavní prvek rootWalls, případně rootElevatorWalls, pokud je definován elevatorParent.
- Vykreslení horizontálních a vertikálních stěn: Algoritmus iteruje přes MazeData a vykresluje stěny dle jejich orientace.
- Vykreslení podlahy a střechy: Podlaha je generována podle viditelnosti stěn a výtahů. Pokud je povoleno renderování střechy, je přidána horní vrstva chunku.

Každá stěna, podlaha nebo střecha je vytvořena pomocí metody InstantiateWall, která instanciuje prefabrikovaný model (prefab) a nastavuje jeho pozici, měřítko a materiál.

10.8.2 HexagonChunkRenderer

Třída HexagonChunkRenderer vykresluje bludiště na základě hexagonální mřížky, což vyžaduje odlišné výpočty pozic a orientace stěn kvůli jiné geometrii.

Hexagonální reprezentace používá mřížku, ve které jsou dlaždice posunuty ve stylu včelí plástve. Kvůli této struktuře obsahuje HexagonChunkRenderer specifické metody pro vykreslování:

- `RenderVerticalWalls` – zajišťuje správnou orientaci stěn, konkrétně to mohou být úhly 60° nebo -60° .
- `RenderHorizontalWalls` – kvůli nepravidelnému uspořádání hexagonální mřížky jsou horizontální stěny umísťovány jiným způsobem než ve čtvercových chunkách. Záleží, jestli je sudý sloupec nebo lichý. Pokud je sudý sloupec je posunut o půlku velikosti hexagonu vzhůru.
- `RenderFloorAndRoof` – funguje podobně jako u `SquareChunkRenderer`, ale přizpůsobuje se hexagonálnímu uspořádání. Podlahy a střechy jsou vykreslovány jako šestiúhelníkové objekty s vlastní Mesh geometrií.

10.9 Minimapa

Minimapa slouží jako orientační prvek, který hráči pomáhá lépe se zorientovat v prostředí bludiště. Zobrazuje aktuální pozici hráče, rozlišuje navštívené a nenavštívené buňky a také vizualizuje výtahy.

10.9.1 Zobrazení buněk

Každá buňka může být na minimapě zobrazena odlišně v závislosti na svém stavu:

- **Navštívená** – Buňka se zobrazí jako kompletně odhalená, přičemž všechny stěny kolem ní jsou vizuálně odstraněny. Díky tomu je možné přesně určit její strukturu a obsah.
- **Nenavštívená** – Buňka je zobrazena jako plná, obklopená stěnami. Není možné z ní vyčíst, zda obsahuje například výtah nebo jiný prvek.

10.9.2 Asynchronní vykreslování

Vzhledem k tomu, že vykreslení celé podlahy může být při rozsáhlém bludišti časově náročné, proto minimapa využívá asynchronní generování pomocí coroutin. Renderování probíhá postupně – v každém snímku (frame) se vykreslí pouze určitá oblast (začíná v okolí hráče). Tento proces pokračuje, dokud není celá minimapa kompletně načtena. Tím se výrazně snižuje zatížení systému a zajišťuje plynulost celé aplikace

10.10 Editační režim

Editační režim umožňuje úpravu bludiště. Uživatel může měnit viditelnost jednotlivých prvků, jako jsou stěny, podlahy a výtahy. Tento systém je řízen třídou `MoveGOFromChunkToOtherChunk`, která zajišťuje přesun objektů mezi chunky a úpravu jejich viditelnosti.

10.10.1 Generování editačního prostředí

Po aktivaci editačního režimu se vygenerují všechny potřebné chunky s atributem `VisibleInEditMode`. To znamená, že stěny a podlahy dostanou editovatelný materiál, který vizuálně odlišuje jednotlivé prvky.

10.10.2 Interakce s objekty

Po kliknutí na objekt se zavolá metoda `ChangeWallVisibility`, která zpracuje změnu jeho viditelnosti a případné přesunutí mezi chunky. Algoritmus probíhá následovně:

1. Získání pozice a názvu objektu

Pomocí služby `PositionCalculatorService` se vypočítá pozice v poli a název objektu. Zároveň se ověří, zda objekt má platného rodiče. Pokud ne, operace je zrušena.

2. Identifikace typu objektu

Podle názvu objektu (`_h`, `_v`, `_f`, `_e`) se určí, zda se jedná o:

- Horizontální stěnu (`_h`),
- Vertikální stěnu (`_v`),
- Podlahu (`_f`),
- Výtahovou plošinu (`_e`).

3. Zpracování změny viditelnosti

Pokud je objekt stěna, provede se přepnutí mezi stavy `VisibleInEditMode` a `VisibleInNormalMode`. Tento stav je uložen v příslušném seznamu stěn. Objektu je následně přiřazen odpovídající materiál:

- `VisibleInEditMode` → `_editMaterial`
- `VisibleInNormalMode` → `_mazeMaterial`

Specifické operace:

- Podlahy: Pokud se podlaha změní na výtahovou plošinu, původní podlaha se odstraní a nahradí platformou generovanou `IElevatorPlatform`. Současně se aktualizuje její stav v seznamu.
- Výtahy: Pokud je objekt výtah, odstraní se a místo něj se vytvoří nová podlaha s využitím `meshe`. Stav objektu se následně aktualizuje v seznamu.
- Minimapa: pokud je v minimapě nastavená možnost `Visible All After Start`, tak se aktualizuje i minimapa.

10.11 Kontrola řešitelnosti

Kontrola řešitelnosti bludiště probíhá ve dvou fázích:

- Zjištění cesty – hledání průchozí trasy od startovního bodu k cíli.
- Výsledné renderování – vizualizace nalezené cesty v prostoru.

10.11.1 Zjištění cesty

Tato fáze je zodpovědná za ověření, zda existuje spojitá cesta mezi dvěma body v bludišti. K tomu slouží abstraktní třída `MazeSolverBase`, která poskytuje základní logiku pro prohledávání cesty.

Metoda `FindPath` využívá BFS (Breadth-First Search) k průchodu bludištěm a nalezení nejkratší cesty mezi startovním a cílovým bodem. Klíčové kroky tohoto algoritmu jsou:

- Použití fronty (`Queue`) pro iterativní prohledávání.
- Kontrola každé prozkoumané buňky a přidání jejích sousedů do fronty.
- Využití množiny (`HashSet`) k uchování navštívených bodů, čímž se zamezí cyklům.
- Pokud algoritmus dosáhne cílového bodu, vrátí seznam bodů představující trasu.
- Pokud žádná cesta neexistuje, metoda vrátí `null`.

Každý konkrétní solver implementuje metodu `GetNeighbors`, která definuje pravidla pro získání sousedních buněk.

10.11.2 Výsledné renderování

Jakmile je cesta nalezena, přichází na řadu vizualizace. K tomu slouží třída `MazeVisualizerBase`, která poskytuje základní funkce pro zobrazení nalezené trasy v prostoru.

Vizualizace je založena na následujících principech:

- Metoda `Visualize` zajišťuje vykreslení cesty tak, že mezi jednotlivé body umístí vizuální objekt a propojí je pomocí kostek.
- Metoda `CreateConnectingCube` generuje vizuální spojení mezi dvěma body pomocí objektu `Cube`.

10.12 Vytvoření assembly definicí

V rámci projektu jsou vytvořeny assembly definice pro oddělení jednotlivých částí kódu.

CustomAttributeEditor

Tato assembly definice obsahuje editory související s vlastními atributy definovanými v `CustomAttribute`. Je určena výhradně pro použití v editoru Unity a není součástí běhového kódu.

CustomAttribute

Assembly definice `CustomAttribute` obsahuje třídy pro vlastní atributy, které se používají v projektu. Tato reference má závislost na `CustomAttributeEditor`.

MazeGeneratorEditor

Definice `MazeGeneratorEditor` zahrnuje skripty určené pro rozšíření editoru Unity v souvislosti s generováním bludiště. Obsahuje vlastní inspektory a nástroje pro vizualizaci. Stejně jako `CustomAttributeEditor` je tato assembly určena pouze pro editor a není zahrnuta do buildu hry.

MazeGenerator

Assembly definice `MazeGenerator` obsahuje hlavní logiku generování bludiště. Tato definice má tři závislosti:

- `TextMeshPro` – využívá se pro vykreslování textů pro interakci s výtahem.

- `AYellowpaper.SerializedCollections` – externí knihovna, která umožňuje serializaci slovníků (`Dictionary`) v Unity, což standardně není podporováno.
- `CustomAttribute` – odkazuje na vlastní atributy definované v rámci projektu pro zajištění specifického chování některých tříd.

11 POPIS NÁSTROJE

Tato kapitola se věnuje popisu jednotlivých skriptů, které umožňují úpravu, generování a vizualizaci bludiště.

11.1 Maze Scaler

Skript Maze Scaler slouží k úpravě velikosti a viditelnosti bludiště. Díky parametrům tohoto modulu lze přizpůsobit rozměry jednotlivých buněk.

Parametry:

- Scaler X & Z – Proměnná typu Vector2.X, která nastavuje velikost jednotlivých buněk bludiště v osách X a Z.
- Scaler Y (Výška škálování) – Proměnná typu Vector2.Y, která nastavuje velikost buněk v ose Y.

11.2 Maze Visible Chunk

Skript Maze Visible Chunk umožňuje definovat, které části bludiště budou viditelné v daném okamžiku.

Parametry:

- Visible All Maze (Viditelnost celého bludiště) – Boolovská proměnná, která umožňuje zapnout nebo vypnout zobrazení celého bludiště. Pokud je nastavena na true, hráč vidí celé bludiště najednou. Pokud je nastavena na false, viditelnost bludiště je omezená kolem pozice hráče.
- Transform Player (Pozice hráče) – Objekt typu GameObject, který slouží jako reference na pozici hráče. Tento parametr je povinný. Na základě této pozice se generují herní chunky a zároveň se aktualizuje minimapa.
- Player Camera – Odkaz na komponentu typu Camera, která se používá pro určení orientace hráče na minimapě. Tento parametr je povinný.
- Visible Terrain (Viditelný terén) – Proměnná typu int, která určuje počet viditelných chunků okolo hráče. Hodnota se pohybuje v rozmezí 1–10 a ovlivňuje renderování chunků kolem hráče.

- Visible Floor (Viditelnost podlahy) – Proměnná typu int, která určuje počet viditelných podlah okolo hráče. Hodnota se pohybuje v rozmezí 2–10.

11.3 Maze Generator Manager

Skript Maze Generator Manager se stará o samotné generování bludiště a umožňuje konfiguraci jeho tvaru, struktury a vizuálních prvků.

Parametry:

- Text Interaction (Interakční text) – Odkaz na TextMeshPro, kde se zobrazují textové interakce, konkrétně teď je využíván, když hráč vstoupí na výtah.
- Make Roof (Vytvořit střechu) – Boolovská proměnná určující, zda má mít bludiště střechu.
- Wall Prefab (Prefab zdí) – Reference na prefab objektu, který představuje zdi bludiště.
- Method Generate (Metoda generování) – Enum, který umožňuje vybrat způsob generování bludiště. Může mít hodnoty:
 - Image – Generování na základě obrázku.
 - Default Shape – Standardní generování na základě obdélníkového tvaru.
- Shape Maze (Tvar bludiště) – Enum, který definuje tvar buněk v bludišti. Možnosti jsou:
 - Čtvercové buňky
 - Hexagonové buňky
- Chunk Size (Velikost chunku) – Proměnná typu int, která určuje velikost jednoho chunku v osách X a Z. Hodnota se pohybuje v rozmezí 2–150.
- Number Of Image (Počet obrázků) – Počet obrázků použitých pro generování bludiště (1–12). Tento parametr je viditelný pouze pokud je metoda generování nastavena na Image.
 - Scale (float) – Určuje, jak moc bude obrázek zvětšen nebo zmenšen. Je omezen na hodnoty 0,1 – 5

- – W: (new width); H: (new height)(label) – Zobrazuje výslednou šířku a výšku po škálování.
- Image (Obrázky bludiště) – Obrázky použité k vytvoření bludiště. Alfa kanál určuje, kde budou prázdné buňky. Počet obrázků odpovídá hodnotě Number Of Image.
- Number Of Floor (Počet podlah) – Počet úrovní bludiště (1–12). Tento parametr je viditelný pouze pokud je metoda generování nastavena na Default Shape.
- Width (Šířka podlahy) – Velikost dané podlahy v bludišti. Odpovídá počtu podlaží definovanému v Number Of Floor. Je možné nastavit v rozsahu 1 – 1 000.
- Height (Výška podlahy) – Velikost dané podlahy v ose z. Odpovídá počtu podlaží v Number Of Floor. Je možné nastavit v rozsahu 1 – 1 000.
- Recalculate Maze (Přepočítat bludiště) – Tlačítko, které odstraní všechny chunky a znovu je inicializuje. Viditelné pouze pokud bylo bludiště vygenerováno a edit mód je vypnutý.
- Generate Wall (Vygenerovat stěny) – Tlačítko pro generování bludiště, kde jsou všechny buňky plné. Užitečné pro ruční úpravy.
- Generate Wall – with destroy Walls (Vygenerovat bludiště s odstraněnými zdmi) – Vygeneruje bludiště se zaručenou průchodností.
- Start Edit Mode (Spustit editační režim) – Tlačítko pro aktivaci editovacího módu. Pokud je mód aktivní, zobrazí se tlačítko Stop Edit Mode.

11.4 Maze Color

Skript Maze Color umožňuje nastavit materiály a barevné schéma bludiště, včetně jednotlivých částí, jako jsou výtahy, podlahy a obrysy zdí.

Parametry:

- Maze Material (Materiál bludiště) – Materiál bludiště (shader).
- Elevator Material (Materiál výtahu) – Materiál použitý pro výtahy.
- Edit Material (Editační materiál) – Materiál použitý v editačním režimu.
- Specifications (Specifikace stylu) – Pole obsahující vizuální parametry pro každé patro bludiště:

- Height Floor Texture – Výška textury podlahy (rozsah 0–1).
- Outline – Velikost obrysu (rozsah 0–1).
- Height Wall Color – Výška, do které sahá barva zdi (rozsah 0–1).
- Scaler Texture – Mění velikost textury podlahy.
- Color Outline – Barva obrysových linií.
- Color Wall – Barva zdí.
- Floor Texture – Textura podlahy.

11.5 Solvable Checker

Skript Solvable Checker slouží k ověření, zda je bludiště řešitelné. Ověřuje, zda existuje průchozí cesta mezi dvěma body.

Parametry:

- Start Floor – nastavuje počáteční bod, ze kterého se hledá cesta.
- End Floor – určuje cílové bod, kam se musí hráč dostat.

11.6 Minimap

Skript Minimap slouží k nastavení vzhledu a funkčnosti minimapy zobrazující bludiště. Minimapa se automaticky aktualizuje podle pozice hráče (Player Transform) a jeho kamery (Player Camera) ze skriptu Maze Visible Chunk. V edit módu mapa není centrována.

Parametry:

- Minimap Show – Určuje, kdy se buňky na minimapě stanou viditelnými. Enum:
 - VisibleAfterPlayerVisit – buňka se zobrazí až po návštěvě hráčem, změni barvu na „navštíveno“.
 - VisibleAllAfterStart – všechny buňky jsou viditelné hned po startu.
- Player – Odkaz na objekt ikony hráče zobrazovaný v minimapě.
- Player Size (0.5 – 0.8) – Velikost ikony hráče v minimapě.
- Image Root From Canvas – GameObject, který je rootem minimapy na UI Canvasu.
- Size Of Cell (8 – 30) – Velikost jedné dlaždice v minimapě v pixelech.

- Elevator Icon Up – Obrázek používaný pro označení výtahu směřujícího nahoru.
- Elevator Icon Down – Obrázek výtahu směřujícího dolů.
- Elevator Icon Both – Obrázek výtahu umožňujícího cestu oběma směry.
- Color For Not Visited Cell – Barva dlaždic, které hráč ještě nenavštívil.
- Color For Visited Cell – Barva dlaždic, které již hráč navštívil.

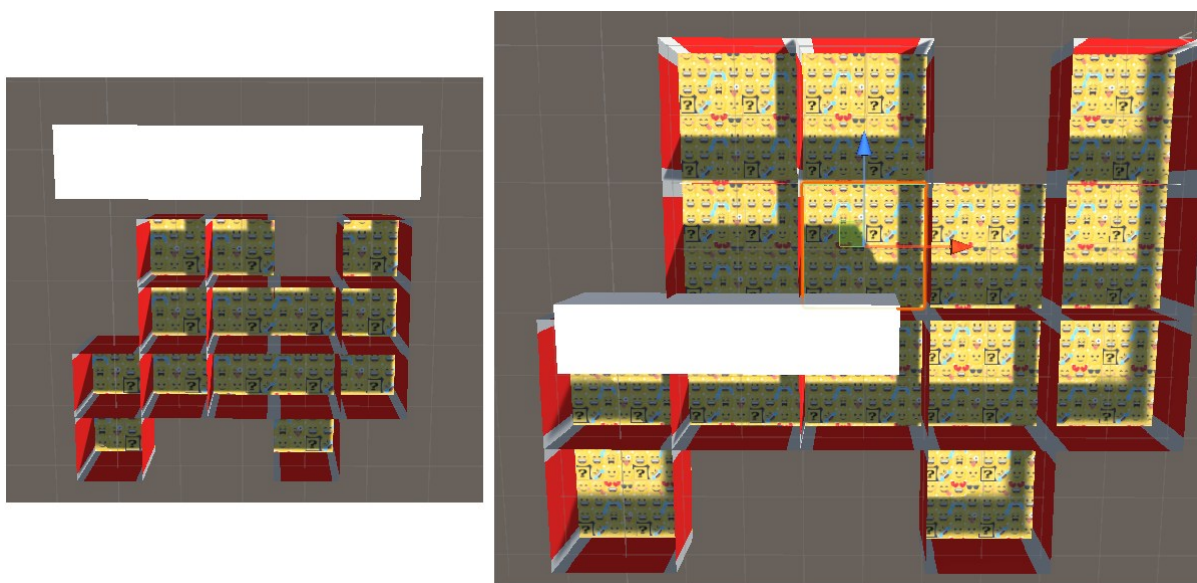
12 TESTOVÁNÍ ASSETU

Tato kapitola popisuje testovací scénáře, které ověřují správnou funkčnost některých částí nástroje. Testy jsou zaměřeny na ověření generování bludiště, vizualizaci, editaci a řešitelnost bludiště.

Scénář 1: Změna velikosti bludiště (Maze Scaler)

Postup testu:

1. Nastavit parametr **Scaler** na hodnotu **2** (pro zvětšení na dvojnásobek).
2. Kliknout na tlačítko **Recalculate Maze** pro zvětšení bludiště.
3. Manuálně zkontrolovat, že struktura a rozmístění prvků zůstaly zachovány (porovnat levý a pravý obrázek).



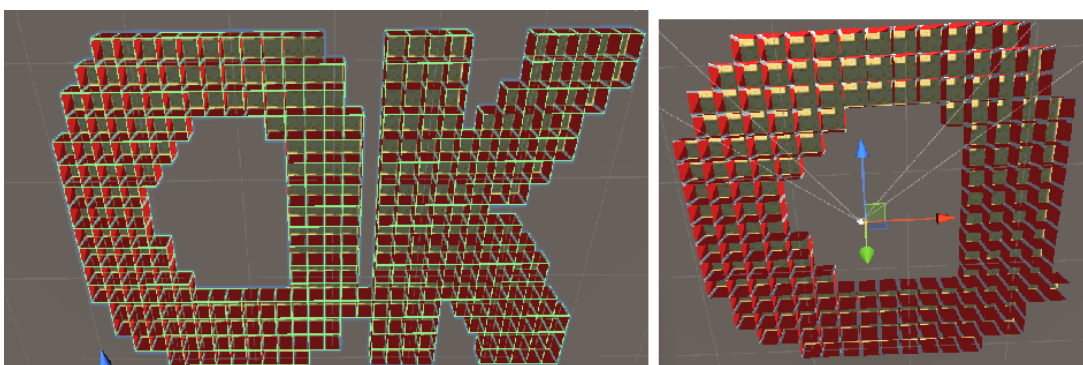
Obrázek 29 – Testování zvětšení

Zdroj: screenshot z Unity

Na obrázku 29 je možné vidět srovnání původního bludiště (vlevo) a jeho zvětšené verze na dvojnásobnou velikost (vpravo), přičemž struktura a rozmístění prvků zůstaly zachovány, což potvrzuje správnou funkčnost škálování.

Scénář 2: Ověření viditelnosti celého bludiště**Postup testu:**

1. Nastavit parametr **Visible All Maze** na hodnotu **false**.
2. Nastavit parametr **Visible Terrain** na **1** (chunk velikosti 8).
3. Kliknout na tlačítko **Recalculate Maze**.
4. Manuálně zkontrolovat, že je viditelná pouze oblast kolem hráče (1 chunk kolem a po jednom na každou stranu).



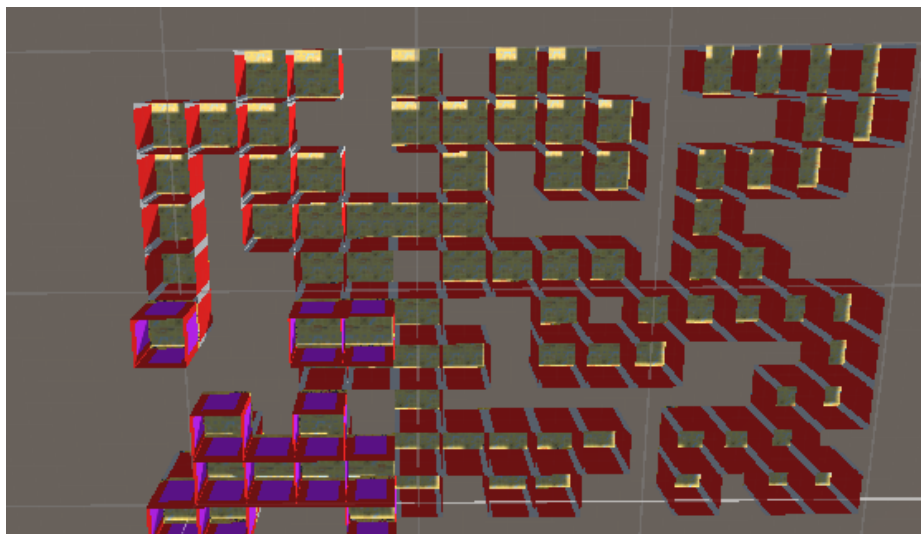
Obrázek 30 – Testování viditelnosti chunků

Zdroj: screenshot z Unity

Na obrázku 30 vlevo je zobrazeno celé bludiště, zatímco napravo je viditelná pouze oblast kolem hráče. Viditelnost byla nastavena na jeden chunk o velikosti 8, což znamená, že celkem by mělo být zobrazeno 9 chunků (1, na kterém se hráč nachází, a po jednom na každou stranu). Test viditelnosti hráče byl úspěšný – zobrazení odpovídá očekávání: viditelná zůstává pouze nejbližší oblast, zatímco vzdálenější chunky bludiště nejsou vykresleny.

Scénář 3: Ověření generování vícepatrového bludiště**Postup testu:**

1. Nastavit parametr **Number Of Floor** na **2**.
2. Nastavit parametry **Width** na **15** a **Height** na **10** pro spodní patro.
3. Nastavit parametry **Width** a **Height** na **5** pro druhé patro.
4. Kliknout na tlačítko **Generate Wall**.
5. Manuálně zkontrolovat správné vygenerování vícepatrového bludiště.



Obrázek 31 – Testování generování vícepatrového bludiště

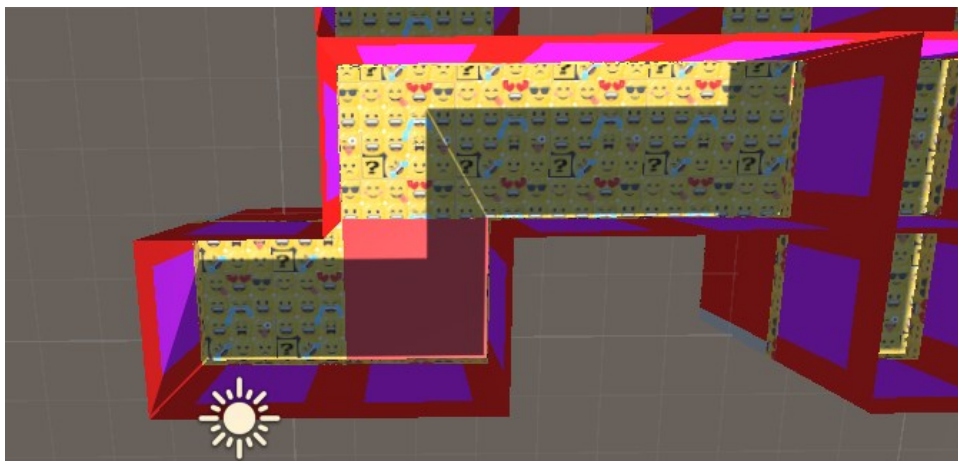
Zdroj: screenshot z Unity

Na obrázku 31 je vidět test generování vícepatrového bludiště. Spodní patro bylo nastaveno na šířku 15 a výšku 10, což odpovídá očekávaným rozměrům. Druhé patro mělo být vytvořeno s rozměry 5x5, což také odpovídá skutečnému výsledku. Test byl úspěšný, protože generované patro přesně odpovídá zadaným parametrům.

Scénář 4: Editace bludiště v Edit Modu

Postup testu:

1. Kliknout na tlačítko **Start Edit Mode**.
2. Odstranit tři vertikální zdi a jednu podlahu pomocí **Edit Tool**.
3. Zkontrolovat automatické vytvoření výtahů mezi patry.
4. Kliknout na **Stop Edit Mode**.
5. Manuálně zkontrolovat upravené bludiště.



Obrázek 32 – Testování edit módu

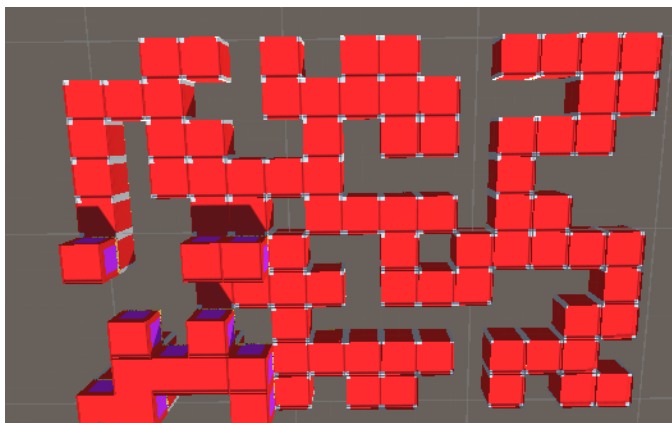
Zdroj: screenshot z Unity

Na obrázku 32 je vidět test editačního režimu, během kterého byly odstraněny tři vertikální zdi a jedna podlaha (bludiště stejné jako na obrázku 31). Kvůli odstranění podlahy bylo nutné vytvořit dva výtahy, každý pro jedno patro, aby zůstalo zachováno propojení mezi úrovněmi bludiště (výtahy byly vytvořeny automaticky). Všechny úpravy proběhly správně a výsledek odpovídá očekávání. Test editačního režimu byl tedy úspěšný.

Scénář 5: Přidání střechy k bludišti (Make Roof)

Postup testu:

1. Zaškrtnout možnost **Make Roof**.
2. Kliknout na tlačítko **Generate Wall**.
3. Manuálně zkontrolovat, že bludiště bylo vygenerováno se střechou.



Obrázek 33 – Testování vytvoření střechy

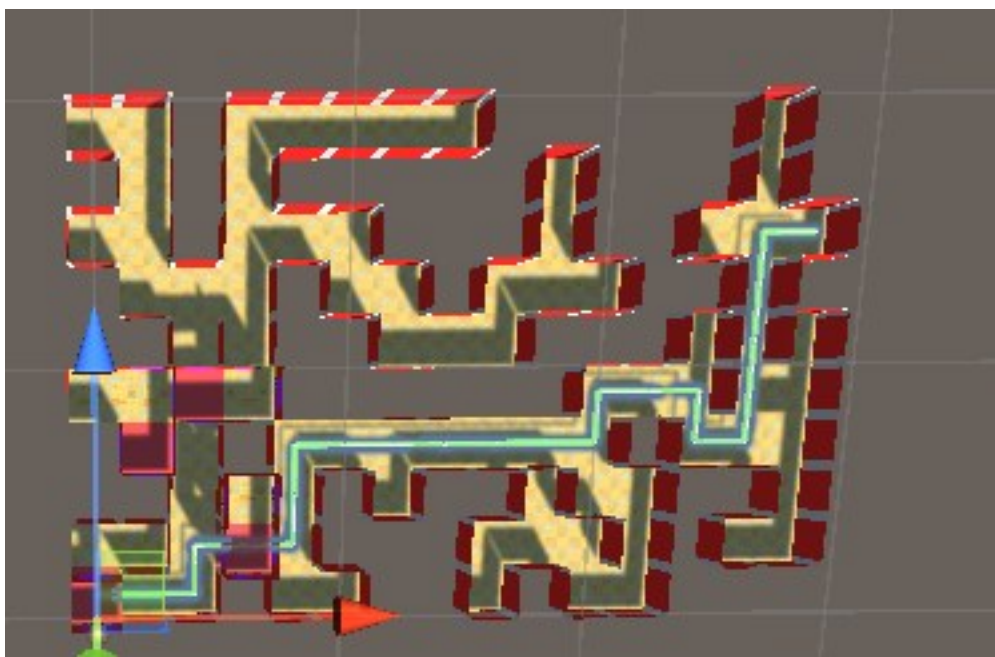
Zdroj: screenshot z Unity

Na obrázku 33 je vidět test tvorby střechy, který měl ověřit správné generování horního krytu bludiště. Střecha byla úspěšně vytvořena a odpovídá očekávanému výsledku. Test byl tedy úspěšný.

Scénář 6: Ověření průchodnosti bludiště (Solvable Checker)

Postup testu:

1. Nastavit **Start Floor** a **End Floor** na vybrané body.
2. Kliknout na tlačítko **Check Solvable**.
3. Automaticky zkontrolovat průchodnost a označení trasy.
4. Manuálně porovnat vyznačenou cestu s očekávanou trasou



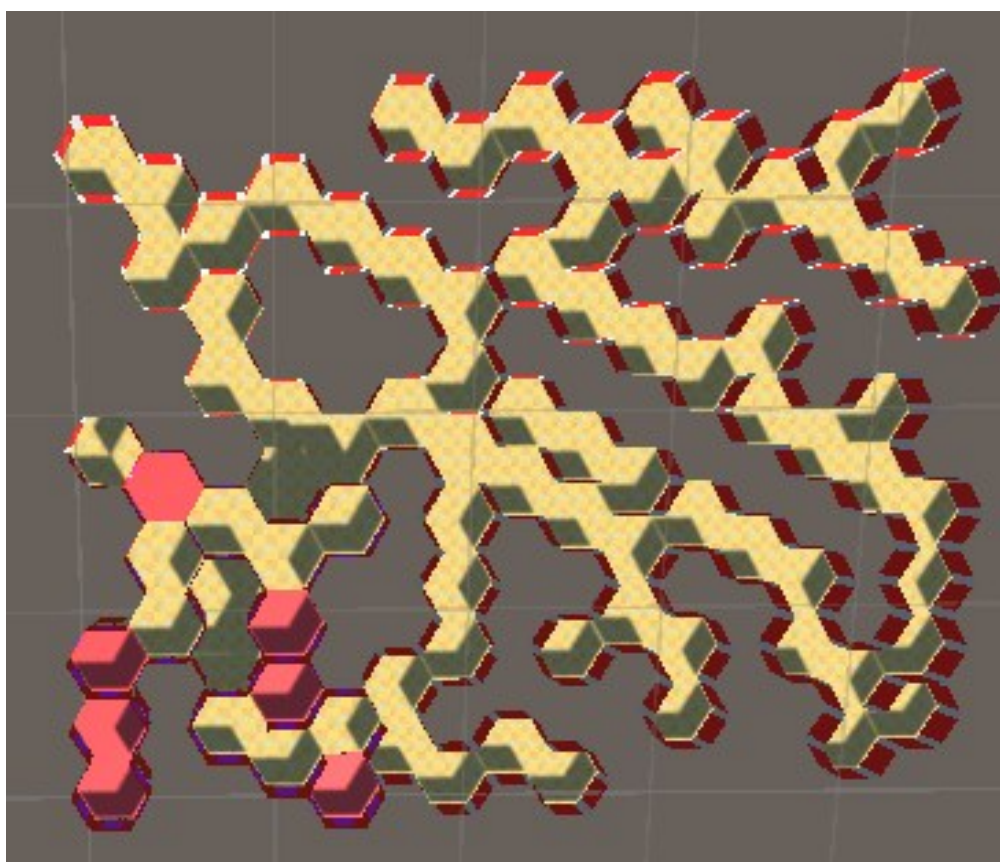
Obrázek 34 – Ověření průchodnosti bludiště

Zdroj: screenshot z Unity

Na obrázku 34 je vidět test ověření průchodnosti bludiště, který proběhl korektně. Algoritmus správně vyhodnotil průchozí cestu a ihned ji vizuálně označil. Test byl úspěšný, protože vyznačená trasa odpovídá očekávanému řešení.

Scénář 7: Ověření generování hexagonálního bludiště**Postup testu:**

1. Nastavit **Shape Maze** na **Hexagonové buňky**.
2. Kliknout na tlačítko **Generate Wall**.
3. Manuálně zkontrolovat, že bludiště má hexagonální mřížku.



Obrázek 35 – Testování generování Hexagonálního bludiště

Zdroj: screenshot z Unity

Na obrázku 35 je vidět test generování hexagonového bludiště, který proběhl úspěšně. Bludiště bylo správně vygenerováno s hexagonální mřížkou a odpovídá očekávané struktuře.

13 TESTOVÁNÍ VÝKONU ASSETU

Tato kapitola se zabývá testováním výkonu vytvořeného assetu. Aspekty, které jsou testovány, zahrnují dobu renderování, čas potřebný pro výpočet zdí a rychlost propojení bludiště. Pro vykreslení bylo nastaveno, aby bylo vždy viditelné celé bludiště.

13.1 První obrázek



Obrázek 36 – První testovací obrázek bludiště

Zdroj [28]

Na obrázku 36 je první testovací bludiště ve tvaru hvězdy. Velikost obrázku je 500x400, což odpovídá i velikosti výsledného bludiště.

Čtvercové bludiště

Tabulka 8 – Časy měření vytvoření čtvercového bludiště pomocí obrázku 1

Proces	Čas (ms)
Kontrola a propojení bludiště	138,7505
Inicializace zdí	145,1232
Vykreslení	3018,388

Tabulka ukazuje naměřené časy pro vytvoření čtvercového bludiště na základě testovacího obrázku 1. Jak je patrné, kontrola propojení bludiště zabrala přibližně 138 ms. Inicializace zdí, tedy jejich výpočet a příprava pro vykreslení, trvala 145 ms. Nejdelší část procesu představovalo samotné vykreslení bludiště, které zabralo přibližně 3018 ms.

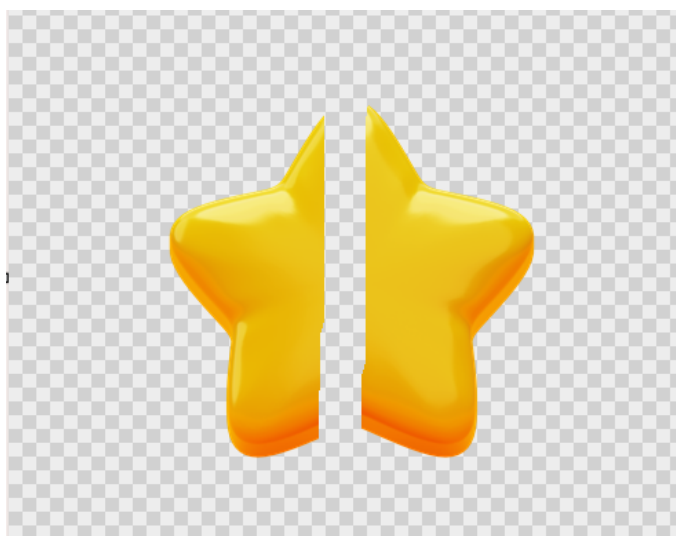
Hexagonové bludiště

Tabulka 9 – Časy měření vytvoření hexagonového bludiště pomocí obrázku 1

Proces	Čas (ms)
Kontrola a propojení bludiště	186,4288
Inicializace zdí	181,9372
Vykreslení	4558,222

Tabulka 9 obsahuje časy měření pro hexagonové bludiště. Oproti čtvercovému bludišti jsou zde patrné vyšší časy výpočtů i renderování, což je dáno složitější geometrií hexagonálních dlaždic. Kontrola propojení zdí trvala 186 ms, výpočet zdí byl delší než u čtvercového bludiště (181 ms oproti 145 ms), a samotné vykreslení zabralo 4558 ms, což je přibližně o 50 % delší čas než u čtvercového bludiště.

13.2 Druhý obrázek



Obrázek 37 – Druhý testovací obrázek bludiště

Zdroj upraven [28]

Na obrázku 37 je druhé testovací bludiště, které je téměř shodné s prvním, avšak byla odstraněna jeho prostřední část. Tato úprava způsobila nutnost propojení jednotlivých částí bludiště.

Čtvercové bludiště

Tabulka 10 – Časy měření vytvoření čtvercového bludiště pomocí obrázku 2

Proces	Čas (ms)
Kontrola a propojení bludiště	54558,52
Inicializace zdí	416,9474
Vykreslení	2582,231

U čtvercového bludiště byl výrazně vyšší čas potřebný pro kontrolu a propojení bludiště, což je způsobeno odstraněním středu a nutností nalezení nové spojitě cesty. Inicializace zdí i vykreslení proběhly v relativně krátkém čase.

Hexagonové bludiště

Tabulka 11 – Časy měření vytvoření hexagonového bludiště pomocí obrázku 2

Proces	Čas (ms)
Kontrola a propojení bludiště	52394,35
Inicializace zdí	503,6055
Vykreslení	3779,486

Podobně jako u čtvercového bludiště bylo propojení jednotlivých částí časově náročné. Hexagonové bludiště se vykreslovalo déle než čtvercové, což odpovídá jeho složitější geometrii.

13.3 Třetí obrázek

Pro třetí testovací scénář byla zvolena velikost 1024x1024 bez alfa kanálu, což znamená, že bludiště bude kompletně zaplněno bez jakýchkoliv mezer.

Čtvercové bludiště

Tabulka 12 – Časy měření vytvoření čtvercového bludiště pomocí obrázku 3

Proces	Čas (ms)
Kontrola a propojení bludiště	11946,25
Inicializace zdí	1278,757
Vykreslení	83686,63

V případě čtvercového bludiště byla nejdelší operací fáze vykreslování, která trvala téměř 84 sekund. To ukazuje, že vykreslení rozsáhlého bludiště je velmi výpočetně náročné. Kontrola spojení a inicializace zdí proběhly relativně rychle.

Hexagonové bludiště

Tabulka 13 – Časy měření vytvoření hexagonového bludiště pomocí obrázku 3

Proces	Čas (ms)
Kontrola a propojení bludiště	13191,78
Inicializace zdí	1470,927
Vykreslení	185911,6

Hexagonové bludiště vyžaduje podstatně více času na vykreslení než čtvercové. Vykreslování trvalo téměř 186 sekund, což je více než dvojnásobek oproti čtvercové variantě.

14 EXPORT A IMPORT UNITY ASSETU

Tato kapitola popisuje postup publikování vytvořeného nástroje tak, aby byl přenositelný do jiných projektů v Unity.

14.1 Vytvoření balíčku UnityPackage

Balíček UnityPackage (.unitypackage) je soubor obsahující vybrané soubory z Unity projektu, který lze snadno importovat do jiných projektů. Následuje postup jeho vytvoření:

1. Příprava souborů

Nejprve je nutné otevřít projekt určený k publikaci (v tomto případě nástroj pro generování bludiště). Doporučuje se seskupit všechny soubory určené k exportu do jedné složky a pojmenovat ji podle názvu assetu, aby se předešlo nejasnostem při importu do jiného projektu. Dále je vhodné přejmenovat scénu, aby se předešlo konfliktům s existujícími soubory v cílovém projektu.

2. Export souborů

V menu Assets vybrat možnost Export Package. Otevře se dialogové okno, kde lze označit soubory určené k exportu.

3. Uložení balíčku

Po potvrzení exportu se zobrazí okno pro výběr umístění souboru .unitypackage. Po uložení je balíček připraven k distribuci, například prostřednictvím Unity Asset Store nebo jiných úložišť. [29]

14.2 Import balíčku UnityPackage do projektu

Níže jsou uvedeny způsoby importu balíčku do jiného projektu v Unity. (Import přímo z Unity Asset Store zde není zahrnut.)

Možnost 1: Import přímo v Unity

1. Otevření projektu

- Otevřít projekt, do kterého bude balíček importován.

2. Zahájení importu

- V menu Assets vybrat Import Package → Custom Package.

- Otevře se Průzkumník souborů, ve kterém se musí vybrat požadovaný .unitypackage.
- Alternativně lze balíček otevřít přímo dvojitým kliknutím v průzkumníku souborů. Tento způsob je rychlejší, ale může způsobit problémy, pokud je otevřeno více projektů v Unity (import do špatného projektu). [29]

3. Výběr souborů pro import

- Po otevření balíčku se zobrazí dialogové okno se seznamem obsažených souborů.
- Je možné vybrat pouze soubory relevantní pro projekt. Pokud již některé soubory v projektu existují, nebudou zobrazeny.

4. Dokončení importu

- Po potvrzení výběru souborů kliknout na Import. Vybrané soubory budou přidány do projektu a připraveny k použití.

Možnost 2: Import pomocí Unity Hubu

Pokud je nainstalován Unity Hub, je možné balíček importovat i bez otevření projektu v Unity.

1. Otevření souboru balíčku

- V průzkumníku souborů se musí vyhledat .unitypackage a otevřít jej dvojitým kliknutím.
- Unity Hub se automaticky spustí a nabídne možnosti importu.

2. Vytvoření nového projektu

- Pokud má být balíček importován do nového projektu, v Unity Hubu se může zvolit New Project, zadat jeho název a vybrat umístění.

3. Import do existujícího projektu

- V Unity Hubu lze rovněž vybrat existující projekt a otevřít jej.
- Po spuštění projektu probíhá import stejným způsobem jako v Možnosti 1 – výběrem souborů a jejich potvrzením. [29]

14.3 Publikování Assetu na Unity Asset Store

Tato část popisuje proces publikace assetu na Unity Asset Store.

Požadavky na publikaci

Před nahráním na Asset Store je nutné:

- Mít účet Unity ID a být přihlášen v Publisher Portalu.
- Zvolit unikátní název balíčku a přiřadit jej ke správné kategorii.
- Připravit UnityPackage k nahrání.

Instalace a použití Asset Store Tools

Pro správu a nahrání balíčků je nutné nainstalovat Asset Store Tools (zdarma dostupné na Unity Asset Store). Po instalaci se v Unity Editoru zobrazí nabídka Tools > Asset Store.

Postup nahrání:

- Otevřít Asset Store Tools a vybrat Uploader.
- Zvolit možnost Pre-exported .unitypackage a vybrat následný balíček. Poté stačí kliknout na Upload.
- V Publisher Portalu přidat požadované propagační materiály (snímky obrazovky, banner, ikonu).
- Doplnit klíčová slova a případně další informace.
- Po dokončení úprav se aktivuje tlačítko Submit. Po odeslání je třeba vyčkat na schválení. Schválení probíhá ručně a zpravidla trvá několik pracovních dnů.

Odkaz na výsledný asset

[30]

ZÁVĚR

Tato diplomová práce se zaměřila na návrh a implementaci nástroje pro generování bludišť v herním enginu Unity. Hlavním cílem bylo vytvořit systém, který podporuje generování různých typů bludišť s více patry, umožňuje uživatelské úpravy zdí a je vhodný pro použití v počítačových hrách.

V teoretické části byly analyzovány herní enginy a popsány pojmy související s bludišti, včetně rozdílu mezi labirintem a bludištěm a vlastností perfektního bludiště. Dále byly rozebrány různé algoritmy generování bludišť, jejich výhody a nevýhody.

Praktická část se zaměřila na návrh a realizaci nástroje v Unity. Nejprve byla provedena analýza existujících řešení, následně byl vyvinut asset umožňující generování bludišť s možností výběru různých parametrů. Nástroj podporuje náhodné generování buněk, tvorbu bludišť na základě obrázku s alfa kanálem a umožňuje uživatelskou úpravu viditelných zdí prostřednictvím zabudovaného editačního režimu.

Mezi klíčové vlastnosti implementovaného řešení patří podpora generování bludišť s více patry, což umožňuje tvorbu komplexnějších struktur. Nástroj navíc obsahuje kontrolu propojenosti buněk a ověřování řešitelnosti, což zajišťuje, že generované bludiště je vždy hratelné. Součástí systému je také minimapa, která zobrazuje tvar buňky až po jejím navštívení. Pro její vykreslování je využito asynchronní generování pomocí coroutine.

Výkon jednotlivých algoritmů byl testován a porovnán, přičemž výsledky ukázaly rozdíly v rychlosti generování v závislosti na použité metodě a velikosti bludiště. Díky systému správy viditelnosti bludiště pomocí chunků bylo možné optimalizovat vykreslování a zobrazovat pouze části bludiště v závislosti na pozici hráče. Součástí nástroje je také vlastní shader.

SEZNAM POUŽITÉ LITERATURY

- [1] *What Is A Game Engine: An Essential Overview for Beginners*. Online. Draw & Code. Copyright 2024 ©. Dostupné z: <https://drawandcode.com/learning-zone/what-is-a-game-engine/>. [cit. 2025-02-04].
- [2] *Technical Overview of Unity Game Engine*. Online. Pubnub. 2010. Dostupné z: <https://www.pubnub.com/guides/unity/>. [cit. 2025-02-05].
- [3] *What Is Unreal Engine?* Online. BairesDev. 2009. Dostupné z: <https://www.bairesdev.com/blog/what-is-unreal-engine/>. [cit. 2025-02-06].
- [4] *Scripting the Unreal Editor Using Python*. Online. UNREAL ENGINE. © 2025. Dostupné z: <https://dev.epicgames.com/documentation/en-us/unreal-engine/scripting-the-unreal-editor-using-python>. [cit. 2025-02-06].
- [5] *Proč začít vyvíjet hry v enginu Godot?* Online. Robot_dreams. 2020. Dostupné z: <https://robotdreams.cz/blog/370-proc-zacit-vyvijet-hry-v-enginu-godot>. [cit. 2025-02-08].
- [6] *Tohle je ten nejlepší herní engine, který musíš znát v roce 2024!*. Online. HERI CORE ACADEMY. © 2025. Dostupné z: <https://kericore.academy/tohle-nejlepsi-herni-engine-ktery-musis-znat-2024/>. [cit. 2025-02-08].
- [7] *Jaký je rozdíl mezi labyrintem a bludištěm? V jednom z nich byste se určitě neztratili*. Online. Prima ZOOM. Dostupné z: <https://zoom.iprima.cz/zajimavosti/bludiste-labyrint-rozdil-104997>. [cit. 2024-11-10].
- [8] *What makes a good maze? – 5 Characteristics of a maze*. Online. Mazy Maze. Dostupné z: <https://mazymaze.com/blog/characteristics-of-effective-mazes>. [cit. 2024-11-15].
- [9] *Maze Generation Algorithms - An Exploration*. Online. Dostupné z: <https://professor-l.github.io/mazes/>. [cit. 2024-11-15].
- [10] *Maze Classification*. Online. Dostupné z: <https://www.astrolog.org/labyrnth/algrithm.htm>. [cit. 2024-11-15].
- [11] BUCK, Jamis. *Mazes for programmers: code your own twisty little passages*. P1.0-July. Pragmatic bookshelf. Dallas, Texas: Pragmatic Programmers, [2015]. ISBN 978-1-68050-055-4.

- [12] *Binary tree algorithm*. Online. Languagelessness: OpenSCAD. ©2024. Dostupné z: <https://openhome.cc/eGossip/OpenSCAD/SimpleGeneratedMaze.html>. [cit. 2024-11-18].
- [13] *Maze generation with the sidewinder algorithm*. Online. 64bitdragon. ©2024. Dostupné z: <https://learn.64bitdragon.com/articles/computer-science/procedural-generation/maze-generation-with-the-sidewinder-algorithm>. [cit. 2024-11-18].
- [14] *Maze Generation: Recursive Backtracking*. Online. The Buckblog. Dostupné z: <https://weblog.jamisbuck.org/2010/12/27/maze-generation-recursive-backtracking>. [cit. 2024-11-18].
- [15] *Maze Generation: Hunt-and-Kill algorithm*. Online. The Buckblog. Dostupné z: <https://weblog.jamisbuck.org/2011/1/24/maze-generation-hunt-and-kill-algorithm>. [cit. 2024-12-01].
- [16] *Access Modifiers (C# Reference)*. Online. Microsoft Learn. © Microsoft 2024. Dostupné z: <https://learn.microsoft.com/en-gb/dotnet/csharp/language-reference/keywords/access-modifiers>. [cit. 2024-12-03].
- [17] *Scope and Access Modifiers*. Online. Unity Learn. Copyright © 2024 Unity Technologies. Dostupné z: <https://learn.unity.com/tutorial/scope-and-access-modifiers#>. [cit. 2024-12-06].
- [18] ADRES, Angelo. *Extending Unity with Editor Scripting*. Birmingham, UK: Packt, 2015. ISBN 9781785281853
- [19] KOK, Benny. *Beginning Unity editor scripting: create and publish your game tools*. New York, NY, USA: Apress, 2021. ISBN 978-1-4842-7166-7.
- [20] *Unity User Manual 2021.3 (LTS)* [online]. San Francisco, USA: Unity Technologies, 2022 [cit. 2025-01-03]. Dostupné z: <https://docs.unity3d.com/Manual/index.html>
- [21] *Introduction to Unity Serialization and Game Data*. Online. Game Developer. © 2025. Dostupné z: <https://www.gamedeveloper.com/business/introduction-to-unity-serialization-and-game-data>. [cit. 2025-01-05].
- [22] *Struct vs Class in C#: Choosing the Right Data Type*. Online. ByteHide. ©2025. Dostupné z: <https://www.bytehide.com/blog/struct-vs-class-csharp>. [cit. 2025-01-05].

- [23] *Creating Custom Unity Attributes: ReadOnly*. Online. Medium. ©2025. Dostupné z: <https://vintay.medium.com/creating-custom-unity-attributes-readonly-d279e1e545c9>. [cit. 2025-01-15].
- [24] *Property Attributes - Part 1 | Unity Editor Scripting* [@Otter Knight]. Online. 2021. Dostupné z: YouTubeVideo, <https://www.youtube.com/watch?v=SvHoOu47pP0>. [cit. 2025-01-15].
- [25] *Property Attributes - Part 2 | Unity Editor Scripting* [@Otter Knight]. Online. 2021. Dostupné z: YouTubeVideo, <https://www.youtube.com/watch?v=zkr6LU3kHE4>. [cit. 2025-01-15].
- [26] *Unity Features 101: Assembly Definitions*. Online. Medium. © 2025. Dostupné z: <https://medium.com/design-bootcamp/unity-features-101-assembly-definitions-44cc8878e141>. [cit. 2025-01-20].
- [27] *Unity Assembly Definition Files*. Online. InnoGames. © Copyright 2025. Dostupné z: <https://blog.innogames.com/unity-assembly-definition-files/>. [cit. 2025-01-20].
- [28] *Star winner rating review icon sign or symbol 3d background illustration*. Online. In: Freepik. © 2010. Dostupné z: https://www.freepik.com/free-psd/star-winner-rating-review-icon-sign-symbol-3d-background-illustration_71291981.htm#fromView=keyword&page=1&position=2&uuid=d20238fb-801d-46f1-9c26-a7ecabab57c9&query=Png. [cit. 2025-02-12].
- [29] MIKESKA, Mikuláš. *Vytvoření nástroje pro generování ostrova v enginu Unity*. Bakalářská práce. Zlín: Fakulta aplikované informatiky | Univerzita Tomáše Bati. 2023.
- [30] *Maze Generator V*. Online. Unity Asset Store. Dostupné z: <https://assetstore.unity.com/packages/tools/level-design/maze-generator-v-310858>. [cit. 2025-05-08].

SEZNAM OBRÁZKŮ

Obrázek 1 – ukázka labyrintů	18
Obrázek 2 – ukázka bludiště	19
Obrázek 3 – Kód ukázka atributů	29
Obrázek 4 – Ukázka kódu vlastního atributu pro zvýraznění	31
Obrázek 5 – Ukázka kódu PropertyDrawer pro zvýraznění	32
Obrázek 6 – Kód pro změnu zvýraznění prvku v Inspectoru	33
Obrázek 7 – Kód pro přidání posuvníku v Inspectoru	33
Obrázek 8 – Kód pro přidání popisu v Inspectoru	34
Obrázek 9 – Ukázka kódu pro přidání více prvků v Inspectoru	35
Obrázek 10 – Graf průměrných hodnot pro generování bludiště.....	46
Obrázek 11 – Kód pro VisibleIf v PropertyDrawer	48
Obrázek 12 – Kód pro LabeledVector2 v PropertyDrawer	49
Obrázek 13 – Kód pro Required v PropertyDrawer	50
Obrázek 14 – Ukázka hexagonového bludiště	51
Obrázek 15 – Kód abstraktní třídy pro inicializaci zdí	53
Obrázek 16 – Zbytek kódu z abstraktní třídy pro inicializaci zdí	54
Obrázek 17 – Kód pro přidání čtvercového patra	55
Obrázek 18 – Kód pro vytvoření všech stěn kolem buňky	55
Obrázek 19 – Kód pro získání sousedů čtvercového bludiště	56
Obrázek 20 – Kód pro odstranění stěny	57
Obrázek 21 – Znázornění nevyužitých vertikálních stěn v hexagonovém bludišti	58
Obrázek 22 – Testovací kód shaderu	59
Obrázek 23 – Kód pro nastavení proměnných v shaderu	60
Obrázek 24 – Kód v shaderu pro čtvercové bludiště	61
Obrázek 25 – Kód výpočtu obrysů v hexagonovém bludišti	62
Obrázek 26 – Barevné rozlišení podmínek pro výpočet obrysů hexagonového bludiště	62
Obrázek 27 – Kód mapování textu u Hexagonového bludiště.....	63
Obrázek 28 – Kód skriptu MazeVisibleChunk	64
Obrázek 29 – Testování zvětšení	76
Obrázek 30 – Testování viditelnosti chunků.....	77
Obrázek 31 – Testování generování vícepatrového bludiště	78
Obrázek 32 – Testování edit módu	79
Obrázek 33 – Testování vytvoření střechy	79
Obrázek 34 – Ověření průchodnosti bludiště.....	80

Obrázek 35 – Testování generování Hexagonálního bludiště	81
Obrázek 36 – První testovací obrázek bludiště.....	82
Obrázek 37 – Druhý testovací obrázek bludiště	83

SEZNAM TABULEK

Tabulka 1 – Výsledky při 10 měření a velikosti 10x10	41
Tabulka 2 – Výsledky při 10 měření a velikosti 20x20	42
Tabulka 3 – Výsledky při 10 měření a velikosti 50x50	43
Tabulka 4 – Výsledky při 10 měření a velikosti 100x100	43
Tabulka 5 – Výsledky při 10 měření a velikosti 200x200	44
Tabulka 6 – Výsledky při 10 měření a velikosti 300x300	45
Tabulka 7 – Výsledky při 10 měření a velikosti 500x500	45
Tabulka 8 – Časy měření vytvoření čtvercového bludiště pomocí obrázku 1	82
Tabulka 9 – Časy měření vytvoření hexagonového bludiště pomocí obrázku 1	83
Tabulka 10 – Časy měření vytvoření čtvercového bludiště pomocí obrázku 2	84
Tabulka 11 – Časy měření vytvoření hexagonového bludiště pomocí obrázku 2	84
Tabulka 12 – Časy měření vytvoření čtvercového bludiště pomocí obrázku 3	85
Tabulka 13 – Časy měření vytvoření hexagonového bludiště pomocí obrázku 3	85

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AI – Artificial Intelligence

API – Application Programming Interface

BFS – Breadth-First Search

DLL – Dynamic-Link Library

HDRP – High Definition Render Pipeline

HTML5 – HyperText Markup Language, version 5.

JSON – JavaScript Object Notation

NPC – Non-Player Character

UI – User Interface

UE – Unreal Engine

URP – Universal Render Pipeline

VR – Virtual Reality

AR – Augmented Reality

SEZNAM PŘÍLOH

Příloha P I: Struktura odevzdaných dat („zip.zip“)

PŘÍLOHA P I: STRUKTURA ODEVZDANÝCH DAT („ZIP.ZIP“)

Tento zip obsahuje:

Diplomovou práci

Asset

Implementaci algoritmů

DP_Mikeska-Mikulas.pdf

MazeAsset.unitypackage

AlgorithmusExample.unitypackage