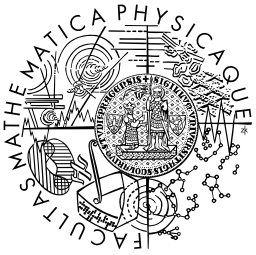


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Ondřej Hlava

Simulátor šíření ohně

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Jiří Švancara, Ph.D

Studijní program: Informatika

Praha 2024

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Chtěl bych poděkovat svému vedoucímu RNDr. Jiřímu Švancarovi, Ph.D. za jeho vedení, čas a ochotu při realizaci této práce. Také bych chtěl upřímně poděkovat úplně všem, kteří mě podporovali nejen během samotného psaní, ale již dávno předtím, neboť bez nich bych tuto práci ani nikdy psát nezačal.

Název práce: Simulátor šíření ohně

Autor: Ondřej Hlava

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Jiří Švancara, Ph.D, Katedra teoretické informatiky a matematické logiky

Abstrakt: Tato práce se zabývá popisem nástroje na simulaci šíření ohně v krajině implementovaném v multiplatformním herním enginu Unity. Implementace přináší nový, vizuálně přitažlivý a velmi jednoduše použitelný nástroj, který lze využít mnoha způsoby. Šíření ohně je založeno na jednoduchých nedeterministických, avšak realistických pravidlech. Simulace se odehrává na zjednodušeném modelu světa, který je procedurálně generovaný. Aplikace je do jisté míry univerzální, a tak má množství různorodých využití, například může sloužit jako podpůrný nástroj ve výuce o faktorech ovlivňujících šíření ohně, dále může posloužit začínajícím programátorům, kterým nabídne možnost vizualizovat si vlastní svět či vygenerovat data pro předpovídání šance rozšíření ohně s následnou možností si tuto předpověď vizualizovat.

Klíčová slova: oheň, simulátor, políčka, krajina, Unity, C#

Title: Fire Spread Simulator

Author: Ondřej Hlava

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Jiří Švancara, Ph.D, Department of Theoretical Computer Science and Mathematical Logic

Abstract: This thesis focuses on the description of a tool designed for simulating fire spread across landscapes, which is developed within the cross-platform Unity game engine. This tool stands out due to its visually appealing interface and exceptional user-friendliness, offering diverse applications. The fire spread is modeled using simple, non-deterministic but realistic rules. The simulation is conducted on a simplified, procedurally generated world model. The application's versatility allows it to be used in diverse contexts. For instance, it can serve as a support tool in teaching about factors affecting the dynamics of fire spread. Additionally, it could be useful for beginner programmers by providing them the opportunity to visualize their landscapes, generate data for predicting fire spread chances, along with the options to visualize their own predictions.

Keywords: fire, simulator, tiles, landscape, Unity, C#

Obsah

1 Úvod **8**
- 1.1 Zařazení simulátoru . . . 8
- 1.2 Vize kompletního simulátoru . . . 9
- 1.3 Cíle práce . . . 9
  - 1.3.1 Požadavky . . . 10

2 Analýza zadání **11**
- 2.1 Související práce . . . 11
  - 2.1.1 Wildfire Explorer . . . 11
  - 2.1.2 FireSpreadingSimulation . . . 12
  - 2.1.3 ForeFire . . . 13
  - 2.1.4 Fire-spreading-Simulation . . . 15
  - 2.1.5 Shrnutí . . . 16
- 2.2 Cíloví uživatelé . . . 17
- 2.3 Cílová platforma . . . 17
- 2.4 Implementační nástroj . . . 17

3 Unity herní engine **21**
- 3.1 Úvodní přehled . . . 21
- 3.2 Spuštění . . . 22
- 3.3 Editor . . . 22
- 3.4 GameObject . . . 22
- 3.5 Scene, Game view . . . 23
- 3.6 Project . . . 23
- 3.7 Inspector . . . 26
  - 3.7.1 Komponentový systém . . . 27
- 3.8 Prefab . . . 29
- 3.9 Programování . . . 29
  - 3.9.1 MonoBehaviour . . . 29
  - 3.9.2 Serializace . . . 31

4 Softwarový návrh řešení **32**
- 4.1 Herní svět . . . 32
- 4.2 Generace světa . . . 33
  - 4.2.1 Perlinův šum . . . 33
  - 4.2.2 Další techniky . . . 35
- 4.3 Ukládání a načítání světa . . . 35
- 4.4 Simulace . . . 36
  - 4.4.1 Predikce a napojení na externí skript . . . 38
- 4.5 Rozdělení simulátoru . . . 38
- 4.6 Uživatelské rozhraní . . . 39
  - 4.6.1 Hlavní menu . . . 39
  - 4.6.2 Kamera . . . 40
  - 4.6.3 Modely vegetace . . . 40
  - 4.6.4 Voda . . . 41

5 Uživatelská dokumentace **43**
5.1 O aplikaci . . . 43
5.2 Spuštění aplikace . . . 43
5.3 Hlavní nabídka . . . 43
5.3.1 Info . . . 43
5.3.2 Settings . . . 45
5.3.3 Quit . . . 46
5.4 Tutorial . . . 46
5.5 Sandbox . . . 46
5.6 Predictions . . . 48
5.7 Složka StreamingAssets . . . 51

6 Vývojářská dokumentace **52**
6.1 Unity projekt . . . 52
6.1.1 Assets . . . 52
6.1.2 Scripts . . . 53
6.1.3 StreamingAssets . . . 54
6.2 WorldClasses.cs . . . 54
6.3 WorldUtilities.cs . . . 55
6.4 Map.cs . . . 55
6.5 MapExtensions.cs . . . 55
6.6 WorldGenerator.cs . . . 55
6.7 WorldBuilder.cs . . . 56
6.8 HeightMapImporter.cs . . . 56
6.9 InputHandler.cs . . . 56
6.10 MainLogic.cs . . . 58
6.11 TutorialManager.cs . . . 58
6.12 FireSimulation.cs . . . 60
6.13 FireSimParameters.cs . . . 60
6.14 SimulationManager.cs . . . 60
6.15 SimulationBase.cs . . . 60
6.16 EventLoggers.cs . . . 61
6.17 Visualizer.cs . . . 61
6.18 WindIndicator.cs . . . 62
6.19 CameraHandler.cs . . . 62
6.20 PredictionLogic.cs . . . 63
6.21 PythonCaller.cs . . . 63
6.22 main.py . . . 64
6.23 WorldSerialization.cs . . . 64
6.24 Settings.cs . . . 64
6.25 FileBrowserHandler.cs . . . 66
6.26 FileManagementService.cs . . . 66

Závěr **67**

Literatura **69**

Seznam obrázků **72**

Seznam tabulek **73**

A Přílohy **74**

1 Úvod

Dnes a denně požáry zasahují různá místa po celém světě. Mohou ničit stavení, statky a domy, ale také celé přírodní ekosystémy, lesy a krajiny. Oheň je pro mnoho lidí nesmírně fascinujícím fenoménem, který může jim i jiným organismům zmařit různé životní snahy, ale také přinést spoustu užitku.

Simulace ohně představuje nelehkou, ale klíčovou vědeckou výzvu. Význam takovýchto simulací může přesahovat do životů lidí na různých místech planety víc, než si běžně uvědomujeme. Simulace může být využita od předpovědí a analýz rizika požárů, přes vzdělávání a školení personálu zapojeného do ochrany před požáry, až po vývoj strategií pro obnovu postižených oblastí. Má tedy význam nejenom pro ekologii, ochranu přírody, ale také například pro správu krajiny a můžeme se díky ní vyhnout i potenciálně obrovským katastrofám.

Problém takové simulace zahrnuje modelování všemožných fyzikálních procesů. I přes svou nezpochybnitelnou komplexitu a množství ovlivňujících faktorů nám moderní vědecké přístupy umožňují stále lepší a přesnější modelování a simulace těchto procesů.

1.1 Zařazení simulátoru

Je důležité zmínit, že kromě simulátorů dnes existuje mnoho dalších nástrojů, které nám pomáhají chápat a řešit výzvy spojené s požáry. Například interaktivní mapy jako tato [1] poskytují aktuální data o požárech na celém světě a mohou sloužit jako doplněk k různým simulacím.

Pojďme se nyní zaměřit na simulátory šíření ohně. Nejprve se podíváme na to, jak by se daly takové simulátory vlastně kategorizovat.

Podle účelu použití

- *Výcvikové simulátory:* Umožňují hasičům a záchranářům procvičovat scénáře hašení požárů a evakuace v bezpečném, kontrolovaném prostředí.
- *Vědecké a výzkumné simulátory:* Slouží k modelování chování ohně a jeho šíření v různých podmínkách, což pomáhá při tvorbě prediktivních modelů a strategií pro boj s požáry.
- *Umělecké a zábavní simulátory:* Využívají se v digitálním umění a videohrách pro vytvoření vizuálně přesvědčivých zobrazení ohně.

Podle míry realismu

- *Fyzikálně založené simulátory:* Tyto simulátory využívají komplexní matematické modely pro simulaci chování ohně, včetně šíření tepla, produkce kouře a interakce s okolním prostředím.
- *Zjednodušené simulátory:* Pro méně náročné aplikace a podle způsobu použití, kde není potřeba vysoký stupeň realismu, lze použít jednodušší algoritmy a vizualizace.

Podle modelovaného prostředí

- *Krajinné (venkovní) simulátory:* Simulují šíření ohně v lesních oblastech, zahrnují faktory jako jsou typ vegetace, počasí a topografie.
- *Městské požáry:* Zaměřují se na šíření ohně v městských prostředích, včetně interakce s budovami, silničními sítěmi a infrastrukturou.
- *Interiérové (vnitřní) simulátory:* Modelují šíření ohně uvnitř budov, včetně evakuace osob a šíření kouře.

Další rozdělení

Podle specifických aspektů ohně se simulátory dále mohou zaměřovat a více do detailu modelovat určité aspekty ohně a vznikající fenomény jako například: Simulátory šíření tepla, které se specializují na modelování šíření tepla a jeho vlivu na okolní materiály a struktury.

Simulátory vývoje kouře, které se zaměřují na generování a šíření kouře, což je klíčové například pro zdraví a rizika při evakuaci osob či zvířat.

Simulátory plamenů se pak specializují na vizualizaci a chování vzduchu a samotných plamenů, včetně jejich barvy, intenzity a pohybu.

1.2 Vize kompletního simulátoru

Uvedeme si, jak zhruba vypadá vize našeho simulátoru. Náš simulátor bude program, který bude snadno spustitelný a široké veřejnosti dostupný. Bude vizuálně přitažlivý, aby uživatelé měli z používání programu pozitivní dojmy. Program umožní simulovat šíření ohně ve zjednodušeném modelu krajiny světa. Také bude natolik intuitivní, že i uživatelé s menší technickou zdatností v něm budou schopni bez větších obtíží se vyznat, navigovat a ovládat jej. Uživatel bude motivován v něm strávit určitý čas, tak aby si stihl všimnout základních faktorů ovlivňujících oheň v reálném světě a alespoň částečně pochopil, jak komplexní a složité je opravdu realisticky předpovídat, kam se oheň potenciálně rozšíří. Nebude se jednat ani o vědecký ani o výcvikový simulátor, bude se jednat o zjednodušený zábavní simulátor.

1.3 Cíle práce

V této práci vybereme vhodnou množinu požadavků a funkcí našeho simulátoru. Jeho unikátnost bude mimo jiné právě v kombinaci nabízených vlastností. Simulátor se budeme snažit držet co nejblíže naší původní vize. Zejména chceme usilovat o vytvoření samostatného a plnohodnotného programu. Zároveň budeme chtít, aby byl navržen s ohledem na částečnou flexibilitu a možnost budoucího rozšíření.

Nyní si uvedeme abstraktní seznam vlastností, které by náš simulátor měl mít. Poté si v následujících kapitolách více do detailu zanalyzujeme, proč je důležité tyto vlastnosti zahrnout a jakým nejlepším způsobem je můžeme implementovat v rámci našeho programu.

1.3.1 Požadavky

Simulátor by měl přinést mezi již stávající simulátory následující vlastnosti:

- Snadnost používání
- Generování krajiny (terénu)
- Možnost ukládání a načítání krajiny
- Zjednodušené pravidla simulace
- Dobrý vizuální dojem
- Logika tříd světa a simulace oddělena od jejich vizualizace
- Rozšiřitelnost programu
- Predikce rozšíření
- Možnost jednoduchého propojení s externími skripty

Všechny tyto vlastnosti si v následující kapitole zanalyzujeme a vysvětlíme více do detailu.

Cílem práce *není* vytvořit nástroj, který by se snažil konkurovat dnešním technologiím co se komplexnosti a realističnosti týče, a proto tedy není ani zamýšlen k přenesení a aplikaci výstupu simulátoru do reálné krajiny. Nechceme se ani snažit simulovat co nejpřesněji všechny spjaté fyzikální jevy, které se při požáru objevují. Naše snaha je spíše přiblížit široké veřejnosti celý tento koncept a usnadnit začátky pro lidi, kteří se něco o ohni chtějí naučit. Mělo by to tedy být formou, při které se člověk něco málo naučí, aniž by si to úplně třeba uvědomil. Tím by nástroj mohl zvýšit povědomí o požárech.

Navíc by mohl sloužit i programátorům začátečníkům, kteří by si pomocí něj mohli vytvořit a vizualizovat vlastní virtuální svět nebo trénovat základní techniky strojového učení na modelování šíření ohně.

V následující kapitole si zařadíme náš simulátor mezi ostatní stávající simulátory tak, aby obsahoval vhodnou množinu funkcí. Zaměříme se na to, aby celý koncept byl adekvátně zjednodušen pro naplnění vize našeho simulátoru ohně v krajině. Co by takové dobré zjednodušení obnášelo si konkrétněji ujasníme v další kapitole.

2 Analýza zadání

Abychom byli schopni dobře a kvalitně navrhnout náš program, musíme si nejprve rozmyslet na jaké aspekty našeho programu konkrétněji cílíme a proč. Tedy dát dohromady očekávání a rozmyslet si, jakým způsobem bychom jich mohli nejlépe dosáhnout.

2.1 Související práce

Nejprve se podíváme na některé již existující hotové programy zabývající se stejným nebo podobným tématem a okruhem. Následující výběr je takový menší příkladový vzorek programů, co jsou blízko naší představě o simulátoru. Podíváme se na to, jaké vlastnosti tyto programy splňují a co nabízejí. Zaměříme se také na věci, co jim naopak chybí ve srovnání s naší vizí programu, nebo s tím, co by někteří uživatelé mohli uvítat. Následně všechny tyto společné vlastnosti a zjištění spojíme dohromady. Podle toho si pak také lépe dokážeme vytyčit jasnější cíl a jaký přínos by naše práce mezi tyto již stávající nástroje přinesla.

2.1.1 Wildfire Explorer

Cílem projektu GeoHazard [2] konsorcia Concord je vzdělávat studenty středních a vysokých škol v oblasti přírodních rizik prostřednictvím jejich modelování a hodnocení. Projekt integruje modely země s různými nástroji pro analýzu dat a snaží se studentům ukázat příčiny, průběh a dopady přírodních nebezpečí.

Wildfire Explorer je online nástroj, který se zabývá šířením ohně v krajině [3].

Projekt nabízí možnost zažehnout plameny na různých místech mapy a poté sledovat, jak se oheň rozšiřuje. Uživatelé na připravené krajině mohou na zvolených místech rozdělat oheň, simulaci pozastavovat a opět pouštět. Nástroj nabízí možnost vizualizace grafu, počtu shořených akrů krajiny v čase.

Obrázek 2.1 ukazuje rozhraní samotného programu.

Obrázek 2.1 Ukázka online nástroje Wildfire Explorer.

Wildfire Explorer umožnuje umístit do terénu *Helitack* či *Fire line*.

Helitack týmy jsou specializované skupiny hasičů, které používají vrtulníky pro rychlou dopravu na místo požáru a efektivní hašení ze vzduchu. Tyto týmy mohou provádět počáteční útoky na požáry v obtížně přístupných oblastech nebo podporovat operace na zemi poskytováním kritických informací, dopravou personálu a materiálu, a v některých případech tedy i vypouštěním vody nebo retardantu na požár.

Fire line neboli požární linie je bariéra, která je vytvořena odstraněním veškerého hořlavého materiálu v určitém pruhu země, aby se zabránilo šíření požáru. Požární linie hrají klíčovou roli v strategiích boje proti požárům tím, že zabraňují jeho rozšíření do dalších oblastí.

Poznatky pro náš simulátor

Naš simulátor by se soustředil primárně na simulaci šíření požáru, na rozdíl od Wildfire Exploreru, který se věnuje spíše jeho hašení. Přesto bychom však chtěli, aby bylo možné do budoucna relativně snadno tuto funkcionalitu doplnit. Měli bychom být tedy schopni přidat do simulátoru právě věci, jako jsou například výše zmíněný Helitack nebo požární linie, aniž by to vyžadovalo složité úpravy celého fungování našeho programu.

Wildfire Explorer sám o sobě nevysvětluje ani nevzdělává uživatele, jaké faktory tedy vlastně ovlivňují ono samotné šíření ohně, což považujeme za nevyužitý potenciál. Jednotlivým částem mapy můžeme pouze měnit základní vlastnosti, jako je například vlhkost. Můžeme zde sice definovat rychlost a směr větru celého světa, ale líbilo by se nám, kdyby zde bylo víc parametrů, například jako je třeba typ vegetace. Také by bylo užitečné, kdybychom mohli měnit reliéf krajiny a vidět tak simulaci v různých terénech s rozdílnými parametry.

2.1.2 FireSpreadingSimulation

Tento program je napsaný v herním enginu Unity a přináší uživatelům zajímavé možnosti manipulace s prostředím, ačkoliv má také několik omezení. Tyto omezující vlastnosti si popíšeme a vysvětlíme si, proč může být vhodné je dělat odlišně.

Projekt je dostupný na GitHubu [4].

Program má subjektivně docela dobré uživatelské prostředí, ačkoliv by podle nás mohlo být o mnoho lepší. Bohužel zde platí podobné nedostatky jako u většiny ostatních programů, například nemožnost měnit krajinu, velmi omezené vlastnosti faktorů šíření ohně nebo nedostatečné vysvětlení samotného fungování programu pro nové uživatele a podobně.

Obrázek 2.2 ukazuje rozhraní samotného programu.

Program nabízí možnost upravit směr a sílu větru. Dále tento program zahrnuje několik základních funkcí za pomoci tlačítek, což umožňuje uživatelům interagovat s prostředím simulace.

Za pomoci tlačítka `Generate` se náhodně a automaticky na mapě vytvoří objekty (rostliny) pro zapálení.

Tlačítko `Clear` umožňuje odstranit všechny objekty ze scény. Tato funkce je užitečná pro resetování simulace a přípravu prostředí na nové experimenty bez nutnosti restartování celého programu. Obdobu tohoto tlačítka budeme chtít v našem simulátoru také.

Poznatky pro náš simulátor

Uživatelé mají možnost přidávat nebo odebírat ze scény objekty pouhým kliknutím. Tyto objekty mohou navíc být i zapáleny a tedy zapojeny do simulace požáru, což rozšiřuje možnosti simulace tím, že umožňuje testování různých scénářů a konfigurací, což také hodnotíme pozitivně.

Jak jsme již zmínili, současná verze programu nenabízí možnost upravovat terén ani měnit pohled kamery. Toto omezení může zhoršit zážitek pro koncové uživatele, tedy pro ty, kteří se neorientují v programování nebo nechtějí do kódu programu zasahovat. To tím pádem pro tuto skupinu uživatelů významně snižuje jejich prostor a možnosti pro experimentování.

Vlastnosti šíření požáru jsou zde velmi základní, což může dále snižovat realističnost a užitečnost simulace pro některé uživatelské scénáře. Program podle nás není dostatečně připravený na rozšiřování vlastností simulace bez rozsáhlejších úprav.

Další problém je třeba fakt, že simulace spoléhá na konkrétní komponenty Unity (zde ve verzi 2019.1.9f). Využití komponenty `Collider` pro simulaci šíření požáru je sice do jisté míry efektivním řešením pro detekci kolizí a interakci mezi objekty, ale může to ztížit sledování simulace, zejména pro ty, kteří nejsou obeznámeni s Unity Editorem. Bylo by vhodnější, pokud by logika simulace a její vizualizace byly navrženy tak, aby byly do jisté míry nezávislé na specifických Unity komponentách, nebo aby tyto komponenty využívaly jen minimálně.

Závislost na komponentách, jako je právě výše zmíněný `Collider`, které se konfigurují zejména v Unity Editoru a navíc se za běhu programu mění s různým směrem a sílou větru, činí jakékoli úpravy simulace bez hlubokého porozumění Unity Editoru téměř nemožnými. To významně ztěžuje uživatelům bez předchozích zkušeností s Unity možnost přizpůsobení simulace svým specifickým potřebám.

Podle nás by bylo lepší částečně oddělit vizualizaci a samotnou logiku simulace v kódu (skriptech). Simulace by nemusela záviset na komponentách specifických pro daný vizualizující nástroj vůbec, nebo by je využívala jen minimálně a jednoduše. Tento přístup se budeme snažit uplatnit v našem simulátoru.

2.1.3 ForeFire

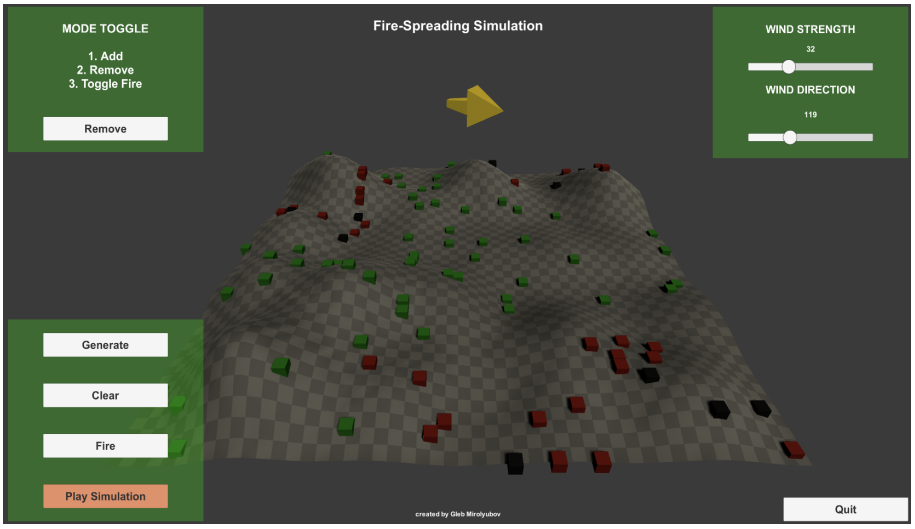
ForeFire je open-source (s otevřeným zdrojovým kódem) projekt pro modelování šíření požárů v divoké přírodě, vyvinutý Universitou de Corse Pascal Paoli [5]. Program dokáže zpracovávat běžná data krajiny a je navržený pro rozsáhlé simulace požárů v Unix systémech.

ForeFire je napsán především v C a C++ jazyce, což mu umožňuje efektivní zpracování a flexibilitu pro integraci s různými programovacími prostředími a nástroji, včetně Pythonu, kde lze využít balíčky jako SciPy a Numpy, a také Fortranu pro integraci s numerickými modely předpovědi počasí.

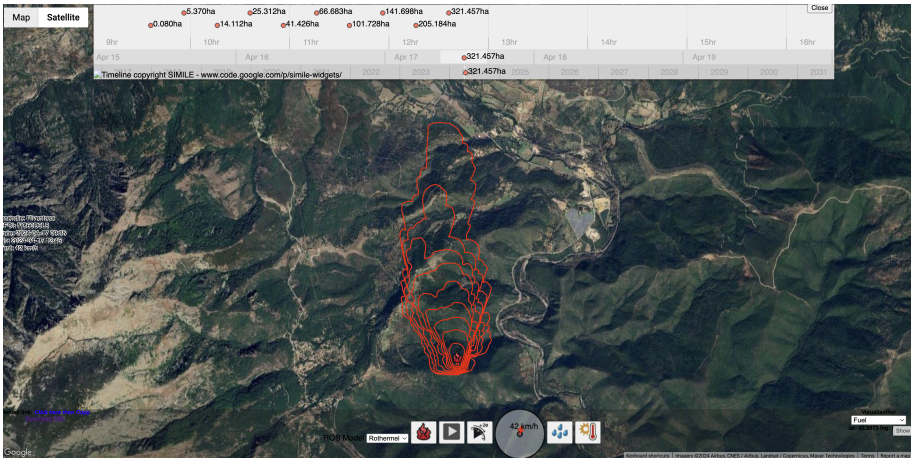
Obrázek 2.3 ukazuje rozhraní samotného programu.

Poznatky pro náš simulátor

Oceňujeme že nástroj nabízí možnost simulace přímo na mapě reálného světa. Uživatel si v jednoduché demoverzi může najet za pomoci Google map na část



Obrázek 2.2 Ukázka jedné z existujících aplikací napsaných v Unity.



Obrázek 2.3 Demo ukázka aplikace FireFront.

světa, která ho zajímá. Dále je možnost zvolit si velice základní vlastnosti mapy jako například vlhkost či směr a sílu větru.

V levém dolním rohu aplikace je navíc možnost zvolit si model na použití pro predikci. Jedná se o takzvaný *ROS* („Rate of Spread“) model. Tyto modely mají zásadní význam pro předpověď rychlosti a způsobu šíření požáru za různých podmínek prostředí.

Možnost měnit použitý model je funkce, která se nám opravdu velmi líbí. Umožňuje do určité míry zkoumat rozmanitost šíření ohně v takových simulátorech a to za specifických podmínek s jinými vlastnostmi modelu. Tuto myšlenku bychom chtěli v našem simulátoru využít jako inspiraci a dovolit uživatelům alespoň částečně upravovat parametry modelu simulace přímo z aplikace.

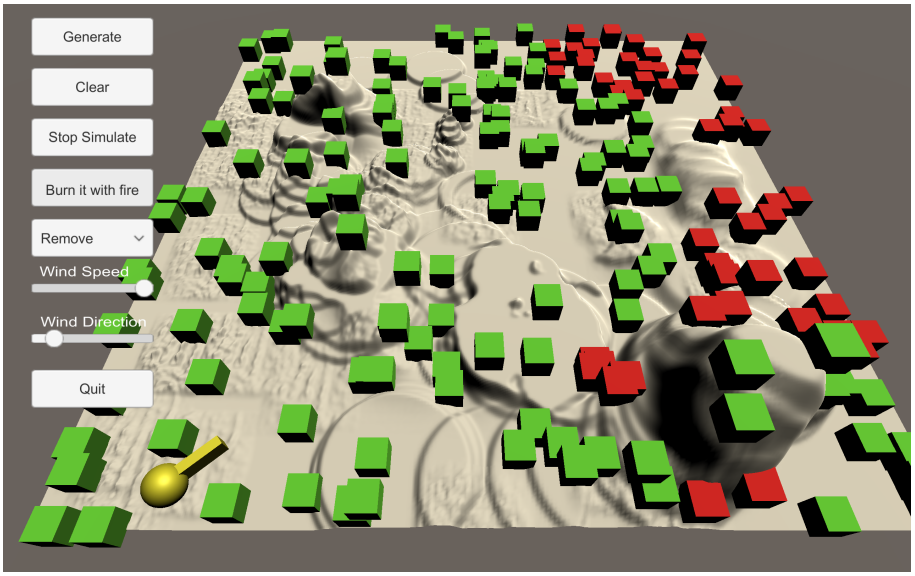
2.1.4 Fire-spreading-Simulation

Na platformě GitHub můžeme dále najít velké množství projektů podobných tomuto.

Jedná se o velmi jednoduchý simulátor ohně, který je dostupný jako projekt na GitHubu zde [6]. Bohužel se jedná o starší projekt. Aplikace se zdá být nedokončená. Projekt se jen těžko dá převzít a nějak smysluplně využít. Kód není napsaný čitelně a chybí jakákoliv dokumentace.

Tlačítko `Generate` rozmístí náhodně body. Ty se dají přidávat i odebírat. V podstatě se ale nedá říct co se v aplikaci děje ačkoliv některé body opravdu zčervenají (shoří) po spuštění simulace.

Obrázek 2.4 ukazuje rozhraní samotného programu.



Obrázek 2.4 Ukázka velmi jednoduchého a volně dostupného simulátoru.

Poznatky pro náš simulátor

Bohužel značné množství podobných aplikací často bývají velmi uživatelsky nepřívětivé, jako je tomu například právě u tohoto projektu. To může mít za následek špatný uživatelský zážitek, protože uživatelé mohou být zmatení, co se vlastně doopravdy v aplikaci děje.

2.1.5 Shrnutí

Prozkoumali jsme množství aplikací volně dostupných na internetu. Po prozkoumání těchto a mnoha dalších již existujících simulátorů jsme narazili na podobné nedostatky a nyní máme lepší představu o tom, jaké nástroje dnes již existují a jakých věcí se budeme chtít držet. Žádný z existujících programů úplně nepokrývá množinu funkcionalit, kterou jsme si představili v úvodu (1.3.1), v jedné aplikaci. Buď se jedná o příliš jednoduché simulátory, nebo naopak jsou tyto simulátory až příliš komplexní a podrobné, většinou bez přívětivého uživatelského prostředí pro běžné uživatele. Chceme tedy najít takový zlatý střed.

Pojďme si nyní shrnout, na co se zaměříme a jakou kombinaci funkcionalit uživatelům poskytneme.

- Snadnost používání - Rozhraní bude vcelku intuitivní a přístupné i pro uživatele bez větších technických zkušeností. Zaměříme se na jasnější vedení uživatele skrze simulaci, od generování krajiny až po analýzu výsledků.
- Generování krajiny (terénu) - Umožníme uživatelům generovat různorodé terény a také s již existujícími manipulovat, což poskytne větší flexibilitu pro simulaci různých scénářů.
- Možnost ukládání a načítání krajiny - Simulátor umožní libovolné ukládání a načítání různých krajin. To uživatelům umožní snadno se vracet a dále pracovat na svých projektech.
- Zjednodušená pravidla simulace - Simulátor nebude využívat žádné složité modely šíření ohně, ale jednodušší, diskrétní kroky.
- Dobrý vizuální dojem - Neopomeneme ani vizuální stránku, která bude podávat docela realistický a vizuálně působivý dojem.
- Logika tříd světa a simulace oddělena od jejich vizualizace - Zvýšíme modularitu a udržitelnost kódu, který nebude natolik závislý na použitém vývojovém prostředí. Usnadníme tím další rozvoj a úpravy simulátoru.
- Rozšiřitelnost programu - Chceme, aby logika simulace nebyla omezena specifickým ovládáním. Například vcelku jednoduchým složením již existujících funkcí nebo drobnou úpravou bude možné implementovat nějaké nové rozumné rozšíření (např. zmíněný Helitack).
- Predikce rozšíření - V simulátoru bude možné si nechat zobrazit predikci rozšíření ohně v závislosti na specifikovaných podmínkách a počátečním zdroji ohně.
- Možnost jednoduchého propojení s externími skripty - Chceme poskytnout možnost programátorům, kteří nechtějí upravovat zdrojový kód simulátoru, aby mohli v jiném programovacím jazyce vytvořit skript, jehož výsledky bude možné načíst a zobrazit v simulátoru (například vlastní predikce).

2.2 Cíloví uživatelé

Podle těchto prací si teď především upřesníme pro koho chceme, aby náš simulátor zejména vlastně byl.

Existují dva hlavní typy cílových uživatelů.

První skupinu tvoří ti, kteří potřebují snadno použitelný nástroj k jednoduché vizualizaci a simulaci šíření ohně bez nutnosti zabývat se technickými detaily nebo programováním. V této skupině mohou být například vyučující, kteří hledají podpůrnou pomůcku, která by oživila výuku pro mladší děti. Dále například hasiči předvádějící preventivní opatření na akcích pro veřejnost. Nebo třeba ti, kteří na simulátor náhodou narazí a kterým by tak mohl poskytnout příjemnou inspiraci a motivaci dozvědět se více o dynamice nebo prevenci požárů.

Druhou skupinou mohou být začínající programátoři a vývojáři, kteří chtějí použít simulátor jako nástroj pro hezkou vizualizaci svých dat nebo jako součást většího projektu. Díky základům simulátoru v jednoduchých diskrétních krocích může například sloužit právě jako opora pro ty, kteří se seznamují s principy programování.

2.3 Cílová platforma

Mobilní zařízení dnes disponují více než dostatečným výkonem pro naši představu jednoduchého simulátoru a jsou velice dostupné a přenositelné.

Pro naše účely se však PC platforma (osobní počítače a notebooky) jeví jako lepší volba. Větší obrazovka znamená lepší vizuální prostor pro simulaci. Kontrola pomocí myši a klávesnice umožňuje uživatelům efektivnější a přesnější interakce. V neposlední řadě je PC primární platforma pro vývoj softwaru. To tak umožní snazší, hlubší a efektivnější zasahování do struktury a logiky simulátoru, což je pro naši druhou skupinu cílových uživatelů velmi důležité. PC platforma tedy bezpochyby lépe vyhovuje potřebám vývojářů.

Aplikaci budeme vyvíjet pro operační systémy Microsoft Windows a macOS, které společně tvoří více jak 85 % trhu s operačními systémy pro osobní počítače na celém světě [7]. Microsoft Windows je dlouhodobě nejrozšířenější operační systém pro osobní počítače a zaujímá významnou část trhu. Operační systém macOS je preferován a používán autorem této práce.

2.4 Implementační nástroj

Volba implementačního nástroje je také jedno z velmi důležitých rozhodnutí, od kterého se bude vyvíjet postup a způsob implementace našeho simulátoru. Bude totiž určovat konečné celkové možnosti simulátoru. Bude mít vliv nejen na to, jakým způsobem a jak efektivně se bude simulátor vyvíjet, ale i na budoucí přizpůsobitelnost našeho simulátoru.

Protože ten kdo zasahuje do programu nemusí být vždy jen autor, ale potenciálně také ostatní vývojáři, kteří by si ho chtěli do jisté míry přizpůsobit svým potřebám. Navíc je potřeba, aby to byl nástroj známý, rozšířený a dobře se s ním pracovalo.

Herní engine

Herní engine je v podstatě takový softwarový balíček určený pro vývoj a design videoher [8]. Herní enginy, navzdory svému názvu, však nejsou omezeny pouze na vývoj her. Jejich široký rozsah funkcí a adaptabilita je činí vhodnými pro mnohem širší spektrum aplikací. Poskytuje vývojářům různé sady nástrojů a knihoven, které usnadňují grafické zobrazení, manipulaci s audiovizuálním obsahem, správu paměti a mnoho dalších aspektů, které jsou nezbytné pro moderní videohry a simulátory.

Chceme si tedy vybrat herní engine, který nám nabídne komplexní řešení a odstraní potřebu zabývat se moc technickými aspekty projektu, které nejsou přímo spojené s naším simulátorem. Vyvíjet vlastní herní engine není naším cílem a zbytečně by to komplikovalo celý vývoj.

Herní engine, který si pro tento účel vybereme, nám musí poskytnout dostatečnou flexibilitu a rozšiřitelnost. Prozkoumáme různé herní enginy, které jsou aktuálně populární v herním průmyslu, abychom určili, který z nich bude pro nás nejvhodnější.

Engine musí nabízet rozsáhlé možnosti pro design uživatelského rozhraní, interakci s uživateli a podpory různých vstupních zařízení.

Také je důležité, aby byl dostatečně rozšířený a populární ve vývojářské komunitě. To znamená, že existuje bohatý ekosystém doplňků, dokumentace a komunitní podpora, které mohou vývojářům pomoci při řešení všemožných problémů. Tím se zvyšuje šance, že simulátor bude moci být snadno přizpůsobován a rozšiřován v budoucnu, což je klíčové pro jeho dlouhodobou udržitelnost a úspěch.

Výběr enginu

Při výběru zvážíme zejména také:

- Renderování 3D objektů a správa materiálů a dalších herních zdrojů.
- Jednoduchost uživatelského rozhraní.
- Kvalitní vývojářská dokumentace a aktivní komunita vývojářů.
- Podpora programovacích jazyků.

Na základě zkušeností autora upřednostňujeme programovací jazyky C#, Python nebo C++.

Při hodnocení dostupných herních enginů jsme narazili na množství možností, mezi nejpopulárnější a nejvhodnější patří *Godot* [9], *Unity* [10] a *Unreal Engine* [11]. Všechny tyto platformy splňují do určité míry naše požadavky. Všechny navíc umožňují vývoj pro naše vybrané cílové operační systémy, Windows a macOS.

Renderování 3D objektů a správa materiálů a dalších herních zdrojů.

- *Godot*: Poskytuje flexibilní systém pro práci s 3D objekty a materiály. Při porovnání s *Unity* a *Unreal Engine* nabízí méně pokročilé nástroje a možnosti vizualizace. Zaostává v rozsahu podporovaných funkcí.

- *Unity*: Vyniká v renderování 3D objektů a efektivní správě scén. Je vhodný pro různorodé projekty a nabízí širokou škálu nástrojů pro práci s nejen 3D grafikou.
- *Unreal Engine*: Nabízí velmi výkonné nástroje pro vizuálně náročné projekty s pokročilými možnostmi renderování a správou materiálů [12].

Jednoduchost uživatelského rozhraní.

- *Godot*: Má relativně jednoduché a přizpůsobitelné uživatelské rozhraní.
- *Unity*: Nabízí docela dobře navržené uživatelské rozhraní s intuitivními nástroji a rozsáhlou podporou pro návrhy uživatelského prostředí.
- *Unreal Engine*: Rozhraní je spíš více zaměřené na pokročilé uživatele a projekty s vysokou vizuální složitostí [13].

Kvalitní vývojářská dokumentace a aktivní komunita vývojářů

- *Godot*: Má rostoucí komunitu s podpůrnými zdroji, avšak menší oproti *Unity* a *Unreal Engine*.
- *Unity*: Vyniká v kvalitní dokumentaci a má velmi aktivní a rozsáhlou komunitu.
- *Unreal Engine*: Také disponuje rozsáhlou a aktivní komunitou s množstvím zdrojů. Oproti *Unity* se nám zdá, že má horší dokumentaci.

Podpora programovacích jazyků

- *Godot*: Podporuje C#, C++ a GDScript.
- *Unity*: Především C#.
- *Unreal Engine*: Primárně využívá C++.

Zhodnocení

Výběr závisí na analýze výše zmíněných vlastností a na osobních preferencích autora. Autor sám předtím nikdy v životě nepracoval ani s jedním ze zmíněných enginů.

Všechny tři enginy jsme si pro tuto práci vyzkoušeli. Autorovy zkušenosti s Unreal Engine a Godotem nebyly příliš pozitivní, což bylo způsobeno především nedostatky v jejich dokumentacích a nefunkčními ukázkovými C++ řešeními problémů. Z těchto důvodů jsme se rozhodli pro vývoj v Unity.

Unity je známé svou schopností adaptovat se na různé typy projektů, od jednoduchých mobilních her až po komplexní simulátory. Nabízí možnost jednoduše exportovat projekt do široké řady platforem včetně Windows, MacOS, Linux, iOS, Android a mnoha dalších, což rozšiřuje potenciální dosah našeho simulátoru do budoucna. Nabízí dobrý kompromis mezi vizuální kvalitou a výkonem, s výbornou podporou pro různé platformy.

S rozsáhlou a aktivní komunitou vývojářů a bohatou sadou dokumentace je mnohem snazší nalézt řešení problémů a při vývoji získat pomoc.

Unity navíc exceluje v renderování 3D objektů a efektivní správě scén, což je nezbytné pro vizuálně působivé simulace.

Navíc využívá programovací jazyk C#, který je autorův oblíbený. Jazyk má vhodnou úroveň abstrakce, což z něj dělá ideální volbu pro projekt, kde je důležitá rychlost vývoje a snadná udržovatelnost kódu. Jedná se o dobře čitelný programovací jazyk. Jeho syntaxe je podle autora čistá a intuitivní, což usnadňuje učení a snižuje riziko chyb.

3 Unity herní engine

V této sekci si představíme vývojářský nástroj Unity. Zaměříme se především na klíčové aspekty tohoto enginu. Tyto základní vlastnosti jsou nutností pro širší pochopení realizace našeho projektu, který pak bude podrobněji popsán v dalších kapitolách.

Tato kapitola je určena těm, kteří s Unity nemají žádné či jen malé zkušenosti, a přesto by chtěli tento projekt lépe pochopit, například pro vlastní potřebu úprav projektu. Dočtete se zde stručné informace o tom, jak Unity engine funguje jako takový, jak s ním pracovat a k čemu která komponenta slouží.

Tato část by tedy mohla působit jako úvodní průvodce pro programátory, kteří s Unity nikdy nepracovali, nebo si chtějí osvěžit paměť ohledně základních konceptů a paradigmat, které jsou pro práci s tímto nástrojem klíčové.

Pro ty, kteří hledají hlubší pochopení tohoto nástroje, doporučujeme nahlédnout na podrobnější oficiální uživatelský manuál [10] nebo oficiální Unity tutoriály [14].

3.1 Úvodní přehled

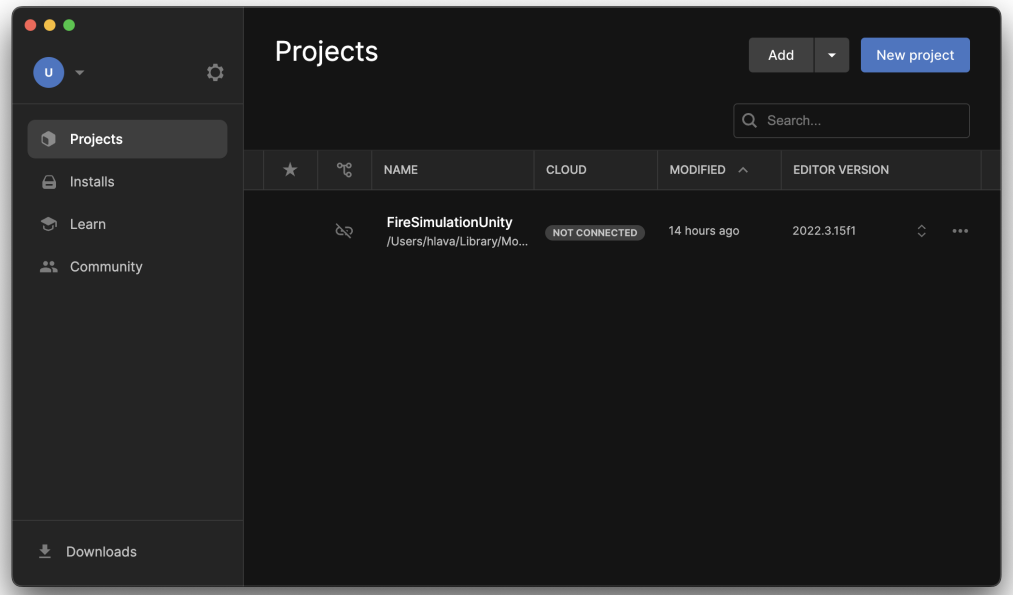
Unity je herní engine a vývojové prostředí, které nám umožňuje vytvářet širokou škálu interaktivních 2D i 3D her a aplikací. Má uživatelsky velice přívětivé rozhraní, které je snadno použitelné i pro začátečníky. Je to nástroj velmi flexibilní se schopností publikovat vytvořené projekty na různé platformy. Mezi tyto platformy patří například mobilní telefony, herní konzole, osobní počítače a zařízení pro virtuální a rozšířenou realitu.

Zprvu si vysvětlíme některé základní navigační části a komponenty Unity, abychom se v tomto docela komplexním systému lépe vyznali. Doporučujeme si přečíst základní představení tohoto editoru a poté pro hlubší porozumění si číst přiloženou vývojářskou dokumentaci tohoto simulátoru současně se zkoušením si některých těchto konceptů přímo v Unity. Doporučujeme zkoušet upravovat některé menší a méně důležité části tohoto simulátoru. Autor je přesvědčen, že pro dostatečné pochopení si nestačí o těchto věcech pouze číst, ale je třeba vidět a osobně zkoušet a experimentovat. Tento simulátor je autorův první projekt v Unity v životě vůbec, a proto tvrdí, že se samotné základní fungování a principy ovládání dají opravdu velmi rychle pochopit a naučit.

Pokud by tyto koncepty nebyly i přesto zcela jasné, nebo by chtěl uživatel pochopit dané věci ještě více do hloubky, tak Unity má širokou základnu tutoriálů na Youtube, kde je navíc vizuálně hned jasnější, jak se s nástrojem může zacházet a pracovat. Zdrojů informací je převážně díky velmi početné a aktivní komunitě na internetu opravdu velké množství. Vřele doporučujeme hledat a využívat tyto zdroje nejen v průběhu vývojového procesu, protože je pravděpodobné, že s podobnými nejasnostmi nebo problémy se už před vámi mnozí také potýkali.

3.2 Spuštění

Po stažení a instalaci Unity a spuštění hlavního programu Unity Hub (obr. 3.1) se ocitneme v centrálním bodě pro správu Unity projektů. Unity Hub slouží jako spouštěč a organizátor pro různé verze Unity Editoru a projektů. V pravém horním rohu okna najdeme možnost „Add project from disk“, kterou použijete pro přidání existujícího projektu. Stačí vybrat adresář validního projektu a projekt se sám otevře již v Unity Editoru.



Obrázek 3.1 Ukázka rozhraní aplikace Unity Hub.

3.3 Editor

Unity Editor je centrem všeho vývoje v Unity. Nabízí nástroje pro vizuální editaci scén, animace, správu aktiv a debuggování (ladění) svého programu. Editor je navržen tak, aby podporoval rychlý vývoj a prototypování, což umožňuje vývojářům rychle testovat a upravovat své programy v reálném čase. Kromě toho Unity poskytuje *Asset Store*, obrovský trh s aktivy a nástroji třetích stran, které mohou vývojáři použít k obohacení svých projektů a urychlení vývojového procesu.

Na obrázku 3.2 je ukázka, jak může vypadat takové rozhraní Unity Editoru.

Nyní si postupně pro lepší orientaci rozebereme jednotlivé části Unity.

3.4 GameObject

`GameObject` neboli herní objekt je základní stavební blok pro vytváření scén v Unity [15]. Herní objekty mohou reprezentovat od předmětů a postav přes světla a text až po kamery, opravdu velmi mnoho typů objektů.

Herní objekty jsou uspořádané do hierarchie, kde každý objekt může být v roli rodiče či potomka vůči dalším objektům.

Na obrázku 3.3 můžeme vidět, jak může vypadat takové okno s objekty a jejich hierarchií.

Herní objekty samy o sobě neobsahují žádné chování a mají jen základní vlastnosti, jako je například pozice v prostoru. Chování a další specifické vlastnosti jsou přidávány prostřednictvím komponent. Komponenty mohou být různého typu, včetně vlastních skriptů, grafických komponent, fyzikálních komponent apod. Pomocí těchto komponent můžete definovat, jak se objekty chovají nebo jak vypadají.

Obecně můžeme herní objekty rozdělit na statické (např. krajina) nebo dynamické (např. postavy). Statické herní objekty jsou ty, které se během hry nebudou pohybovat. Systém osvětlení v Unity díky tomu, že je nějaký objekt statický, může předpočítat světelné interakce s těmito objekty, což může výrazně snížit výpočetní náročnost osvětlení ve scéně a zlepšit výkon aplikace.

3.5 Scene, Game view

Scéna je takový kontejner, který drží všechny herní objekty aplikace. Tím vlastně scéna představuje úroveň nebo také prostředí hry.

Různých scén s různými objekty můžeme mít v aplikaci více a přepínat mezi nimi. To buď přímo v editoru nebo dokonce i za běhu programu za pomocí skriptů. Musíme však mít na paměti, že při změně scén musí Unity uvolnit všechny herní objekty a jiné „assety“ současné scény z paměti a načíst nové pro další scénu. To může způsobit dočasný pokles výkonu, zvlášť pokud je objektů hodně.

Používat scény pro větší aplikace je doporučené. Mohou totiž být použity pro různé účely, jako je třeba hlavní menu, nastavení, různé úrovně hry nebo třeba pro testování určitých funkcí během vývoje.

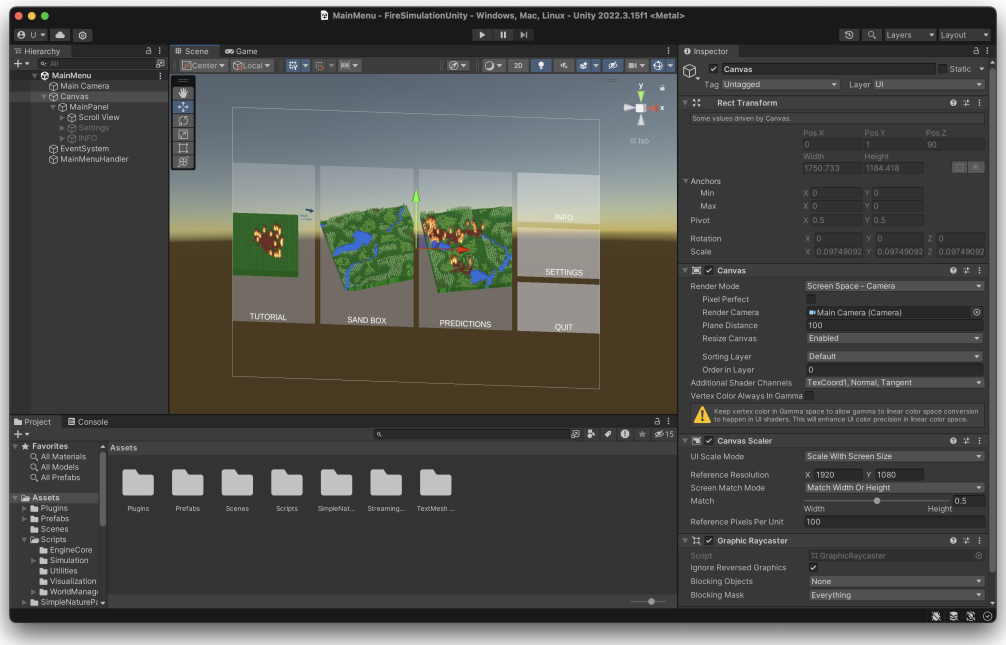
Záložka `Game` v Unity Editoru slouží jako náhled toho, co uživatel uvidí, když bude aplikace (hra) spuštěna. Zobrazuje výstup z aktivních kamer ve scéně. S aplikací můžeme interagovat již přímo v editoru, což umožňuje rychle testovat a upravovat chování a vzhled herních objektů bez nutnosti kompletního buildu (plného sestavení) celé aplikace.

Na obrázku 3.4 můžeme vidět, jak vypadá záložka scény (`Scene`) a hry (`Game`), mezi kterými můžeme v Unity přepínat.

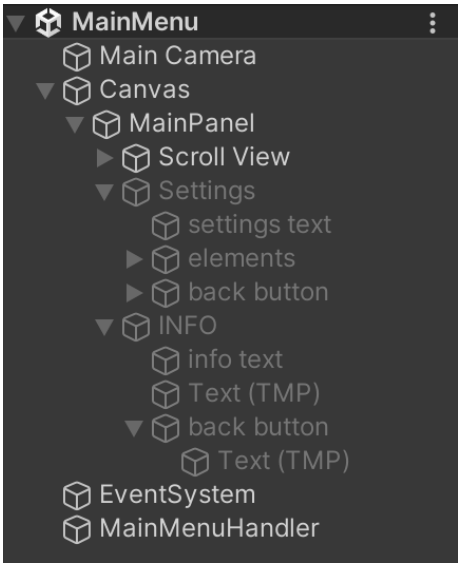
3.6 Project

Záložka `Project` v Unity umožňuje správu složek a souborů v našem projektu. Zde se obvykle nachází všechny zdroje, skripty, materiály, textury a další soubory, které jsou součástí celého projektu. Tato záložka je organizována do hierarchie složek, což umožňuje například snazší navigaci a organizaci mezi soubory a to přímo z editoru. Není tedy třeba se k souborům dostávat přes správce souborů operačního systému. Jednoduchým způsobem se pak dají například skripty a nebo Prefaby, o kterých bude řeč později, vložit přímo do scény nebo připojit k danému objektu pouhým přetažením.

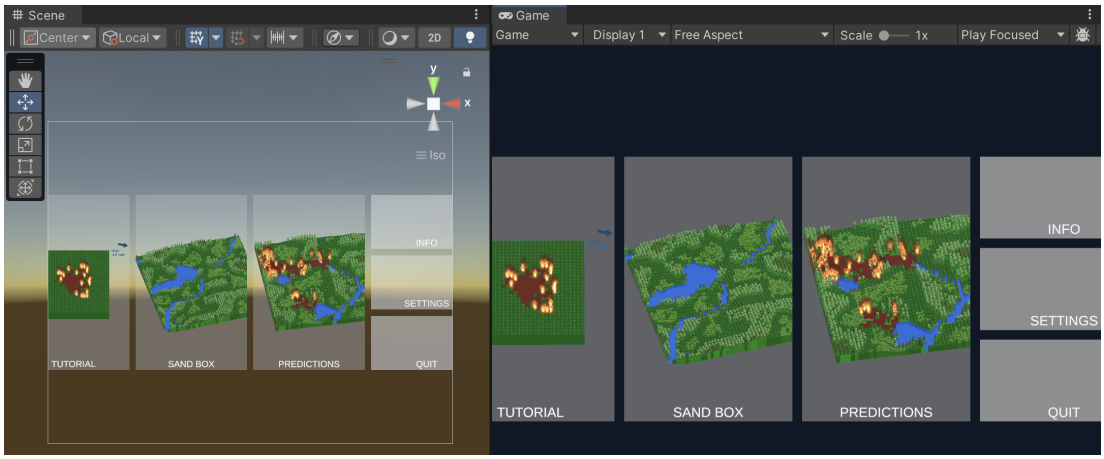
Na obrázku 3.5 můžeme vidět, jak vypadá okno projektu v Unity Editoru.



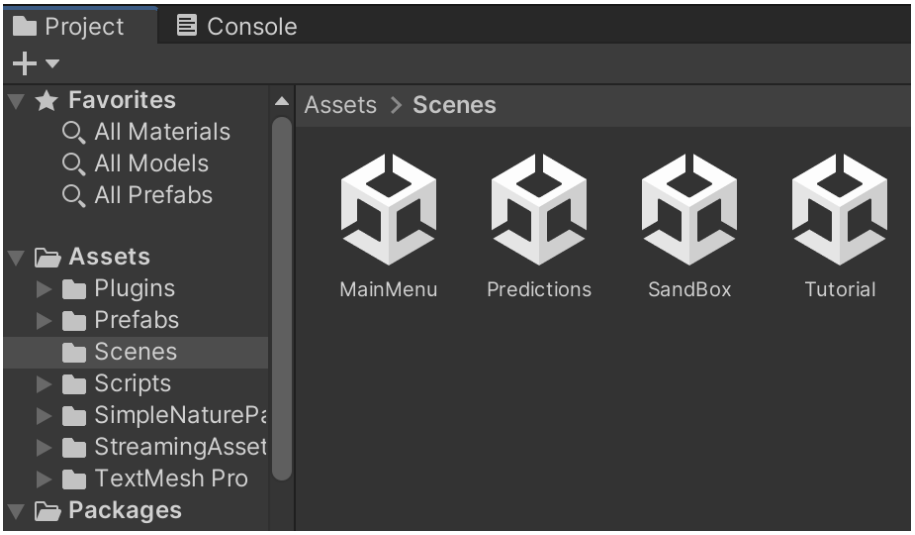
Obrázek 3.2 Ukázka rozhraní Unity Editoru.



Obrázek 3.3 Ukázka rozhraní okna s herními objekty (GameObject) aplikace Unity Editor.



Obrázek 3.4 Ukázka rozhraní oken Scene and Game aplikace Unity Editor.



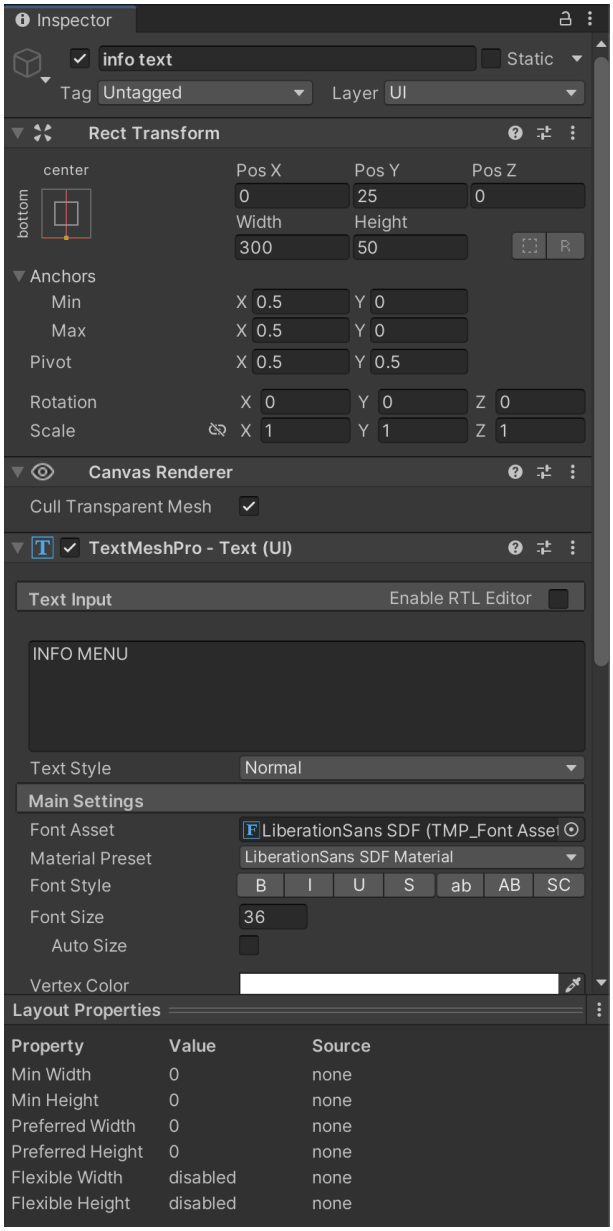
Obrázek 3.5 Ukázka rozhraní okna projektu aplikace Unity Editor.

3.7 Inspector

`Inspector` se používá k zobrazení a úpravě vlastností herních objektů a jejich komponent. Když vyberete herní objekt ve scéně, tak inspektor automaticky zobrazí všechny přiřazené komponenty tohoto objektu a umožní upravit jejich vlastnosti.

Komponenty přiřazené herním objektům se zobrazují ve vertikálním uspořádání, v pořadí v jakém byly přidané. Jejich pořadí můžeme změnit. Komponenty nebo skripty jsou aplikovány či vykonávány od shora dolů.

Na obrázku 3.6 můžeme vidět, jak může vypadat taková záložka Inspector v Unity Editoru.



Obrázek 3.6 Ukázka rozhraní okna Inspector aplikace Unity Editor.

3.7.1 Komponentový systém

Jedním z klíčových rysů Unity je jeho komponentový systém, který vývojářům umožňuje definovat chování herních objektů přidáváním nezávislých komponent objektům. Tyto komponenty mohou být vše od vizuálních aspektů, jako jsou meshe (sítě, množiny bodů) a textury, přes fyzikální vlastnosti, jako jsou hmotnost a tření, až po skripty, které definují vlastní logiku a chování objektu.

Díky komponentovému systému mohou vývojáři snadno experimentovat, iterovat a upravovat své projekty bez nutnosti zásadních změn kódu. Tento modulární design nejenže zvyšuje efektivitu a flexibilitu vývoje, ale také podporuje lepší spolupráci mezi členy týmu, protože umožňuje individuální práci na komponentách bez rizika zásadnějších konfliktů.

Navíc, systém komponent v Unity usnadňuje opětovnou použitelnost kódu, umožňuje vývojářům snadno sdílet a opětovně používat komponenty napříč různými projekty, což dále zvyšuje produktivitu a snižuje čas potřebný k vývoji.

Camera

Komponenta `Camera` v Unity definuje „oči“ hráče ve virtuálním světě [16]. Určuje, co a jak bude hráč vidět na svém displeji. Každá kamera v scéně může mít různá nastavení, jako je zorné pole, clipping planes (hranice, za které už kamera nevidí), nebo specifické efekty, jako je hloubka ostrosti nebo motion blur (rozmazání pohybu). Kamery lze také používat pro vytváření mini-map, zobrazení objektů z různých úhlů apod.

Light

Světla v Unity dodávají scéně atmosféru a realismus tím, že simulují různé typy světelných zdrojů [17]. Existují různé typy světel, včetně `Directional Light` pro simulaci slunce, `Point Light` pro simulaci žárovky nebo `Spot Light` pro simulaci světlometu. Světla mohou být nastavena s různou intenzitou, barvou nebo třeba vlastnostmi stínů.

Renderer

`Renderer` je obecný termín pro komponenty v Unity takového typu, které zajišťují vykreslování objektů na obrazovku [18]. Pro různé typy objektů existují různé druhy rendererů, jako například `MeshRenderer` pro 3D modely nebo `SpriteRenderer` pro 2D sprites (obrázky). `Renderer` se stará o to, aby byl objekt viditelný, a umožňuje nastavení materiálů, textur a shaderů, které dále definují vzhled objektu.

Canvas

`Canvas` je komponenta používaná pro vytváření uživatelského rozhraní (UI) ve hrách a aplikacích v Unity [19]. Slouží jako kontejner pro další UI prvky. Mohou jimi být například texty (`Text - TextMeshPro`), tlačítka (`Button`), obrázky (`Image`), posouvátka (`Slider`), přepínací tlačítka (`Toggle`) a zadávací políčka (`Input field`). `Canvas` může být nastaven tak, aby UI elementy byly vždy vykresleny nad herními objekty, nebo aby se pohybovaly a měnily velikost v závislosti

na rozlišení a orientaci obrazovky, což je zásadní pro tvorbu responzivního uživatelského rozhraní (optimalizovaného pro různá zařízení).

Canvas Scaler

Komponenta `Canvas Scaler` v Unity je nástroj, který pomáhá zajistit, aby se uživatelské rozhraní správně zobrazovalo při různých rozlišeních a velikostech obrazovek. Tato komponenta je přidána objektu `Canvas` a umožňuje nastavit pravidla pro škálování UI elementů. Můžeme zvolit mezi různými režimy škálování, jako je konstantní fyzická velikost, škálování podle šířky nebo výšky okna apod. Díky tomu je UI responzivní a vypadá dobře na jakémkoliv zařízení, od mobilních telefonů po velké monitory.

Scroll Rect

Komponenta `Scroll Rect` se používá v rámci UI pro vytvoření oblasti, kterou lze procházet posouváním. Toto je zvláště užitečné, když chcete zobrazit více obsahu, než se vejde do oblasti obrazovky, například seznam položek nebo dlouhý text. `Scroll Rect` může být nastaven tak, aby umožňoval posouvání horizontálně, vertikálně nebo oběma směry. K jeho funkčnosti je obvykle nutné přidat komponenty jako `Mask` pro oříznutí obsahu mimo určitou oblast a `Scrollbar` pro zobrazení posuvníku.

Transform

Každý herní objekt v Unity má komponentu `Transform`, která určuje jeho polohu, rotaci a škálování (natažení) v prostoru. `Transform` může také odkazovat na `Transform` komponentu jiného objektu, čímž se vytváří zmíněná hierarchická struktura.

Je důležité rozlišovat mezi lokálními a globálními souřadnicemi. Lokální souřadnice určují polohu, rotaci a škálování vzhledem k předkovi, zatímco globální souřadnice reprezentují skutečnou pozici v herním světě. Unity poskytuje metody pro konverzi mezi těmito souřadnicovými systémy.

Rigidbody a Collider

`Rigidbody` je komponenta, která přidává fyzikální vlastnosti herním objektům, umožňuje jim například podléhat gravitaci a reagovat na síly. Mezi tyto vlastnosti patří například hmotnost, tření a odpor vzduchu. Díky této komponentě mohou herní objekty provádět realistické pohyby a interakce mezi sebou, jako jsou odrazy od sebe, skluz apod.

Často je také `Rigidbody` používán s komponentou `Collider`. `Collider` umožňuje detekci a reakci na kolize s jinými objekty. Definuje v podstatě tvar objektu pro účely této kolize. `Collider` může být například tvaru koule (`Sphere Collider`), krychle (`Box Collider`), nebo může přesně kopírovat tvar modelu (`Mesh Collider`).

3.8 Prefab

`Prefab` se dá chápat jako předpřipravený objekt. Jedná se v podstatě o šablonu pro objekty. Prefaby můžeme vytvářet předem a pak instanciovat v rámci herní scény nebo skriptů podle potřeby [20].

Tento fakt vede k několika výhodám.

Výhodou je například možnost opakovaného použití. Můžeme nastavit komponenty a vlastnosti jednou a pak je používat opakovaně, třeba i klidně ve více scénách nebo projektech.

Pokud se změní vlastnosti Prefabu, změny se automaticky projeví ve všech instancích tohoto Prefabu v projektu. Vlastnosti jednotlivých instancí Prefabu můžeme měnit a upravovat, bez nutnosti změny původního Prefebu. Tyto změny pak mají přednost před vlastnostmi této šablony Prefabu.

3.9 Programování

V Unity se programování herní logiky provádí v jazyce C#. Kompilátor Roslyn [21], který Unity využívá, podporuje verzi C# 9.0, ale s určitými omezeními.

Nepodporované (C# 9.0 v Unity)

- Potlačení emitování lokálního flagu `localsinit`.
- Kovariantní návratové typy.
- Modulové inicializátory.
- Init-only settery.

Použití nepodporovaných metod pak vede k chybám při kompilaci skriptů.

3.9.1 MonoBehaviour

Základem pro vytváření nových komponent je třída `MonoBehaviour`, která umožňuje integraci uživatelsky definovaných metod s herním enginem. Jedná se o velmi důležitou a často používanou třídu.

Unity a třída `MonoBehaviour` poskytují robustní systém událostí, který umožňuje objektům komunikovat mezi sebou bez nutnosti být vzájemně odkazované (reference). Toho je dosaženo pomocí speciálních metod této třídy.

`MonoBehaviour` poskytuje rámec, který umožňuje připojit skript k hernímu objektu v editoru. To se udělá například tak, že se nejprve vytvoří potomek třídy `MonoBehaviour`, který umístíme do souboru skriptu pojmenovaného stejně jako název tohoto potomka. Poté se skript přidá jako komponenta na námi zvolený `GameObject`. Pro přidání stačí přetáhnou skript na tento objekt. Po kliknutí na něj můžeme tuto komponentu poté vidět i v inspektoru.

Zde jsou některé klíčové metody definované `MonoBehaviour` třídou, pro detailnější přehled a popis viz dokumentace Unity v části *Order of execution for event functions* [22].

- `Awake()`: Unity zavolá tuto metodu při načítání instance skriptu. Zavolá se například v těchto případech: Rodičovský (parent) `GameObject`, který má tento skript přidaný jako komponentu je aktivní a inicializuje se při načtení scény nebo rodičovský `GameObject` přejde z neaktivního do aktivního stavu.

  Unity volá `Awake` pouze jednou za celou dobu existence instance skriptu. Životnost skriptu trvá, dokud je načtená scéna, která ho obsahuje.

- `Start()`: Tato metoda se volá, když začne `GameObject` existovat, buď při načítání scény, nebo při vzniku instance tohoto objektu. Volá se vždy po metodě `Awake` a před jakýmkoli voláním metody `Update`.

- `Update()`: Tato metoda se volá vždy jednou za snímek. Samotný objekt musí být aktivní.

- `FixedUpdate()`: Na rozdíl od `Update` metody, které nemá garantovanou frekvenci volání, tak má `FixedUpdate` metoda pevně daný časový interval volání.

Další užitečné metody:

- `Reset()`: Tato metoda se zavolá, když uživatel v Unity Editoru klikne na tlačítko „Reset“ na panelu s vlastnostmi této komponenty.

- `OnValidate()`: Volá se po každé deserializaci komponenty nebo při změně jakékoliv její serializované hodnoty. (viz dále sekce 3.9.2 o serializaci)

- `OnDestroy()`: Nastane, když hra nebo scéna skončí, těsně před tím, než objekt doopravdy přestává existovat.

- `OnEnable()`: Tato metoda se volá, když komponenta přechází z neaktivního stavu do aktivního.

- `OnDisable()`: Volá se, když komponenta přechází z aktivního stavu do neaktivního a to i v případě zničení objektu s danou komponentou.

Více informací o `MonoBehaviour` si je možné přečíst v manuálu Unity [23].

Coroutines

Coroutines jsou v Unity implementovány jako metody, které mohou běžet v průběhu několika snímků nebo sekund a přitom neblokují provádění ostatního kódu [24]. To umožňuje vývojářům psát asynchronní kód, jako je čekání na uplynutí času, dokončení načítání zdrojů, nebo dokončení animace, aniž by museli přerušit ostatní činnosti hry.

K zahájení coroutine slouží metoda `StartCoroutine()`, k zastavení pak `StopCoroutine()`.

Další události

Mezi další události patří například `OnCollisionEnter`, `OnTriggerEnter`, `OnDestroy`, `OnEnable`, `OnDisable`, které Unity automaticky volá při určitých událostech.

3.9.2 Serializace

Serializace je proces konverze objektů a dat do formátu, který lze snadno ukládat, přenášet a opětovně načítat. Opačným procesem je pak deserializace. Serializace umožnuje a je klíčová pro ukládání dat mezi scénami, do trvalého úložiště mezi různými spuštěními aplikace nebo pro práci s Unity Editorem.

Unity používá svůj vlastní serializační systém, který pracuje s veřejnými (public) i soukromými (private) členy (field) tříd. Soukromé členy tříd však musí být označeny atributem [SerializeField], jako například v ukázkovém programu 1. Člen nesmí být statický (static), konstantní (const), nebo pouze pro čtení (readonly).

Program 1 Příklad pro použití SerializeField

```
using UnityEngine;

public class Player : MonoBehaviour
{
    [SerializeField] private int health = 100;
    [SerializeField] private string playerName = "Unknown";
}
```

Většinu základních datových typů Unity automaticky umí serializovat (např. int, float nebo string).

Dále pak například reference na typy odvozené od `UnityEngine.Object` (např. `GameObject`, `Component`) a některé generické typy jako jsou `List<T>`.

Naopak, typy jako jsou slovníky (`Dictionary<TKey, TValue>`) nebo vlastní generické kolekce nejsou přímo podporovány.

V C# 9.0 byly představeny tzv. záznamy (records), což jsou typy, které usnadňují vytváření neměnných objektů. Záznamy jsou obzvláště užitečné pro definování objektů, u kterých je důraz kladen na hodnoty objektu, nikoliv jeho identitu.

Unity nepodporuje možnost serializovat tyto záznamy. To znamená, že pokud byste se pokusili použít záznam jako serializovaný typ (např. pro uložení dat mezi scénami nebo do trvalého úložiště), tak by Unity nedokázalo tyto objekty správně serializovat a deserializovat.

Více o serializaci je možné si přečíst v manuálu Unity [25].

4 Softwarový návrh řešení

Rozhodli jsme se implementovat simulátor v herním enginu Unity. Také již máme dobrou představu o tom, pro koho simulátor vytváříme a co od něj požadujeme.

V této kapitole tedy rozebereme jednotlivé části projektu a jednotlivé komponenty a mechaniky, které budeme využívat. Konkrétněji si zanalyzujeme všechny části simulátoru tak, aby byl pro naše cílové uživatele co nejpřínosnější. Pojďme si tedy rozebrat problémy, které je potřeba vyřešit v rámci našeho projektu a vybrat jejich vhodná řešení.

4.1 Herní svět

Herní svět bude jednou z hlavních částí našeho simulátoru, od které se odvíjejí naše další rozhodnutí. Pojďme si rozmyslet, jak by bylo pro naše účely nejlepší tento svět, ve kterém následně samotná simulace probíhá, reprezentovat.

Unity nabízí vestavěnou komponentu `Terrain` pro 3D terény. Komponenta `Terrain` umožňuje vytvářet terén přímo v Unity Editoru a následně za pomoci nástrojů ho velmi snadným způsobem upravovat [26]. Rozhodli jsme se však tuto komponentu nevyužít, protože si chceme moct svět tvořit podle vlastních pravidel s možností měnit jeho podobu přímo ve vlastně napsaných C# skriptech. Tedy máme tak menší potřebu znát a spoléhat na funkce Unity jako takové.

Použitím vlastní reprezentace světa budeme mít úplnou kontrolu nad pravidly jeho vytváření a následném chováním simulace ohně na něm.

Existují takové dva základní druhy, jak by se dal svět reprezentovat - kontinuální a diskrétní.

V kontinuálním světě není prostor rozdělen do předem definovaných segmentů. Místo toho objekty a terén existují v nepřetržitém, plynule měnícím se prostoru. Pro určité typy simulací, kde je důležitý vysoký stupeň realismu a detailní interakce s prostředím, může tato reprezentace přinést značné výhody. V našem případě si však vystačíme s jednodušší a méně detailní diskrétní reprezentací. Navíc, pokud bychom chtěli, můžeme se v budoucnu přiblížit větší realističnosti alespoň tím, že zmenšíme tyto diskrétně definované segmenty a upravíme způsob simulace.

Jaký zvolit tvar těchto segmentů? Nabízí se množství možností. Běžně se setkáme v různých programech se čtvercovou mřížkou. Mohli bychom také ale například zvolit šestiúhelníkovou mřížku, která nabízí plynulejší a realističtější možnosti šíření ohně, protože každý segment má šest sousedů namísto čtyř. Navíc vzdálenosti mezi středy šestiúhelníků jsou stejné ve všech směrech, což přináší určité výhody v simulacích založených na vzdálenosti. My vzdálenost políček sice používat budeme, avšak na realističnosti, o kterou nám zas až do takovéto hloubky tolik nejde, nám to tolik nepřidá.

Jakékoliv jiné možnosti oproti čtvercové mřížce nám nepřinesou významné výhody, a proto bude pro nás nejlepší právě čtvercová mřížka. Ta je velice jednoduše implementovatelná a intuitivně pochopitelná. Umožňuje relativně snadný výpočet vzdáleností a snadnější reprezentaci nebo indexování těchto segmentů. Od této chvíle budeme těmto segmentům říkat políčka.

Protože chceme, aby simulátor využil celý 3D prostor, implementujeme něco, čemu se říká 2,5D svět. To je něco mezi 2D a 3D světem. Je to prostor, který kombinuje 2D mřížku s elementem výšky.

2,5D svět umožňuje simulovat objekty a terén v trojrozměrném prostoru, aniž by bylo nutné řešit složitost plnohodnotného 3D modelování a výpočtů pro zobrazení terénu. Nemusíme tak řešit komplexní 3D reprezentaci, když nás u políčka bude zajímat i pro naše účely pouze jeho výška.

4.2 Generace světa

Pro generování (vytváření) herního světa budeme implementovat algoritmy, které umožní dynamickou tvorbu terénu podle definovaných pravidel. Tyto algoritmy budou mít za úkol vytvářet vizuálně atraktivní prostředí, které je přesvědčivé a dostatečně realistické. Nové světy by měly být generované vždy automaticky, určitým způsobem odlišně, aby působily pokaždé alespoň trošku jiným dojmem.

Tomuto procesu se obvykle říká procedurální generování terénu. Jedná se tedy o způsob vytváření terénu, kdy je terén vytvářen automaticky za pomoci algoritmu, nikoli ručně.

Chceme umožnit tvorbu krajin, včetně řek, jezer a různých typů vegetace.

4.2.1 Perlinův šum

Perlinův šum je algoritmus, který generuje náhodné gradientní textury, které se hodí například právě pro simulaci různorodých přírodních povrchů. Tyto textury jsou hodnoty (intenzity) bodů nějakého prostoru. Jednotlivé hodnoty se potom dají interpretovat třeba právě jako výška v terénním modelu nebo jako hodnoty udávající vlhkost apod.

Tento algoritmus pracuje s několika klíčovými parametry, jako jsou frekvence, amplitudy, oktávy a další. Tyto parametry ovlivňují výsledný vzhled generovaných struktur. Význam jednotlivých parametrů si uvedeme názorně na generaci výškové mapy terénu.

Frekvence určuje, jak těsně jsou rozloženy jednotlivé hodnoty šumu, což určuje hustotu výskytu prvků. Nižší frekvence znamená, že prvky (rysy) terénu jsou rozmístěny dále od sebe, vyšší frekvence naopak znamená, že prvky jsou blíže u sebe.

Amplituda ovlivňuje rozsah výšek terénu. Vyšší amplitudy způsobují, že terén má výraznější vrcholy a údolí.

Oktávy určují počet vrstev šumu. Šum můžeme překrývat, aby vytvořil složitější a komplexněji vypadající vzory s různou frekvencí apod.

Vytrvalost (persistence) ovlivňuje, jak rychle amplituda klesá s každou oktávou. Nižší hodnota persistence znamená rychlejší pokles, což vede k jemnějším detailům při vyšších oktávách.

Lakunarita občas známá jako mezerovitost (lacunarity) obvykle značí a kontroluje, jak rychle se mění frekvence s každou oktávou.

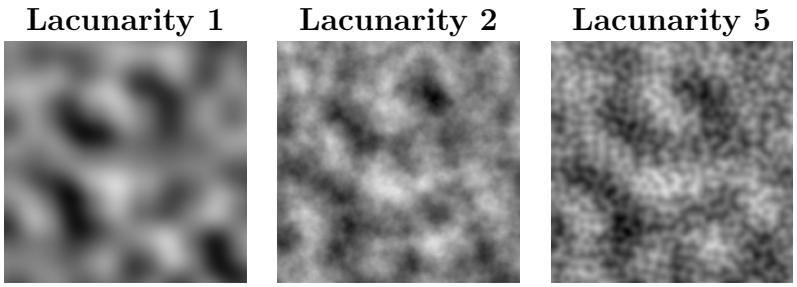
Více o tomto tématu si lze přečíst například zde v tomto článku [27].

Pro konkrétní implementaci je potřeba experimentovat, které hodnoty parametrů se nám hodí nebo líbí více.

Příklady

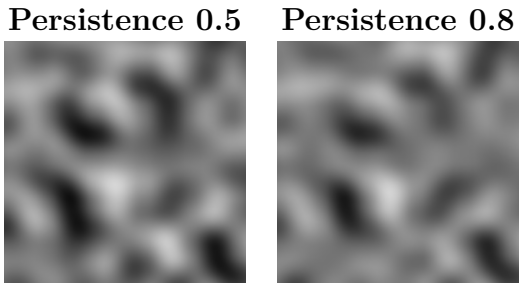
Pro ilustraci vlivu různých parametrů jsme vytvořili několik ukázkových obrázků textur. Abychom zajistili konzistenci výsledků, tak jsme použili pro generátor (pseudo) náhodných čísel stálý počáteční bod (seed), což zaručuje, že jsou experimenty dobře reprodukovatelné díky identické sekvenci generovaných čísel.

Níže je ukázka 4.1 několika vygenerovaných textur, ve kterých je názorně vidět rozdíl při použití rozdílných parametrů (oktávy: 10, persistence 0.5, počáteční frekvence a amplituda 1).



Tabulka 4.1 Ukázky Perlinova šumu s různými hodnotami lacunarity

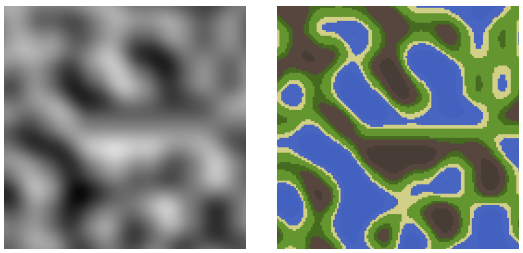
Pro následující ukázku 4.2 jsou všechny ostatní parametry nastavené stejně (oktávy: 10, lacunarity 1, počáteční frekvence a amplituda 1).



Tabulka 4.2 Ukázky Perlinova šumu s různými hodnotami persistence

Následně je už na nás, jak bude hodnoty této textury interpretovat.

Například můžeme zvolit určité rozsahy hodnot, které budeme obarvovat totožnou barvou a dostaneme texturu, která nám může připomínat třeba právě terén světa. Viz příkladová ukázka 4.3.



Tabulka 4.3 Ukázka Perlinova šumu a jeho obarvené varianty pro 8 rozsahů hodnot

4.2.2 Další techniky

Mezi další techniky a algoritmy patří například Diamond-Square Algoritmus [28], zvláště vhodný pro vytváření horských masivů a terénních výškových map. Pracuje na principu rozdělování čtverců na menší diamanty a postupného přidávání náhodné výšky.

Dále například Voronoi diagramy [29], které se často používají k vytváření realistických struktur terénů, jako jsou pukliny, cesty a další geologické rysy.

Mohli bychom sice používat a kombinovat další techniky, jako třeba hydraulické eroze, kde simulujeme vodní erozi podle které upravujeme mapu, avšak my si vystačíme právě se základem v Perlinově šumu a dalším jednoduchém kombinováním map pro vlhkost, výšku a typ vegetace našeho světa.

Máme-li tedy například mapu vlhkosti vygenerovanou pomocí Perlinova šumu, můžeme vytvářet mapu vegetace, protože můžeme předpokládat, že některé rostliny preferují více vlhkosti než jiné. Podobných způsobem si vytvoříme vlastní zjednodušené generátory (dále popsané v sekci 6.6).

Generátory se specializují na jednotlivé vlastnosti světa a tyto vlastnosti pak dále kombinují podle daných pravidel, aby společně vytvořily celý herní svět. Dále implementujeme funkce, které by pracovaly na těchto mapách tak, aby se podle potřeby daly přizpůsobovat podle konkrétnějších potřeb a požadavků. Funkce mohou tyto jednotlivě vygenerované mapy kombinovat nebo filtrovat (rozmazání a další transformace).

4.3 Ukládání a načítání světa

Jedním z našich cílů bylo uložit a znovu načíst konkrétní herní svět. Tuto funkcionalitu mohou využít uživatelé, aby se mohli vracet ke stejnému světu opakovaně a testovat na něm například jiné podmínky simulace. To umožňuje uživatelům pokračovat v simulaci kdykoliv si přejí, experimentovat s různými scénáři, svět sdílet nebo analyzovat dopady změn v simulovaném prostředí.

Data o světě lze ukládat různými způsoby. Každý způsob má jiné přednosti a omezení:

Binární formáty umožňují rychlé načítání a menší velikost souborů. Data jsou ve formě binárních souborů, což jsou soubory v téměř nečitelné formě pro člověka, ale velmi efektivní pro zpracování počítačem. Kvůli horší čitelnosti pro člověka jsou tudíž méně flexibilní pro manuální úpravy a ladění.

Databáze mohou být vhodné pro rozsáhlé nebo dynamicky se měnící světy. Jsou obvykle složitější na implementaci a správu. Podporují například snazší dotazování na data a mohou poskytovat pokročilé funkce pro správu těchto dat. Pro náš projekt se však nehodí.

Textové formáty (JSON, YAML, XML) nabízí lepší čitelnost a jednodušší manipulaci s daty. Přináší však větší velikost souborů a pomalejší zpracování.

Simulátor má být uživatelsky co možná nejpřívětivější. Podobu uložení světa však nejvíce ocení zejména začínající vývojáři, kteří tak s programem mohou pracovat snadněji. Svět navíc bude vždy omezený a relativně malý, tudíž nám nebude vadit ani větší velikost souborů. Vzhledem k tomuto faktu a tomu, že se nesnažíme o co nejefektivnější uložení dat, ale spíše oceníme čitelnost a jednodušší

Program 2 Příkladová reprezentace světa ve formátu JSON

```
{
  "world": {
    "width": 1,
    "depth": 2,
    "tiles": [
      {
        "widthPosition": 0,
        "depthPosition": 0,
        "moisture": 10,
        "vegetation": "Grass",
        "height": 0.7
      },
      {
        "widthPosition": 0,
        "depthPosition": 1,
        "moisture": 55,
        "vegetation": "Forest",
        "height": 0.9
      }
    ]
  }
}
```

manipulaci s daty, implementujeme ukládání a načítání světa právě za pomocí jednoho z tohoto textového formátu.

Budeme používat právě textový formát `JSON` [30] (JavaScript Object Notation), což je formát určený pro výměnu dat, který je snadno čitelný jak pro lidi, tak pro stroje. Je velmi univerzální, protože je podporován mnoha programovacími jazyky a platformami, což usnadní přenositelnost.

Ve formátech jako právě `JSON` můžeme reprezentovat složitější struktury dat ve stromové struktuře, kde každý uzel může mít jednoho nebo více potomků. Toto je vhodné pro hierarchické organizace dat, jako například políček našeho světa, která tak mohou být reprezentována jako objekty s jasně definovanými vlastnostmi (např. výška, vlhkost, typ vegetace). Tyto objekty pak mohou být uloženy v seznamu nebo poli, což je jednoduchá a efektivní forma ukládání pro naše účely.

V programu 2 je příklad takové možné reprezentace ve formátu `JSON`. Vidíme, že se sice nejedná o velmi efektivní způsob uložení, ale je to velmi přirozeně a jasně čitelný formát. I když není nutné specifikovat oba rozměry světa, názvy polí se opakují apod, tak nám to nevadí. Takto usnadníme pozdější čtení souboru a umožníme jednodušší následnou kontrolu při načítání dat zpět do tříd programu.

4.4 Simulace

Samotná simulace šíření ohně může být implementována mnoha různými způsoby. Každý z těchto způsobu má vlastní specifické vlastnosti a použití. My jsme si již vysvětlili, že se budeme zaměřovat především na relativní jednoduchost této simulace. Tím tak bude simulátor dostupnější více uživatelům a přitom

splní požadavky, které jsme si stanovili. První skupině uživatelů se bude jevit jako vcelku realistický a zároveň svou jednoduchou architekturou přinese o to větší užitek i druhé skupině našich námi dříve definovaných cílových uživatelů zmíněných v sekci 2.2.

Uvedeme si některé z nejběžnějších typů simulací, jejichž aplikace sahá od akademického výzkumu po praktické aplikace ve vývoji her.

Modelování za pomocí agentů. Tento přístup si můžeme představit tak, že entity (agenti) mají autonomní chování a nejsou všechny řízeny nějakou jinou entitou. Při simulaci šíření ohně může každý agent představovat například strom, který reaguje na ohně okolo sebe.

Celulární automaty jsou velmi populární modely pro simulaci šíření ohně. Svět je, podobně jako je to v našem případě, rozdělen do buněk (políček). Každá buňka má definovaný stav (například hořící, nespálený, spálený). Stav každé buňky se aktualizuje na základě jednoduchých pravidel závislých na stavech a vlastnostech nastavených pro sousední buňky. Pravidla jsou obvykle jednoduchá a vcelku snadno pochopitelná.

Automaty můžeme rozdělit na deterministické a nedeterministické (stochastické) v závislosti na tom, jak jsou navrženy.

Deterministické automaty můžeme chápat tak, že pro stejné počáteční podmínky bude mít každý běh simulace vždy stejný výsledek, protože jsou pravidla pevně stanovena.

Stochastické automaty naopak zahrnují element náhody v pravidlech nebo ve vývoji stavů. Při stejných počátečních podmínkách tak může mít každý běh simulace odlišné výsledky. V takovém případě je obvykle potřeba simulaci spustit několikrát, aby bylo možné získat statistický přehled o možných výsledcích.

Simulací bude moci probíhat v simulátoru několik zároveň. Nejen tedy například simulace ohně, ale i také simulace větru apod. Každá simulace se může tedy zaměřovat na vlastní specifické parametry světa. Simulátor navrhneme tak, aby byla nějaká centrální logika, která dokáže v krocích spouštět tyto různé simulace, které poté přepisují vlastní nebo sdílené hodnoty parametrů v daném světě.

Bude tak možné přidat vlastní simulace respektující architekturu světa a ostatních simulací. Také bude možné přidat vlastní parametr (celému světu) nebo parametry (jednotlivým políčkům světa), následně přidat simulaci ovlivňující a využívající tyto parametry a nakonec upravit logiku pravděpodobnosti rozšíření ohně respektující i tyto parametry.

Například by uživatel mohl chtít přidat parametr intenzity tepla každému políčku. Tento parametr by mohl mít vlastní logiku simulace. A to by poté mohlo ovlivnit další pravděpodobnosti nějakého jiného fenoménu jako například vlhkosti. Políčka by tak tedy neměla pouze stavy hoří a nehoří, ale vlastně i míru intenzity hoření.

Simulátor bude vybaven několika základními funkcemi, které by mohly být využity různými simulacemi. Budou například řešit otázku vzdálenosti mezi políčky. V čtvercové mřížce může být komplikované určit, jak daleko by jedno políčko mělo ovlivňovat políčko druhé. Je tedy otázka, zda sousední políčka jsou pouze ta, která se dotýkají hranou nebo i ta, která sousedí vrcholem. Implementujeme tedy více možností, které nás napadnou, aby si pak sám uživatel mohl vybrat, kterou funkci chce použít a jakým způsobem.

Architektura simulace je podrobněji popsána ve vývojářské dokumentaci 6.12.

4.4.1 Predikce a napojení na externí skript

Část simulátoru bude věnována k předpovídání pravděpodobností rozšíření ohně za určitých počátečních podmínek. To bude možné dělat interně přímo v projektu, nebo externě za pomoci vlastních skriptů.

Nabídneme uživatelům prostředky k tomu, aby si mohli sami ve vlastních skriptech takovou předpověď udělat a následně si ji vizualizovat a to v jiném programovacím jazyce, jako je například Python. Jazyk Python je známý pro svou jednoduchost a čitelnost, což umožňuje rychlé učení a snadné pochopení kódu. Je tedy vhodný i pro ty, kteří si ještě nevěří na programování v C# nebo to není jejich preferovaný programovací jazyk. Pro tyto uživatele to je další možná alternativa, jak si zkusit předpovědět rozšíření ohně a následně si tuto předpověď vizualizovat.

Poskytneme potřebné propojující součásti, které si pak uživatel může doplnit nebo podle vlastních potřeb upravit. Bude se tedy jednat o takové jednoduché rozhraní, které bude schopné ze C# projektu (našeho simulátoru) spouštět skript napsaný v Pythonu a následně si přečíst jeho výstup a ten zobrazit.

Už například víme, že díky možnosti načítání světa z formátu `JSON` je v Pythonu možné vytvořit data pro vlastní svět bez nutnosti použít algoritmus simulátoru. Tento svět stačí uložit podle pravidel respektujících jeho strukturu právě do formátu `JSON` a kdykoliv ho následně načíst do simulátoru, jak je popsáno výše v sekci 4.3. Dále pak bude možné díky tomuto přímému propojení na Python si na tomto světě zkusit vlastní predikci rozšíření ohně a porovnat ji s vestavěnou predikcí přímo v simulátoru. To bude možné dělat rychleji a snáze, protože nebude nutné simulátor zavírat, ale pouze přepsat algoritmus přímo v Pythonu a zavolat tento skript ze simulátoru znovu. Vestavěná predikce v simulátoru bude jednoduše předpovídat rozšíření ohně tak, že mnohokrát spustí simulaci a vyhodnotí pravděpodobnosti rozšíření do jednotlivých částí (políček) světa.

Pro predikování rozšíření ohně tedy umožníme přímější integraci a spouštění vlastního Python skriptu přímo ze simulátoru.

Připojíme i možnost vygenerovat si trénovací data. Trénovací data bude možné si uložit a mohou posloužit o něco pokročilejším uživatelům, kteří si za jejich pomoci chtějí natrénovat model, který je schopný rozšíření predikovat. Tedy provést tuto predikci rychleji, lépe nebo s menším počtem výpočtů bez nutnosti mnohokrát spouštět simulaci nebo jiným způsobem počítat pravděpodobnosti šíření ohně.

Na predikci za pomoci umělé inteligence se dnes zaměřuje mnoho prací a uživatel tak může najít inspiraci pro vlastní testování nápadů v našem modelu světa, nebo na trénování vlastního modelu kdekoliv na internetu. Do projektu přidáme ukázkový Python skript, který bude demonstrovat komunikaci a předpověď skriptu s následným zobrazením. Poslouží tak i jako praktická ukázka pro ty, kteří by bez delšího studování struktury projektu chtěli rovnou začít programovat predikci v Pythonu.

4.5 Rozdělení simulátoru

Náš simulátor bude obsahovat různé scény sloužící rozdílným účelům. Nechceme uživatele zahltit v jedné scéně mnoha tlačítky a možnostmi. Každá scéna se tedy

pokusí zaměřit na trošku jiné nabízené funkce. Například zcela určitě budeme potřebovat uživatelské menu, tutoriál nebo scénu se samotnou hlavní logikou simulovaní a další jiné.

Díky tomuto designu simulátoru bude možné do budoucna přidat vlastní scény simulátoru zaměřující se na jiné další funkce. Navíc bude simulátor přehlednější a snazší ovládat.

Naše druhá skupina cílových uživatelů, tedy vývojáři, mohou chtít do budoucna přidávat vlastní scény s vlastními pravidly a logikou. Usnadníme jim to implementací komponenty, která se bude starat o zpracování uživatelského vstupu. Budeme mít připravené určité funkce nebo systém a bude možné napojit na tyto funkce další skripty, které už podle vlastní logiky a stavu zváží, jak se vlastně zachovají. Některá napojená tlačítka totiž mohou volat funkce, které mohou a nemusí být doopravdy vykonány podle stavu simulátoru.

Tato komponenta by mohla sloužit jako rozhraní pro přidání takovýchto tlačítek nebo zadávacích políček. Ty mohou poskytovat vizuální zpětnou vazbu uživateli. Tato zpětná vazba může být rozdílná v závislosti na tom, zda se příslušná funkce úspěšně vykonala, nebo ne.

Další funkce jako například pohyb kamery nebo možnost klikat na políčka bude pravděpodobně sdílet více scén.

Scény tedy budou využívat některé stejné komponenty, ale také budou moci si snadno do určité míry implementovat nebo změnit hlavní logiku na svoji vlastní.

4.6 Uživatelské rozhraní

Při návrhu uživatelského rozhraní (UI) pro náš simulátor se chceme zaměřit na zásady jednoduchosti a intuitivního uspořádání. Chceme, aby ovládání nebylo překážkou pro používání, ale naopak dovolovalo co nejsnadněji navigovat a ovládat náš simulátor při různých úkonech.

Snažili jsme se pro návrh dobrého UI vycházet z některých známých nebo základních principů uvedených v knize [31].

Jednou z klíčových zásad dobrého UI designu je minimalizace kognitivního úsilí potřebného k pochopení rozhraní. Naši uživatelé by měli být schopni rychle rozpoznat dostupné volby a efektivně se rozhodnout, aniž by museli dlouze přemýšlet, kterou možnost zvolit.

4.6.1 Hlavní menu

Hlavním menu bude sloužit jako centrální bod pro navigaci v simulátoru. Mělo by být přehledné a minimalistické s jasnou strukturou navigace.

Z hlavního menu bude uživatel moct vcházet jak do jednotlivých scén, tak by měl mít přístup k ovládacím a informačním prvkům týkající se celého simulátoru.

Tlačítko, které nás přímo přenese do scény s tutoriálem k simulátoru, bude umístěno na nejvýraznějším místě vlevo, což přirozeně vede uživatele k jeho vyzkoušení jako prvního kroku.

Přechody mezi jednotlivými částmi aplikace si představujeme jako je následovně naznačené na obrázku 4.1.

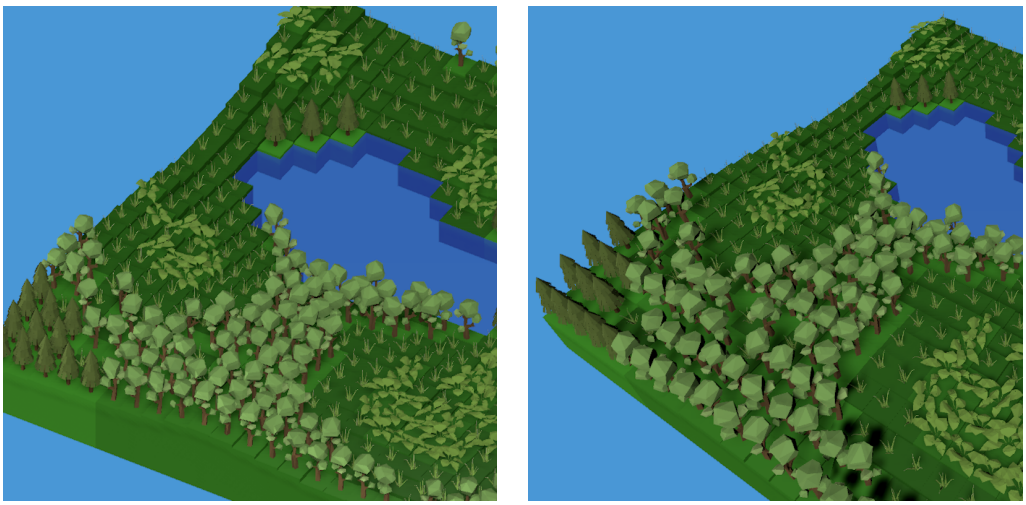
4.6.2 Kamera

Existují dvě základní perspektivy, které můžeme v Unity využít pro zobrazení světa. Mohou být dosaženy za pomoci ortografické nebo perspektivní kamery.

Perspektivní kamera simuluje způsob, jakým lidské oko vidí svět a jedná se tedy o přirozenější způsob zobrazení. Objekty jsou menší, čím dále jsou od kamery. Tato kamera je ideální tam, kde je žádoucí realistické zobrazení prostoru a hloubky.

Ortografická kamera zobrazuje objekty ve stejné velikosti, bez ohledu na jejich vzdálenost od kamery. Je naopak ideální tam, kde je důležitá konzistentnost velikosti a tvaru objektů.

Na obrázcích 4.4 vidíme porovnání těchto perspektiv.



Tabulka 4.4 Porovnání ortografické (vlevo) a perspektivní kamery v Unity.

Pro náš simulátor se více hodí *Ortografická kamera*, protože je podle autora celá simulace přehlednější a lépe se také kliká na vzdálenější políčka, jelikož mají stejnou velikost, jako políčka nacházející se blíže kameře.

4.6.3 Modely vegetace

Pro modely vegetace našeho světa jsme se rozhodli využít balíček „Low Poly Simple Nature Pack“ volně a zdarma dostupný z *Unity Asset Store* [32].

Tento balíček obsahuje vizuálně přitažlivé 3D modely rostlin a stromů. V projektu našeho simulátoru bude celý tento balíček importovaný. Pokud by si uživatel přál použít jiné 3D modely, nebo rozšířit množství použitých, tento balíček mu to velmi usnadní.

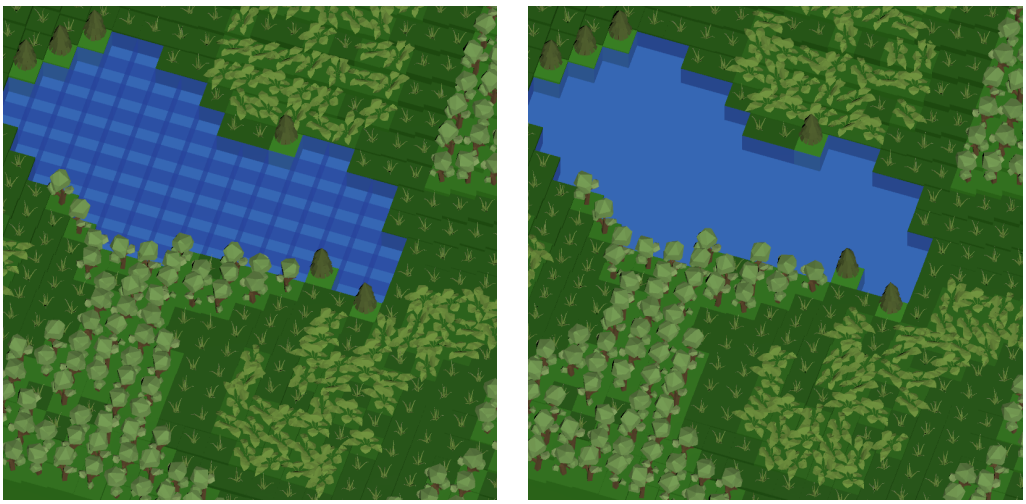
Pro implementaci jsme vybrali modely jako jsou nízké trávy, keře a různé typy stromů. Do budoucna je možné nahradit tyto modely i vlastními 3D modely.

Chtěli bychom nabídnout možnost vizualizace světa úplně bez jakýchkoliv modelů vegetace, bez nutnosti odebírat tento vegetační faktor z logiky simulace jako takové. V simulátoru tedy bude možnost si zobrazit svět ve zjednodušeném režimu, který tyto nebo jiné 3D modely vegetace vůbec nebude používat. Ušetříme tak navíc výkon a zdroje počítače.

4.6.4 Voda

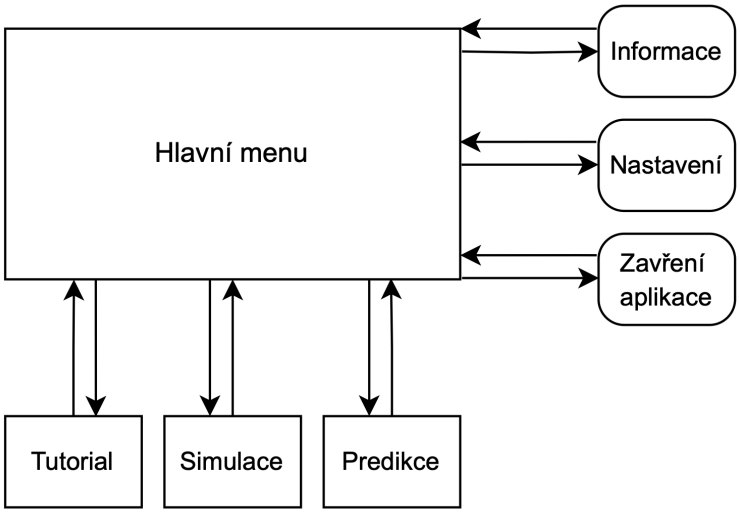
Políčka vody budeme chtít mít průhledné, tak aby měl svět přirozeněji vypadající vzhled. V Unity se obvykle vodní plochy realizují pomocí shaderů (program ovlivňující vykreslování) a materiálů, které dovolují simulaci průhlednosti, odrazu a refrakce světla. My použití těchto pokročilých technik nemáme v úmyslu. Vystačíme si s průhlednými vodními políčky, které zapadnou do designu našeho simulátoru. Díky tomu se také vyhneme použití složitějších grafických technik jako jsou právě shadery, renderery a jejich nastavení, které jsou také často náročnější na pochopení.

Nejen u průhledných objektů se ale setkáváme s problémem, kdy dochází k nežádoucím vizuálním efektům při jejich překryvu. Na příkladovém obrázku 4.5 takový efekt můžeme vidět na místech, kde se průhledné políčka vody potkávají. V těchto místech jsou body, ze kterých jsou políčka tvořena (jejich hranice), na stejných místech a dochází k nepěknému zdůraznění těchto překryvů.



Tabulka 4.5 Porovnání vzhledu nespojených a spojených vodních políček.

Takovýmto vizuální efektům a překryvům budeme chtít algoritmy předejít, a proto spojíme sousední vodní políčka do jednoho velkého objektu. Tím odstraníme vnitřní hranice mezi políčky a ponecháme pouze vnější hranice. To značně zlepší vizuální spojitost vodní plochy. Jak se toho konkrétněji dosáhne je vysvětleno později ve vývojářské dokumentaci 6.17.



Obrázek 4.1 Diagram hlavního menu, šipky značí možné přechody mezi scénami a ovládacími prvky

5 Uživatelská dokumentace

Tato kapitola je věnována uživatelské příručce pro tento simulátor. Jedná se o průvodce pro začátečníky, který vás provede procesem spuštění aplikace a krok za krokem seznámí se všemi klíčovými aspekty simulátoru, přičemž poskytne užitečné informace o tom, jak simulátor efektivně ovládat. Všechny potřebné soubory je možné nalézt v příloze této práce.

5.1 O aplikaci

Aplikace je v anglickém jazyce. Je zdarma a volně dostupná pro každého bez jakýchkoli omezení, takže způsob využití je tedy plně v rukou uživatelů.

5.2 Spuštění aplikace

Pro spuštění aplikace stačí otevřít vhodný spustitelný soubor dle použité platformy.

Pro počítače s operačním systémem *Microsoft Windows* se jedná o soubory končící příponou `.exe`, konkrétně tedy `FireSimulationUnity.exe`. Verze této aplikace je určena výhradně pro 64-bitové architektury.

Pro počítače s operačním systémem *macOS* se jedná o soubory končící příponou `.app`, konkrétně tedy `FireSimulatorMac.app`. Aplikace podporuje jak procesory Intel, tak i novější Apple Silicon čipy. Díky Apple technologii nazvané Universal binary aplikace kombinuje podporu pro oba typy procesorů a automaticky detekuje a spouští ten správný spustitelný soubor podle toho, který typ procesoru zařízení používá [33].

5.3 Hlavní nabídka

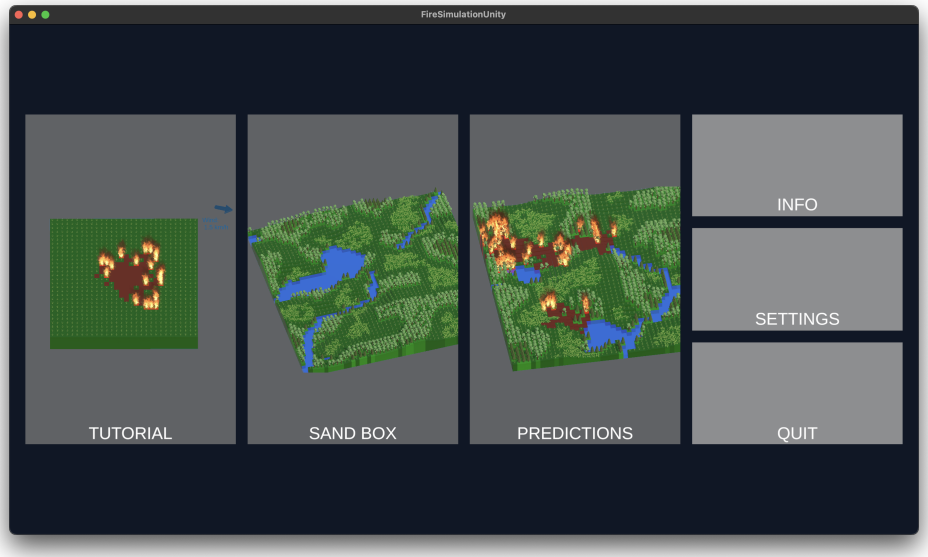
Ihned po spuštění aplikace se dostaneme do *hlavní nabídky* (obr. 5.1). Zde se nacházejí tři velké tlačítka hlavních scén aplikace a tři vedlejší tlačítka. Mezi hlavní scény aplikace patří `Tutorial`, `Sandbox` a `Predictions`.

Tři vedlejší tlačítka jsou umístěny pod sebou v pravé části okna programu a slouží k různorodým účelům. Od shora jsou jimi `Info`, `Settings` a `Quit`.

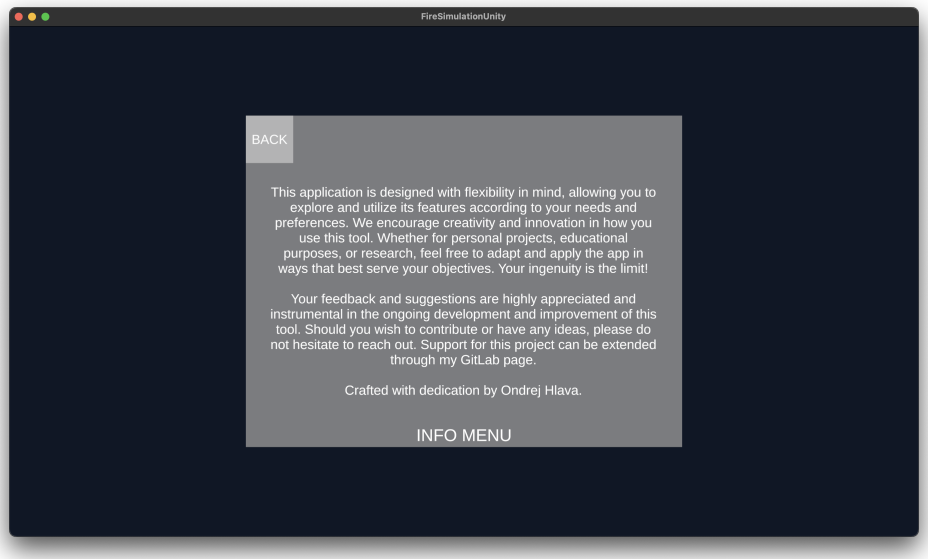
5.3.1 Info

Pod kliknutí na `Info` se zobrazí nové okno s informacemi o aplikaci (obr. 5.2). Můžeme si zde přečíst nějaké drobné základní informace o účelu a možnostech programu, zpětné vazbě a podpoře.

V levém horním rohu se nachází tlačítko `Back`, které nás vrátí zpět do *hlavní nabídky*. Návrat do *hlavní nabídky* se také provede kdykoliv za běhu programu stisknutím klávesy `Ecs`.



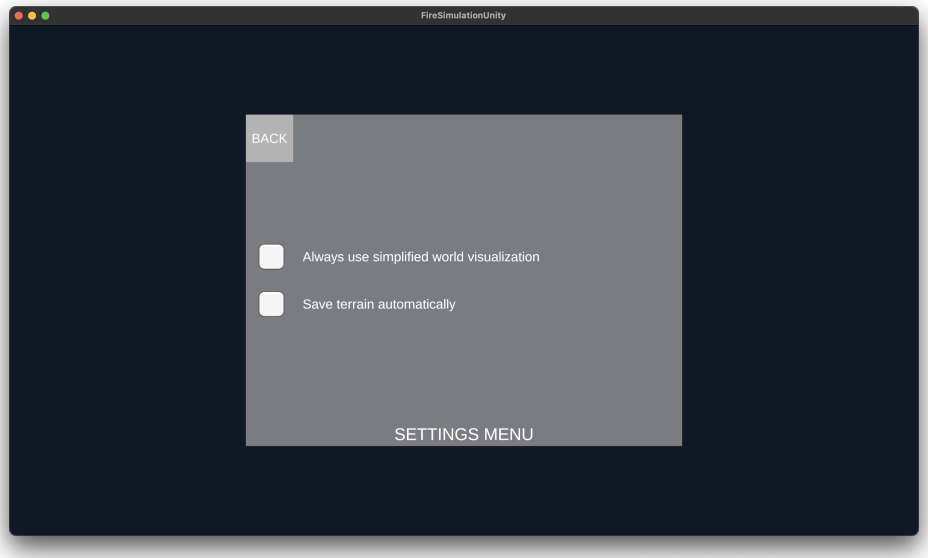
Obrázek 5.1 Okno hlavní nabídky (Main menu) simulátoru.



Obrázek 5.2 Okno informace o simulátoru.

5.3.2 Settings

`Settings` slouží jako záložka, kde nalezneme možné nastavení a přizpůsobení aplikace (obr. 5.3). Tato nastavení jsou trvalá. To znamená, že při příštím spuštění aplikace se obnoví tato nastavení přesně tak, jak je uživatel před posledním zavřením aplikace ponechal.



Obrázek 5.3 Okno nastavení simulátoru.

V levém horním rohu se opět nachází tlačítko `Back`, které nás vrátí zpět do *hlavní nabídky*.

Konkrétně se tu pak nachází dvě možnosti nastavení, které ovlivňují chování v celé aplikaci:

Always use simplified world visualization

Pokud je zaškrtnutá tato možnost, tak se vždy za jakýchkoli okolností pro zobrazení světa používá zjednodušený mód. To znamená, že políčka světa a krajiny budou zobrazena bez 3D modelů rostlin. Toto nastavení však nemá vliv na samotnou logiku simulace, která tak zůstává stejná.

Takováto zjednodušená vizualizaci světa může pomoct k ušetření výkonu počítače nebo zrychlení a větší plynulosti aplikace.

Program i bez zaškrtnutí této možnosti automaticky používá zjednodušený mód zobrazení ve scéně `Predictions` nebo v ostatních scénách pokud je zároveň svět (počet políček) moc velký.

Save terrain automatically

Pokud je zaškrtnutá tato možnost, tak dochází k automatickému ukládání nově generovaných světů. Světy se ukládají to složky `SavedWorlds`, kterou najdeme v rámci speciální složky `StreamingAssets` popsané v sekci 5.7.

Uživatel má tedy sice možnost terén ukládat manuálně, ale s touto funkcí nemusí mít obavy, že by o nějaký zajímavý terén světa přišel, pokud by na jeho uložení náhodou zapomněl.

5.3.3 Quit

Ukončí a zavře celou aplikaci. Funkce tohoto tlačítka je podobná tomu, jako když uživatel ukončí aplikaci standardními způsoby poskytovanými operačním systémem, například zavřením okna aplikace nebo ukončením aplikace přes správce úloh.

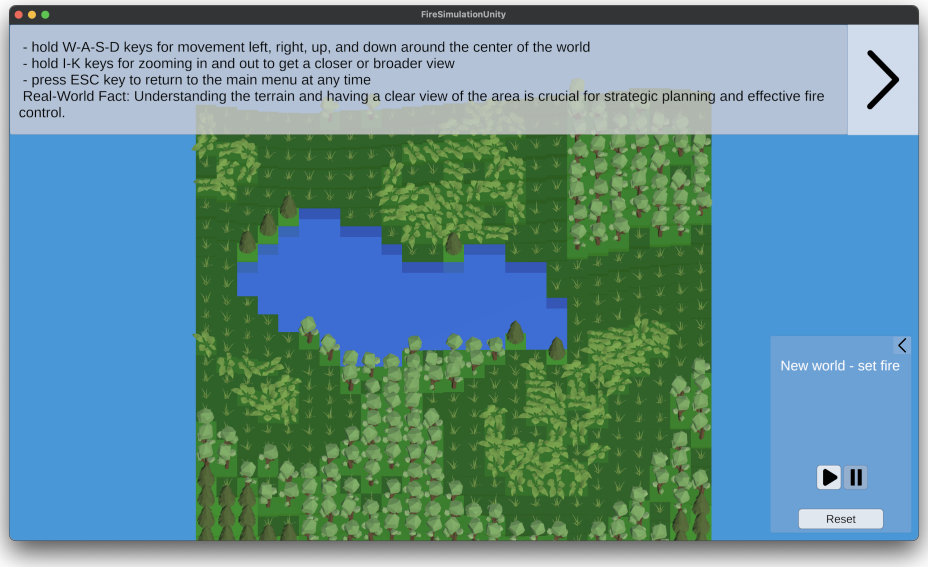
5.4 Tutorial

Scéna tutoriálu učí základy ovládání programu a principy simulace požáru. To je uskutečněno především díky panelu s textem a vysvětlujících komentářů v horní části okna. Tutoriál nabízí 6 různých sekcí a světů, které by měly být pro uživatele dostačující k pochopení způsobu ovládání programu.

K přechodu mezi sekcemi stačí jednoduše kliknout na šipky nacházející se v levém nebo pravém horním rohu.

Pro návrat do *hlavní nabídky* opět stačí stisknout klávesu `Ecs`.

Příkladová ukázka z tutoriálu je na obrázku 5.4.



Obrázek 5.4 Ukázka scény tutoriálu simulátoru.

5.5 Sandbox

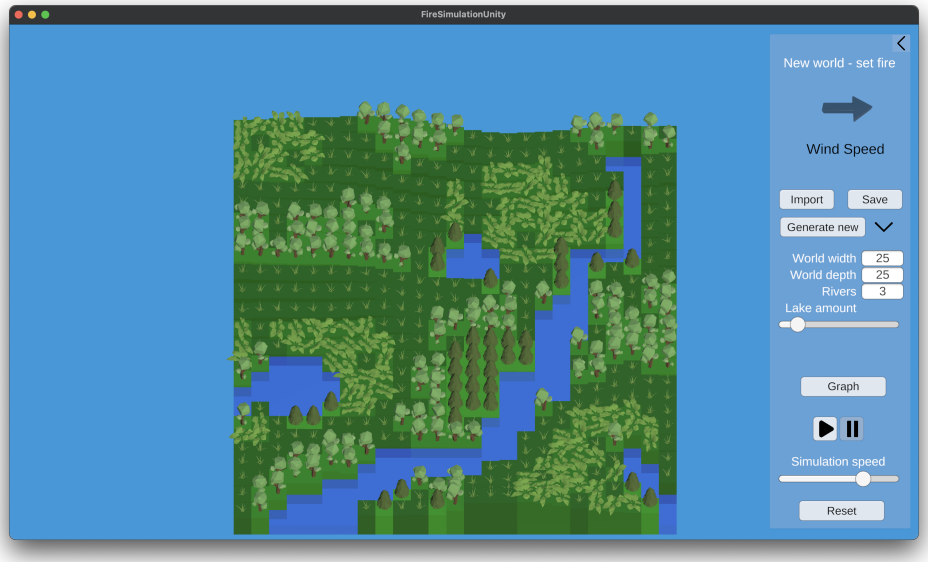
`Sandbox` je scéna (prostředí), kde se dá bez omezení experimentovat a testovat simulace. Slouží k tomu, aby si uživatel mohl nahrávat nebo ukládat vlastní světy a živě pak sledovat vývoj ohně v čase.

Ovládání

Veškeré ovládání je za pomoci myši nebo kláves.

Klávesy `W-A-S-D` slouží pro náklon kamery, `I-K` pro přiblížení a oddálení světa (zoom), `Esc` pro návrat do *hlavní nabídky*.

Svět lze také přibližovat kolečkem na myši. Myší dále můžeme klikat nebo popotahovat prvky na panelu v pravé části okna. Prvky slouží k všemožným účelům od ovládání generace terénu světa, až po ovládání simulace (obr. 5.5).



Obrázek 5.5 Ukázka SandBox scény simulátoru.

Panel a možnosti světa

V horní části panelu se nachází informační text ohledně stavu simulace a tlačítko zpět pro návrat do *hlavní nabídky*. Hned pod nimi najdeme ukazatel síly a směru větru.

Ve střední části panelu se nachází prvky ovlivňující parametry použité při vytváření světa. Parametry zahrnují šířku světa (`World width`), hloubku světa (`World depth`, počet řek (`Rivers`) a množství políček vody (`Lake amount`). Po změně jakékoliv z hodnot se automaticky daný svět vytvoří znovu za použití těchto nově nastavených parametrů. Pokud uživateli svět nevyhovuje, tak může kliknout na tlačítko pro vytvoření světa nového (`Generate new`).

Další dvě tlačítka `Import` a `Save` slouží k načtení světa ze souboru, či naopak uložení jeho dat do souboru.

Kliknutím na `Import` se v programu otevře okno, které pomáhá s vybráním cesty k požadovanému souboru. Tato cesta tedy slouží k jednoznačnému a konkrétnímu umístění souboru se světem v souborovém systému počítače.

Příklad takového okna můžeme vidět na obrázku 5.6.

Podporované soubory

Je možné vybrat soubory končící s příponou `.json` nebo obrázky s příponou `.jpeg`, `.jpg` a `.png`.

Podporované obrázky (přípony `.jpeg`, `.jpg`, `.png`) jsou načteny a chápány jako výšková mapa. Mělo by se jednat o černo-bílé obrázky. Velikost světa je pak určena podle již zmíněných parametrů `World width` a `World depth`, které musí být menší, než rozměry obrázku.

Předpřipravené soubory

Některé ukázkové soubory jsou již uložené a připravené v balíčku samotného programu. Tyto soubory najdeme ve složkách, které se nachází ve výchozí cestě umístění souborů (složka `StreamingAssets`). Můžeme využít například soubory ve složce `TerrainMaps` nebo `TutorialWorlds`.

Simulace šíření ohně

Nejdříve je potřeba vybrat a zapálit políčka, ze kterých se oheň během simulace začne šířit. Zapálení políček se provede kliknutím na políčka světa. Až poté lze simulaci opravdu spustit.

Běh simulace je možné kdykoliv pozastavit a opět spustit za pomoci tlačítek umístěných dole v panelu. Simulaci je také možné spustit zmáčknutím klávesy mezerníku.

Dále tu najdeme posuvník s textem `Simulation speed`, který slouží pro nastavení rychlosti aktualizací simulace. Pro rychlejší simulaci je třeba popotáhnout posuvník vpravo, naopak pro pomalejší aktualizace simulace stačí popotáhnout posuvník vlevo.

Pod posuvníkem nalezneme tlačítko `Reset`, které uvede svět do stavu před začátkem simulace. Obnoví všechny hořící a shořelé políčka a připraví svět na možnost opětovně rozdělat na políčkách oheň.

Graph

Poslední tlačítkem v panelu je tlačítko `Graph`. Toto tlačítko zobrazuje, nebo naopak skrývá graf, který zobrazuje počet hořících políček v čase. Graf se automaticky aktualizuje za běhu simulace. Příklad takového grafu je na obrázku 5.7.

5.6 Predictions

`Predictions` je scéna a prostředí, které slouží k analýze pravděpodobností a předpovědi rozšíření požáru.

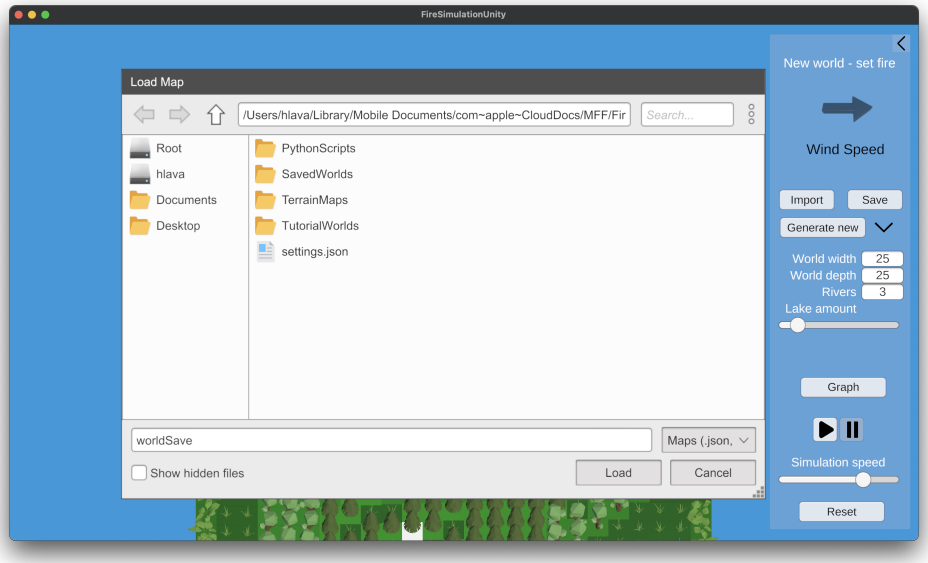
Pro režim zobrazení světa se zde používá vždy zjednodušený mód, tedy bez modelů rostlin. Scéna se zaměřuje na faktory ovlivňující šíření ohně. Pro zjednodušení je zde úmyslně vypnutý faktor větru.

Veškeré ovládání je podobně jako ve scéně `SandBox` přes levé tlačítko myši a klávesnici.

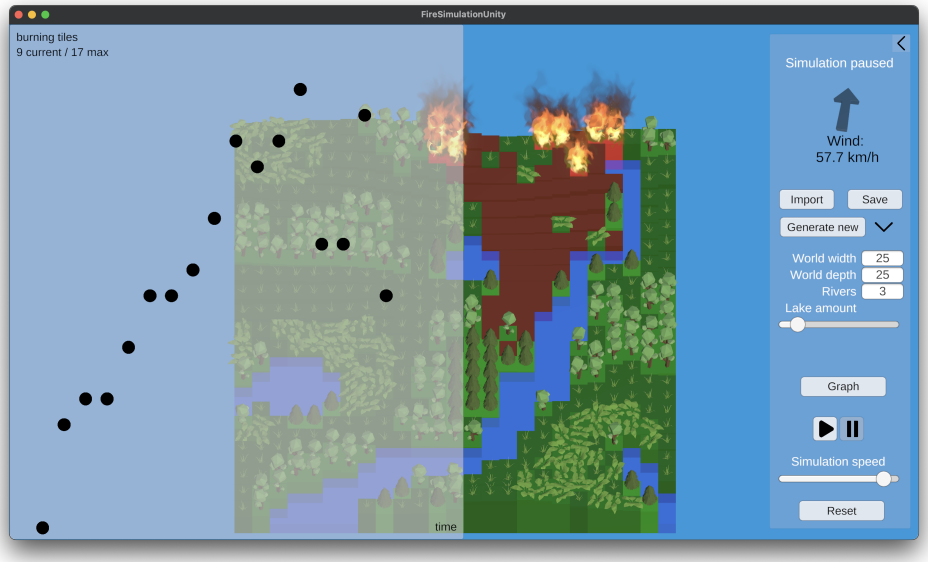
Také na pravé straně okna simulátoru se nachází panel ovládání.

Generate new, Import a Save

V panelu se nachází tlačítka `Generate new`, `Import` a `Save`. Tato tlačítka fungují naprosto stejně, jako stejně pojmenované tlačítka ve scéně `SandBox` popsané v sekci 5.5. Doporučujeme si uložit nějaký zajímavý svět v jedné scéně a načíst si ho ve scéně druhé.



Obrázek 5.6 Okno simulátoru sloužící k importování a načtení světa ze souboru.



Obrázek 5.7 Příklad okna simulátoru při zobrazení grafu.

Základní pravděpodobnost a faktory

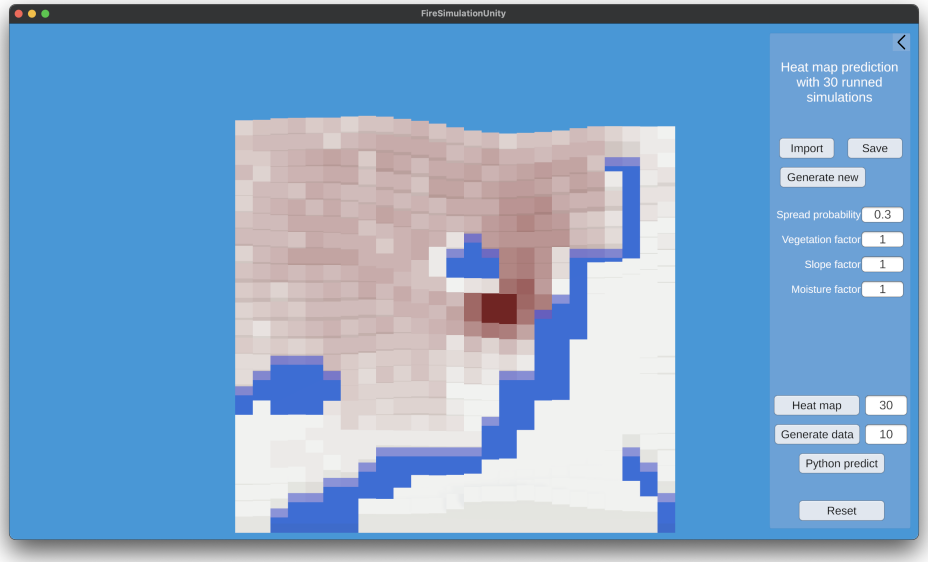
Střední část panelu slouží k možnosti změny a experimentování s faktory. Základní pravděpodobnost (`Spread probability`) slouží jako výchozí pravděpodobnost rozšíření ohně na sousedící políčka.

Ostatní faktory, jako je faktor podle typu rostlin na políčkách (`Vegetation factor`), faktor rozdílu výšek políček (`Slope factor`) nebo faktor vlhkosti políček (`Moisture factor`) je možné upravovat změnou jejich příslušných hodnot. Hodnoty vyšší než 1 efekt zesilující, zatímco hodnoty nižší než 1 efekt naopak oslabují. Hodnota faktoru je procentuální vůči základní hodnotě, takže stačí docela malé změny pro velké rozdíly ve výsledcích.

Heat map

Nejdříve je potřeba vybrat a zapálit políčka, z kterých se oheň během simulace začne šířit. Zapálení políček se provede kliknutím na políčka světa. Poté je možné kliknout na tlačítko `Heat map`.

`Heat map` zobrazí pravděpodobností mapu rozšíření ohně dle zvolených faktorů, jako je vidět na příkladovém obrázku 5.8.



Obrázek 5.8 Ukázka Predictions scény simulátoru.

Program spustí simulaci vícekrát za stejných podmínek a určí, jak často se oheň na jaké políčko rozšířil. Čím častěji tomu tak bylo, tím tmavší odstín červené je pro políčko použito. Pro přesnost odhadu je možné zvětšit, nebo naopak zmenšit hodnotu vedle tohoto tlačítka. Hodnota udává, kolikrát se simulace spustí.

Generate data a Python predict

Tato scéna je také zamýšlena pro ty, kteří by si chtěli vyzkoušet za pomoci vlastního skriptu a algoritmu odhadnout, kam se požár vlastně rozšíří. K tomu slouží dvě tlačítka `Generate data` a `Python predict`.

`Generate data` vygeneruje podle příslušné hodnoty zadané množství světů a do souboru uloží pro každý tento svět jeho odpovídající `Heat mapu`. Tento soubor (`datafile.json`) se ukládá do složky `PythonScripts` nacházející se ve složce `StreamingAssets` popsané v sekci 5.7. Takto vygenerujte data mohou být použita například pro predikční modely a strojové učení.

Tlačítko `Python predict` poté zavolá skript napsaný v programovacím jazyce `Python`. Program volá skript s názvem `main.py` uloženým ve složce `PythonScripts` ve složce `StreamingAssets`.

Pro více informací o přesnějším fungování metod `Generate data` a `Python predict`, odkážeme čtenáře na sekci ve vývojářské dokumentaci 6.20, protože se jedná o pokročilejší funkce. Uživatel se zde mimo jiné dočte jakým způsobem si skript předává s programem data, které poté simulátor přímo zobrazí.

`Python predict` tedy je určen pro programátory, kteří by chtěli mít možnost si vizualizovat vlastní předpověď teplotní mapy úpravou zmíněného skriptu.

Reset

Tlačítko `Reset` uvede svět do stavu před začátkem simulací požáru. Zruší zobrazení `Heat mapy` a připraví svět na možnost opětovného rozdělávání ohně na políčkách.

5.7 Složka StreamingAssets

Tato speciální složka se nachází přímo v balíčku aplikace simulátoru a její umístění se liší podle operačního systému, na kterém aplikace běží. Tato složka se automaticky otevře jako výchozí při kliknutí na jakékoliv tlačítko `Import` nebo `Save` v aplikaci.

Najdeme v ní uložené nastavení simulátoru (`settings.json`), připravený Python skript (složka `PythonScripts`), uložené světy pro tutoriál (složka `TutorialWorlds`) nebo obrázky map terénu (složka `TerrainMaps`).

Pro aplikace běžící na *Windows* je složka `StreamingAssets` obvykle umístěna v adresáři aplikace, spolu se spustitelným souborem aplikace (přípona `.exe`). Pokud je tedy název aplikace `FireSimulatorWin` a aplikace byla nainstalována do složky:
`C:\Program Files\FireSimulatorWin`
tak by cesta k `StreamingAssets` byla následující:
`C:\Program Files\FireSimulatorWin\FireSimulatorWin_Data`

Na *macOS* jsou aplikace zabaleny jako `.app` balíčky, které se chovají jako samostatné aplikace, ale ve skutečnosti jsou složkami obsahujícími všechny potřebné soubory.

Pokud je tedy název aplikace `FireSimulatorMac` a aplikace by byla umístěna ve složce:
`/Applications`
tak by cesta k `StreamingAssets` byla následující:
`/Applications/FireSimulatorMac.app/Contents/Resources/Data`

6 Vývojářská dokumentace

Tato kapitola je věnována vývojářské příručce pro tento simulátor.

Čtenář by měl být před čtením této části obeznámen s principy fungování Unity z kapitoly 3. Předpokládá se tedy také znalost programovacího jazyka C#.

Cílem této kapitoly *není* vysvětlení všech jednotlivých funkcí a mechanismů simulátoru. Slouží jen jako přehledová kapitola doplňující předešlé kapitoly pro lepší orientaci v projektu. Jednotlivé funkce jsou sami o sobě napsány co nejčitelněji a navíc jsou doplněny komentáři přímo ve skriptech.

Zaměříme se maximálně tedy na hlavní funkce každé třídy, nebo poskytneme přehled o jejich vzájemném propojení a fungování.

6.1 Unity projekt

Celý projekty vyvíjen v Unity ve verzi 2022.3.15f1. Doporučujeme tedy použít právě tuto verzi pro otevření projektu. Unity umožňuje aktualizaci projektu na novější verzi Unity, avšak funkčnost aktualizovaného projektu nemůžeme garantovat. Minimální systémové požadavky pro tuto verzi jsou uvedeny v manuálu Unity [34].

Celý projekt je přiložen v příloze této práce. Projekt je možné otevřít vybráním složky `FireSimulationUnity` v aplikaci Unity Hub. Po prvním otevření je potřeba v Unity Editoru otevřít některou ze scén. Doporučujeme otevřít scénu `MainMenu` ve složce `Scenes` dvojitým kliknutím a následně spustit program kliknutím na tlačítko `Play`, nacházejícím se uprostřed nahoře.

Veškeré hlavní soubory projektu jsou uložené ve složce `Assets`. Zde jsou soubory systematicky rozčleněny do několika podadresářů. Pro lepší orientaci v projektu uvedeme tyto podadresáře a jejich obsah.

6.1.1 Assets

Struktura složky `Assets`:

- **Plugins**: Složka obsahující knihovny nebo rozšíření třetích stran a pluginy pro rozšíření funkcí Unity.
- **Scenes**: Tato složka obsahuje scény projektu. V projektu jsou celkem 4 scény (`MainMenu`, `Tutorial`, `SandBox`, `Predictions`).
- **Prefabs**: Obsahuje předpřipravené objekty a jejich materiály pro možnost opakovaného použití ve scénách.
- **Scripts**: Obsahuje veškeré C# skripty, které řídí logiku a chování objektů.
- **SimpleNaturePack**: Balíček s modely rostlin jako jsou stromy, trávy a keře.
- **StreamingAssets**: Obsahuje soubory, které mohou být načteny za běhu simulátoru a kam standartně ukládáme i další data ze simulátoru.
- **TextMesh Pro**: Obsahuje soubory pro textový systém TextMesh Pro.

6.1.2 Scripts

Struktura složky **Scripts**:

- **WorldManagement**
  - WorldBuilder.cs
  - WorldClasses.cs
  - WorldFileManager.cs
  - WorldGenerator.cs
  - WorldSerialization.cs
  - WorldExtensions.cs
  - **Maps**
    - Maps.cs
    - MapExtensions.cs
- **EngineCore**
  - InputHandler.cs
  - MainLogic.cs
  - PredictionsLogic.cs
  - TutorialManager.cs
- **Simulation**
  - FireSimParameters.cs
  - FireSimulation.cs
  - SimulationBase.cs
  - SimulationManager.cs
  - WindSimulation.cs
  - **Events**
    - Events.cs
    - EventLoggers.cs
- **Utilities**
  - FileBrowserHandler.cs
  - FileManagementService.cs
  - FirePredictor.cs
  - HeightMapImporter.cs
  - PythonCaller.cs
  - RandomUtility.cs
  - Settings.cs

- **Visualization**
  - CameraHandler.cs
  - GraphVisualizer.cs
  - MainMenu.cs
  - TileInstancesExtensions.cs
  - UIManager.cs
  - Visualizer.cs
  - WindIndicator.cs

6.1.3 StreamingAssets

Tato složka slouží pro uložení souborů, které aplikace potřebuje načítat za svého běhu. Zároveň je důležité, že všechny soubory, které uložíme do této složky Unity zkopíruje do stejně pojmenované složky pro postavenou aplikaci na cílovou platformu. Soubory v ní jsou tedy také odkudkoliv a kdykoliv dostupné za pomoci vlastnosti `Application.streamingAssetsPath`, která tedy vždy ukazuje na správné umístění na platformě, kde je aplikace spuštěna [35].

Struktura a soubory složky **StreamingAssets**:

- **settings.json** Konfigurační soubor simulátoru.
- **PythonScripts** Výchozí složka pro volání Python skriptů. Obsahuje základní skript pro predikci šíření ohně. Také se zde ukládají trénovací data.
- **SavedWorlds** Prázdná složka pro automatické ukládání vygenerovaných světů uživatelem.
- **TerrainMaps** Obrázky terénu - mapy stupňů šedi.
- **TutorialWorlds** Složka s uloženými světy použitými pro tutoriál.

6.2 WorldClasses.cs

Třída `World` je centrální komponentou našeho simulačního modelu. Jedná se o nosnou třídu vlastností světa. Obsahuje jak klasický konstruktor, tak kopírovací, který vytvoří zcela nezávislou kopii instance světa. Třída kromě svých vlastností také sdružuje jednotlivá políčka.

Třída `Tile` představuje políčka, základní stavební bloky světa, s vlastnostmi definujícími její fyzikální stav a stavové proměnné. Hodnoty proměnných se podílí na simulaci (výška, vlhkost, vegetace a další).

Třída `Wind` spravuje informace o větru, jako je jeho směr nebo síla. Vítr je vlastností celého světa (`World`).

6.3 WorldUtilities.cs

Obsahuje nástroje pro manipulaci se světem a políčky. Rozšiřuje funkce a možnosti tříd `World` a `Tile` přidáním pomocných metod.

Třída `TileUtilities` obsahuje metody pro nastavení a resetování stavů (nestatických vlastností) políček, které je možné volat přímo na jednotlivých instancích políček.

Třída `WorldUtilities` poskytuje metody pro získávání informací o sousedních políčkách, resetování světa a manipulaci s jeho strukturou. Tyto metody lze volat přímo na instancích světa.

Obsahuje například metody pro získání sousedních políček různými způsoby. Můžeme se zajímat o políčka sousedící pouze hranou (metoda `GetEdgeNeighborTiles`), sousedící i vrcholy (metoda `GetNeighborTiles`), tak i políčka nacházející se v určité kruhové vzdálenosti (metoda `GetCircularEdgeNeighborTiles`). Tyto metody se využívají například při simulaci nebo při hledání navazujících vodních políček.

Metoda `GetRandomInitBurningTiles` vrací list náhodně vybraných políček pro zapálení. Metodu využíváme při vytváření trénovacích dat pro predikci šíření ohně.

6.4 Map.cs

V některých případech chceme pracovat s pouze mapy reprezentující jednu danou vlastnost světa. Proto abychom nemuseli pracovat přímo s políčky světa využijeme generickou třídu `Map<T>`.

Tato generická třída je pro jednodušší správu map jednotlivých vlastností světa. Nabízí metody pro práci nad těmito daty pro různý typ proměnných. Například metoda `FillWithDefault` nastavuje všechny hodnoty mapy na jednu zvolenou základní hodnotu. Dále obsahuje několik implicitních konverzí umožňujících převody mezi dvourozměrnými poli (`T[,]`, `T[][]`) a instancemi třídy Map<T>.

6.5 MapExtensions.cs

Poskytuje rozšiřující metody na třídách `Map<T>` a `System.Array`. Pro `Map<float>` umožňuje například normalizovat (`Normalize`) hodnoty mapy tak, že nejnižší hodnota bude 0 a nejvyšší 1, vyhladit (`Smooth`) průměrovacím filtrem mapu a další.

6.6 WorldGenerator.cs

Skript obsahuje hlavní třídu `WorldGenerator`, která kombinuje různé mapy generované specifickými generátory do jednoho nového světa.

Každá mapa je nejprve vytvořena příslušným generátorem, poté může být upravena nebo jinak kombinována za pomoci rozšiřujících metod definovaných ve třídě `MapExtensions`.

Zde je výčet specifických generátorů:

- `LakeMapGenerator` rozpozná oblasti pod určitou prahovou hodnotou a tyto oblasti označuje jako vodní políčka.
- `RiverMapGenerator` vytváří zjednodušenou simulaci toků řek, kde řeky jsou generovány náhodně s přihlédnutím k existující topografii a jezerům (vodním políčkám).
- `MoistureMapGenerator` generuje mapu vlhkosti, kde vodní políčka mají maximální vlhkost. Vlhkost se snižuje se vzdáleností od zdroje vody.
- `VegetationMapGenerator` přiřazuje typy vegetace na základě mapy vlhkosti.

6.7 WorldBuilder.cs

Statická třída `WorldBuilder` obsahuje metody, které aplikují různé typy map (`Map<T>`) na danou třídu světa (`World`). Díky těmto metodám je tak například možné získat mapu, kterou aplikujeme na již existující svět a tím můžeme změnit různými způsoby jeho jednotlivé vlastnosti.

V simulátoru jeden takový příklad na získání mapy (jiný než za pomoci některého generátoru) je a to z takzvané „heightmapy“. „Heightmapa“ neboli výšková mapa může být obrázek se stupni šedi, kde interpretace intenzity každého pixelu může být chápána jako výška políčka na dané pozici.

Třída, která právě výše zmíněnou výškovou mapu získá z obrázku se nachází ve skriptu `HeightMapImporter`.

6.8 HeightMapImporter.cs

Třída `HeightMapImporter` realizuje rozhraní `IMapImporter`, které vyžaduje implementaci metody `GetMap` (pro načtení mapy libovolného typu).

`HeightMapImporter` využívá vlastnost `HeightMultiplier`, která umožňuje přeškálování výškových hodnot mapy před jejím vytvořením. Výškovou mapu s požadovanými rozměry metoda `GetMap` získává z textury obrázku při poskytnutí cesty k tomuto obrázku.

6.9 InputHandler.cs

Třída `InputHandler` zpracovává vstupy od uživatele, jako jsou kliknutí myši, zmáčknutí kláves nebo stisknutí některého z tlačítek ve scéně. Její hlavní role je zpracování uživatelských vstupů a vyvolání odpovídajících událostí.

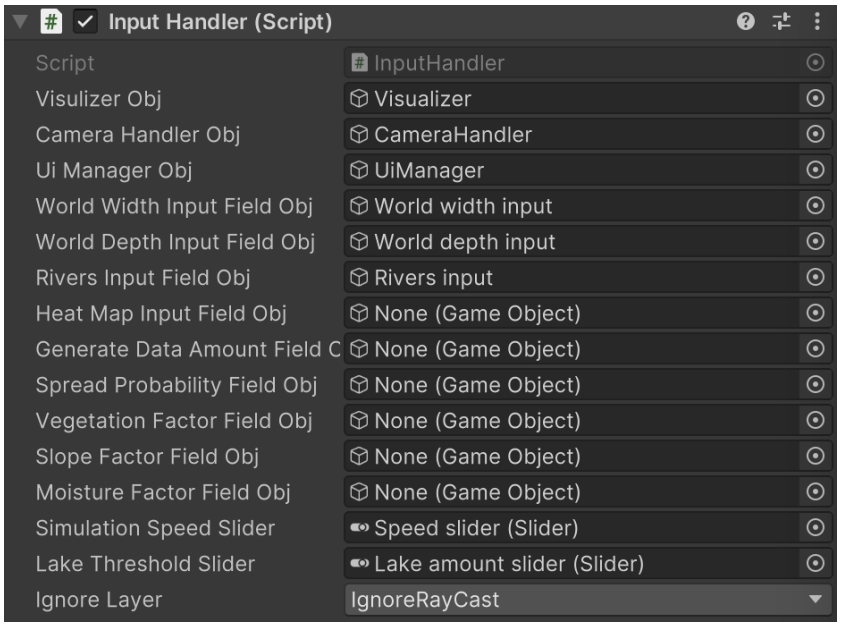
Třída je navržena tak, aby bylo snadné ji použít ve více scénách. Je potomkem (dědí od) třídy `MonoBehaviour`. Skript je připojen k prázdnému objektu ve scéně v Unity Editoru, což umožňuje snadné přiřazení tlačítek, posuvníků, textových polí a dalších pouhým přetažením daného prvku na odpovídající vlastnost definovanou v tomto skriptu. Díky této referenci poté můžeme přímo ze skriptu nastavit výchozí hodnoty těmto prvkům.

Například `visulizerObj`, `cameraHandlerObj` jsou reference na objekty ve scéně, které `InputHandler` využívá pro vizualizaci světa a ovládání kamery.

`simulationSpeedSlider`, `lakeThresholdSlider` jsou posuvníky uživatelského rozhraní pro nastavení rychlosti simulace a množství vygenerovaných jezer.

`worldWidthInputField`, `worldDepthInputField`, `riversInputField` jsou textová pole pro hodnoty, které se dále používají pro vytvoření světa.

Na následujícím obrázku je skript `InputHandler.cs` připojen ke stejně pojmenovanému objektu `InputHandler` v Unity Editoru. Následně díky tomu v záložce `Inspector` vidíme a můžeme upravovat reference na připojené prvky (obr. 6.1). Pro úpravu reference stačí přetáhnout objekt scény na odpovídající vlastnost.



Obrázek 6.1 Ukázka InputHandler skriptu připojeného k objektu InputHandler ve SandBox scéně Unity Editoru.

Další klíčovou funkcionalitou třídy je zpracování výběru políčka myší. K tomu slouží metody `HandleTileClick`, `HandleTileHover` reagující na akce myši nad políčky světa.

Pro detekci interakce s políčkem vysíláme z pozice kamery směrem k místu, kam uživatel klikl paprsek (ray). Následně detekujeme průsečík tohoto paprsku s objektovou reprezentací políček světa. Tyto objekty vytváří třída `Visualizer`, o které bude řeč později. Zatím nám stačí vědět, že každé políčko `Tile` má i svou vizuální reprezentaci vytvořenou právě třídou `Visualizer`.

Pokud bychom chtěli, aby paprsek ignoroval některé objekty, stačí tomuto objektu přidělit vrstvu s názvem `Ignore Raycast`, nebo vlastní vytvořenou vrstvu, kterou v Unity Editoru přidělíme proměnné `ignoreLayer`. Pro více informací o vrstvách doporučujeme přečíst odpovídající sekci v manuálu Unity [36]. Použití těchto vrstev najdeme například u modelů vegetace. Nechceme, aby modely vegetace blokovaly možnost kliknutí na objekty políček pod nimi. Proto mají tyto modely přidělenou vrstvu `Ignore Raycast`, což můžeme vidět zobrazené v záložce `Inspector` (obr. 6.2).

Dále blokujeme možnost klikat nebo vybírat políčka pokud je políčko překryté panelem (například panely objektu `Canvas`). Pro tuto funkcionalitu je vlastnost panelů nastavena na „True“ pro vlastnost `Raycast Target`. To je možné také jednoduše vypnout v záložce `Inspector` po vybrání tohoto panelu.

Stejný objekt `InputHandler`, ale napojený na jiné ovládací prvky nalezneme celkem ve třech scénách projektu (`Tutorial`, `SandBox`, `Predictions`).

`InputHandler` má definované delegáty a události, které mohou být spuštěny na základě různých akcí. Reakci na vstup od uživatele tak vyhodnotí částečně `InputHandler`, například omezením nějaké hodnoty pro zadávací políčko a následně vyvolá odpovídající událost.

Více skriptů jako například hlavní logika `MainLogic` dané scény může být k těmto událostem přihlášena (metoda `SubscribeToInputEvents`) a reaguje na ně svými zvolenými příslušnými metodami.

Aby vůbec například tlačítka zavolaly některou z metod samotného `InputHandleru`, je potřeba těmto tlačítkům přidat `OnClick` událost. Tlačítku přetáhneme objekt `InputHandler` a vybereme jakou metodu chceme při kliknutí zavolat - jako je vidět na obrázku 6.3. Více informací o událostech lze nalézt v manuálu Unity [37].

6.10 MainLogic.cs

Třída `MainLogic` koordinuje hlavní logiku simulace, včetně generování světa, spouštění a pozastavení simulace. Reaguje také na uživatelské vstupy z `InputHandleru` a podle svého stavu rozhoduje jak na ně reaguje nebo jaké metody dále zavolá.

Například po tom, co uživatel vybere políčka, na kterých začne oheň hořet je třeba nejdřív inicializovat simulaci. Tlačítko, které přípravu a začátek této simulace vyvolá je to samé tlačítko jako to, které po kliknutí na pozastavení simulace, simulaci opět nechá pokračovat.

`MainLogic` je také potomkem třídy `MonoBehaviour`. Tento skript je také tedy zamýšlen k připojení k objektu ve scéně. Odkud v záložce `Inspector` je pak možné jednoduše měnit některé vlastnosti jako například rychlost simulace. To je dobré například pro testovací účely, kdy za běhu programu můžeme v Unity Editoru tuto rychlost měnit, aniž bychom museli napojit změnu této hodnoty na posuvník přímo v uživatelském rozhraní simulátoru, nebo měnit tuto hodnotu ve skriptu.

V metodě `Awake` inicializujeme komponenty a registruje události od `InputHandleru`.

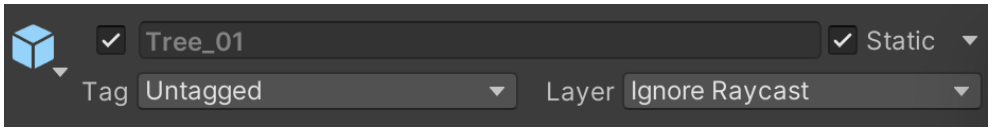
Metoda `Update` pravidelně aktualizuje stav simulace podle nastavené frekvence.

Najdeme zde další funkce jako například `GenereteNewWorld`, `OnImportClicked`, `OnSaveClicked` zajišťují generování nového světa, import mapy či celého světa a ukládání aktuálního světa.

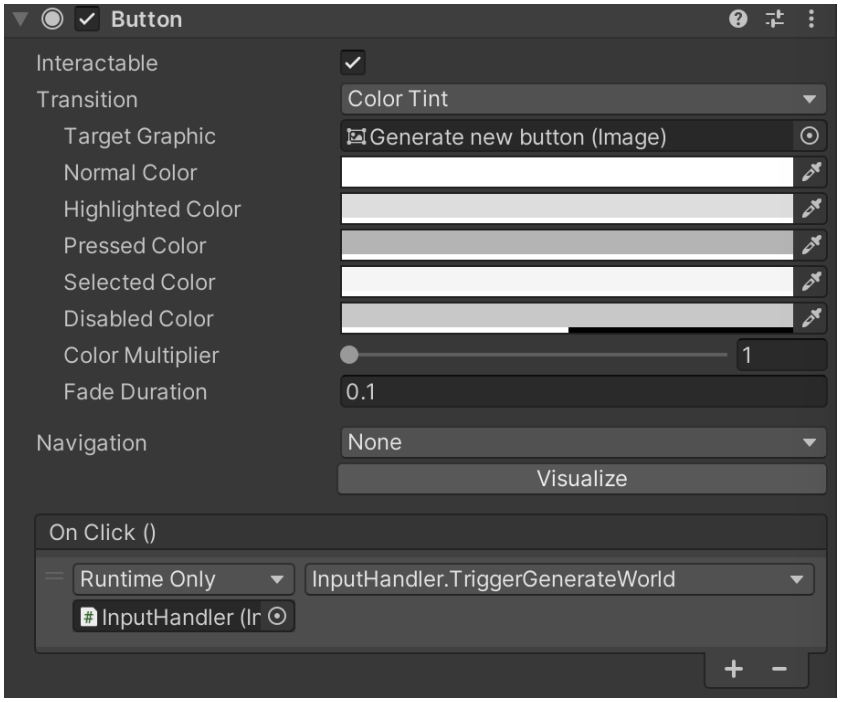
Hlavní logiku využíváme ve dvou scénách (`Tutorial` a `SandBox`), protože mají podobný společný základ. Jednoduše tedy stačí v každé scéně vybrat a napojit požadovanou podmnožinu metod na interakční elementy.

6.11 TutorialManager.cs

Tento skript je určen především pro scénu `Tutorial`. Skrze uživatelské rozhraní umožňuje hráči postupovat mezi různými sekcemi tutoriálu. Základem všech těchto sekcí je vždy hlavní logika (`MainLogic`), kterou tak pouze rozšiřuje.



Obrázek 6.2 Ukázka vybrané vrstvy Ignore Raycast pro objekt (model) vegetace.



Obrázek 6.3 Ukázka připojení metody třídy InputHandler při kliknutí na tlačítko.

Stará se o zobrazení správného textu na panelu a načítání odpovídajícího světa (metoda `ImportTutorialMap`) pro každou sekci. Dále nastavuje parametry simulace ohně v závislosti na aktuální sekci a zobrazuje nebo skrývá ukazatel větru v závislosti na tom, zda je v dané sekci potřebný.

6.12 FireSimulation.cs

Třída `FireSimulation` dědí základní třídy `SimulationBase` a je zodpovědná za hlavní simulaci šíření požáru v daném světě.

Uchovává parametry simulace (`parameters`), seznam hořících políček (`burningTiles`) a logger pro záznam událostí (`fireLogger`).

Metoda `Update` posouvá simulaci o krok vpřed, aktualizuje stavy hořících a sousedních políček a zaznamenává změny.

Metoda `CalculateFireSpreadProbability` určuje pravděpodobnost rozšíření ohně na základě faktorů jako vegetace, sklon terénu, vlhkost, vítr a hodnot ve třídě `FireSimParameters`.

6.13 FireSimParameters.cs

Třída `FireSimParameters` je konfigurační objekt pro simulaci síření ohně. Umožňuje základní a zjednodušenou možnost upravit způsob výpočtu pravděpodobnosti rozšíření ohně pro jednotlivé faktory (metoda `CalculateFireSpreadProbability`).

Například je tak velice snadno možné naprosto potlačit vlivy určitých faktorů nastavením jejich odpovídajících hodnot na nulu.

6.14 SimulationManager.cs

Třída `SimulationManager` spravuje spuštění a koordinaci různých simulací. Simulace je možné přidávat a tento manager je pak postupně všechnu aktualizuje. Všechny simulace musí implementovat rozhraní `ISimulation`, který požaduje metodu `Update` pro aktualizaci (jeden výpočetní krok simulace).

Pro přidání vlastní simulace proto tedy stačí takovou simulace pouze vytvořit podobně jako třída `WindSimulation` starající se o vítr celého světa a přidat ji do `SimulationManageru` za pomoci funkce `AddSimulation`. Metoda `UpdateAllSimulations` poté sama aktualizuje všechny simulace v pořadí, v jakém byly přidány.

Některé simulace navíc implementují třídu `SimulationBase` a mají tedy i svůj kalendář využívaný pro záznam událostí.

6.15 SimulationBase.cs

Doporučujeme vytvářet další třídy simulací, jako potomky této třídy. Od této třídy dědí jak `FireSimulation`, tak `WindSimulation`.

Tato třída obsahuje `SimulationCalendar`, což je kalendář, který slouží k uchování času některých změn v simulacích. Díky tomu je později možné vyhodnocovat, co se dělo za běhu simulace.

Čas se uchovávají společně s typem události přímo v loggeru simulace. Například pro `FireSimulation` takto zaznamenaná data o vývoji využijeme poté pro zobrazení v grafu, jako změny počtu hořících políček v čase.

6.16 EventLoggers.cs

Třída `EventLogger<T>` je generická třída pro logování událostí typu T, kde T musí být potomkem vlastně definované třídy `Event`. V simulátoru jsou implementovány dva potomci třídy `Event` - `FireEvent` a `WindEvent`. Třída `EventLogger<T>` umožňuje uchovávání událostí podle času jejich výskytu, což usnadňuje jejich následné zpracování.

`FireLogger` a `WindLogger` jsou specializované třídy pro logování specifických typů událostí souvisejících s požárem a větrem. Tyto třídy rozšiřují funkcionalitu základního `EventLoggeru<T>` o metody specifické pro daný typ události, jako jsou metody pro logování změn směru a rychlosti větru u `WindLoggeru` a metody pro sledování změn stavu hořících políček u `FireLoggeru`. Obě tyto třídy mají navíc metody pro vypsání souhrnů událostí do Unity Editoru.

6.17 Visualizer.cs

Třída `Visualizer` slouží k vizuální reprezentaci světa (jeho políček). Vytvořené 3D objekty neznají logiku samotné simulace a jsou měněny pouze na základě volaných metod hlavní logiky.

Při každé aktualizaci simulace získáme nově vzniklé události a podle nich poté aktualizujeme a zobrazíme změny.

`Visualizer` pracuje ve dvou režimech a to standardním a zjednodušeném. Režimy ovlivňují, zda-li jsou instance modelů vegetace a ohně vytvářeny a zobrazeny, nebo se pouze mění barvy (materiály) políček.

Třída je potomkem třídy `MonoBehaviour`. Skript `Visualizer.cs` se tak přidává do scén vlastnímu objektu `Visualizer` (obr. 6.4). Díky tomu poté můžeme v Unity Editoru jednotlivým objektům políček přiřazovat modely rostlin podle typu vegetace. Také lze přiřazovat materiál pro různé stavy nebo vlastnosti políček. V neposlední řadě můžeme upravit přímo takto z editoru vlastnost `TileHeightMultiplier`, který se zasluhuje o to, že více či méně zvětšuje nebo zmenšuje výškové zobrazení, což však nemá vliv na vlastnost políček. Jde pouze o způsob zobrazení.

Mezi metody třídy `Visualizer` patří `CreateWorldTiles`, která vytvoří reprezentaci všech políček světa instancováním odpovídajícího Prefabu (více o Prefabech je možné si přečíst v sekci 3.8).

Metody jako `SetAppropriateMaterial` a `MakeTileBurned` umožňují měnit vizuální vlastnosti políček na základě jejich stavu nebo vlastností, jako jsou vlhkost, typ vegetace či zda hoří nebo jsou shořelá.

Pro zlepšení výkonu (snížením počtu vykreslovaných objektů) a pro přitažlivější vizualizaci dochází ke kombinování vodních políček do větších celků. Tato

funkcionalita je implementována v metodách `CombineAllWaterTiles`, `FindContiguousWaterTileGroups` a `CombineWaterTilesOfGroup`. Tyto metody pracují s algoritmem pro hledání spojitých skupin vodních políček a následně vytvářejí jednotný mesh pro celou tuto skupinu.

Vlastnost Prefabu políček je nastavena na *static*. To znamená, že instance políček nechceme a nebudeme za běhu programu přesouvat, otáčet a podobně měnit. Toto nastavení může pomoci při výkonu aplikace.

6.18 WindIndicator.cs

Třída `WindIndicator` slouží k zobrazování aktuálního směru a rychlosti větru.

Třída má přiřazený objekt šipky (indikátoru `WindArrow`), který se nachází ve scéně. Tímto objektem otáčíme podle směru větru (metoda `UpdateIndicator`) správně vůči vizualizovanému světu.

Problém je, že při natáčení nebo přibližování hlavní kamery chceme držet tento indikátor na stejném místě, aby se hezky zobrazoval uživateli přesně tam, kde je to zapotřebí a nepřekrýval ostatní prvky uživatelského rozhraní. Nechceme přitom neustále měnit jeho pozici, protože by to bylo nepraktické.

Využijeme proto několika komponent:

Využijeme druhé kamery `WindArrowCamera`, která bude zachycovat pouze tento indikátor. Toho je docíleno tak, že nastavíme vlastnost `Culling mask` této kamery na vrstvu `wind`, na které je právě a pouze indikátor `WindArrow`. Pouze tedy objekt `WindArrow` je tak snímán touto kamerou. Výstup této kamery však nezobrazíme přímo kamerou druhou (hlavní kamerou), ale budeme místo toho tento výstup renderovat na speciální objekt s texturou.

`Target Texture` je textura, na kterou `WindArrowCamera` renderuje ukazatel větru. V našem případě je tato textura nazvaná `arrowTexture`. Obraz ukazatele větru je vykreslen tedy do této textury, která je následně použita v hlavním panel a tudíž je vždy hezky viditelná pro uživatele. Navíc se s ní snadno manipuluje, mění její umístění nebo rozměry.

V závislosti na pozici hlavní kamery vůči světu měníme pozici kamery `WindArrowCamera` vůči indikátoru. Při změně natočení hlavní kamery voláme metodu `UpdateIndicatorCameraAngle`. Indikátor se tak jeví, že ukazuje vždy správným směrem.

6.19 CameraHandler.cs

Třída `CameraHandler` je odpovědná za ovládání hlavní kamery, což umožňuje uživateli pohled na simulovaný svět z různých úhlů a vzdáleností. Zajišťuje a poskytuje možnosti rotace kamery podle uživatelských vstupů. Je ovládána třídou `InputHandler` zodpovědnou za zpracování vstupů od uživatele, popřípadě třídou `MainLogic` pokud je potřeba stav simulátoru pro vyhodnocení jestli, nebo kdy a jak nabízené funkce zavolat.

`CameraHandler` nabízí metodu `SetWorldCenter`, která nastavuje střed otáčení kamery, metoda `ZoomCamera` slouží pro ovládání zoom (přiblížení) kamery a

`RotateCamera` umožňuje hlavní rotaci kamery okolo nastaveného středu otáčení (střed zobrazeného světa). Do základní pozice nastaví kameru metoda `SetCamera`.

6.20 PredictionLogic.cs

Třída `PredictionLogic` představuje hlavní skript přidaný a spravující scénu `Predictions` našeho simulátoru. Slouží pro koordinaci logiky, stavů a metod souvisejících s predikcí šíření ohně ve vybraném světe.

Podle vlastního vnitřního stavu rozhoduje, jak interpretovat vstupy od uživatele. Důležitými metody jsou `GenerateData` a `PythonPredict`.

`GenerateData` využívá asynchronní operace v Unity, protože může využívat déle trvající zpracování datově náročných úkolů a my tak nemusíme blokovat uživatelské rozhraní, jako například pohyb kamery během generování těchto dat apod.

Třída načítá parametry ze třídy `InputHandler` definovaných uživatelem pro vlastnosti vegetace, vlhkosti, sklonu terénu a pravděpodobnosti šíření ohně.

Následně vygeneruje množství světů a na nich následně s náhodně vybranými počátečními hořícími políčky spouští několikrát simulaci. Díky tomu za pomocí třídy `FirePredictor` a funkce `GenerateHeatMap` pro každý svět získáme teplotní mapu (`heatMap`). Teplotní mapa představuje procentuálně vyjádřenou četnost, se kterou každé políčko na mapě za běhu mnoha těchto simulací hořelo. Pokud tedy určité políčko hořelo ve více simulacích, jeho hodnota na teplotní mapě bude vyšší.

Následně jsou data připojeny do souboru `datafile.json` do složky `PythonScripts`, která je ve složce `StreamingAssets`.

Metoda `PythonPredict` je určena k zavolání externího Python skriptu pro predikci rozšíření ohně. Serializuje vstupní data světa a zapálených políček a zavolá Python skript `main.py` za pomoci třídy `PythonCaller`.

Zpracuje výstup a pokud je úspěšný, zobrazí předpověď pomocí metody `ApplyHeatMapToWorld` třídy `Visualiser`.

6.21 PythonCaller.cs

Třída `PythonCaller` zajišťuje volání externího Python skriptu a následně si dokáže přečíst a vrátit jeho výstup.

V této třídě je možné a doporučené si změnit a nastavit vlastní cestu k Python interpretu (`pythonPath`), popřípadě jinému Python skriptu (`scriptPath`).

Standardní vstup, výstup a chybový výstup jsou přesměrovány, což umožňuje třídě `PythonCaller` číst a psát data přímo do běžícího procesu a snadněji tak komunikovat. Po spuštění procesu `PythonCaller` zapíše serializovaná vstupní data do standardního vstupu tohoto procesu. Tato data jsou ve formátu `JSON` popsaným v sekci 4.3 a reprezentují stav světa a další informace potřebné pro predikci. Pro načtení zpět do simulátoru stačí, aby Python skript napsal svou výslednou odpověď na standardní výstup, jako je v přiložené příkladové ukázce (`main.py`). Data této odpovědi (výsledky predikce), které Python skript vypočítal, by měly také být ve formátu `JSON`. Doporučujeme se podívat na tento příkladový

způsob komunikace, tedy jak na třídu `PythonCaller`, tak na samotný `main.py` skript.

6.22 main.py

Tento ukázkový Python skript slouží jako demonstrace propojení se simulátorem. Skript je možné i za běhu simulátoru kdykoliv upravovat. Soubor je nutné po úpravě vždy uložit, aby se změny projevily. Díky tomu je možné rychleji testovat svá řešení, protože není nutné zavírat a opětovně otevírat celý program simulátoru pro testování jiných algoritmů předpovědi Python skriptu.

Tento skript se nachází ve složce `PythonScripts`, která je ve složce `StreamingAssets`.

Při chybách Python skriptu zapisujeme tyto chybové zprávy přímo do logu v Unity Editoru, aby bylo snadněji možné odhalit chyby (obr. 6.5).

6.23 WorldSerialization.cs

Tento skript slouží pro ukládání a načítání stavu simulátoru a dalších tříd světa do formátu `JSON` s následnou možností uložení těchto dat v tomto formátu do souboru.

Metoda `ConvertToInputDataSerializationPackage` třídy `SerializableConversion` vytváří balíček pro vstupní data pro Python skript. Metoda `ConvertToMap` převede data zase zpět na mapu specializovanou na typ float (`Map<float>`) pro simulátor.

Uživatelům, kteří chtějí svůj svět serializovat pro jakékoliv budoucí využití, poslouží metoda `ConvertToWorldSerializable`, která dokáže převést `World` objekt na jeho serializovatelnou formu. Zde je tedy například možné přidat, pokud bychom chtěli serializovat i jiné (nové) vlastnosti světa nebo políček. Nebo je zde možné naopak některé vlastnosti jednoduše odebrat.

6.24 Settings.cs

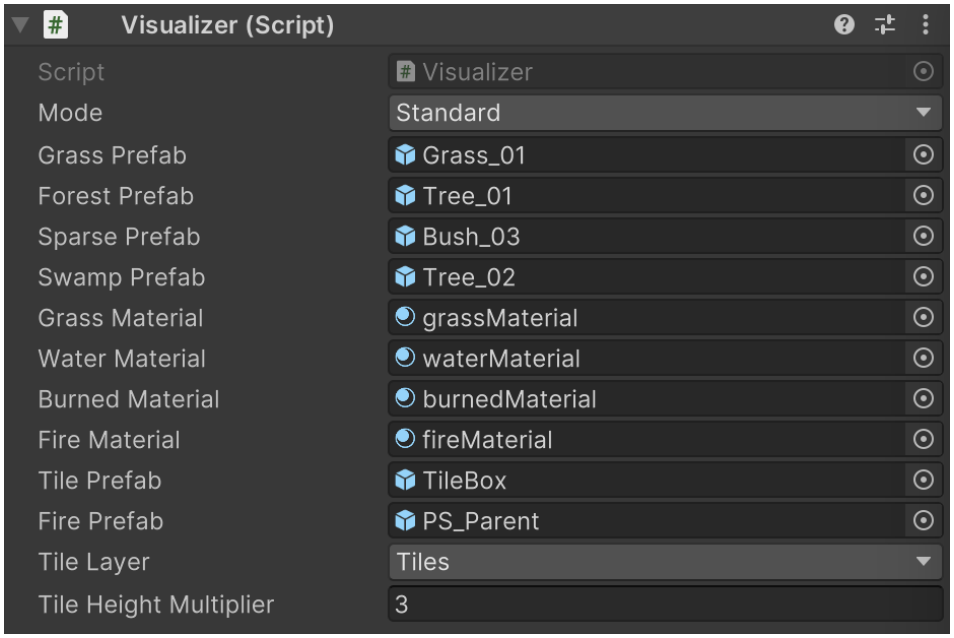
Třída `Settings.cs` poskytuje flexibilní způsob, jak nakládat s uživatelskými preferencemi a konfiguracemi.

Tento skript obsahuje třídy a metody pro načítání a ukládání nastavení, které mohou být dále použity na různých místech programu.

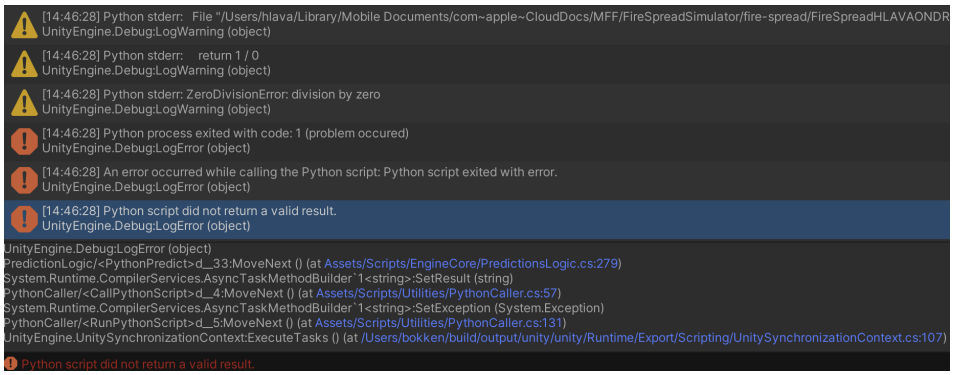
Například, pomocí přepínače `useSimplifiedWorldVisualization` můžeme dynamicky měnit detailnost vizualizace světa v závislosti na výkonnosti cílového zařízení uživatele (použité v metodě `PrepareForNewWorld` třídy `MainLogic`).

Podobně pokud je volba `saveTerrainAutomatically` aktivní, program automaticky ukládá nově generované světy do složky `SavedWorlds`.

Díky této třídě mohou uživatelé nastavit některé vlastnosti chování aplikaci podle svých preferencí jednou a očekávat, že tyto nastavení budou platné i pro budoucí spuštění této aplikace.



Obrázek 6.4 Ukázka Visualizer skriptu připojeného k objektu Visualizer ve SandBox scéně Unity Editoru.



Obrázek 6.5 Ukázka úmyslně vyvolané chyby v Python skriptu zavolaného při spuštění simulátoru přes Unity Editor.

6.25 FileBrowserHandler.cs

Pro import světů využíváme `UnitySimpleFileBrowser` [38], který poskytuje grafické rozhraní pro navigaci v souborovém systému operačních systémů Windows i macOS. Toto grafické rozhraní usnadňuje uživatelům výběr souborů. Tento balíček také umožňuje nastavit filtry pro specifické typy souborů, což přispívá k lepší organizaci a přehlednosti.

Třída `FileBrowserHandler` využívá tento `UnitySimpleFileBrowser` plugin.

Metoda `ImportFile` nastavuje filtry pro soubory a otevírá dialog pro výběr souboru k načtení. Tato funkce je asynchronní a využívá Unity coroutines k zobrazení dialogu bez blokování uživatelského rozhraní. Podobně pracuje metoda `SaveFile` pro ukládání souborů.

6.26 FileManagementService.cs

Třída `FileManagementService` slouží pro správu souborů při načítání a ukládání světů. Spoléhá a těsně spolupracuje se třídou `FileBrowserHandler` pro interakci s uživatelem během výběru souborů.

Metoda `ImportFile` zajišťuje načítání souborů skrze `FileBrowserHandler`. Přes pomocnou funkci `LoadWorldFromFile` validuje soubory a extrahuje a zpracovává data (světa) uložené buď v `JSON` formátu nebo jako obrázek (výšková mapa).

Závěr

V této kapitole si shrneme a zhodnotíme, jak se nám povedlo naplnit cíle z úvodu této práce. Uvedeme si také možná budoucí rozšíření a směřování tohoto projektu.

Projekt, který jsme vytvořili, modeluje a simuluje šíření ohně v krajině a poskytuje uživatelům nástroje pro vizualizaci a pochopení složitosti tohoto dynamického procesu.

Jak v samotné aplikaci simulátoru, tak i v Unity projektu je možné snadno nalézt a měnit parametry ovlivňující způsob průběhu simulace. Simulace je založena na jednoduchých nedeterministických pravidlech. Díky tomu je možné snadněji simulátor uzpůsobit svým potřebám. Přestože je simulace zjednodušená, uživatelé si mohou všimnout, jak se i na základě několika málo faktorů ovlivňujících tuto simulaci dá jen velmi obtížně předpovídat konečné rozšíření požáru.

Udělali jsme maximum pro to a myslíme si, že se nám povedlo splnit všechny naše vytyčené cíle pro tento projekt 2.1.5.

Simulátor umožňuje vytvářet zcela nové terény a světy, ukládat je a opětovně načítat. Vytvářet tyto různě vypadající světy mohou všichni uživatelé. To jak ti, kteří používají postavenou aplikaci simulátoru, tak programátoři a to jednoduše a přímo v kódu projektu kombinací připravených funkcí. Simulátor má také velmi jednoduchá pravidla pro simulaci, kterou tak lze snadněji upravit svým potřebám. Aplikace je uživatelsky přívětivá a přehledná - třídy a skripty jsou rozdělené podle funkcí, část reprezentující svět je oddělena od Unity komponent a je tak možné naopak měnit například vlastnosti tohoto světa bez nutnosti hlubšího porozumění Unity. Aplikace je určitými směry velmi dobře rozšiřitelná.

Vývojáři tedy mohou na různých úrovních komplexnosti zasahovat do skriptů a měnit tak odlišné části a komponenty simulátoru. Na základě jednoduchých změn hodnot dále mohou přizpůsobit mnoho důležitých proměnných nebo upravovat celé algoritmy. Bez zásahu do kódu programu je také možné ze samotného Unity Editoru jednoduše měnit další řadu věcí, jako například přidávat vlastní modely rostlin, měnit použité materiály, upravovat rychlost pohybu kamery a mnoho dalšího. Samotný simulátor také umožňuje zobrazit predikci šíření ohně a podporuje napojení vlastního Python skriptu pro vlastní předpověď bez nutnosti znát programovací jazyk C# použitý pro tento projekt.

Díky jednoduchému uživatelskému rozhraní a přizpůsobitelnému designu mohou uživatelé snadno ovládat simulátor a upravit jej k vlastním vzdělávacím účelům bez nutnosti rozsáhlého studia. Tutoriál zabudovaný v aplikaci mnohdy eliminuje potřebu čtení uživatelského manuálu.

Projekt je navržen s myšlenkou na rozšiřitelnost, což nám otevírá široké možnosti pro jeho další rozvoj.

Pro zvýšení interaktivnosti a zapojení uživatelů můžeme do budoucna do simulátoru přidat herní prvky. Simulátor má spoustu připravených tříd jejichž kombinací bychom mohli implementovat například právě tuto herní část. V hlavním menu by tak snadno mohlo přibýt další tlačítko `Game`, které by uživatele přeneslo do nově vytvořené scény využívající aktuální nebo nově vytvořené třídy a komponenty v Unity. Například bychom mohli do této hry přidat úkoly na zastavení šíření ohně nebo strategické plánování umístění protipožárních bariér. Po pár krocích

simulace by tak například měl hráč možnost opět klikat na políčka a přidat tak další bariéry. Podle obtížnosti („levelu“) by mohl program se snažit zvolit takové ohniska požáru, které by měly potenciál se co nejvíce rozšířit. Pro to bychom využívali algoritmus nebo vlastní formu umělé inteligence a připravené funkce pro předpovídaní rozšíření ohně. Další možnost zvyšování obtížnosti by mohlo být počtem zapojených faktorů ovlivňujících šíření tohoto ohně. Program je připravený pro relativně snadné změny a přizpůsobení i v tomto ohledu. Alternativně by hra také ale mohla být navržena obráceně a hráč by se tak snažil rozdělat oheň v místech, kde by měl největší devastující dopad na krajinu.

Další nápady se týkají možností pro úpravu terénu. V aktuální verzi je možné pouze omezeně měnit parametry generace světa, jako je například množství řek či jezer, velikost světa apod. Do budoucna chceme umožnit i uživatelům, kteří neovládají programování, podrobněji ovlivnit vzhled generovaného světa. Uživatelé by mohli upravovat terén a typ vegetace v reálném čase pomocí nástroje podobného štětci. Tažení štětce by vybíralo políčka, stejně jako aktuálně vybíráme políčka pro zapálení, přičemž by se podle typu štětce upravovala například výška terénu. K tomu bychom mohli využít mimo jiné funkce pro výběr sousedních políček.

S rostoucím významem technologií virtuální reality (VR) vidíme velký potenciál v adaptaci simulátoru pro tyto platformy. Toto rozšíření by mohlo zvýšit a zlepšit zážitek z používání a poskytnout uživatelům ještě zábavnější způsob ovládání tohoto nástroje.

Například hasiči by mohli na dnech otevřených dveří poskytnou tento simulátor k zábavnému a interaktivnímu představení své práce dětem. To by mohlo zvýšit zájem dětí o pochopení, jak se šíří požáry, nebo dokonce motivovat některé z nich k budoucí kariéře v hasičském sboru. Tato forma by přinesla netradiční způsob zábavy formou hry a zvýšila tak zájem o pochopení dynamiky šíření ohně.

Díky tomu, že jsme použili herní engine Unity, bude vývoj pro jiné platformy v budoucnu o mnoho snazší. Projekt sám jako takový může posloužit začínajícím vývojářům jako jeden z jejich prvních rozsáhlejších projektů, a to buď jako zdroj inspirace, nebo jako základ pro učení se strojovému učení v Pythonu či práce se samotným Unity.

S předpokládaným rozvojem a rozšířením funkcí a dosahu našeho simulátoru může sloužit opravdu širokému spektru uživatelů.

Literatura

1. NASA-FIRMS. *Fire Information for Resource Management System* [online]. [cit. 2024-05-01]. Dostupné z: https://firms.modaps.eosdis.nasa.gov/map/.
2. Consortium, Concord (ed.). *GeoHazard: Modeling Natural Hazards and Assessing Risks* [online]. [cit. 2024-05-01]. Dostupné z: https://concord.org/our-work/research-projects/geohazard/.
3. Janik, Piotr. *Wildfire Simulation* [online]. [cit. 2024-05-01]. Dostupné z: https://wildfire.concord.org.
4. Mirolyubov, Gleb. *Fire Spreading Simulation - Unity 2019.1.9f* [online]. [cit. 2024-05-01]. Dostupné z: https://github.com/glebmirolyubov/fire-spreading-simulation.
5. Corse Pascal Paoli, Université de. *ForeFire* [online]. [cit. 2024-05-01]. Dostupné z: https://github.com/forefireAPI/firefront.
6. Chaiyasirisuwan, Nattapon. *Fire-spreading-Simulation* [online]. [cit. 2024-05-01]. Dostupné z: https://github.com/Haleluyaz/Fire-spreading-Simulation.
7. Stats, StatCounter Global. *Desktop Operating System Market Share Worldwide* [online]. [cit. 2024-05-01]. Dostupné z: https://gs.statcounter.com/os-market-share/desktop/worldwide.
8. Wikipedia. *Game Engine* [online]. [cit. 2024-05-01]. Dostupné z: https://en.wikipedia.org/wiki/Game_engine.
9. *Godot Docs – 4.2 branch* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.godotengine.org/en/stable/index.html#.
10. Technologies, Unity. *Unity - Manual: Unity User Manual 2022.3 (LTS)* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/2022.3/Documentation/Manual/.
11. Epic Games, Inc. *Unreal Engine 5.4 Documentation* [online]. [cit. 2024-05-01]. Dostupné z: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-4-documentation.
12. Wirtz, Bryan. *Unity vs Unreal Game Engine: Which Is Better?* [online]. [cit. 2024-05-01]. Dostupné z: https://www.gamedesigning.org/engines/unity-vs-unreal/.
13. Tristem, Ben. *Unity vs. Unreal: Which Game Engine is Best For You?* [online]. [cit. 2024-05-01]. Dostupné z: https://blog.udemy.com/unity-vs-unreal-which-game-engine-is-best-for-you/.
14. Technologies, Unity. *Unity - Manual: Unity User Manual 2022.3 (LTS)* [online]. [cit. 2024-05-01]. Dostupné z: https://learn.unity.com.
15. Technologies, Unity. *Unity - Scripting API: GameObject* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/2022.3/Documentation/ScriptReference/GameObject.html.

16. Technologies, Unity. *Unity - Scripting API: Camera* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/2022.3/Documentation/ScriptReference/Camera.html.

17. Technologies, Unity. *Unity - Manual: Types of Light* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/2022.3/Documentation/Manual/Lighting.html.

18. Technologies, Unity. *Unity - Scripting API: Renderer* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/2022.3/Documentation/ScriptReference/Renderer.html.

19. Technologies, Unity. *Canvas | Unity UI | 2.0.0* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.ugui@2.0/manual/class-Canvas.html.

20. Technologies, Unity. *Unity. Unity - Manual: Prefabs* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/2022.3/Documentation/Manual/Prefabs.html.

21. Technologies, Unity. *Unity - Manual: C# Compiler* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/Manual/CSharpCompiler.html.

22. Technologies, Unity. *Unity - Manual: Order of Execution for Event Functions* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/2022.3/Documentation/Manual/ExecutionOrder.html.

23. Technologies, Unity. *Unity - Manual: MonoBehaviour* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/Manual/class-MonoBehaviour.html.

24. Technologies, Unity. *Unity - Manual: Coroutines* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/2022.3/Documentation/Manual/Coroutines.html.

25. Technologies, Unity. *Unity - Manual: Script Serialization* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/2022.3/Documentation/Manual/script-Serialization.html.

26. Technologies, Unity. *Unity - Manual: Terrain* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/Manual/script-Terrain.html.

27. Biagioli, Adrian. *Understanding Perlin Noise* [online]. [cit. 2024-05-01]. Dostupné z: https://adrianb.io/2014/08/09/perlinnoise.html.

28. *Generating Random Fractal Terrain* [online]. [cit. 2024-05-01]. Dostupné z: https://web.archive.org/web/20170701142426/http://www.gameprogrammer.com/fractal.html#diamond.

29. *Voronoi diagram - Wikipedia* [online]. [cit. 2024-05-01]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Voronoi%20diagram&oldid=1155703644.

30. *JSON* [online]. [cit. 2024-05-01]. Dostupné z: https://www.json.org/json-en.html.

31. Jenifer, Tidwell. *Designing Interfaces: Patterns for Effective Interaction Design*. O'Reilly Media, Inc., 2010. Second Edition. ISBN 978-1-449-302733.

32. *Low-Poly Simple Nature Pack | 3D Landscapes | Unity Asset Store* [online]. [cit. 2024-05-01]. Dostupné z: https://assetstore.unity.com/packages/3d/environments/landscapes/low-poly-simple-nature-pack-162153.

33. APPLE. *Porting your macOS apps to Apple silicon* [online]. [cit. 2024-05-01]. Dostupné z: https://developer.apple.com/documentation/apple-silicon/porting-your-macos-apps-to-apple-silicon.

34. TECHNOLOGIES, Unity. *Unity - Manual: System Requirements for Unity 2022.3* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/2022.3/Documentation/Manual/system-requirements.html.

35. TECHNOLOGIES, Unity. *Unity - Manual: Streaming Assets* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/2022.3/Documentation/Manual/StreamingAssets.html.

36. TECHNOLOGIES, Unity. *Unity - Manual: Layers* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/2022.3/Documentation/Manual/Layers.html.

37. TECHNOLOGIES, Unity. *Unity - Manual: Event System* [online]. [cit. 2024-05-01]. Dostupné z: https://docs.unity3d.com/2022.3/Documentation/Manual/EventSystem.html.

38. YASIRKULA. *UnitySimpleFileBrowser - A uGUI based runtime file browser for Unity 3D* [online]. [cit. 2024-05-01]. Dostupné z: https://github.com/yasirkula/UnitySimpleFileBrowser.

Seznam obrázků

2.1 Ukázka online nástroje Wildfire Explorer. 11
2.2 Ukázka jedné z existujících aplikací napsaných v Unity. 14
2.3 Demo ukázka aplikace FireFront. 14
2.4 Ukázka velmi jednoduchého a volně dostupného simulátoru. . . . 15

3.1 Ukázka rozhraní aplikace Unity Hub. 22
3.2 Ukázka rozhraní Unity Editoru. 24
3.3 Ukázka rozhraní okna s herními objekty (GameObject) aplikace Unity Editor. 24
3.4 Ukázka rozhraní oken Scene and Game aplikace Unity Editor. . . 25
3.5 Ukázka rozhraní okna projektu aplikace Unity Editor. 25
3.6 Ukázka rozhraní okna Inspector aplikace Unity Editor. 26

4.1 Diagram hlavního menu, šipky značí možné přechody mezi scénami a ovládacími prvky 42

5.1 Okno hlavní nabídky (Main menu) simulátoru. 44
5.2 Okno informace o simulátoru. 44
5.3 Okno nastavení simulátoru. 45
5.4 Ukázka scény tutoriálu simulátoru. 46
5.5 Ukázka SandBox scény simulátoru. 47
5.6 Okno simulátoru sloužící k importování a načtení světa ze souboru. 49
5.7 Příklad okna simulátoru při zobrazení grafu. 49
5.8 Ukázka Predictions scény simulátoru. 50

6.1 Ukázka InputHandler skriptu připojeného k objektu InputHandler ve SandBox scéně Unity Editoru. 57
6.2 Ukázka vybrané vrstvy Ignore Raycast pro objekt (model) vegetace. 59
6.3 Ukázka připojení metody třídy InputHandler při kliknutí na tlačítko. 59
6.4 Ukázka Visualizer skriptu připojeného k objektu Visualizer ve SandBox scéně Unity Editoru. 65
6.5 Ukázka úmyslně vyvolané chyby v Python skriptu zavolaného při spuštění simulátoru přes Unity Editor. 65

Seznam tabulek

4.1 Ukázky Perlinova šumu s různými hodnotami lacunarity 34
4.2 Ukázky Perlinova šumu s různými hodnotami persistence 34
4.3 Ukázka Perlinova šumu a jeho obarvené varianty pro 8 rozsahů hodnot 34
4.4 Porovnání ortografické (vlevo) a perspektivní kamery v Unity. . . 40
4.5 Porovnání vzhledu nespojených a spojených vodních políček. . . . 41

A Přílohy

Přehled elektronických příloh

1. `README.md` - Soubor popisující strukturu příloh s kontaktem a odkazem na Gitlab repozitář Unity projektu.

2. Unity projekt - `FireSimulationUnity` - Adresář projektu simulátoru obsahující všechny potřebné soubory pro otevření a práci s tímto projektem v Unity.

3. Build pro macOS - `FireSimulatorMac.app` - Vytvořená a přímo spustitelná aplikace pro operační systém macOS společnosti Apple.

4. Build pro Windows - `FireSimulatorWin` - Složka s vytvořenou a přímo spustitelnou aplikací pro operační systém Microsoft Windows.