VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HRA PRO OCULUS QUEST

GAME FOR OCULUS QUEST

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB KRYŠTŮFEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JIŘÍ POMIKÁLEK,**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Kryštůfek Jakub**
Program: Informační technologie
Název: **Hra pro Oculus Quest**
Game for Oculus Quest
Kategorie: Počítačová grafika

Zadání:

1. Seznamte se s brýlemi Oculus Quest 2 pro virtuální realitu, jejich ovládacími periferiemi a nástroji pro vývoj aplikací.
2. Nastudujte herní engine Unity a techniky tvorby her pro virtuální realitu.
3. Navrhněte počítačovou hru v prostředí virtuální reality.
4. Implementujte navrženou hru na platformě Unity. Při implementaci dbejte na čitelnost a přehlednost zdrojových kódů.
5. Proveďte uživatelské testování vytvořené počítačové hry, analyzujte jeho výsledky.
6. Vytvořte krátké video pro prezentování projektu.

Literatura:

- dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1., 2., 3. a částečně bod 4.

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

Vedoucí práce: **Pomikálek Jiří, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 1. listopadu 2021

Abstrakt

Cílem této bakalářské práce je vytvořit prototyp hry ve virtuální realitě pro cílovou platformu Oculus Quest 2. Hra byla vytvořena v herním enginu Unity s kladeným důrazem na optimalizaci a rozšiřitelnost implementovaných systémů. V rámci hry byl vytvořen systém řízení autonomních agentů pomocí konečných automatů a plánovací architektury s názvem cílem řízené plánování akcí. Jednou z nejzajímavějších herních mechanik je systém šíření ohně implementovaný pomocí celulárního automatu.

Abstract

The aim of this bachelor thesis is to create a prototype of a videogame in virtual reality for target platform Oculus Quest 2. The game was created with game engine Unity with emphasis on optimalization and extensibility of implemented systems. Within the implemented game was created system of autonomous agents via finite state machines and planning architecture goal-oriented action planning. One of the most insteresting game mechanics is fire propagation system implemented with cellular automaton.

Klíčová slova

Unity, Virtuální Realita, VR, hra, herní prototyp, C#, GOAP, FSM, automaty, konečný automat, celulární automat, šíření ohně, multiagentní systém, autonomní agenti

Keywords

Unity, Virtual reality, VR, game, game prototype, C#, GOAP, FSM, automatons, finite-state machine, cellular automaton, fire spread, multi-agent system, autonomous agents

Citace

KRYŠTŮFEK, Jakub. *Hra pro Oculus Quest*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Pomikálek,

Hra pro Oculus Quest

Prohlášení

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

.....
Jakub Kryštůfek
9. května 2022

Poděkování

Tímto bych rád upřímně poděkoval vedoucímu mé bakalářské práce, panu Ing. Jiřímu Pomikálkovi za jeho ochotu vést mou práci, vřelý přístup a cenné rady. Dále bych rád poděkoval všem, kteří byli ochotni zúčastnit se uživatelského testování mé práce.

Obsah

Seznam použitých zkratek

FSM Konečný automat (z anglického „Finite-state machine“)

GOAP Cílem řízené plánování akcí (z anglického „Goal-Oriented Action Planning“)

GPU Grafická karta

GUI Grafické rozhraní

NPC Nehratelná postava ve hře (z anglického „Non-player character“)

VR Virtuální realita

XR Rozšířená realita (z anglického „Extended reality“)

Kapitola 1

Úvod

V dnešní době se na herním trhu začíná objevovat čím dál více herních titulů pro virtuální realitu. Je tomu tak především kvůli nedávnému vydaní virtuálních brýlí Oculus Quest 2, které jsou cenově dostupné pro širší škálu uživatelů, přičemž kvalitou se vyrovnají konkurenčním brýlím i s trojnásobnou cenou. Kromě ceny má Oculus Quest 2 také zásadní výhodu v tom, že k němu není potřeba žádné další vybavení nebo výkonný počítač, je totiž naprosto samostatný. Z těchto důvodů byl právě Oculus Quest 2 vybrán jako cílová platforma, pro kterou bude v rámci této práce vytvořen prototyp hry. Seznámení s těmito virtuálními brýlemi je věnována Kapitola 2.

Cílem této bakalářské práce je tedy vytvořit prototyp hry ve virtuální realitě, který bude sloužit jako rozšiřitelný základ pro plnohodnotnou hru. Hra je navržena a implementována pro virtuální brýle Oculus Quest 2 s důrazem kladeným na dobrou optimalizaci, intuitivní ovládání a dobrou rozšiřitelnost.

Námětem na téma hry je herní řada Black & White, která se řadí mezi netypický žánr her na boha. To jsou hry, ve kterých se hráč vžije do role boha, který může pomocí svých božích sil ovlivňovat svět kolem něj. V navržené hře této práce je hráč v herním světě bohem, jehož údělem je pomáhat a řídit vesnici tak, aby co nejvíce prosperovala. Myšlenka hry je detailně rozvinuta v Kapitole 5.

Vesnice je navržena jako autonomní multiagentní systém, který hráč může ovlivňovat svými interakcemi. Jednou z hráčových výzev při snaze o vývoj vesnice je dynamické šíření ohně realizované pomocí celulárního automatu. Právě autonomní multiagentní systém a systém šíření ohně jsou spolu s implementací ve virtuální realitě hlavními obory, kterými se práce do detailu zabývá. Popisu funkcionality všech systémů výsledné hry je věnována Kapitola 6.

Pro implementaci autonomního systému agentů bylo využito konečných automatů a plánovací architektury cílem řízeného plánování akcí. Tyto dvě metodologie byly zkombinovány pro efektivní a dobře rozšiřitelný systém. O výpočetních modelech využitých pro systémy této práce pojednává Kapitola 4.

Hra je implementována v herním vývojovém prostředí Unity, které je v dnešní době jedním z nejpoužívanějších pro vývoj virtuálních her. Pro vytvoření realistických interakcí hráče ve hře je využito také fyzikálního jádra PhysX, který je vestavěný přímo v Unity. O procesu vývoje v tomto vývojovém prostředí se zabývá Kapitola 3.

Po dokončení implementace hry bylo pro otestování jejích herních systémů zorganizováno uživatelské testování, jehož průběh je popsán v Kapitole 7.

Kapitola 2

Oculus Quest 2

Oculus Quest 2 jsou virtuální brýle, vytvořeny společností Oculus, která byla v roce 2014 koupena společností Facebook[22] (k dnešnímu dni již společnost Meta). Na trh byl uveden 13. října 2020 s bezkonkurenční cenou 299 $ (přibližně 6 500 Kč). Na rozdíl od většiny konkurenčních virtuálních brýlí, jako je například Valve Index[1] nebo VIVE Pro 2[2], které podporují pouze tzv. PC VR a potřebují být pro renderování a výpočetní sílu připojeny k výkonnému PC, je Oculus Quest 2 tzv. „All-in-One Device“. To znamená, že brýle mají vlastní výpočetní sílu a fungují tak naprosto samostatně. Oculus Quest 2 nabízí také možnost připojení k PC přes internetovou síť nebo USB-C kabel a využívat jej jako standardní PC VR brýle. [15]

Součástí Oculus Quest 2 nejsou pouze virtuální brýle, ale také ovladače sloužící k interakci ve virtuální realitě. Celou sadu lze vidět na obrázku 2.1.

Obrázek 2.1: Obrázek virtuálních brýlí Oculus Quest 2 i s jejich ovladači. Obrázek je převzatý z internetu[3].

Díky tomu, že se na trhu objevují VR brýle jako Oculus Quest 2, které jsou finančně dostupné pro větší skupinu lidí, roste poptávka nejen po těchto zařízeních, ale také po

[1] https://store.steampowered.com/valveindex
[2] https://www.vive.com/us/product/vive-pro2-full-kit/overview/
[3] https://www.ign.com/articles/oculus-quest-2-review

aplikacích a hrách s možností využití virtuální reality. Předpokládá se, že v nadcházejících letech popularita virtuální reality stále poroste, a proto bude vývoj aplikací a her pro virtuální realitu čím dál tím více vyhledávaným a finančně výnosným artiklem. [11]

Aktuálně se ve světě mluví o pojmu „metaverse“, což je představa jakéhosi nového virtuálního internetu. Mělo by se jednat o síť virtuálních světů, mezi kterými se bude moci uživatel volně přemísťovat tak, jako na internetu přepíná z jedné stránky na druhou. V těchto světech bude možné vyvíjet různé aktivity typu společenských událostí, pracovních pohovorů, hraní videoher nebo sledování virtuálního koncertu. Metaverse není něco, co je vyvíjeno jednou společností, ale je to kolektivní projekt, který má sloužit k propojení produktů všech zapojených společností. Aktuálně jsou největšími iniciátory tohoto projektu společnosti Meta, Microsoft a Nvidia, herní platforma Roblox a herní studio Epic Games. [12]

2.1 Specifikace

Oculus Quest 2 je nyní prodáván ve dvou variantách, které se liší pouze ve velikosti úložiště. Jedna varianta má úložiště 128GB a druhá 256GB. Všechny ostatní specifikace jsou pro obě varianty stejné a jsou vypsány v tabulce 2.1. Pro srovnání je v tabulce uvedena i specifikace předchozí generace Oculus Quest. [13]

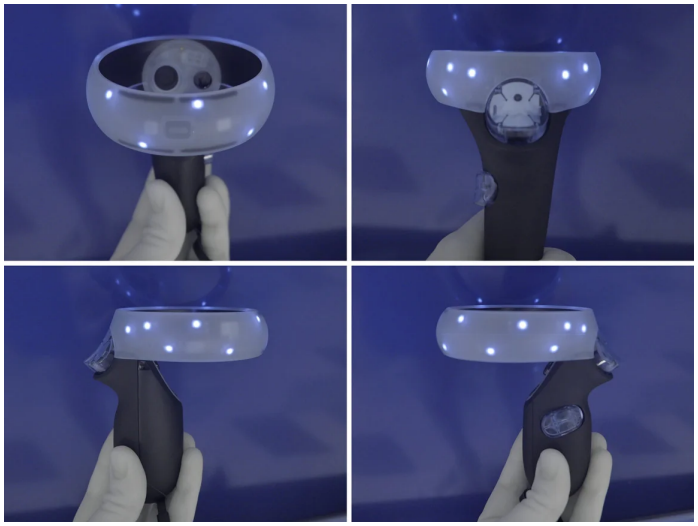
	Oculus Quest 2	Oculus Quest
Displej	LCD, 1832 × 1920px na oko	OLED, 1440 × 1600px na oko
Obnovovací frekvence	60/72/90/120Hz	60/72Hz
Zorný úhel	100°	100°
Sledování pohybu	6DOF, 4 kamery	6DOF, 4 kamery
CPU	Qualcomm® Snapdragon XR2 Platform	Qualcomm® Snapdragon 835
RAM	6GB	4GB
Úložiště	128/256GB	64/128GB
Zvuk	Integrované reproduktory, 3.5mm konektor	Integrované reproduktory, 3.5mm konektor
Hmotnost	503g	571g

Tabulka 2.1: Specifikace Oculus Quest 2 v porovnání s předchozí generací. [13]

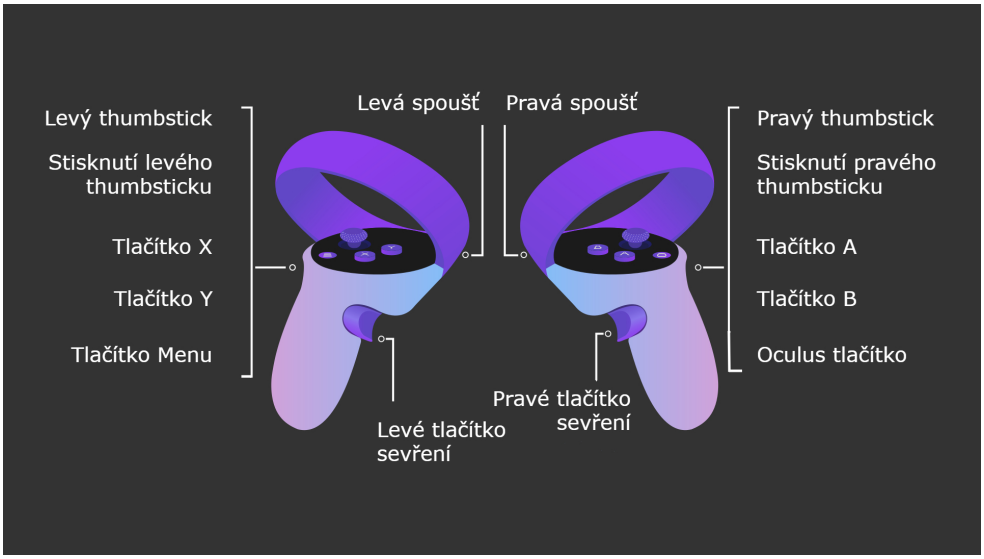
2.2 Ovladače

Oculus Quest 2 má dva ovladače, jejichž pozice je snímána čtyřmi kamerami umístěnými na přímo na brýlích. Kamery snímají pozici ovladačů pomocí patnácti infračervených bodů rozmístěných po celé ploše obou ovladačů. Ty lze vidět na obrázku 2.2, který byl pořízen infračervenou kamerou. Ovladače mají na horní části tři tlačítka a jeden joystick, který slouží zároveň jako tlačítko. Na boku je tlačítko pro sevření, tzv. „Grip button“ a na zadní straně ovladače je tlačítko spouště, tzv. „Trigger button“. Přesné rozložení tlačítek lze vidět na obrázku 2.3.

Každý z ovladačů také disponuje gyroskopickými senzory, díky kterým dokáží brýle odhadovat pozici ovladačů i v situaci, kdy se ovladače nachází mimo zorné pole kamer. [15]



Obrázek 2.2: Snímky Oculus Quest 2 ovladačů pořízené infračervenou kamerou znázorňují infračervené body, které slouží ke sledování ovladačů v prostoru. Obrázek převzatý z internetu[4].



Obrázek 2.3: Schéma tlačítek na ovladačích Oculus Quest 2. Obrázek převzatý z internetu[5].

[4] www.reddit.com/r/OculusQuest/comments/ca28kn/15_tracking_dots_per_touch_controller_seen/

[5] www.docs.microsoft.com/cs-cz/windows/mixed-reality/altspace-vr/getting-started/oculus-controls

Kapitola 3

Unity

Unity je multiplatformní herní engine, podporující vývoj her na Windows, macOS i Linux. Byl vyvinut společností Unity Technologies v roce 2005. Od té doby byl však významně rozšířen a vylepšen. Podporuje vývoj her pro různé platformy, např. iOS, Android, Windows, Mac, PlayStation, Xbox a mnoho dalších. Umožňuje vývoj her ve 2D, 3D a má také podporu pro XR[1].

Unity je pro jednotlivce i firmy zcela zdarma, pokud jejich roční obrat nepřekračuje 100 000 $. Pakliže jejich obrat tuto částku překročí, musí si zvolit jeden z placených plánu[19].

Další z jeho předností je vestavěné fyzikální jádro PhysX. PhysX je fyzikální jádro fungující v reálném čase vyvíjené společností Nvidia. Obsahuje efektivní multithreading[2] a akceleraci fyzikální simulace pomocí GPU. [21]

3.1 Úspěšné tituly pro virtuální realitu

V Unity byla vytvořena řada velice kvalitních a úspěšných VR her. Mezi ty nejúspěšnější patří např. Beat Saber, Job Simulator a Superhot VR.

3.1.1 Beat Saber

Beat Saber patří do skupiny rytmických videoher, u kterých musí hráč zapojit smysl pro rytmus a koordinaci těla. Hra byla vydána v roce 2018 českým studiem Beat Games. Ve hře má hráč modrý a červený meč, každý v jedné ruce. Směrem k nim se přibližují barevné kostky a cílem hráče je tyto kostky v rytmu hudby přeseknout. Hráč však musí kostky přeseknout mečem odpovídající barvy, tedy červené kostky červeným mečem a modré kostky modrým mečem. [23]

Celkově měla hra velký úspěch. Bylo prodáno více než 4 milióny kopií hry, což z ní dělá prozatím nejprodávanější VR hru. [14]

[1] XR (Extender Reality) - je to skupina, do které patří: virtuální realita, mixovaná realita a rozšířená realita

[2] multithreading - možnost výpočtu na více jádrech procesoru zároveň

Obrázek 3.1: Náhledový obrázek hry Beat Saber, převzatý z internetu[3].

3.1.2 Job Simulator

Job Simulator je komediálním simulátorem povolání z reálného světa. Hráči si mohou vyzkoušet práci automechanika, šéfkuchaře, prodavače nebo třeba práci v kanceláři. Během hry hráč interaguje s roboty, kteří mají podobu starých CRT monitorů a ti hru doprovází skvělými hláškami a celkově kvalitním humorem. [23]

Hra byla vydána studiem Owlchemy Labs v roce 2019 a prodalo se přes milion kopií. [5]

Obrázek 3.2: Náhledový obrázek hry Job Simulator, převzatý z internetu[4].

3.1.3 Superhot VR

Superhot je hra vydaná v lednu 2016 původně pro PC, bez podpory VR, ale později v roce 2016 byla vydána varianta hry pro virtuální realitu. Hráč prochází úrovněmi, ve kterých má za úkol zneškodnit všechny červené protivníky. Může k tomu využít různé předměty, jako jsou střelné zbraně, ruční zbraně a nebo třeba láhev od piva. Hlavní mechanika hry

[3] https://store.steampowered.com/app/620980/Beat_Saber/
[4] https://store.steampowered.com/app/448280/Job_Simulator/

spočívá v netradičním pojetí plynutí času, kdy čas hry plyne pouze, když se hráč pohybuje, což hráči umožňuje zorientovat se v aktuální situaci a vymyslet taktiku jak protivníky co nejefektivněji porazit. [23]

Hra byla vyvinuta nezávislým studiem Superhot Team a prodalo se přes 2 miliony kopií. [6]

Obrázek 3.3: Náhledový obrázek hry Superhot VR, převzatý z internetu[5].

3.2 Základní práce s Unity

V této kapitole jsou objasněny základní pojmy spojené s vývojem her v herním enginu Unity, které jsou nezbytné pro pochopení popsaných implementací.

3.2.1 Scéna

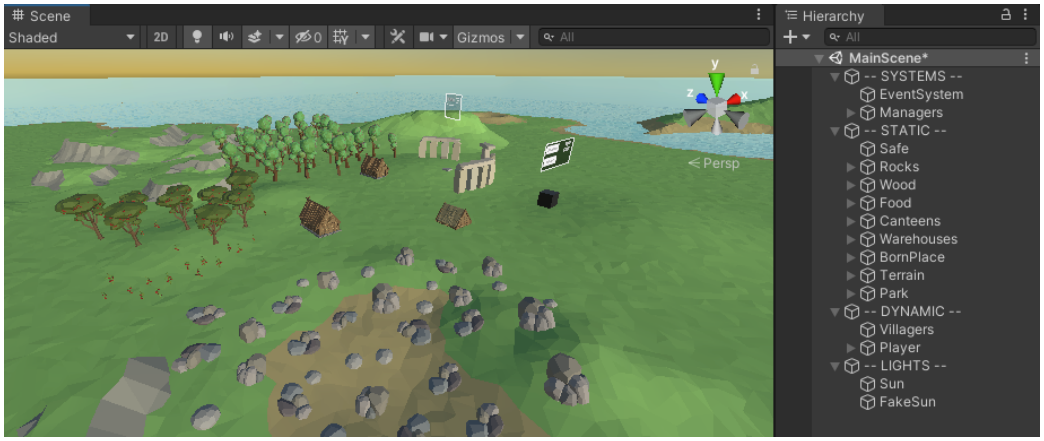
Scéna je prostředí, ve kterém je hra tvořena. Obsahuje všechny objekty, osvětlení a celkově vše, co může v herním světě být. Většina her je tvořena pomocí většího množství scén, které reprezentují např. úrovně hry nebo hlavní menu. Každá scéna obsahuje hierarchii herních objektů, které definují danou scénu a jsou uspořádány ve stromové struktuře vyobrazené na obrázku 3.4. [20]

3.2.2 Inspektor

Inspektor je jedním z oken v Unity editoru. Jak toto okno v editoru vypadá, lze vidět na obrázku 3.5. Inspektor slouží k modifikaci herních objektů následujícími způsoby:

- Nastavení jména, štítku a vrstvy.
- Přidání a odebrání komponent.
- Modifikace komponent.
- Nastavení materiálu.

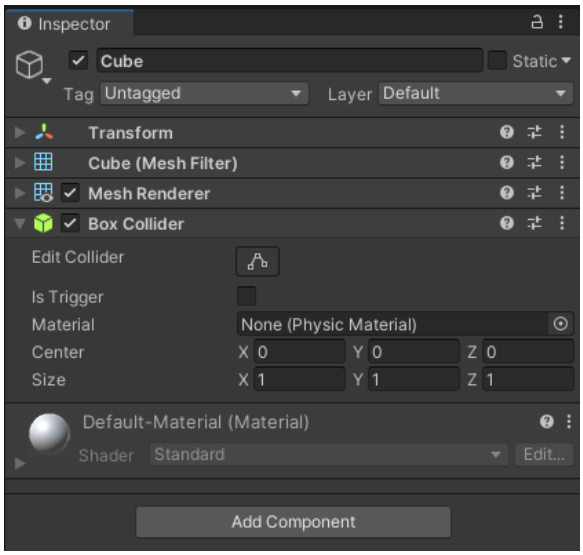
[5] https://store.steampowered.com/app/617830/SUPERHOT_VR/



Obrázek 3.4: Názorná ukázka hierarchie scény v editoru Unity.

3.2.3 GameObject

`GameObject`, neboli herní objekt, je základní třídou pro všechny entity v herní scéně. Sám o sobě však nemá velký význam, chová se totiž pouze jako kontejner pro komponenty, které implementují funkcionalitu daného objektu. Komponenty se mohou k objektu volně přidávat pomocí editoru z menu komponent v inspektoru, jak je zobrazeno na obrázku 3.5. [20]



Obrázek 3.5: Okno inspektora v editoru Unity. Je zde vidět možnost modifikace objektu „Cube“ se čtyřmi komponentami a nastaveným výchozím materiálem.

3.2.4 Component

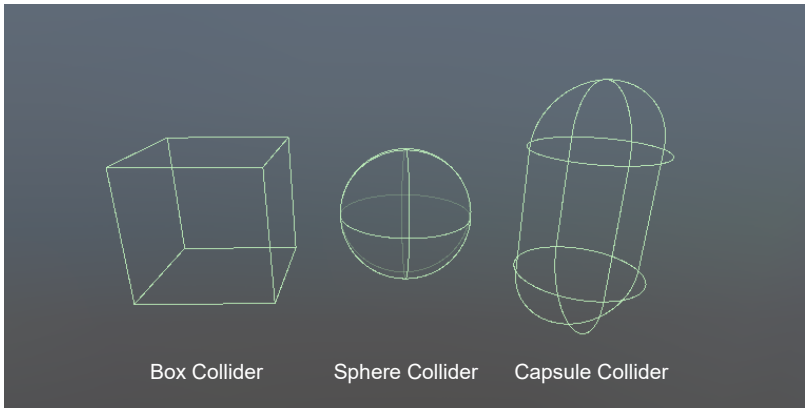
`Component`, neboli komponenta, je základní třídou pro vše, co je potřeba přiřadit k herním objektům. Komponenty slouží k definici chování a vzhledu objektů. Bez nich by herní objekty zkrátka neměly žádný význam ani funkcionalitu. [20]

V Unity je jedna komponenta, kterou má povinně každý herní objekt, a tou je komponenta `Transform`. Ta reprezentuje pozici a rotaci objektu v herní scéně, a proto ji ani nelze odstranit. Bez ní by objekty v herní scéně nemohly vůbec existovat. [20]

Všechny důležité komponenty, které byly při této práci využity jsou popsány níže.

3.2.5 Collider

`Collidery` lze označit za kolizní obálky, které objektu umožňují kolidovat s ostatními objekty. Přidávají mu fyzikální tvar, který slouží pro výpočet fyzikálních kolizí. Tento tvar není renderovaný, ale slouží pouze pro fyzikální jádro PhysX. Ty nejjednodušší a nejméně náročné pro procesor jsou `Box Collider` (tvar kvádru), `Sphere Collider` (tvar koule) a `Capsule Collider` (tvar kapsle), viz. obrázek 3.6.

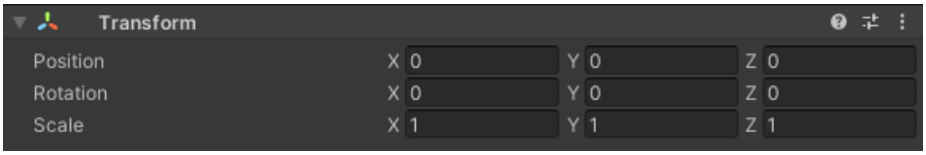


Obrázek 3.6: Porovnání základních Colliderů v herním enginu Unity.

Objekt může obsahovat více těchto kolizních obálek. Pokud jich má více, jedná se o složenou kolizní obálku, díky které lze docílit dostatečně přesné reprezentace tvaru s použitím jednoduchých kolizních obálek a tím ušetřit práci procesoru. [20]

3.2.6 Transform

Komponenta `Transform` slouží k uchování pozice, rotace, škály a rodičovských vztahů objektu. Je to jediná povinná komponenta každého herního objektu a nelze ji z objektu odstranit. V `Inspektoru` lze touto komponentou nastavit pozici, rotaci a škálu objektu, jak je zobrazeno na obrázku 3.7. [20]

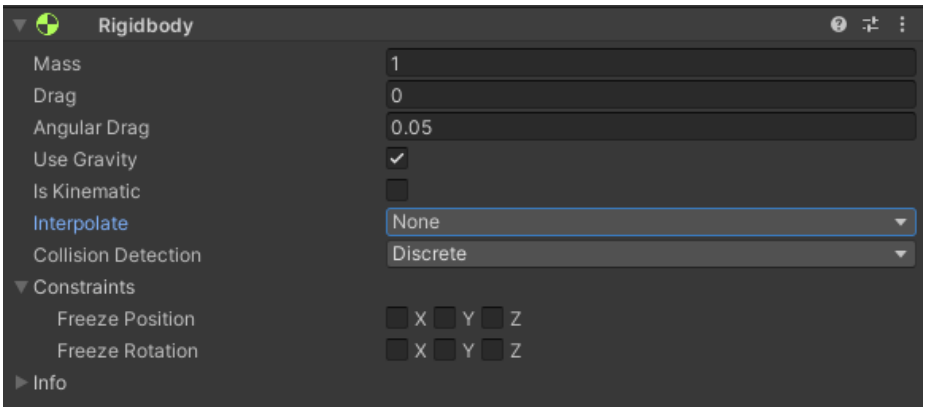


Obrázek 3.7: Nastavení komponenty `Transform` v okně `Inspektoru`.

3.2.7 Rigid Body

Komponenta `Rigid Body` umožňuje fyzikálnímu jádru řídit simulaci fyziky daného objektu. Jeho rozhraní umožňuje na daný objekt působit fyzikálními silami přímo ze skriptů a vytvářet tak pohyby objektu, které působí na pohled realisticky.

V `Inspektoru` pak lze nastavovat veličiny, které budou ovlivňovat fyzikální chování objektu jako např. hmotnost, odpor vzduchu, valivý odpor apod. Všechny veličiny, které lze nastavit, jsou uvedeny na obrázku 3.8. [20]



Obrázek 3.8: Nastavení komponenty `Rigid Body` v okně `Inspektoru`, kde lze nastavovat širokou škálu veličin ovlivňujících fyzikální simulaci.

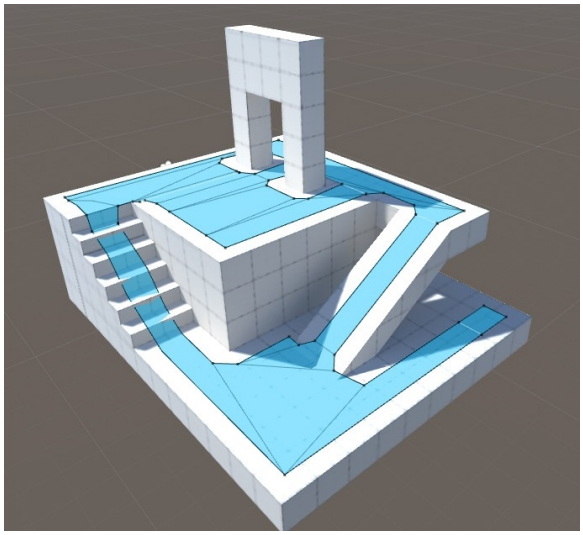
3.2.8 Nav Mesh

`Nav Mesh`, neboli navigační mřížka, je obecně forma reprezentace prostoru, který je pochozí inteligentními agenty. [18] Jednodušeji řečeno, jedná se o jakousi síť, která má možnost určovat, kde se budou moci agenti v prostoru pohybovat.

Unity má vlastní implementaci navigační mřížky v třídě `Nav Mesh`. Díky tomu lze navigační mřížku generovat přímo z geometrie objektů. Tomuto procesu se říká „NavMesh Baking“. Vygenerovaná navigační mřížka pak může vypadat jako na obrázku 3.9. [20]

3.2.9 Nav Mesh Agent

`Nav Mesh Agent` je komponenta, která implementuje navigaci agentů po navigační mřížce. Ve skriptech je možné tuto komponentu využít k nalezení cesty pro agenty a případně navigování agenta až do cíle. Podporuje vyhýbání se ostatním agentům a překážkám na navigační mřížce. V inspektoru lze nastavit různé parametry, které ovlivňují výslednou navigaci agentů. [20]



Obrázek 3.9: Ukázka vygenerované navigační mřížky, kde modrá plocha reprezentuje pochozí povrch, po kterém se mohou inteligentní agenti pohybovat. Obrázek převzatý z internetu[6].

[6] https://learn.unity.com/tutorial/navmesh-baking-1#5ce58ae8edbc2a0ff933255a

Kapitola 4

Výpočetní modely

V této kapitole jsou vysvětleny výpočetní modely a rozhodovací architektury, které byly využity pro vývoj této práce. Nejprve jsou přiblíženy konečné a celulární automaty, které byly ve hře využity pro implementaci chování agentů a šíření ohně. Následně je vysvětlena architektura cílem řízeného plánování akcí, která je v praxi používaná pro simulaci komplexního chování agentů.

Automaty jsou nedílnou součástí vývoje her. Pomocí nich se implementují různé části her, od herních smyček přes složité simulační systémy až po chování umělé inteligence. Pro každý z těchto problémů je však vhodný jiný typ automatů. Ve hrách se obecně využívá široká škála typů automatů, jako jsou například konečné automaty, zásobníkové automaty, adaptivní automaty či celulární automaty. Pro účely této práce je třeba znát pouze dva typy, a to konečné automaty a celulární automaty. Ty jsou popsány v následujících sekcích. [1]

4.1 Konečné automaty

Konečné automaty (anglicky „Finite State Machines“) jsou výpočetní modely, které mohou být implementovány pomocí hardwaru nebo softwaru a být využity k simulování sekvenčních obvodů nebo počítačových programů. Jsou využívány v široké škále oborů, například v matematice, umělé inteligenci, ve hrách, uplatnění najdou také v lingvistice. [3]

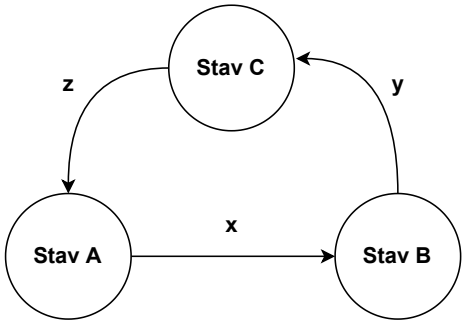
Konečný automat je formálně definován jako uspořádaná pětice $(Q, \Sigma, \delta, q_0, F)$, kde [3]:

- Q je konečná množina stavů
- Σ je neprázdná vstupní abeceda
- δ je konečná množina pravidel typu $q_1 \times a \to q_2$, kde $q_1, q_2 \in Q$ a $a \in \Sigma$
- q_0 je počáteční stav a $q_0 \in Q$
- F je množina koncových stavů a $F \in Q$

Konečné automaty mohou být reprezentovány stavovou tabulkou, viz. ukázka tabulky 4.1. Tato podoba reprezentace se však stává u větších systému značně nepřehlednou, proto je v takovém případě vhodnější reprezentace grafická. Nejčastěji se znázorňuje v podobě diagramu přechodů. Příklad tokového diagramu je na obrázku 4.1. [9]

Stav \ Vstup	x	y	z
Stav A	Stav B	-	-
Stav B	-	Stav C	-
Stav C	-	-	Stav A

Tabulka 4.1: Příklad reprezentace konečného automatu pomocí stavové tabulky.



Obrázek 4.1: Příklad reprezentace konečného automatu pomocí diagramu přechodů. Uzly v těchto diagramech představují jednotlivé stavy konečného automatu. Šipky představují přechody mezi stavy. Nad šipkami jsou pak napsané podmínky přechodu. Ty mohou být v diagramu vynechány v případě, že jsou příliš složité. Následně jsou popsány slovně pod diagramem.

4.1.1 Konečné automaty ve hrách

Ve hrách se pomocí konečných automatů dají implementovat jednoduchá chování takzvaných NPC[1]. NPC jsou postavy ve hře, které nejsou ovládané hráčem. Jejich chování je řízeno určitým systémem. Pokud jejich chování není komplexní, může být implementace tohoto systému pomocí konečného automatu ideální, jelikož režie tohoto systému bývá zpravidla velice malá a samotný systém je jednoduchý na implementaci. Pokud je však chování komplexní, je vhodné konečné automaty zkombinovat s jinou metodikou nebo zvolit metodiku úplně jinou. A to z toho důvodu, že komplexní systémy realizované čistě konečnými automaty bývají zpravidla chaotické a nepřehledné. To může vést k tomu, že jen samotné zjištění, do jakého stavu může který přejít, je velice zdlouhavé. Příklad složitého konečného automatu je na obrázku 4.2. [7]

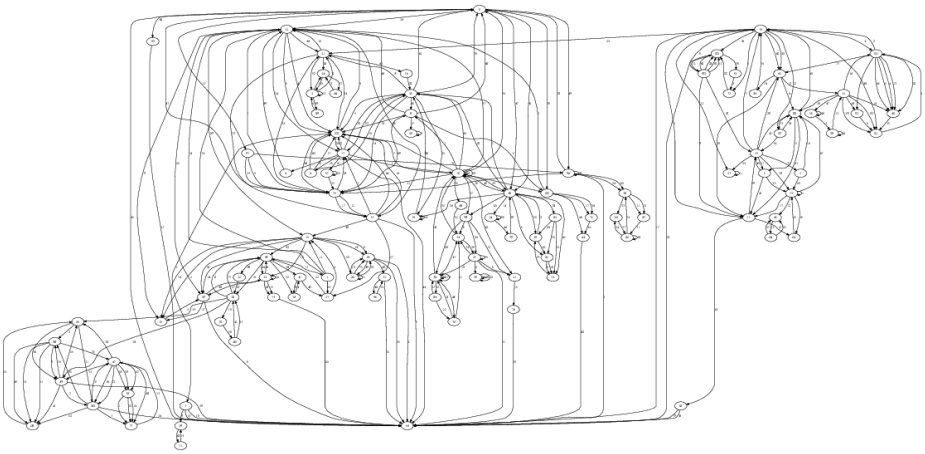
4.2 Celulární automaty

Celulární automat je diskrétní model, který je v dnešní době velice populární při modelování fyzikálních simulací. Principiálně jde o n-dimenzionální mřížku, ve které mají její buňky určité stavy. Tyto stavy jsou pak ovlivněny stavy sousedících buněk. Formálně je celulární automat čtveřice (d^n, S, N, f), kde [4]:

- d^n je n-dimenzionální mřížka buněk
- S je konečná množina stavů, které mohou buňky nabýt

[1] NPC - non-player character (nehratelná postava)

- N je konečná množina sousedících buněk a $N \subseteq d^n$
- f je konečná množina pravidel pro přechod mezi stavy



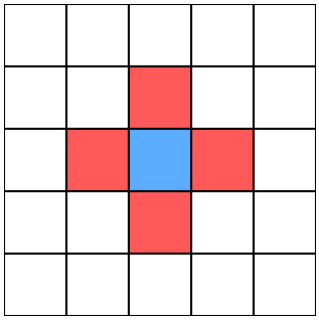
Obrázek 4.2: Příklad složitého konečného automatu. Obrázek převzatý z internetu[2].

Za prvním celulárním automatem stojí John von Neumann, který se ve 40. letech 20. století snažil vytvořit model samoreprodukujících organismů. Tento model byl navržen jako dvou-dimenzionální celulární automat, kde je každá buňka reprezentována malým čtverečkem a dohromady tvoří jednu velkou mřížku. Každá buňka mohla nabývat dvou stavů, černá nebo bílá. To, jaký stav bude buňka mít, je závislé na jejích sousedících buňkách. Za sousedící buňky jsou považovány buňky z okolí podle obrázku 4.3a, kterému se v dnešní době říká Von Neumannovo sousedství. [4]

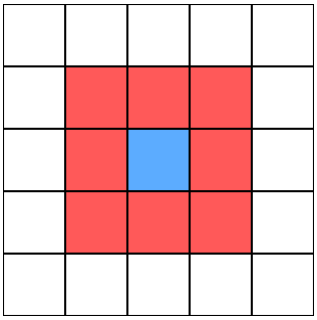
V roce 1970 vymyslel britský matematik John Horton Conway nejznámější celulární automat s názvem Game Of Life. Jedná se opět o dvou-dimenzionální automat, jehož buňky mohou nabývat dvou stavů, živé a mrtvé. Definuje ale sousedící buňky podle Moorova sousedství, které je zobrazeno na obrázku 4.3b. Automat poté funguje na velice jednoduchém principu. Buňka zůstává naživu, pouze pokud má dvě nebo tři sousedící živé buňky. Toto pravidlo má simulovat myšlenku, že pokud je organismus osamocený, tak nemůže přežít a pokud je jeho okolí přeplněné organismy, tak je utlačen. Dalším pravidlem je, že pokud je buňka mrtvá, může se stát živou, právě když má tři sousedící živé buňky. To ve hře znamená, že každý organismus má právě tři rodiče. Těmito pravidly lze dosáhnout vizuálně velice pěkných simulací, které mohou působit jako živé organismy. Příklad obrazců vytvořených tímto systémem lze vidět na obrázku 4.4. Pro vizualizaci jednotlivých kroků automatu je však vhodnější například video Life in life[3]. [8]

[2] https://www.semanticscholar.org/paper/Improving-software-remodularisation-Hall/f0e4063e1c981049404069bd5efa5d6fa3bf74d6/figure/1

[3] Life in life - https://www.youtube.com/watch?v=xP5-iIeKXE8

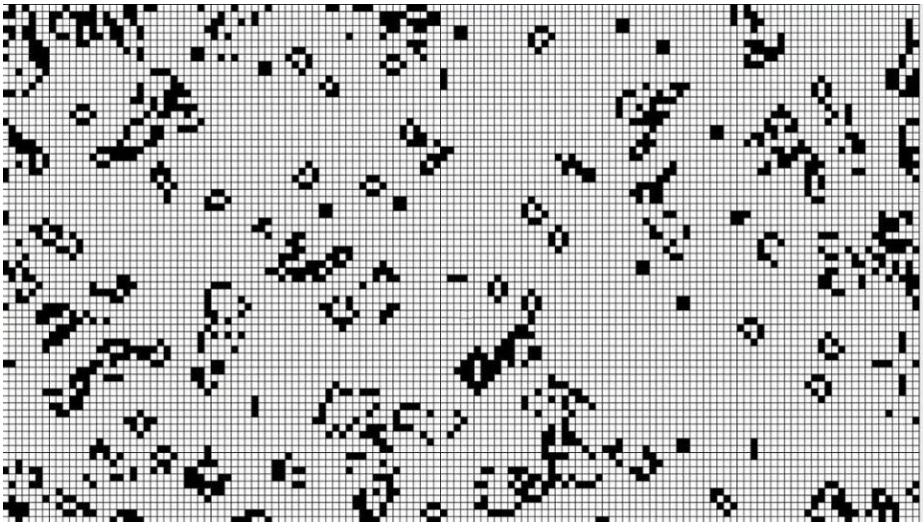


(a) Von Neumannovo sousedství, kde jsou za sousedící buňky považovány čtyři přilehlé buňky znázorněny červenou barvou. Obrázek převzatý z internetu[4].



(b) Moorovo sousedství, kde je za sousedící buňky považováno všech osm přilehlých buněk znázorněných červenou barvou. Obrázek převzatý z internetu[5].

Obrázek 4.3: Porovnání Von Neumannova a Moorova sousedství v celulárních automatech.



Obrázek 4.4: Příklad fungujícího systému v Game Of Life. Obrázek převzatý z internetu[6].

4.3 Cílem řízené plánování akcí

Ve videohrách se v průběhu času vyžadovalo čím dál složitější a věrohodnější chování autonomních agentů, a proto lidé hledali nová řešení, jak simulovat chování bytostí ve hrách. Jedním z řešení je právě cílem řízené plánování akcí, zkráceně GOAP z anglického „Goal-Oriented Action Planning“, které bylo poprvé uvedeno v roce 2003. Využívá se převážně u systémů s menším počtem agentů kvůli jeho výpočetní náročnosti. [17]

GOAP je tedy systémem umělé inteligence pro řízení autonomních agentů, který umožňuje dynamicky plánovat sekvenci akcí vedoucí ke splnění nějakého požadovaného cíle agenta. Jelikož plánování sekvence akcí je závislé nejen na aktuální stavu agenta, ale také na

[4] https://commons.wikimedia.org/wiki/File:Von_neumann_neighborhood.svg

[5] https://commons.wikimedia.org/wiki/File:Moore_neighborhood.svg

[6] https://medium.com/starts-with-a-bang/it-from-bit-is-the-universe-a-cellular-automaton-4a5b1426ba6d

stavu světa kolem něj, může udělení stejného cíle dvěma agentům vyvolat rozdílné sekvence jejich akcí. [2]

Každý z agentů má množinu cílů a množinu akcí, které tyto cíle splňují. Obě tyto množiny jsou dynamické, lze je měnit v reálném čase a všechny akce jsou na sobě naprosto nezávislé. Přináší to značné výhody. Systém nemusí být navržen jako jeden komplexní celek, jak je tomu například u konečných automatů, kde je pevně dáno, z jakého stavu se přejde na jaký, a kde musí být systém velice dobře navržen, jelikož v průběhu vývoje je těžké jej upravovat. [2]

Agent je řízený plánovačem, který mu určuje jakou akci má v danou chvíli provádět. Přesné role cílů, akcí a plánovače jsou popsány v navazujících podsekcích.

4.3.1 Cíl

Cíl je stavem, který se agent snaží splnit. Každý z agentů může mít v danou chvíli přidělený libovolný počet cílů. Každý z cílů má nastavenou úroveň priority, ale aktivován je pouze cíl s nejvyšší prioritou. Podle tohoto aktivovaného cíle je pak řízeno aktuální chování agenta. Každý cíl obsahuje metody pro výpočet jeho priority a pro zjištění, zda byl splněn. [16]

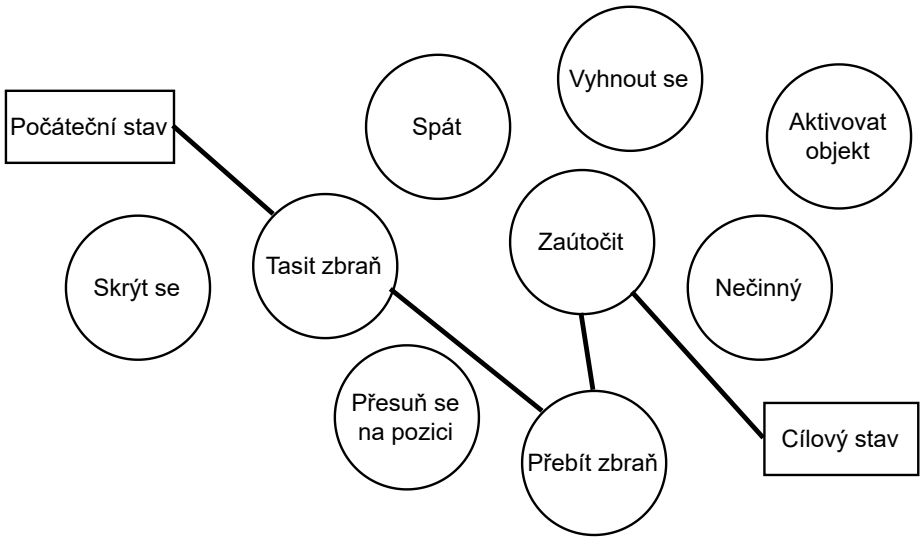
4.3.2 Akce

Akce je implementací jednoduchého chování, jako například „přesunout se na místo“. Důležité je, že akce jsou od sebe navzájem naprosto izolovány, takže nemají žádné závislosti. Samy o sobě akce neimplementují žádné komplexní chování a jsou skládány do sekvence akcí, které se říká plán. Každá z akcí má podmínky pro spuštění a účinek po splnění akce. Tyto účinky mohou poté splnit podmínky jiné akce, která se díky tomu aktivuje. Například podmínka pro spuštění akce „donést objekt na místo“ může být „drží objekt v ruce“ a její účinek „objekt donesen“. [16]

4.3.3 Plánovač

Plánovač se stará o vytvoření sekvence akcí pro splnění cíle, neboli plánu. Plánovači je zadán cíl, který je třeba splnit a on na základě akcí, které má agent k dispozici, vytvoří sekvenci akcí tak, aby byl z aktuálního stavu splněn daný cíl. Příklad vytvořeného plánu je ilustrován na obrázku 4.5.

Plánovač musí vytvořit všechny možné sekvence akcí, které povedou ke splnění zadaného cíle, a následně z nich vybrat tu nejvhodnější. To má za následek, že vytváření plánu je bohužel výpočetně poměrně náročné, a proto je využití tohoto systému častější u her s menším počtem autonomních agentů, u kterých je ale požadováno komplexní chování. [16]



Obrázek 4.5: Příklad vytvořené sekvence akcí GOAP plánovačem pro splnění zadaného cíle. Obrázek převzatý z [16].

Kapitola 5

Návrh

Děj hry se odehrává v jedné vesnici, která je na počátku svého vývoje, a v jejím okolí. Hráč se ve hře vžije do role boha. Jeho cílem je pomáhat této vesnici, aby se rozvíjela a prosperovala. Hráč má různé možnosti, jak toho docílit. Buď může přímo vlastnoručně zajišťovat suroviny nutné pro přežití a rozšíření vesnice, což ale nebude sám stíhat při větším počtu vesničanů. Nebo může určovat práci vesničanům a ti mu budou následně suroviny obstarávat sami.

Vesničané jsou řízeni autonomně. To znamená, že nejsou ovládáni hráčem, ale mají svůj řídící systém, který rozhoduje o tom, co budou v danou chvíli dělat. Tento systém je navržen v sekci 5.1.

Součástí vesnice jsou také budovy, které má hráč možnost stavět výměnou za vytěžené suroviny. Tyto budovy pak přímo ovlivňují chování vesničanů, kteří s nimi mohou interagovat. Návrh budov je v sekci 5.3.

Ve hře jsou k dispozici tři typy surovin: dřevo, kámen a jídlo. Dřevo a kámen jsou suroviny potřebné pro stavění budov. Jídlo slouží vesničanům k přežití. Mohou totiž dostat hlad a pokud budou hladoví příliš dlouho, zemřou. Suroviny se dají dále rozdělit na dva typy: obnovitelné a neobnovitelné. Pokud je úplně vytěžena obnovitelná surovina, samovolně se po určitém čase doplní, zatímco neobnovitelná surovina je vytěžena na dobro. Obnovitelné jsou dřevo a jídlo a neobnovitelný je kámen.

5.1 Návrh autonomních agentů

Vesničané jsou v této hře plně autonomní, proto je lze označit za autonomní agenty. Každý z agentů má svůj vlastní rozhodovací systém, který řídí jeho aktuální činnost. Tento systém je rozdělen do dvou vrstev, kde jedna se stará o rozhodování o aktuální činnosti a druhá o vykonávání vybrané činnosti.

První vrstva je rozhodovací a je založena na plánovací architektuře GOAP. Díky této architektuře se dají vytvářet poměrně komplexní systémy, aniž by byl nutný složitý návrh celistvého systému. Je třeba navrhovat pouze konkrétní části chování a ty se dají poté agentům dynamicky přidávat.

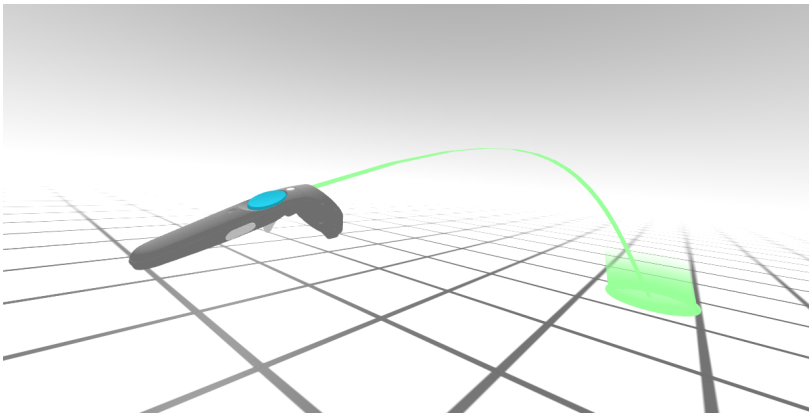
Druhá vrstva je implementací akcí z GOAP architektury a je implementována pomocí FSM. Správa FSM je výpočetně nenáročná. Celá tato vrstva je zaštítěná vrstvou GOAP, není tedy třeba vymýšlet komplexní automaty, které by musely brát v potaz přechod mezi velkým počtem stavů.

5.2 Interakce ve hře

Veškeré interakce, které může hráč ve hře provádět, jsou realizovány pomocí rozhraní ovladačů Quest 2. Ve hře jsou ovladače reprezentovány viditelnými modely rukou, kterými může hráč interagovat s herními objekty.

5.2.1 Ovládání pohybu

Ve hře má hráč dva způsoby pohybu. První způsob je plynulý pohyb, při kterém se hráč posouvá po herním terénu. Druhým způsobem je teleportace na místo určené pomocí ukazovátka vycházejícího z pozice ovladače. Příklad, jak může teleportační ukazovátko vypadat, je na obrázku 5.1.



Obrázek 5.1: Příklad ukazovátka pro pohyb pomocí teleportu. Obrázek převzatý z internetu[1].

5.2.2 Interakce s agenty

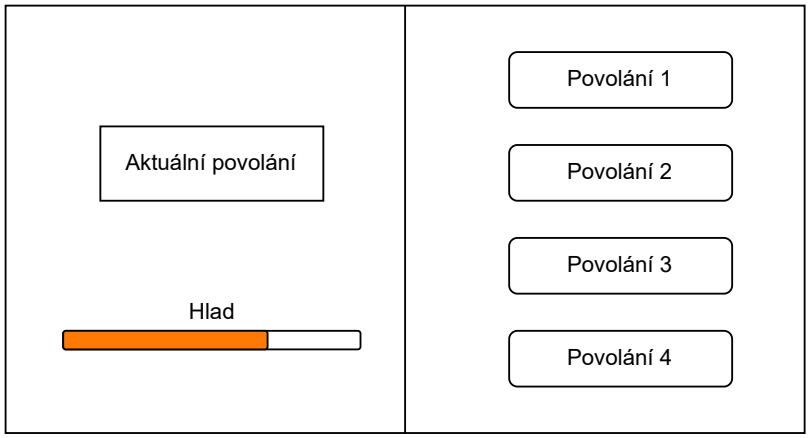
Hráč může uchopit jakéhokoli agenta do rukou. Po uchopení si může zobrazit agentovo GUI. V něm jsou vypsány informace o daném agentovi a také tlačítka pro změnu jeho povolání. Návrh tohoto rozhraní je na obrázku 5.2.

5.2.3 Boží schopnosti

Hráč může využívat božích schopností, které mají formu kouzel. Ve hře se neustále doplňuje hodnota many a kouzla je pak možné sesílat výměnou za tuto manu. Kouzlo má formu objektu, který může hráč uchopit do ruky a následně hodit do vzduchu. Při dopadu na zem bude mít kouzlo účinek na objekty v určité vzdálenosti.

Kromě sesílání kouzel může hráč také těžit vlastnoručně suroviny. Aby se nespoléhal pouze na své schopnosti, bude pro těžbu opět potřebovat manu.

[1] https://aframe.io/blog/teleport-component/



Obrázek 5.2: Návrh grafického rozhraní pro výběr povolání dospělého obyvatele

5.3 Návrh budov

Hráč má možnost stavět budovy výměnou za suroviny. Každá z těchto budov pak má nějaký přímý dopad na chování agentů. Ve hře jsou čtyři druhy budov s použitými modely na obrázku 5.3. Jednotlivé budovy jsou popsány v navazujících podsekcích.

Obrázek 5.3: Navrhované modely pro budovy ve hře. Modely byly použity z volně přístupného balíčku RPG Poly Pack[2].

5.3.1 Skladiště

Skladiště agentům slouží jako místo, kde mohou ukládat natěžené suroviny. Agenti vždy surovinu donesou do nejbližšího skladiště, takže jich může být ve hře libovolný počet. Pokud nebude postavené žádné skladiště, budou agenti nosit suroviny přímo hráči.

5.3.2 Jídelna

Do jídelen mohou agenti nosit nasbírané jídlo. Ve chvíli kdy dostanou hlad, budou chodit do jídelen a nají se zde. Opět, agenti využívají nejbližší jídelnu, takže jich může být postaven

[2] https://assetstore.unity.com/packages/3d/environments/landscapes/rpg-poly-pack-lite-148410

libovolný počet. Pokud nebude postavena žádná jídelna a agent dostane hlad, najde si ve světě nejbližší zdroj jídla a nají se z něj.

5.3.3 Stan

Stan agenti využívají v noci ke spánku. Každý stan má omezenou kapacitu agentů, takže agenti budou využívat pouze ty, které mají volné místo.

5.3.4 Táborák

Táborák agentům slouží jako náhradní místo ke spánku. Pokud nenajdou volné místo v žádném ze stanů, jdou spát k nejbližšímu táboráku.

5.4 Návrh systému šíření ohně

V herním světě může dojít k náhodným událostem, které mají pro hráče vytvářet výzvy. Jednou z těchto událostí je náhodné vzplanutí budov a právě v této situaci bude využito šíření ohně k vytvoření obtížnější výzvy, kde hráč musí zareagovat včas, aby se oheň příliš nerozšířil a neměl tak destruktivní dopad na vesnici.

Systém bude fungovat na principu celulárního automatu, kde buňky tohoto automatu budou mapovány na konkrétní oblast v herním světě. Jelikož terén světa nemusí být prostou plochou, ale může mít libovolný tvar, je nutné, aby systém dokázal buňky mapovat korektně podle tvaru terénu. Aby mohly být ohněm ovlivňovány objekty ve světě, musí být jasné, který objekt se nachází v jaké buňce, a to i včetně dynamicky se pohybujících objektů.

Kapitola 6

Implementace

V této kapitole jsou popsány implementační detaily systémů, které jsou součástí výsledné hry. Nejprve jsou shrnuty obecné implementační informace a následně jsou v navazujících sekcích detailně popsány všechny dílčí systémy.

Hra byla vyvíjena v Unity verze 2020.3.26f1 a veškeré skripty byly psány v jazyce C# s využitím vývojového prostředí Visual Studio Code[1]. Pro projekt byl nastaven vykreslovací řetězec Universal Render Pipeline[2], který je doporučený pro vývoj her na Oculus Quest, jelikož je dobře optimalizovaný pro méně výkonná zařízení.[21]

6.1 Obecné implementační informace

V této sekci jsou uvedeny informace o metrice a struktuře zdrojových kódů a o balíčcích, které byly využity.

6.1.1 Seznam použitých balíčků

Pro usnadnění vývoje hry bylo využito dostupných balíčků pro Unity. V rámci této práce byly použity kromě balíčků dostupných zdarma i některé placené balíčky. Valná většina z nich jsou balíčky s grafickými modely nebo speciálními efekty. Seznam využitých balíčků, které nejsou ve standardní knihovně Unity:

- **Polybrush**[3] byl využit při vytváření herní mapy pro efektivní rozmísťování prefabrikátů (stromy, kameny, apod.).
- **Tool Set - PolyPack**[4] - modely nástrojů.
- **RPG Poly Pack**[5] - modely budov.
- **Low Poly Modular Terrain Pack**[6] - vytvoření terénu mapy.

[1] https://code.visualstudio.com/
[2] https://unity.com/srp/universal-render-pipeline
[3] https://docs.unity3d.com/Packages/com.unity.polybrush@1.0/manual/index.html
[4] https://assetstore.unity.com/packages/3d/props/tools/tool-set-polypack-207678
[5] https://assetstore.unity.com/packages/3d/environments/landscapes/rpg-poly-pack-lite-148410
[6] https://assetstore.unity.com/packages/3d/environments/low-poly-modular-terrain-pack-91558

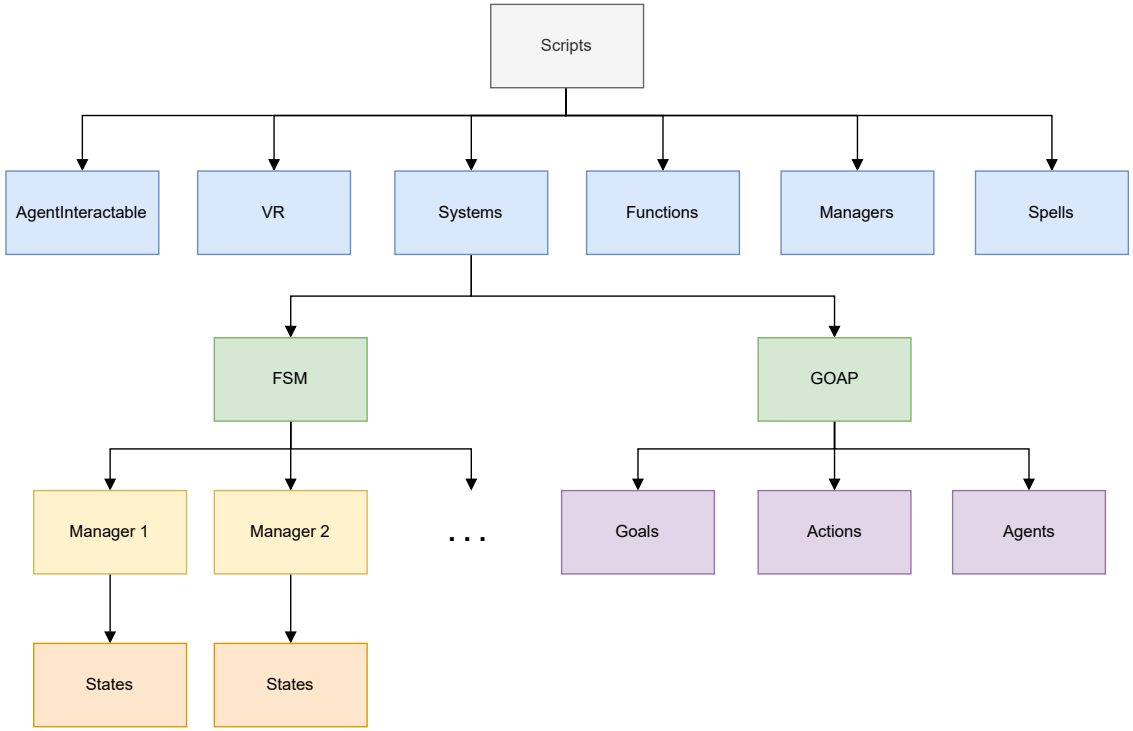
- **Low Poly Rocks Pack**[7] - doplňkové modely.
- **Sci-Fi Arsenal**[8] - speciální efekty kouzel a ohně.
- **POLYGON Farm**[9] - modely stromů a doplňkové modely.
- **Low-Poly Simple Nature Pack**[10] - další modely stromů a kamenů.

6.1.2 Metriky kódu

Všechny systémy, které byly vytvořeny v rámci této práce jsou definovány v 99 zdrojových souborech psaných v jazyce C# a obsahují přibližně 3500 řádků kódu. Struktura těchto souborů je popsána v následující podsekci 6.1.3.

6.1.3 Struktura zdrojových souborů

Všechny zdrojové soubory vytvořené v rámci této práce se nachází v podadresářích */Assets/FirePropagation* a */Assets/Scripts*. V adresáři *FirePropagation* se nachází všechny soubory spjaté se systémem šíření ohně popsaném v sekci 6.4. Všechny ostatní zdrojové soubory jsou v adresáři *Scripts* se strukturou zobrazenou na diagramu 6.1.



Obrázek 6.1: Diagram hierarchie podadresářů se zdrojovými soubory projektu. Každý uzel diagramu reprezentuje jeden adresář.

[7] https://assetstore.unity.com/packages/3d/environments/low-poly-rocks-pack-70164
[8] https://assetstore.unity.com/packages/vfx/particles/sci-fi-arsenal-60519
[9] https://assetstore.unity.com/packages/3d/environments/industrial/polygon-farm-low-poly-3d-art-by-synty-146192
[10] https://assetstore.unity.com/packages/3d/environments/landscapes/low-poly-simple-nature-pack-162153

Jednotlivé podadresáře z diagramu 6.1 pak obsahují následující:

AgentInteractable Obsahuje implementaci všech objektů, se kterými může agent interagovat. Těmito objekty jsou všechny budovy a suroviny.

VR Obsahuje implementaci objektů spojených čistě s interakcemi ve virtuální realitě.

Functions Obsahuje podpůrné funkce, které nelze zařadit do žádné z ostatních skupin.

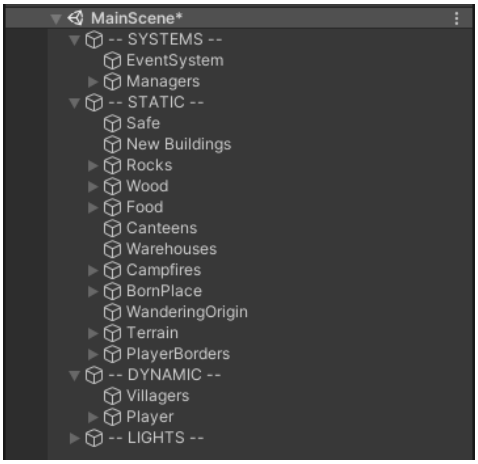
Managers Obsahuje správce herních systémů, které běží na pozadí. Jedná se například o správce cyklu dne a noci, správce hráčových statistik apod.

FSM Obsahuje implementaci základních abstraktních tříd pro stavy konečného automatu a správce stavů. Ty jsou popsány v sekci 6.3.2. Dále také obsahuje adresáře konkrétních stavových správců a jejich stavů.

GOAP Obsahuje implementaci základních abstraktních tříd cílů, akcí a agentů systému GOAP. Dále obsahuje také implementaci GOAP plánovače. V jeho podadresářích jsou pak implementace konkrétních cílů, akcí a agentů.

6.1.4 Hierarchie objektů

Hierarchie objektů herní scény projektu je pro přehlednost rozdělena do čtyř hlavních skupin: statické objekty, dynamické objekty, systémy a osvětlení. Tato hierarchie je vidět na obrázku 6.2 a její části jsou popsány dále.



Obrázek 6.2: Hierarchie objektů v herní scéně této práce.

Skupiny v hierarchii obsahují objekty podobné povahy. Ve skupině osvětlení jsou pouze objekty, které ovlivňují osvětlení herní scény. Ve skupině systémů jsou převážně správci implementovaných systémů, kteří nejsou ve výsledné scéně viditelní, ale musejí mít herní objekty, aby mohli běžet na pozadí. Skupina statických objektů obsahuje objekty, jejichž pozice se v průběhu hry nemění. Obsahuje tedy všechny prvky terénu, budovy a suroviny, viz obrázek 6.2 pod návěštím -- *STATIC* --. Skupina dynamických objektů pak naopak obsahuje objekty, jejichž pozice se v průběhu hry mění. Těmito objekty jsou pouze vesničané a hráč.

6.2 Optimalizace

Při vývoji hry bylo potřeba udělat několik optimalizací, aby hra fungovala plynule na Oculus Quest 2, jelikož výkon tohoto zařízení odpovídá dnešním chytrým telefonům. K nalezení procesů, které hru zpomalovaly, bylo využito profilování běhu hry pomocí vestavěného Unity profilovače[11].

V Unity projektu bylo třeba kompletně vypnout stínování v reálném čase, jelikož s přibývajícím množstvím objektů ve scéně začaly v Oculus Quest 2 klesat snímky za vteřinu.

V rámci systému řízení agentů bylo třeba zjednodušit systém GOAP tak, aby jej bylo možné využít na cílové platformě i s větším počtem agentů. Toho bylo docíleno rozdělením systému na dvě vrstvy, kde první je založena na GOAP a druhá na FSM. Detailní popis navrženého systému je v sekci 6.3.

Navržený systém řízení ohně pomocí celulárního automatu fungoval s menšími mřížkami dostatečně efektivně, ale při mřížkách s desítkami tisíc buněk systém velice snižoval snímkovou frekvenci hry. Bylo proto nutné zavést optimalizaci tohoto systému. Hlavním problémem bylo, že systém každý snímkem aktualizoval úplně všechny buňky konečného automatu. Proto bylo určeno, že při každém snímku bude aktualizováno pouze x buněk, kde x je volitelně zvolená hodnota v inspektoru. Detailní popis této optimalizace lze nalézt v sekci 6.4. Dalším problémem tohoto systému je, že při velkém rozšíření ohně je ve scéně pro každou hořící buňku vytvořena instance hořícího částicového efektu (z anglického „particle effect“). Toto bohužel při velkých požárech dramaticky snižuje snímkovou frekvenci. Tento problém by bylo možné řešit omezením počtu instancí daného efektu ve scéně a následným zvětšováním již existujících efektů. Tato optimalizace dosud nebyla v rámci práce implementována a je jednou z možností rozšíření práce.

6.3 Systém řízení agentů

V této sekci je nejdříve popsáno, jak systém řízení agentů funguje, a následně jsou přiblíženy jednotlivé části chování a jejich implementace.

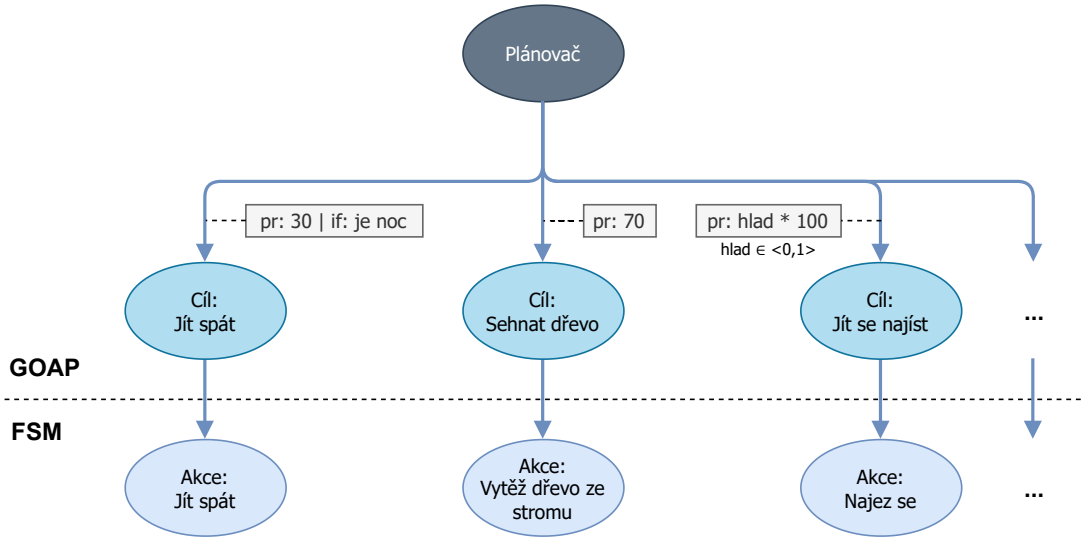
Před implementací výsledného systému, který je popsaný v této sekci byl vytvořen prototyp systému řízení agentů pouze pomocí GOAP. V tomto prototypu se však projevila jeho podstatná nevýhoda, kterou je výpočetní náročnost. Konkrétně byl systém zpomalován nalézáním nejvhodnější sekvence akcí pro splnění zadaného cíle. Z toho důvodu byl navržen nový systém, který eliminuje tento problém.

Systém je rozdělen na dvě vrstvy. První vrstva je založena na GOAP a má na starosti rozhodování o cíli, který bude v danou chvíli agent mít. Jednotlivé cíle mohou být splněny akcemi, jejichž implementace reprezentuje druhou vrstvu systému, která může, ale nemusí být implementována pomocí konečného automatu, zkráceně FSM z anglického „Finite State Machine“. Jestli je akce implementována pomocí FSM, závisí čistě na složitosti dané akce. Pokud je akce atomická, např. přehrání animace a vynulování stavové proměnné, pak je zbytečné pro takovou akci implementovat konečný automat. Proto je zde možnost volby, zdali FSM využít nebo ne. Schéma tohoto systému je vidět na obrázku 6.3.

6.3.1 Systém cílem řízeného plánování akcí

Čistá implementace systému cílem řízeného plánování akcí, neboli GOAP, má jednu velkou nevýhodu, kterou je výpočetní náročnost vytváření sekvence akcí pro nalezení nejvhodněj-

[11] https://docs.unity3d.com/Manual/Profiler.html



Obrázek 6.3: Schéma řídícího systému autonomních agentů. Přerušovanou čárou je naznačeno rozdělení na GOAP a FSM vrstvy systému. U hran vedoucích od plánovače k cílům jsou vidět priority cílů označené „pr:“ a podmínka jejich aktivace označená „if:“.

šího plánu. Jelikož je pro účely této práce výpočetní nenáročnost kritická, bylo nutné systém upravit. V rámci systému navrženého v této práci nepovede ke splnění cíle sekvence akcí, ale vždy pouze jenom jedna akce. Stále je možné cíl splnit různými akcemi, ze kterých je vybrána akce nejlevnější, ale není již třeba složitě nalézat nejvhodnější sekvenci akcí. Aby tato úprava mohla být provedena, bylo třeba vyřešit jak implementovat komplexní akce. To je vyřešeno zavedením konečných automatů. V této sekci je dále popsána implementace GOAP systému a v navazující sekci 6.3.2 pak implementace konečných automatů.

Všechny cíle i akce jsou implementovány pomocí komponent, které jsou přiřazeny ke konkrétním herním objektům. Plánovač následně bere v potaz pouze akce a cíle, které má agent přidělené ke svému hernímu objektu. Plánovač je implementován jako komponenta a v každém snímku je vyhodnoceno rozhodnutí o aktuálním nejdůležitějším cíli a akci, která by měla být provedena pro dosažení cíle. Po dokončení rozhodování je předáno řízení akci, která byla vybrána a její implementace může být buď konečným automatem nebo obyčejným skriptem. Všechny části systému jsou detailně popsány v následujících podsekcích.

GPlanner

`GPlanner` implementuje GOAP plánovač a je hlavní řídící komponentou, která má na starosti jednak vyhodnocení priorit jednotlivých cílů a také cenu akcí, které jsou tyto cíle schopné splnit. Na základě této priority a ceny vybere cíl s nejvyšší prioritou a nejlevnější akci, která může tento cíl splnit. Poté zaktivuje vybraný cíl a akci a předá jim řízení.

Plánovač funguje následovně. Každý snímek iteruje přes všechny cíle, které objekt obsahuje. Z těchto cílů vybere takový, který splňuje následující podmínky:

- Může být aktivovaný (získáno metodou cíle `CanRun()`).
- Má nejvyšší prioritu (získáno metodou cíle. `CalculatePriority()`).
- Objekt má připnutou alespoň jednu akci, která může tohoto cíle dosáhnout. Zároveň vybere akci s nejnižší cenou.

Akce i cíle mají metody pro aktivaci `OnActivated()` a deaktivaci `OnDeactivated()`, ty jsou využity v dalších krocích, které mohou být následující:

(a) Pokud není aktivní žádný cíl, je aktivován nově vybraný cíl a akce.

(b) Pokud jsou aktivní cíl a akce stejné jako nově vybraný cíl a akce, nic nemění.

(c) Pokud je aktivní cíl stejný jako nově vybraný, ale akce jsou rozdílné, deaktivuje aktuální akci a aktivuje nově vybranou akci.

(d) Pokud je aktivní cíl rozdílný od nově vybraného, deaktivuje aktivní cíl i akci a aktivuje nově vybraný cíl a akci.

GGoalBase

`GGoalBase` je abstraktní třídou[12], která slouží třídám konkrétních cílů jako základní třída, ze které jsou odvozeny. Implementuje GOAP cíl, který má nějakou prioritu.

Implementuje základní virtuální metody[13]:

`float CalculatePriority()` Tato metoda vypočítá a vrátí ve `float` hodnotě prioritu cíle, přičemž cíle nižší priority jsou v rozmezí 0–30, střední priority 30–70 a vysoké priority 70–100.

`bool CanRun()` Pokud může být cíl aktivovaný, vrací metoda hodnotu `true`, a pokud nemůže být aktivovaný, tak vrací `false`.

`void OnTickGoal()` Tato metoda slouží pro aktualizaci vnitřního stavu instance cíle. Měla by být zavolána vždy před kontrolou priority a možnosti aktivace cíle.

`void OnActivated(GAction _likedAction)` Je zavolána vždy při zaktivování cíle. Parametr `_linkedAction` obsahuje akci, která byla aktivována.

`void OnDeactivated()` Je zavolána vždy při deaktivaci cíle.

Očekává se, že tyto metody budou v konkrétních třídách cílů přepsány pro požadované chování.

GActionBase

`GActionBase` je základní virtuální třídou pro konkrétní třídy akcí, které jsou z ní odvozeny. Implementuje GOAP akci, která může splnit některé cíle a má nějakou cenu.

Implementuje základní virtuální metody:

`List<System.Type> GetSupportedGoals()` Vrací seznam cílů, které může daná akce splnit.

`float GetCost()` Tato metoda vypočítá a vrátí ve `float` hodnotě cenu provedení akce.

`void OnTick()` Pokud je akce aktivní, je tato metoda volána každý snímek plánovačem. Měla by v ní být implementace chování (nejčastěji pomocí FSM).

[12] Abstraktní třída - třída kterou nelze instanciovat, ale lze z ní dědit

[13] Virtuální metody - metody, které mohou být přepsány v odvozených třídách pomocí klíčového slova `override`

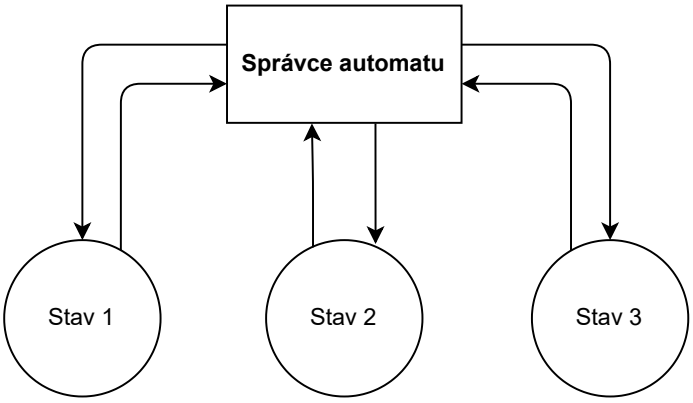
`void OnActivated(GAction _likedGoal)` Je zavolána vždy při zaktivování akce. Parametr `_linkedGoal` obsahuje cíl, který byl aktivován.

`void OnDeactivated()` Je zavolána vždy při deaktivaci akce.

Opět je očekávána úprava těchto metod v odvozených třídách, pro dosažení správného chování.

6.3.2 Systém konečných automatů

Celý systém je řízen správcem automatu. Ten má uložené reference na všechny stavy, které mohou nastat, a udržuje si, který stav je právě aktivní. Stavy jsou implementovány třídou, ve které je definované chování v daném stavu. Pro přechod na jiný stav ale musí zažádat správce. Správce je prostředníkem mezi scénou a jednotlivými stavy, jelikož stavy jsou obyčejnými C# skripty, které nemají přístup k datům ze scény. Na schématu 6.4 je vidět ucelení sytému správcem stavů. Všechny části systému jsou podrobně popsány v následujících podsekcích.



Obrázek 6.4: Schéma systému konečného automatu znázorňující ucelení systému díky správci automatu.

StateManagerBase

Je abstraktní třídou, která implementuje správce stavů. Umožňuje přepínaní aktivního stavu a je prostředníkem mezi scénou a stavy. Udržuje si přehled, který stav je aktivní a mezi jakými stavy může přepínat.

Implementované virtuální metody:

`void OnActivated()` Je zavolána vždy při aktivaci správce. Má na starosti inicializaci správce a nastavení a aktivování počátečního stavu.

`void OnDeactivated()` Je zavolána vždy při deaktivaci správce. Slouží pro uklizení a je v ní také volána metoda `ExitState()` aktivního stavu, pro jeho správné ukončení.

`void OnTick()` Tato metoda je zavolána každý snímek, pokud je správce aktivní. Je v ní volána metoda `UpdateState()` aktivního stavu.

`void SwitchState(StateBase nextState)` Je to metoda kterou využívají stavy, pro přechod do jiných stavů. Ukončí aktuální stav a zaktivuje stav daný v parametru `nextState`.

StateBase

StateBase je abstraktní třídou, která implementuje FSM stav. Jsou z ní odvozeny konkrétní třídy stavů, které rozšiřují její metody. Implementuje 3 virtuální metody: `void EnterState()`, `void UpdateState()`, `void ExiteState()`. `Void EnterState()` je metoda která je zavolána vždy při aktivaci stavu a slouží pro inicializaci stavu. Pokud je stav aktivován, je každý snímek volána metoda`void UpdateState()`, obsahující implementaci chování. `Void ExiteState()` je zavolána vždy při deaktivaci stavu a slouží pro uklizení.

6.3.3 Autonomní chování agentů

Chování agentů je implementováno kompletně pomocí systémů, které jsou popsány v předchozích podsekcích. Aktuálně je ve hře implementováno šest cílů. Každý z cílů má právě jednu akci, která ho může splnit. Jsou cíle, které mají všichni agenti, ale také jsou cíle, které někteří agenti mít nemusí.

Součástí hry je také systém dne a noci, podle kterého se povolují nebo zakazují některé cíle. Myšlenka cyklu jednoho dne je taková, že agenti v noci spí, ráno a večer mají volný čas a odpoledne konají své povolání. Mohou také dostat hlad, takže pokud mají volný čas, jdou se najíst, i když mají malý hlad. Pokud však pracují nebo spí, jdou se najíst pouze v případě hladu kritického. Tímto se dosáhne toho, že budou agenti preferovat jídlo ráno a večer. Může se však stát, že se ráno nenajedí a budou se muset najíst v pracovní době. Stejně tak, pokud se nenajedí večer, musejí vstát a najíst se v noci.

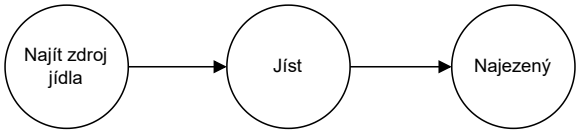
Cíle lze tedy rozdělit na tři skupiny: cíle potřeb, cíle denního cyklu a cíle povolání. Cíle potřeb a denního cyklu mají neustále všichni agenti, zatímco z cílů povolání má každý z agentů pouze jeden cíl. V cílech potřeb je pouze cíl jít se najíst `GoalEat`. V cílech denního cyklu je cíl spánku `GoalSleep` a cíl mít volný čas `GoalHaveFreeTime`. Jelikož jsou ve hře aktuálně tři povolání (dřevorubec, sběrač jídla a horník), jsou zde také tři cíle povolání `GoalGetWood` pro dřevorubce, `GoalGetFood` pro sběrače jídla a `GoalGetStone` pro horníky. Jak už bylo řečeno, tak z těchto tří cílů povolání může mít agent v danou chvíli pouze jeden.

Aby bylo dosaženo zamýšleného chování, bylo třeba dobře nastavit priority jednotlivých cílů. Jak byly priority nastaveny je popsáno spolu s implementací jednotlivých cílů a jejich akcí v následujících oddílech.

GoalEat

Cíl `GoalEat` simuluje potřebu agentů jíst. Každý z agentů má hodnotu hladu, která se v průběhu času zvyšuje. Na základě této hodnoty je dynamicky počítána priorita cíle, podle závislosti $hladNorm \times 90$, kde $hladNorm$ je hodnota hladu normalizovaná do intervalu $\langle 0, 1 \rangle$, kde 0 představuje nejmenší hlad a 1 největší hlad. Díky tomu je priorita cíle nízká, pokud má agent malý hlad a vysoká, pokud má velký hlad. Cíl může být aktivován pouze pokud je normalizovaná hodnota hladu větší než $0,5$. Tato podmínka je přidána, aby se nemohlo stát, že žádná jiná akce nemůže být aktivována a agent bude neustále jíst.

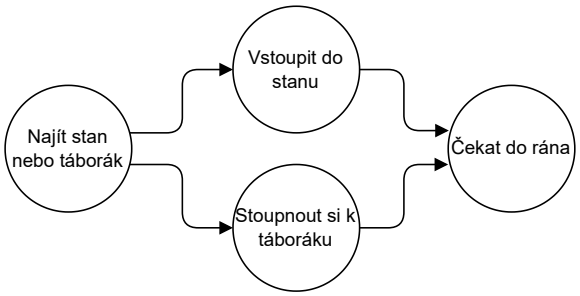
Tento cíl je splnitelný akcí `ActionEat`, která je implementována systémem konečného automatu, jehož schéma je na obrázku 6.5. Při aktivované akci nejprve agent najde zdroj jídla. Pokud je postavena jídelna a je v ní jídlo, zvolí ji jako cíl. Pokud žádnou vhodnou jídelnu nenajde, vyhledá nejbližší zdroj jídla ze zdrojů surovin (bobule nebo ovocný strom). Ve chvíli kdy dojde ke zvolenému cíli, odebere z něj pět jednotek jídla a nají se. Najedení se je implementováno pouze jako uspání procesu agenta na tři sekundy. Následně je agentovi vynulován hlad, čímž priorita cíle `GoalEat` klesne na nulu.



Obrázek 6.5: Diagram konečného automatu akce `ActionEat`.

GoalSleep

Cíl `GoalSleep` simuluje potřebu spánku Je aktivován pouze pokud je ve hře noc a má konstantní prioritu 70. Může být splněn akcí `ActionSleep`, která probíhá následovně. Agent vyhledá nejbližší stan s volným místem (každý stan má kapacitu pro tři agenty). Pokud se mu stan nepodaří najít, najde nejbližší oheň. V obou případech pak čekají na ráno, kdy se cíl `GoalSleep` deaktivuje. Akce je opět implementována konečným automatem, jehož diagram je na obrázku 6.6.



Obrázek 6.6: Diagram konečného automatu akce `ActionSleep`.

GoalFreeTime

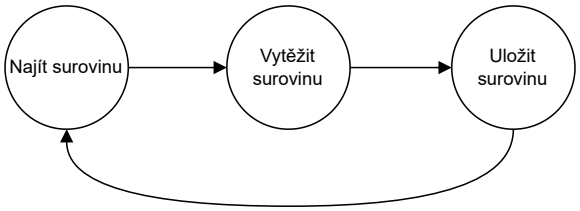
Cíl `GoalFreeTime` může být aktivován v libovolný čas herního dne. Má statickou prioritu 10, takže je aktivován pouze pokud jiný z cílů nemůže být aktivován. Může být splněn akcí `ActionHaveFreeTime`, která jako jediná z akcí není implementována konečným automatem, ale pouze obyčejným skriptem, jelikož je její implementace velice jednoduchá. Agent si pouze zvolí v okruhu 120 jednotek náhodný bod a k němu jde. Když k němu dojde, zvolí si nový náhodný bod a opět k němu jde. Tento proces se opakuje, dokud se agentovi neaktivuje nějaký cíl s vyšší prioritou.

Cíle povolání

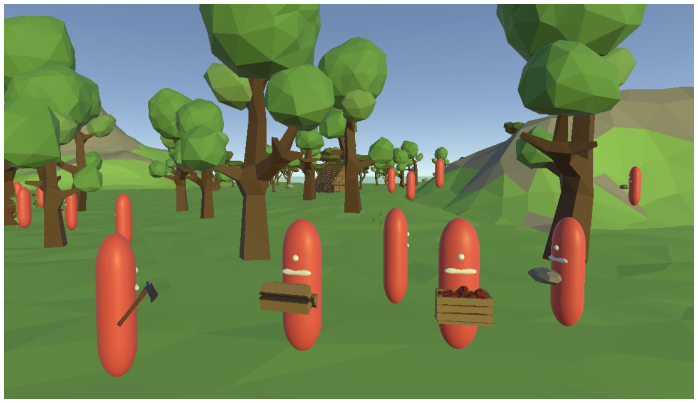
Cíle povolání udělují agentům povinnost sbírat určenou surovinu. Všechny mají statickou prioritu 70 a mohou být aktivovány pouze pokud je v herním čase odpoledne. Mezi tyto cíle patří `GoalGetWood` pro těžbu dřeva, `GoalGetFood` pro sběr jídla a `GoalGetStone` pro těžbu kamene.

Při všech těchto cílech je chování agentů naprosto identické, až na cílové suroviny, které mají agenti při těchto cílech vytěžit. Proto jsou jejich akce implementovány jedním konečným automatem, u kterého se akorát nastaví cílová surovina. Diagram tohoto automatu je na obrázku 6.7. Při aktivované akci agent najde nejbližší surovinu, u které je volné místo. Každá surovina může být totiž v danou chvíli těžena omezeným počtem agentů. Ve chvíli,

kdy agent najde surovinu a dojde k ní, simuluje její těžbu. Simulace probíhá uspáním procesu agenta na určitý počet sekund podle suroviny, kterou právě těží (například kámen se těží déle než probíhá sběr jídla). Po dokončení těžby je ze zdroje suroviny odebráno určité množství suroviny, které je opět závislé na typu suroviny (jídla vytěží agent více než kamene). Potom agent najde nejbližší skladiště a surovinu do něj odnese. Pokud není žádné skladiště postaveno, přinese surovinu přímo k hráči. Během manipulace se surovinou je agentovi přidán model příslušné natěžené suroviny, viz. obrázek 6.8. Následně se celý proces opakuje, dokud je akce aktivní.



Obrázek 6.7: Diagram konečného automatu akce pro získání surovin.



Obrázek 6.8: Snímek vesničanů během plnění akce zadané jejich řídícím systémem. Konkrétně se jedná o akci splňující cíle `GoalGetWood`, `GoalGetFood` a `GoalGetStone`. Vlevo je vidět vesničan ve stavu hledání suroviny a vpravo jsou vesničané ve stavu ukládání suroviny.

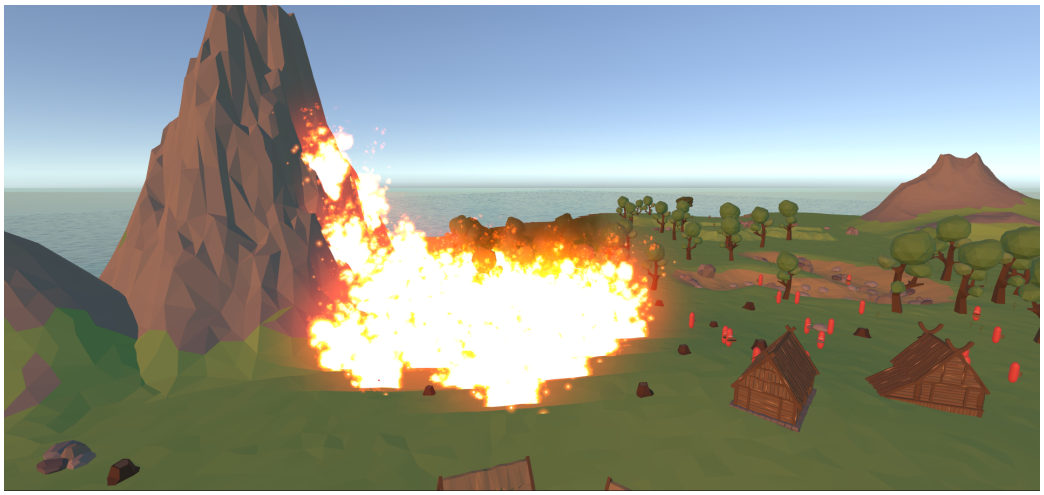
6.4 Systém šíření ohně

Součástí hry je systém šíření ohně, jehož inspirací pro mne byl systém šíření ohně ze hry Far Cry 2, který navrhl a zrealizoval Jean-Francois Levesque. Levesque se se základní architekturou podělil v článku Far Cry: How the Fire Burns and Spreads [10]. Tento systém, který Levesque v článku popisuje, je však pro účely této práce příliš komplexní, proto byl tento systém zjednodušen. Principy funkcionality a architektury tohoto systému jsou popsány v podsekcích 6.4.1 a 6.4.2. V následujících podsekcích je pak detailně popsána funkcionalita jednotlivých částí systému.

Systém byl vyvíjen nezávisle na samotné hře, tak, aby jej bylo možné napojit na libovolný projekt vyvíjený v enginu Unity. Všechny komponenty, které tento systém nabízí, mají konfigurační parametry, kterými lze systém přizpůsobit požadavkům projektu. O jaké

parametry se jedná a jaký mají vliv na chování systému, je popsáno dále u jednotlivých komponent, které tento systém nabízí.

Rozšířený oheň vytvořený tímto systémem je znázorněn na obrázku 6.9.



Obrázek 6.9: Ukázka ohně vytvořeného implementovaným systémem šíření ohně.

6.4.1 Princip

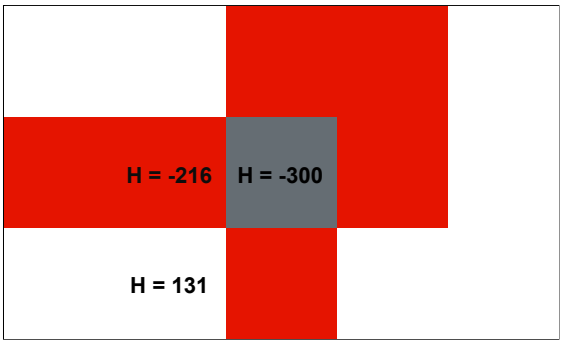
Systém funguje na principu celulárního automatu. Ve scéně je vytvořena mřížka libovolné velikosti, jejíž buňky mají hodnotu zdraví a prahovou hodnotu spálení. Ty jsou klíčové pro simulaci šíření ohně. Automat se poté řídí následujícími pravidly:

1. Pokud buňka hoří, způsobuje svým sousedícím buňkám poškození. Za sousedící buňky jsou považovány buňky podle Von Neumannova sousedství.
2. Pokud hodnota zdraví buňky klesne na hodnotu ≤ 0, buňka se zapálí.
3. Pokud hodnota zdraví buňky klesne na hodnotu $\leq$ prahová hodnota spálení, buňka přestane hořet a je označena za spálenou.

Toto chování je reprezentováno na obrázku 6.10.

Ve scéně mohou být také hořlavé objekty, které mají hodnotu teploty a prahovou hodnotu vznícení. Ty slouží k možnosti zapálení objektu. Hořlavé objekty nejsou řízeny automatem, ale pouze přizpůsobují své chování na základě toho, v jaké buňce se nacházejí. Mají také pravidla, kterými se řídí:

1. Pokud je v hořící buňce, zvyšuje svoji teplotu.
2. Pokud není v hořící buňce, snižuje svoji teplotu.
3. Teplota se pohybuje v intervalu $\langle 0, \text{prahová hodnota vznícení} \rangle$
4. Pokud teplota stoupne na prahovou hodnotu vznícení, začne hořet.
5. Pokud teplota klesne na 0 přestane hořet.
6. Pokud hoří, nebo se nachází v hořící buňce, přijímá poškození.



Obrázek 6.10: Celulární automat systému šíření ohně, kde je prahová hodnota spálení nastavena na hodnotu -300. Hodnota „H“ v buňkách reprezetuje zdraví buňky.

Tato jednoduchá pravidla jsou navržena tak, že objekt začne hořet až po delším setrvání v ohni a přestane samovolně hořet, pokud je dostatečně dlouho mimo oheň. Je možné přidat výjimku pro některé hořlavé objekty, která zabrání samovolnému snižování teploty. To může být vhodné například u budov. U těch se předpokládá, že se samovolně neuhasí.

6.4.2 Architektura

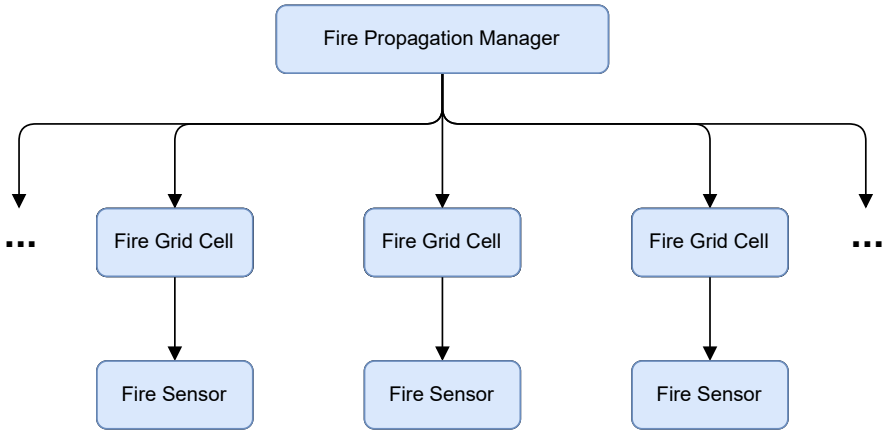
Nejprve je nutné chápat, jakou strukturu má systém v herní scéně. Automat je reprezentován objektem s komponentou `Fire Propagation Manager` a každá buňka je jeho samostatný synovský objekt s komponentami `Fire Grid Cell` a `Fire Sensor`. Každá buňka musí mít také komponentu `Box Collider`, která je využita k detekci objektů vstupujících do buňky.

Systém má hierarchii, která je znázorněna ve schématu 6.11. Na jejím vrcholu je `Fire Propagation Manager`, který je nejdůležitějším prvkem tohoto systému. Stará se totiž o řízení celé simulace šíření ohně, generuje mřížku s buňkami a uceluje celý systém. Jednotlivé buňky, neboli `Grid Cell`, uchovávají stavové proměnné sloužící k simulaci. Jedná se například o proměnné: zda-li buňka hoří, jaké má zdraví nebo zda-li je buňka vyhořelá. Udržují si také reference na hořlavé objekty, které v buňce jsou. O samotnou detekci objektů v buňce se stará `Fire sensor`. Ten pouze pracuje s `Colliderem` objektu, který využívá pro detekci vstupujících a vystupujících objektů z buňky. O těchto událostech pak informuje `Grid Cell`.

6.4.3 Fire Propagation Manager

`Fire Propagation Manager` je nejdůležitější komponentou celého systému. Řídí totiž celou simulaci šíření ohně a má také na starosti generaci mřížky buněk. Krok simulace celulárního automatu provádí každý snímek. To znamená, že musí každý snímek iterovat přes všechny buňky automatu a provádět výpočty pro aktualizaci stavů buněk. To by však bylo u mřížek s větším počtem buněk příliš náročné, proto byla do systému zavedena optimalizace, díky které je možné určit kolik buněk bude za snímek aktualizováno. To by mohlo vést ke zpomalení celé simulace, a proto je třeba některé řídící hodnoty, jako například poškození udělené buňkám, násobit vhodnou konstantou. Ta je vypočítána jako n/m, kde n je počet buněk v mřížce automatu a m je počet buněk aktualizovaných každý snímek.

Má několik parametrů, kterými lze v inspektoru měnit buď průběh simulace, nebo rozměry a tvar mřížky. Těmito parametry jsou:



Obrázek 6.11: Schéma hierarchie systému šíření ohně.

`int GridWidth` Počet buněk na jeden řádek mřížky.

`int GridHeight` Počet buněk na jeden sloupec mřížky.

`float CellSize` Rozměr jedné buňky. Přesněji řečeno, jelikož je buňka považována za kvádr v herní scéně, tak **`Cell size`** určuje rozměry v osách X a Z, neboli šířku a hloubku.

`float CellHeight` Výška jedné buňky.

`bool EnableFollowTerrain` Pokud je **`true`**, tak budou buňky kopírovat výšku terénu. Pokud je **`false`**, tak budou buňky v jedné rovině. Tato možnost je užitečná v případě herních scén, které mají plochý terén, jelikož zásadně urychlí generaci buněk.

`float TerrainMaxHeight` Hodnota maximální výšky terénu, se kterou bude počítáno při generaci mřížky.

`LayerMask TerrainMask` Je maska, kterou má nastavenou terén, jehož výšku má mřížka kopírovat. V případě že je **`EnableFollowTerrain`** nastaven na **`false`**, nemá tento parametr žádný efekt.

`int MaxIterationPerFrame` Určuje maximální počet buněk, které budou aktualizovány za 1 snímek. Tj. pokud je nastaven na 1000, **`Fire Propagation Manager`** iteruje každý snímek přes 1000 buněk.

`GameObject CellPrefab` Prefabrikát objektu buňky, který je instanciován při generaci buněk mřížky.

`float DamagePerCell` Množství poškození, které buňka dostane za každou sousedící hořící buňku včetně sama sebe.

Generace mřížky

Mřížka je vždy vygenerována hned v prvním snímku po načtení scény a po zbytek času zůstává statická. To je naprosto nezbytné, jelikož je generace výpočetně náročná operace, která by v průběhu hry mohla způsobit prodlevu mezi snímky.

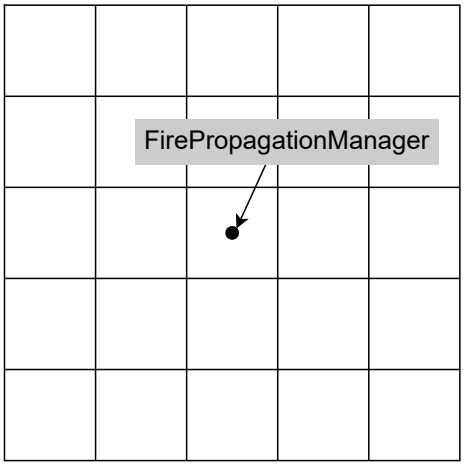
Cílem je vytvořit mřížku tak, aby byl objekt `FirePropagationManager` v jejím středu. To je ilustrováno na obrázku 6.12. Abychom toho docílili, jsou pozice v ose x $PosX$ a pozice v ose z $PosZ$ jednotlivých buněk vypočteny následovně:

$$PosX = cellSize * (gridWidth * 0.5 + x + 0.5) + FPMx \tag{6.1}$$

$$PosZ = cellSize * (gridWidth * 0.5 + y + 0.5) + FPMz \tag{6.2}$$

kde

- hodnota $cellSize$ je určena parametrem `CellSize`
- x je indexem sloupce buňky v mřížce a y je indexem řádku buňky v mřížce
- $FPMx$ a $FPMz$ jsou pozicemi objektu `Fire Propagation Manager` v osách X a Z



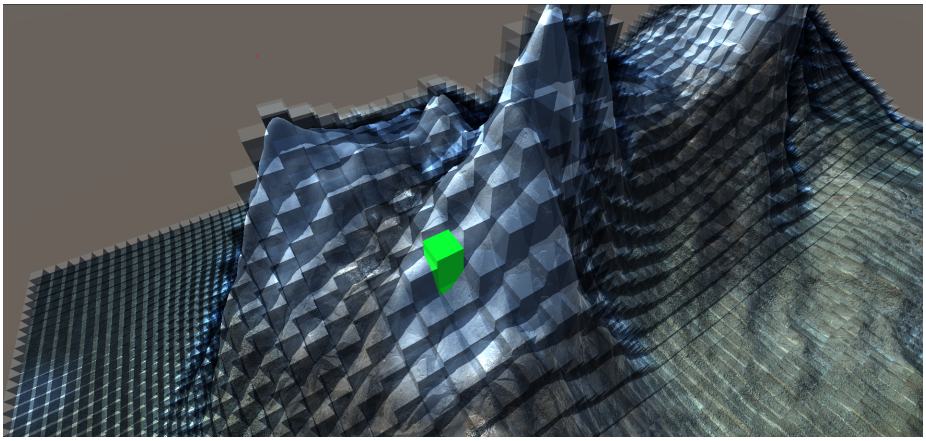
Obrázek 6.12: Pozicování buněk celulárního automatu v závislosti na poloze objektu `Fire Propagation Manager`.

Pozice buňky v ose Y závisí na zvoleném parametru `EnableFollowTerrain`. Pokud byl nastaven na `false`, pozice buňky nezávisí na terénu pod ní, ale pouze se převezme hodnota pozice objektu `Fire Propagation Manager` v ose Z. Pokud je ale nastavena na `true`, tak je pomocí funkce fyzikálního enginu `Raycast(...)`[14] vyslán paprsek z pozice $[PosX, \texttt{TerrainMaxHeight}, PosZ]$ se směrovým vektorem $[0, -1, 0]$ a kolizní maskou nastavenou parametrem `TerrainMask`. Výstupem této funkce je pak přesný bod na terénu, kde jej paprsek zasáhl. Pozice tohoto bodu je výslednou pozicí buňky.

Ve chvíli, kdy je vypočítána pozice buňky, tak je na jejím místě instanciován prefabrikát buňky z parametru `CellPrefab`. Jeho lokální škála v ose X a Z je nastavena na hodnotu parametru `CellSize`/2 a v ose Y na hodnotu parametru `CellHeight`.

Jak může vygenerovaná mřížka vypadat je ilustrováno na obrázku 6.13, kde byla mřížka vygenerována nad hornatým terénem a buňky byly pouze pro účely vizualizace vyrenderovány. Pro lepší orientaci byla jedna konkrétní buňka zvýrazněna zelenou barvou.

[14] Raycast(...) - https://docs.unity3d.com/ScriptReference/Physics.Raycast.html



Obrázek 6.13: Vizualizace vygenerované mřížky s jednou zvýrazněnou buňkou.

6.4.4 Fire Grid Cell

Komponenta `Fire Grid Cell` reprezentuje buňku celulárního automatu. Nemá reference na okolní buňky, pouze si udržuje svůj vlastní stav a zpřístupňuje komponentě `Fire Propagation Manager` své rozhraní. Jejími stavovými proměnnými jsou:

`float Health` Zdraví buňky.

`bool IsOnFire` Zdali buňka hoří.

`bool IsBurned` Zdali je buňka vyhořelá.

`IsOnFire` a `IsBurned` jsou veřejné a `Fire Propagation Manager` je využívá k řízení simulace. `Health` je soukromou proměnnou, podle jejíž hodnoty jsou nastavovány ostatní veřejné proměnné.

V rozhraní `Fire Grid Cell` jsou tři nejpodstatnější metody, které jsou využívány jinými komponentami. Jedou z nich je `void TakeDamage(float damage)`, kterou `Fire Propagation Manager` může buňce udělit poškození. Buňka při udělení poškození snižuje svůj život a v případě, kdy klesne pod jednu z prahových hodnot (viz. 6.4.1), upraví vhodně svůj stav. Další 2 metody jsou `void AddFireableObject(FireableObject fireableObject)` a `void RemoveFireableObject(FireableObject fireableObject)`. Ty využívá komponenta `Fire Sensor` pro informování buňky o vstupu hořlavého objektu do ní. Buňka si udržuje reference na všechny hořlavé objekty, aby je mohla informovat o svém stavu. Ty potom na tento stav samy reagují. To, jak `Fire Sensor` objekty detekuje, je popsáno v následující sekci 6.4.5.

6.4.5 Fire Sensor

`Fire Sensor` je komponentou, která se stará o detekci hořlavých objektů v buňce. Pro detekci využívá `Collideru`, který má na sobě přidaný prefabrikát buňky. `Collider` má nastavené chování spouště. To znamená, že neregistruje kolize s komponentami `Rigid Body` a neovlivňuje tak fyzikální simulaci, ale pouze reaguje na jejich vstup a výstup. Spouště zasílají zprávy `OnTriggerEnter` při vstupu objektu do spouště a `OnTriggerExit` při východu objektu ze spouště. Na ty právě reaguje `Fire Sensor`. Zprávy obsahují i referenci na `Collider` který spoušť spustil a díky tomu `Fire Sensor` může zkontrolovat, zdali je objekt

hořlavý nebo ne. To je dáno tím, zdali objekt vlastní komponentu `Fireable Object` nebo komponenty, které z ní dědí. Pokud je hořlavý objekt detekován, je jeho vstup, případně výstup hlášen příslušné buňce, jak je popsáno v předchozí kapitole 6.4.4. Ukázkový kód pro reakci na zprávu `OnTriggerEnter` a informování příslušné buňky o vstupu objektu je na výpisu 6.1.

```
private void OnTriggerEnter(Collider other) {
    if(other.gameObject.TryGetComponent<FireableObject>(out var
        fireableObject))
    {
        OnObjectEntered?.Invoke(fireableObject);
    }
}
```

Výpis 6.1: Kód pro registraci vstupu hořlavého objektu do buňky

6.4.6 Fireable Object

`Fireable Object` je abstraktní třídou, která implementuje chování hořlavých objektů v herní scéně, které by měly reagovat na systém šíření ohně. Abstraktní je z toho důvodu, že různé hořlavé objekty mohou mít rozdílnou implementaci chování při hoření. Například vesničanům se uděluje poškození na zdraví, ale hořícím stromům se postupně snižuje množství dřeva, které je z nich možné vytěžit. Proto implementaci třídy `Fireable Object` rozšiřují 3 odvozené třídy: `Fireable Object Person`, `Fireable Object Resource` a `Fireable Object Building`.

Jeho funkcionalita je závislá na informacích poskytovaných buňkami systému. Ty `Fireable Object` informují v případě, že se buňce změnil stav (např. při vzplanutí buňky). `Fireable Object` pak na tyto informace reaguje podle pravidel popsaných v sekci o principu systému 6.4.1.

Podmínkou pro správné fungování tohoto systému je, že má hořlavý objekt připnutou komponentu `Collider`. Bez `Collideru` by totiž buňky nemohly detekovat vstup objektu do nich a tím pádem ani informovat `Fireable Object` o jejich stavu.

6.5 Interakce ve virtuální realitě

Pro vývoj prvků virtuální reality byly ve hře využity tři vývojové balíčky od Unity. Dva balíčky umožňující spuštění Unity her v široké nabídce virtuálních brýlí: Oculus XR Plugin[15] a OpenXR Plugin[16]. A balíček XR Interaction Toolkit, který obsahuje framework umožňující interakce virtuálních brýlích s herním světem.

Všechny interakce jsou prováděny pomocí komponenty `XR Ray Interactor` z unity balíčku XR Interaction Toolkit. Tato komponenta umožňuje interagovat s objekty a grafickými rozhraními pomocí paprsku vedoucího z rukou hráče, viz. obrázek 6.14.

[15] https://docs.unity3d.com/Manual/com.unity.xr.oculus.html
[16] https://docs.unity3d.com/Manual/com.unity.xr.openxr.html

Obrázek 6.14: Ukázka paprsků vedoucího z rukou hráče, kterými jsou prováděny veškeré interakce ve hře.

6.5.1 Interakce s vesničany

Hráč má možnost uchopit vesničana do ruky namířením interaktivního paprsku a následného stisknutí tlačítka pro uchopení. Po stisknutí tlačítka je vesničan přesunut do ruky hráče a zůstává tam do uvolnění tlačítka.

Při uchopení vesničana jsou deaktivovány komponenty `Nav Mesh Agent` a `GPlanner`. `Nav Mesh Agent` musí být deaktivovaný, aby se nesnažil ovládat pozici vesničana. `GPlanner` musí být deaktivovaný, jelikož není žádoucí, aby se snažil provádět nějaké akce. Po uvolnění vesničana je jeho objekt přepnut do stavu takzvané hadrové panenky (anglicky „ragdoll“). To znamená, že jeho řídící systémy jsou pořád pozastavené, objekt je pouze řízen fyziálním enginem a padá k zemi. Tento stav přetrvává, dokud se objekt pohybuje směrem alespoň jedné osy rychlostí 0,05 jednotek za sekundu. Poté jsou opět komponenty `Nav Mesh Agent` a `GPlanner` aktivovány a vesničan je opět řízen autonomně.

Pokud je vesničan uchopen, má hráč možnost otevřít jeho grafické rozhraní pomocí primárního tlačítka příslušného ovladače, ve kterém jej hráč drží. S rozhraním může hráč následovně interagovat pomocí interakčního paprsku druhé ruky. V tomto menu má možnost zvolit povolání vesničana pomocí tlačítek, které lze vidět na obrázku 6.15.

6.5.2 Magie

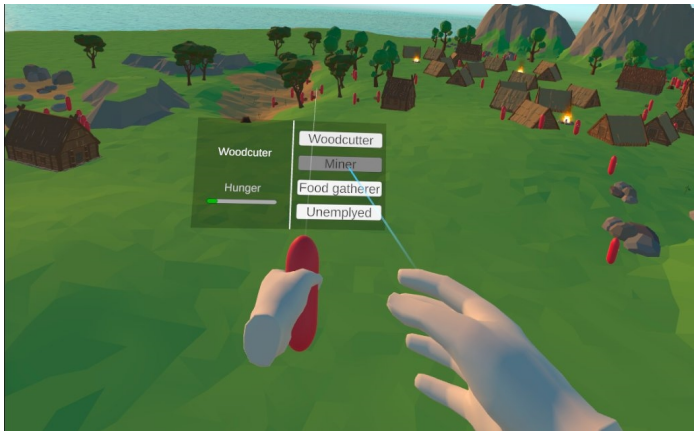
Hráč má na levé ruce menu, které lze zobrazit pomocí sekundárního tlačítka levého ovladače. Na něm je několik tlačítek, pomocí kterých lze sesílat kouzla, viz. obrázek 6.16. Ty může hráč seslat pouze v případě, že má dostatek many, což je hodnota, která je vidět pod tlačítky kouzel. Mana je spjata pouze se sesíláním kouzel a při každém seslání kouzla je z její hodnoty odebrána cena daného kouzla. Obnovuje se postupně časem s konstantním přírůstkem. Každé kouzlo má vlastní cenu seslání a má jiný účinek ve hře.

Ve hře jsou aktuálně dostupná jsou 3 kouzla:

Ohnivé kouzlo Na místě dopadu vytvoří zdroj surovin kamene a v okolí dopadu zapálí všechny buňky a hořlavé objekty ze systému šíření ohně (6.4).

Vodní kouzlo Uhasí všechny buňky a hořlavé objekty v okolí dopadu kouzla.

Kouzlo života V okolí dopadu obnoví všechny zdroje surovin, které jsou obnovitelné (zdroje dřeva a jídla) a všem vesničanům obnoví hodnotu zdraví.



Obrázek 6.15: Výsledné grafické rozhraní vesničana zobrazené po jeho uchycení do rukou. Lze v něm vidět jeho aktuální povolání a hodnotu hladu. Také v něm lze pomocí tlačítek na pravé straně změnit vesničanovo povolání.

Obrázek 6.16: Výsledná podoba grafického menu, pro sesílání kouzel. Po stisknutí jednoho z tlačítek se po pravé straně tohoto GUI instanciuje objekt kouzla, který může hráč následně uchopit do ruky. Na spodní části GUI je viditelná hráčova aktuální hodnota many.

Po instanciaci objektu kouzla jej může hráč uchopit a odhodit. Objekt kouzla poté čeká na první kolizi s nějakým objektem. Jakmile tato kolize nastane, detekuje pomocí funkce fyzikálního enginu `SphereCast(...)`[17] všechny objekty v oblasti definované koulí o průměru nastaveného v parametru prefabrikátu kouzla. Implementace akce provedené s těmito objekty se u každého kouzla liší, jak je popsáno výše. Příklad implementace ohnivého kouzla, ilustrující práci s funkcí `SphereCast(...)`, lze vidět na výpisu 6.2.

[17] https://docs.unity3d.com/ScriptReference/Physics.SphereCast.html

```
private void OnCollisionEnter(Collision other)
{
    var colliders = Physics.OverlapSphere(transform.position, effectArea);

    foreach (var collider in colliders)
    {
        if(collider.TryGetComponent<FireGridCell>(out var fireCell))
            fireCell.SetOnFire();
        else if(collider.TryGetComponent<FireableObject>(out var
            fireableObject))
            fireableObject.IgniteObject();
    }

    RunExplosionEffect();

    Destroy(gameObject);
}
```

Výpis 6.2: Kód implementující chování ohnivého kouzla, ilustrující získání objektů pomocí funkce fyzikálního enginu `SphereCast(...)`.

Kapitola 7

Uživatelské testovaní

Cílem této práce není pouhé vytvoření funkční virtuální hry, ale také aby byla hra uživatelsky přívětivá a byla dobrým základem pro rozšíření a dokončení v plnohodnotnou hru. Z toho důvodu bylo uskutečněno uživatelské testování, jehož cílem bylo potvrdit, zdali vytvořená hra splňuje očekávání.

Testování se uskutečnilo ve čtyřech kolech a každého kola se zúčastnili jiní uživatelé. Tři kola proběhla v domácím prostředí a jedno na půdě Fakulty informačních technologií v Brně. Celkem se testování zúčastnilo deset uživatelů. Uživatelé měli různé zkušenosti jak s virtuální realitou, tak s hraním videoher.

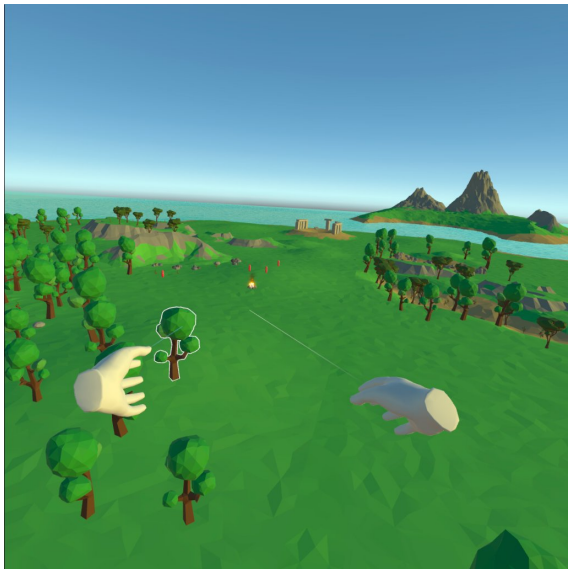
Proces testování probíhal ve čtyřech fázích. V první fázi byl uživateli sděleny základní informace o Oculus Quest 2 a jak bude proces testování probíhat. Byl také obeznámen se základními informacemi o této práci a co může ve hře očekávat.

V druhé fázi byl uživatel seznámen s ovládacími prvky virtuálních brýlí, kterými jsou prováděny veškeré interakce ve hře. Především se jednalo o správné držení a používání ovladačů. Následně byl instruován pro zajištění bezpečnosti při nasazených brýlích.

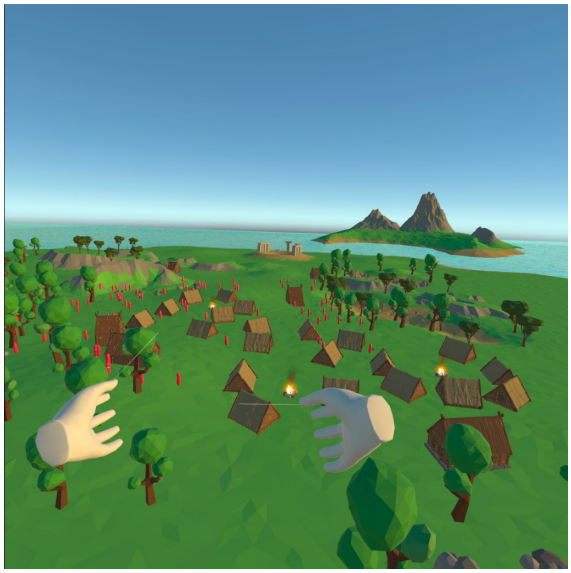
Ve třetí fázi měl už uživatel nasazené brýle, viz. obrázek 7.1, ve kterých byla spuštěna finální hra této práce. Pro každého uživatele byly připravené dvě scenérie. První scenérie znázorňovala, jak vypadá hra při rané fázi hry bez jakýchkoli postavených budov a malým počtem vesničanů (obrázek 7.2a). Druhá scenérie byla v pokročilé fázi hry, kde byl postaven velký počet budov a v herním světě bylo mnoho vesničanů (obrázek 7.2b). Uživatel byl postupně seznamován se všemi herními mechanikami, které jsou ve hře implementované. Po ukázání všech herních mechanik mohl uživatel dělat ve hře libovolné činnosti.

Obrázek 7.1: Fotka z uživatelského testování.

Ve čtvrté fázi byl uživateli předložen pro vyplnění testovací dotazník, který lze vidět v Příloze A. Po vyplnění dotazníku měl možnost položit doplňující dotazy.



(a) Scenérie rané fáze hry.



(b) Scenérie pokročilé fáze hry.

Obrázek 7.2: Snímky obrazovky z připravených scenérií pro testování.

7.1 Vyhodnocení

Jelikož virtuální realita není v dnešní době ještě příliš rozšířená, většina testovacích uživatelů s ní měla minimální nebo žádné zkušenosti. Pouze jeden uživatel měl dobré zkušenosti s virtuální realitou. Statistiky z vyplněných dotazníků ohledně ovládání mohou být z toho důvodu lehce zkresleny. I přesto byla ale valná většina odpovědí pozitivní. Detailní vyhodnocení testování je napsáno dále.

Cílem vyhodnocení testování je potvrzení nebo vyvrácení hypotéz, které byly určeny před testováním:

Hypotéza 1 Ovládání implementované hry ve VR je intuitivní a dá se rychle naučit.

Hypotéza 2 Pohybování pomocí teleportace je preferovaným způsobem pohybu.

Hypotéza 3 Implementované herní mechaniky jsou dobrým základem pro rozšíření na plnohodnotnou hru.

Hypotéza 4 Žánr implementované hry je vhodný pro implementaci ve VR.

Pro potvrzení nebo vyvrácení hypotéz se vychází převážně z vyplněných dotazníků od testovacích uživatelů a také z diskuzí vedených s uživateli v průběhu testování.

Otázky v dotazníku, na které bylo možné odpovědět stupnicí 1-10 bodů (kde 1 znamená zápornou odpověď a 10 kladnou), lze zprůměrovat od všech respondentů a zjistit u nich následující:

Dojem z vizuální prezentace - 9,1b Lze vyvodit, že „low-poly“[1] grafika, které byla zvolena pro vizualizaci výsledné hry, byla správným krokem.

Rychlost zorientování v ovládání - 7,8b Výsledek odpovídá dobré rychlosti zorientování v korelaci s průměrně nízkou zkušeností testovacích uživatelů s virtuální realitou.

Intuitivnost ovládání - 7,7b Podobně jako u předchozího bodu, je intuitivnost ovládání dobrá v korelaci s průměrně nízkou zkušeností testovacích uživatelů s virtuální realitou.

Zájem o dokončený produkt - 8,6b Většina respondentů projevila velký zájem o hru v případě, že by byly rozšířeny a dokončeny její herní mechaniky.

Výhoda virtuální reality v tomto žánru - 9,3b Valná většina respondentů se shodla, že virtuální realita v tomto herním žánru nabízí zajímavější a zábavnější herní mechaniky, než hraní na klasickém monitoru.

Součástí dotazníku byly také otevřené otázky, ve kterých mohli respondenti více specifikovat jejich pocity. Z odpovědí na tyto otázky lze vyvodit následující:

- Jediná mechanika, která byla vícekrát označena jako neintuitivní, je používání kouzel. To z toho důvodu, že po vybrání kouzla se kouzlo okamžitě objeví vedle menu a začne padat na zem. Ostatní mechaniky jsou intuitivní a jednoduché na naučení.
- Teleport preferovalo 5 lidí a plynulý pohyb také 5 lidí, takže ani jedna z variant není preferovaná většinou.
- Lehce nevolno se dělalo pouze respondentům používajícím plynulý pohyb.
- Respondenti vypsali výrazně více věcí, které je bavily/zaujaly, než které jim vadily/-nebavily.
- Celkově respondenti hodnotili hru kladně s velkým potenciálem pro rozšíření.

Z dat, které byly při testování nasbírány, lze vyhodnotit jednotlivé hypotézy následovně:

Hypotéza 1 Nebyla vyvrácena, naopak se výsledky naklánějí k jejímu potvrzení.

Hypotéza 2 Byla vyvrácena, jelikož plynulý pohyb byl stejně preferován jako teleportování.

Hypotéza 3 Byla potvrzena nejen na základě testů, ale také na základě diskuzí vedených s respondenty po testování.

Hypotéza 4 Byla potvrzena. Většina respondentů se shodla, že virtuální realita nabízí zajímavější interakci se světem.

[1] low-poly - modely jsou tvořené malým počtem trojúhelníků

Kapitola 8

Závěr

Cílem této práce bylo vytvořit funkční prototyp hry pro virtuální brýle Oculus Quest 2 v herním enginu Unity, který implementuje dobře rozšiřitelné herní systémy, na kterých lze v budoucnu vybudovat plnohodnotnou hru.

V první fázi byly získány znalosti z naučného programu od Unity pro začínající vývojáře her, který seznamuje nejen s herním enginem Unity, ale také s programovacím jazykem C#. Po jeho dokončení byl v rámci semestrálního projektu navržen a implementován první prototyp systému řízení agentů, který byl demonstrován pomocí jednoduchého multiagentního systému.

Druhá fáze byla zaměřena na studii procesu tvorby her ve virtuální realitě pro platformu Oculus Quest 2. V rámci této studie bylo vytvořeno několik herních prototypů, na kterých byla vyzkoušena základní integrace virtuální reality do her v Unity.

Ve třetí fázi byla navržena a implementována finální hra v netradičním žánru hry na boha. Jádrem hry je autonomní vesnice, které má hráč jakožto bůh pomáhat v rozvoji. Hráč může interakcemi ve virtuální realitě přímo ovlivňovat běh vesnice a jednou z výzev, které hráč čelí, je dynamické šíření ohně ve vesnici.

Poslední fází bylo uskutečnění uživatelského testování. Jeho cílem bylo získání zpětných vazeb na vytvořenou hru, které budou zohledněny při budoucím vývoji.

Hlavními úspěchy práce jsou systém řízení autonomních agentů, který je dostatečně optimalizovaný pro cílovou platformu Oculus Quest 2, systém šíření ohně implementovaný celulárním automatem, jehož buňky jsou mapovány na 3D terén, a úspěšná implementace interakcí ve virtuální realitě.

Tato práce mi přinesla spoustu nových znalostí a zkušeností. Naučil jsem se základům programovacího jazyka C# a vývoji her v Unity, nahlédl jsem do teorie inteligentních systémů a v neposlední řadě jsem si vyzkoušel praktické využití celulárních a konečných automatů. Celkově pro mne byla práce velice cennou zkušeností.

Ve vývoji hry je možné dále pokračovat. Lze přidávat a rozšiřovat herní mechaniky. Jedna z možností je přidání evoluce vesnice v čase. Vesnice by procházela různými epochami vývoje a každá epocha by přinášela nové herní mechanismy a výzvy. Zajímavou novou herní mechanikou by mohlo být přidání víry vesničanů v jejich boha. Tuto víru by si musel hráč budovat, aby mohl využívat mocnějších schopností. Nabízí se také možnost přidání hry ve více hráčích, kde by lidé mohli soupeřit, čí vesnice prosperuje lépe. Obohacením by také mohl být soubojový systém vesničanů, aby proti sobě mohly vesnice hráčů bojovat.

Literatura

[1] Almanasra, S., Suwais, K. a Mohd Arshad, M. The Applications of Automata in Game Theory. In:. Květen 2013, s. 204 – 217. DOI: 10.4018/978-1-4666-4038-2.ch011. ISBN 978-146-6640-38-2.

[2] Bjarnolf, P., Gustavsson, P., Brax, C. a Fredin, M. Threat Analysis Using Goal-Oriented Action Planning. In:. Září 2008.

[3] Constable, R. The Role of Finite Automata in the Development of Modern Computing Theory*. *Studies in Logic and the Foundations of Mathematics*. Prosinec 1980, sv. 101. DOI: 10.1016/S0049-237X(08)71253-9.

[4] Gong, Y. A survey on the modeling and applications of cellular automata theory. *IOP Conference Series: Materials Science and Engineering*. Září 2017, sv. 242, s. 012106. DOI: 10.1088/1757-899X/242/1/012106.

[5] Hayden, S. *'Job Simulator' Goes Platinum, Selling Over 1 Million Copies to Date* [online]. Leden 2020 [cit. 03.05.2022]. Dostupné z: https://www.roadtovr.com/ces-2020-job-simulator-1-million-units-sold/.

[6] Hayden, S. *'Superhot VR' Has Now Sold More Than 2 Million Copies* [online]. Květen 2020 [cit. 03.05.2022]. Dostupné z: https://www.roadtovr.com/superhot-vr-now-sold-2-million-copies/.

[7] Jagdale, D. Finite State Machine in Game Development. Říjen 2021, s. 384–390. DOI: 10.48175/IJARSCT-2062.

[8] Kari, J. Theory of cellular automata: A survey. *Theoretical Computer Science*. 2005, sv. 334, č. 1, s. 3–33. DOI: https://doi.org/10.1016/j.tcs.2004.11.021. ISSN 0304-3975. Dostupné z: https://www.sciencedirect.com/science/article/pii/S030439750500054X.

[9] Lawson, M. V. *Finite Automata* [online]. Listopad 2009 [cit. 19.04.2022]. Dostupné z: https://www.macs.hw.ac.uk/~markl/teaching/AUTOMATA/kleene.pdf.

[10] Levesque., J.-F. *Far Cry: How the Fire Burns and Spreads* [online]. Prosinec 2012 [cit. 29.03.2022]. Dostupné z: https://jflevesque.com/2012/12/06/far-cry-how-the-fire-burns-and-spreads/.

[11] Lin, Y. *10 VIRTUAL REALITY STATISTICS* [online]. Únor 2022 [cit. 15.04.2022]. Dostupné z: https://www.oberlo.com/blog/virtual-reality-statistics.

[12] O'Brien, M. a Chan, K. *What is the metaverse and how will it work?* [online]. Říjen 2021 [cit. 04.05.2022]. Dostupné z: https://abcnews.go.com/Business/wireStory/explainer-metaverse-work-80842516.

[13] Oculus. *Oculus Device Specifications* [online]. [cit. 27.01.2022]. Dostupné z: https://developer.oculus.com/resources/oculus-device-specs/.

[14] Oculus. *HOW OCULUS QUEST 2 IS CHANGING THE GAME FOR VR* [online]. Únor 2021 [cit. 28.01.2022]. Dostupné z: https://www.oculus.com/blog/from-bear-to-bull-how-oculus-quest-2-is-changing-the-game-for-vr/.

[15] Oculus VR. *Introducing Oculus Quest 2, the Next Generation of All-in-One VR* [online]. Září 2020 [cit. 15.04.2022]. Dostupné z: https://developer.oculus.com/blog/introducing-oculus-quest-2-the-next-generation-of-all-in-one-vr/.

[16] Orkin, J. *Applying Goal-Oriented Action Planning to Games* [online]. 2003 [cit. 19.04.2022]. Dostupné z: http://alumni.media.mit.edu/~jorkin/GOAP_draft_AIWisdom2_2003.pdf.

[17] Studiawan, R. Tactical Planning in Space Game using Goal-Oriented Action Planning. *JAREE (Journal on Advanced Research in Electrical Engineering)*. Květen 2018, sv. 2. DOI: 10.12962/j25796216.v2.i1.32.

[18] Toll, W. van, Iv, A. a Geraerts, R. A Navigation Mesh for Dynamic Environments. *Computer Animation and Virtual Worlds*. Listopad 2012, sv. 23, s. 535–546. DOI: 10.1002/cav.1468.

[19] Unity. *Compare plans* [online]. 2022 [cit. 27.01.2022]. Dostupné z: https://store.unity.com/compare-plans.

[20] Unity. *Unity manual* [online]. Květen 2022 [cit. 27.01.2022]. Dostupné z: https://docs.unity3d.com/Manual/.

[21] Unity. *Unity Real-Time Development Platform* [online]. 2022 [cit. 27.01.2022]. Dostupné z: https://unity.com/.

[22] VR, O. *OCULUS JOINS FACEBOOK* [online]. Březen 2014 [cit. 03.05.2022]. Dostupné z: https://www.oculus.com/blog/oculus-joins-facebook/.

[23] Wikipedia contributors. *Wikipedia, The Free Encyclopedia*. [cit. 03.05.2022]. Dostupné z: https://en.wikipedia.org/.

Příloha A

Testovací dotazník

Dotazník k bakalářské práci - Hra pro Oculus Quest

Dobrý den, před Vámi leží dotazník sloužící pro testovací část mé bakalářské práce. Cílem mé práce bylo vytvořit hru pro virtuální realitu a následně za pomoci dobrovolníků otestovat kvalitu výsledné realizace. Děkuji Vám za ochotu vyplnit tento krátký dotazník, který mi umožní zpracovat výsledek mé práce.

Prohlašuji, že tento dotazník je anonymní a veškeré informace uvedené v něm budou použity pouze pro testovací účely mé bakalářské práce.

Jakub Kryštůfek

Informace o Vás:

Věk: _____

Jak často hrajete videohry?

nikdy jsem nehrál ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10 hraji každý den

Hrál/a jste někdy VR hry, případně jak často?

nikdy jsem nehrál ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10 hraji každý den

Dotazy ke hře:

Jaký byl Váš první dojem z vizuální prezentace hry?

vůbec se mi nelíbila ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10 velice se mi líbila

Jak rychle jste se zorientoval/a v ovládání hry?

vůbec ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10 ihned

Bylo pro vás ovládání intuitivní?

vůbec ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10 naprosto intuitivní

Případně co pro vás bylo neintuitivní nebo nelogické? _____

_____

Jaké ovládání pohybu pro Vás bylo přívětivější? ○ teleport ○ plynulý pohyb

Dělalo se Vám při hraní nevolno? ○ ano ○ ne

Pokud ano, při čem? _____

Co Vás na hře bavilo nebo zaujalo? _____

_____

_____

Co Vás na hře nebavilo nebo Vám vadilo? _____

_____

_____

Pokud by byla hra rozšířena a dokončena, měl/a byste zájem ji hrát?

určitě ne ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10 určitě ano

Napadá Vás něco, co Vám ve hře opravdu chybělo nebo byste to ve hře ocenili? _____

_____

_____

Myslíte si, že je hra zábavnější ve virtuální realitě, než kdyby byla na klasickém monitoru?

určitě ne ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10 určitě ano

Proč si to myslíte? _____

_____

Máte nějaké doplňující připomínky, nápady nebo výtky? _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Příloha B

Obsah přiložené SD karty

Přiložená SD karta obsahuje:

- zdrojové soubory implementované hry
- zdrojový tvar technické zprávy
- technickou zprávu ve formátu PDF
- binární soubor implementované hry spustitelný na Oculus Quest 2
- prezentační video výsledné hry