FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Strategická 3D hra se strojovým učením
Student:	Richard Smělý
Vedoucí:	Ing. Petr Pauš, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Počítačová grafika
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem práce je navrhnout a vytvořit jednoduchou strategickou 3D hru, která bude využívat principy strojového učení. Součástí práce je i vytvoření sady 3D modelů.

1. Analyzujte dostupné hry využívající strojové učení.
2. Analyzujte, jaké modely strojového učení lze v této situaci použít.
3. Analyzujte dostupné 3D herní enginy a na základě analýzy vhodný vyberte.
4. Pomocí nástrojů softwarového inženýrství navrhněte prototyp hry včetně uživatelského rozhraní.
5. Implementujte hru s použitím vlastních 3D modelů.
6. Proveďte vhodné testování.

Elektronicky schválil/a Ing. Radek Richtr, Ph.D. dne 31. srpna 2022 v Praze.

Bakalářská práce

STRATEGICKÁ 3D HRA SE STROJOVÝM UČENÍM

Richard Smělý

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Petr Pauš, Ph.D.
3. ledna 2023

Obsah

Seznam obrázků

Seznam tabulek

Děkuji svému vedoucímu a všem ostatním, kteří mne při vypracování bakalářské práce podporovali.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 3. ledna 2023 .....

Abstrakt

Bakalářská práce se zabývá vývojem třírozměrné strategické počítačové hry a využitím umělé inteligence v této hře. Obsahuje analýzu již dostupných her, porovnává výhody a nevýhody vybraných herních enginů. Zkoumá využitelnost strojového učení ve hrách. Popisuje návrh a implementaci vlastní třírozměrné strategické hry se strojovým učením v herním enginu Unity. V této hře jsou využity vlastní vymodelované a otexturované 3D modely. Hlavním výstupem práce je spustitelná aplikace – 3D tower defense hra, která ilustruje využití evoluční neuronové sítě pro samostatnou navigaci nepřátelských jednotek. Hra je na závěr otestována a kladně zhodnocena uživateli. Postupy uvedené v práci umožňují čtenáři nahlédnout do procesu vývoje počítačových her a míry využitelnosti strojového učení v tomto procesu.

Klíčová slova strategická 3D počítačová hra, evoluční neuronová síť, strojové učení, herní engine, Unity

Abstract

The bachelor thesis deals with the development of a three-dimensional strategy computer game and the use of artificial intelligence in this game. It includes an analysis of already available games, compares the advantages and disadvantages of selected game engines. It explores the applicability of machine learning in games. Describes the design and implementation of a self-developed three-dimensional machine learning strategy game in the Unity game engine. This game uses custom modeled and textured 3D models. The main output of the thesis is an executable application – a 3D tower defense game that illustrates the use of an evolutionary neural network for autonomous navigation of enemy units. The game is then tested and positively evaluated by users. The procedures presented in the paper give the reader insight into the process of computer game development and the extent to which machine learning can be used in this process.

Keywords strategic 3D computer game, evolutionary neural network, machine learning, game engine, Unity

Seznam zkratek

2D	dvoudimenzionální
3D	třídimenzionální
FPS	first-person shooter
GPU	graphics processing unit
NS	neuronová síť
PC	personal computer
UE	Unreal Engine

Úvod

Umělá inteligence a strojové učení se v posledním desetiletí vyskytuje od nejodbornějších přírodovědeckých odvětví meteorologie po masově zaměřené přizpůsobování reklam na sociálních sítích. A lze očekávat další nárůst poptávky po nových, lepších systémech umělé inteligence. Nejde jen o vozidla bez lidského řidiče, ale také o virtuální asistenty, zákaznickou podporu a dalších aplikace. Je tedy vhodné a pro společnost přínosné prozkoumávat možnosti tohoto konceptu samoučenlivých algoritmů.

Experimentům snadno přístupným médiem pro toto zkoumání jsou například právě počítačové hry. Vývoj je oproti jiným alternativám extrémně rychlý, protože není potřeba dbát na žádná bezpečnostní rizika. Výstupy jsou velmi názorné a prezentovatelné. A když se pojem počítačová hra rozšíří i na mobilní zařízení, tak dosah může být takřka globální.

Tato práce se tedy podílí na rozvoji chápání a využití umělé inteligence, která může základní principy přiblížit i široké veřejnosti.

Hlavním cílem bakalářské práce je navrhnout a vytvořit jednoduchou strategickou 3D hru ve dle analýzy vybraném prostředí, která bude využívat principy strojového učení. Součástí práce je i vytvoření a využití sady 3D modelů pro tuto hru.

Díky analytické části si může čtenář udělat představu nejen o herní využitelnosti umělé inteligence, ale také o dostupných možnostech vývoje třírozměrných počítačových her ve specializovaných prostředích – *herních enginech*. Text obsahuje také příklady již existujících her, které určitým způsobem využívají strojové učení.

V práci je popsán postup tvorby jedné takové hry od úplných začátků po funkční aplikaci. Při návrhu a implementaci je kladen důraz na rozšiřitelnost a přehlednost projektu. Práce může sloužit ostatním vývojářům jako inspirace a odhalovat použitelné postupy využití možností strojového učení ve hrách, vizualizacích či jiných grafických aplikacích.

Implementovaná hra je nakonec otestována a zhodnocena samotnými hráči.

Kapitola 1
Vymezení pojmů a definic

1.1 Strojové učení

Aby mohl být definován pojem strojové učení, je potřeba nejdříve definovat pojem *umělá inteligence*.

▶ **Definice 1.1.** ***Umělá inteligence*** *(AI) je takový program, který určitý úkol nebude zvládat hůř než člověk.* [1]

Nyní se již dá definovat pojem strojové učení, které je podkategorií algoritmů umělé inteligence.

▶ **Definice 1.2.** ***Strojové učení*** *je odvětví umělé inteligence a počítačových věd, které se zaměřuje na využití dat a algoritmů k imitaci způsobu, jakým se učí lidé. Postupně tímto zvyšuje svoji přesnost.* [2]

Samotný proces zlepšování algoritmu je *trénink*. Souboru dat určených pro trénování modelu se někdy říká *dataset*.

Také se dále bude hodit definice *příznaku*.

▶ **Definice 1.3.** ***Příznak*** *je vlastnost dat, která se algoritmům strojového učení dává jako vstup.*

Strojové učení lze dále rozdělit do čtyř typů takzvaných *modelů* – způsobů, kterými se algoritmy učí: *s dohledem*, *s polodohledem*, *bez dohledu* a *posilovací*.

1.1.1 Model strojového učení

▶ **Definice 1.4.** ***Učení s dohledem*** *využívá k učení předem označená data, které mají vstupní hodnoty a očekávané (správné) výstupní hodnoty.*

Tento model strojového učení se využívá, pokud je třeba co nejpřesněji predikovat očekávanou hodnotu na základě vstupních dat.

▶ **Definice 1.5.** ***Učení bez dohledu*** *využívá k učení neoznačená data, která mají pouze vstupní hodnoty.*

Použití tohoto modelu je zejména v oblastech, kde je potřeba automatická kategorizace dat.

▶ **Definice 1.6.** ***Učení s polodohledem*** *využívá k učení data předem označená i neoznačená.* [3]

Model je kombinací 1.4 a 1.5 (v některých zdrojích se ani ve výčtu modelů neuvádí). Vypadá a využívá se v podobných situacích jako 1.4 s rozdílem, že z různých důvodů jsou k označeným datům přidána data neoznačená.

▶ **Definice 1.7.** ***Posilovací učení*** *je průběžné učení algoritmu (takzvaného agenta), který dokáže vnímat stav prostředí, ve kterém se nachází, jako vektor příznaků a provádět akce, které mohou měnit stav prostředí.* [4]

Pro tuto práci nejzajímavější model, který bude ve větším detailu probrán dále v textu.

1.2 3D počítačová grafika

Pojem *počítačová grafika* je často alespoň podvědomě známý, avšak uvést jeho přesnou definici není zdaleka tak triviální. Počítačovou grafiku lze (korektně) vnímat více různými způsoby a s různou úrovní obecnosti. Tato práce se pro jednoduchost bude držet definice následující.

▶ **Definice 1.8.** ***Počítačová grafika*** *je disciplína počítačových věd zabývající se zobrazením grafických prvků – tvarů, barev a jejich přechodů na zobrazovací zařízení.*

Do počítačové grafiky se samozřejmě řadí jak softwarové tak hardwarové řešení. Text se zabývá převážně softwarovým řešením, které zobrazuje do rasterizované (diskrétní) mřížky barevných bodů na displeji.

▶ **Definice 1.9.** ***3D*** *neboli třírozměrná počítačová grafika se zabývá zobrazením třírozměrného prostoru.*

Většina současných zobrazovacích zařízení je realizována dvourozměrnou mřížkou světelných bodů, které mohou být monochromatické nebo barevné. Aby bylo možné zobrazit třírozměrný prostor, je tedy potřeba provést transformaci, která jeden rozměr odstraní.

Zde se hodí zadefinovat dva druhy takových transformací – *projekcí.*

▶ **Definice 1.10.** ***Ortografická projekce*** *je afinní transformace, která zachová rovnoběžnost přímek.*

Vzdálenost objektů od kamery při použití této transformace nemění jejich velikost.

▶ **Definice 1.11.** ***Perspektivní projekce*** *je transformace, která nemusí zachovávat rovnoběžnost přímek. Pohledové přímky konvergují do středu projekce.*

Perspektivní projekce se snaží simulovat prostorové vnímání obrazu lidským okem. Vzdálenější objekty se vykreslují zmenšené.

Pro vykreslování 3D scény se dají využít různé metody, jednou z nich je *ray tracing* – trasování paprsků.

▶ **Definice 1.12.** ***Ray tracing*** *je algoritmus, který vypočítává trasu paprsků z pozorovatelova oka skrze každý pixel do nejbližšího průsečíku s povrchem ve scéně. Následně provede výpočet odražených paprsků a dle povrchu pixel vystínuje.* [5]

Aby mohl být definován *herní engine*, je třeba nejdříve zavést definici *softwarového frameworku.*

▶ **Definice 1.13.** ***Softwarový framework*** *je software, který slouží jako základ pro vývoj určitého druhu aplikace.*

Pro vývoj aplikací, které z velké části využívají 1.9 je často vhodné použít *herní engine*, který automatizuje části procesu vývoje.

▶ **Definice 1.14.** ***Herní engine*** *je softwarový framework, který poskytuje vývojářům sadu nástrojů a prvků pomáhajících zefektivnit vývoj počítačových her.* [6]

1.3 Strategická hra se strojovým učením

▶ **Definice 1.15.** ***Hra*** *je aktivita nebo sport. Účastník hry využívá dovednosti, znalosti či náhodu k tomu, aby podle pravidel této hry zvítězil nad protivníkem nebo vyřešil hlavolam.* [7]

▶ **Definice 1.16.** ***Počítačová hra*** *je hra, kterou hráč hraje na stroji výpočetní techniky.*

Počítačové hry tedy nejsou pouze hry určené pro stolní počítače. Lze tak označit i hry pro mobilní zařízení, systémy pro virtuální realitu a podobně.

▶ **Definice 1.17.** ***Strategická hra*** *je hra, ve které mají hráčova rozhodnutí významný dopad na výsledek.*

Strategická počítačová hra se strojovým učením je tedy potom hra dle definic 1.15, 1.16 a 1.17, která při vývoji nebo za běhu využívá algoritmy 1.2.

Implementovaná hra bude typu *tower defense*.

▶ **Definice 1.18.** ***Tower defense*** *hra je strategická hra, ve které se hráč snaží zastavit nebo zpomalit nepřátelské jednotky pokládáním ofenzivních budov.* [8]

Kapitola 2
Analytická část

Analýza témat potřebných k vypracování praktické části.

V následujících kapitolách jsou analyzovány hlavní tři pilíře této práce: strojové učení, herní enginy a již existující aplikace – hry využívající strojové učení. V těchto kapitolách není popsáno zdaleka všechno z jejich respektivních odvětví počítačových věd. Slouží spíše pro orientaci v tématu modelů strojového učení a náhled na řešení a postupy využitelné v praktické části práce. Pojem *učení* v textu je pouze zkrácená verze pojmu *strojové učení*.

2.1 Analýza modelů strojového učení

Dle definic v podsekci 1.1.1 jsou algoritmy strojového učení rozděleny do 4 modelů. Každý má své výhody a nevýhody. Text se bude zabývat hlavně možnostmi využití těchto modelů při návrhu, implementaci a běhu her.

2.1.1 Učení s dohledem

Prvním modelem je strojové učení *s dohledem* (někdy také *pod dohledem*). Velmi rozšířeným ukázkovým příkladem využití tohoto modelu je na souboru obrázků ručně psaných číslic zvaném *MNIST Dataset*. Pokud dáme dobře naučenému algoritmu na vstup obrázek ručně psané číslice, tak dokáže s přesností i přes 99 % určit její hodnotu [9].

Tento model je v praxi využíván nejčastěji. Prvním krokem v procesu strojového učení s dohledem je sběr datových párů – vstupní a výstupní. Vstupní i výstupní data mohou být jakákoli, v dalším kroku je však potřeba tyto data vyjádřit v číselné podobě. Například v předchozím odstavci dataset MNIST již má data v číselné podobě. Každý pixel reprezentuje jeden *příznak* s hodnotou stupně šedi v intervalu $\langle 0; 1\rangle$. Když jsou ovšem data nečíselná, jako příklad lze uvést třeba typy herních jednotek, je třeba určitým způsobem tyto informace převést na číselné vyjádření. Jedním z možných způsobů řešení je *one-hot encoding*, který pro každý možný typ vstupu vytvoří nový příznak. Pro one-hot encoding platí, že všechny příznaky jsou nulové kromě toho, který určuje typ vstupu. Vstupní data je často také vhodné převést na hodnoty jednoho intervalu (nejčastěji $\langle 0; 1\rangle$ nebo $\langle -1; 1\rangle$). Tento proces se nazývá *normalizace*. Nyní již může začít *trénink* modelu, který postupně zlepšuje přesnost predikce. Při trénování může nastat problém *přetrénování* (*overfitting*). Model se velmi dobře naučí predikovat správné hodnoty pro trénovací data, ale ztratí schopnost *generalizace* pro data, která mu nebyla poskytnuta při tréninku. [3]

Formát výstupu závisí na aplikaci, na kterou je strojové učení využito a dělí se na dva základní typy: *klasifikaci* a *regresi*. Přiřazování *označení* (*label*) – do jaké třídy vstup patří se nazývá

klasifikace. Detekce spamu je slavný příklad podtypu *binární* klasifikace, která určuje pouze dva stavy (spam a nespam). *Diskrétní* (*multiclass*) klasifikace rozhoduje mezi třemi a více třídami. Pro jednotlivý vstup je typicky určena pravděpodobnost, se kterou patří do každé třídy[1]. Naproti klasifikaci stojí regrese, která na základě vstupu určuje co nejpřesněji výstupní souvislou hodnotu. Výstup může tvořit i vektor hodnot. [3]

Model strojového učení s dohledem může hra využít ve chvíli, kdy lze pro vstup určit „správnou“ výstupní hodnotu. Příkladem algoritmu, který je v některých hrách implementovaný (může, ale nemusí však nutně být na bázi strojového učení) je *chat protection*. Tento algoritmus určuje, jestli jsou textové zprávy napsané jednotlivými hráči vulgární nebo *ofenzivní* – jestli vyjadřují nějakou formu rasové, homofobní a podobné nenávisti. Naivní algoritmus může označit za vulgární všechny zprávy obsahující sprostá slova. Tímto však tento jednoduchý algoritmus končí. Identifikace kontextu, v jakém je například slovo *black* vyjádřením rasové nenávisti, je úkol spíše pro strojové učení. V tomto případě jde o binární klasifikaci: text je ofenzivní / text je v pořádku.

Učení s dohledem také dokáže generovat nový náhodný obsah s rysy trénovacích dat. U her, které se neodehrávají v reálných lokacích, lze strojovým učením generovat například prostředí, a to rovnou dvěma způsoby. Buď vývojář využije strojové učení pouze při vývoji hry. Nechá tedy algoritmus vygenerovat zvolené množství různých prostředí, které následně do hry nastálo přidá. Výhoda tohoto přístupu je možnost preciznější ručního výběru a úpravy před přidáním do hry. Nebo vývojář pouze natrénuje model tak, aby generoval uspokojivě vypadající prostředí a nechá prostředí generovat až při běhu hry. Tímto dosáhne téměř neomezené zásoby různých prostředí, což může hře pomoci se znovuhratelností (*replayability*).

Limitujícím faktorem využití strojového učení s dohledem ve vývoji her je nutnost předchozího sběru dat. Pokud vývojář nemá k dispozici dostatek dat, tak žádný model nenatrénuje. Zvláště pro začínající vývojáře může toto být problém.

2.1.2 Učení bez dohledu

Strojové učení bez dohledu je využíváno na řešení úloh, u kterých nejsou výstupní data označena. Data se tedy nesbírají v párech, stačí pouze vstupní. Možnosti transformace vstupních dat jsou stejné jako u učení s dohledem. Model se dá využít pro rozřazení dat do předem neznámých kategorií zvaných *clusters* (*clustering* algoritmus). Posuzovaní správnosti je u tohot modelu obtížné, protože vývojář nemá žádný referenční bod, se kterým může kvalitu trénovaného modelu porovnat. Reálné využití může najít například v:

- Kategorizaci ve chvíli, kdy není předem znám počet ani typ kategorií
- Hledání *outlierů* – datových bodů, které jsou značně vzdálené od ostatních
- Redukce dimenze vstupu [3]

V herním průmyslu by se dalo využít strojové učení bez dohledu na kategorizaci hráčů. Hry často umožňují uživatelům hrát různým stylem. Díky kategorizaci těchto stylů může vývojář hry získat cennou infromaci o většinově využívaných taktikách a strategiích. Hry, u kterých je nutné připojení k internetu, jsou ideálním kandidátem na implementaci tohot modelu. Vývojáři stačí sbírat dostupné informace při běhu hry na uživatelově počítači a odesílat tyto informace na vybraný server, kde se časem nastřádá dostatek dat pro clustering.

Na principu hledání *outlierů* může fungovat i identifikace hráčů, kteří při online hrách podvádí (takzvaných *cheaterů*). Problematika podvádění v online hrách se zprvu může čtenáři jevit zanedbatelná. Faktem však zůstává, že na světové e-sportové soutěži ve hře *Dota 2* pořádané pod jménem *The International 2021* si soutěžící odnesli peněžní výhru v celkové hodnotě $40 018 400 amerických dolarů. Taková suma může jistě pro mnohé být motivující sáhnout i po zakázaných

[1] Jde tedy o jakousi míru „jistoty“.

programech, které poskytnou hráči nesportovní výhodu. Za předpokladu, že většina hráčů nepodvádí, tak pomocí nalezení outlierů lze identifikovat hráče, kteří se výrazně od ostatních liší. Nemusí to samozřejmě hned znamenat, že podvádějí – třeba jsou akorát velmi dobří. Pro finální verdikt je možné zapojit lidskou pracovní sílu[2], která již precizně zhodnotí pouze vybrané případy.

2.1.3 Učení s polodohledem

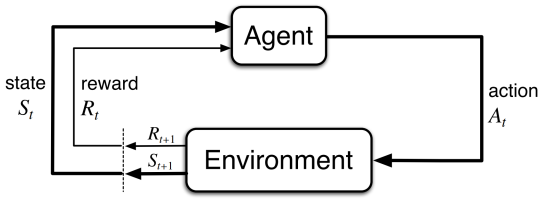
V případech, kdy získání dostatečně velkého[3] označeného datasetu je příliš nákladné (ať už časově nebo finančně), se dá pro zpřesnění výstupu vyzkoušet model strojového učení s polodohledem z definice 1.6. Co se týče reálného využití tohoto modelu, tak se téměř neliší od učení s dohledem, takže dále nebude analyzováno. [3]

2.1.4 Posilovací učení

Model posilovacího strojového učení se od ostatních modelů značně liší. Na obrázku 2.1 lze vidět princip fungování *agenta* na základě Markovova rozhodovacího procesu.

Samotná implementace číselného výpočtu výstupu na základě vstupu může být stejná jako u ostatních modelů. Význam těchto vtupů a výstupu je jiný. Agent interaguje přímo s *prostředím*, ve kterém se nachází. Vstup vyjadřuje momentální *stav* prostředí. Výstup vyjadřuje *akci*, kterou agent v tomto stavu chce provést. Tato akce může změnit momentální stav a přínést agentovy *odměnu* (ve formě číselného ohodnocení). Agent se snaží celkovou odměnu maximalizovat. [4]

Obrázek 2.1 Diagram fungování agenta [4]



Oproti ostatním modelům dělají tyto vlastnosti z posilovacího učení univerzálnější nástroj. Agent vnímá problém jako celek – nerozděluje problém na podproblémy, takže dokáže upřednostnit dlouhodobé cíle (reprezentované vyšší odměnou) před krátkodobými.

V herním světě se této vlastnosti dá využít pro ovládání jednotek řízených počítačem. Ukázkovým příkladem je umělá inteligence pro hru šachy. Aby dokázala porazit zkušeného lidského hráče, musí optimalizovat své tahy za účelem eventuálního vítězství v kontextu celé hry. Musí tedy umět rozpoznat situace, kde se vyplatí například místo vyhození nepřátelské figurky (za což dostane odměnu) dostat svou armádu do lepší pozice, která následně vede k jistějšímu vítězství.

Další výhodou posilovacího učení je, že algoritmus nepotřebuje žádný předcházející krok sběru dat. Formát vstupních dat sice stále potřebuje náležitou péči, ale vývojář nepotřebuje mít předem nachystaný dataset. Agent je pouze „vypuštěn“ do prostředí, ze kterého se samostatně naučí jeho kontext.

Tento model tedy lze využít v případech, kdy by vývojář jen těžko získával data pro trénování. Při vývoji nové hry, která má mít navigační algoritmus pro jednotky ve hře, může dokonce posilovací učení objevit způsob procházení herní úrovně, který je optimálnější a vývojáři by jinak unikl a tudíž by ho do trénovacích dat ani nezahrnul.

Posilovací učení lze využít i v aplikacích s prostředím, které se postupně mění[4]. Tato postupná adaptabilita u ostatních modelů strojového učení chybí.

Když vývojář nechá agenta, aby se učil i po vydání / za běhu hry, tak může docílit toho, že se adaptuje na herní styl hráče a je mu tak rovnějším soupeřem. Dalo by se takto docílit něčeho

[2] I se strojovým učením se tomuto nákladnému zdroji občas nelze vyhnout.
[3] Dostatečně velkého k uspokojivému natrénování modelu.
[4] Prostředí, které se mění jako celek – nejen jeho stav vůči agentovi.

jako *samovyvažující* se hry: pokud je hráč lepší než stroj, tak se od něj stroj může učit, pokud se stroj dostane na stejnou úroveň jako hráč, tak už se učit nemusí.

Mezi výzvy posilovacího učení patří vyvážení takzvaného *objevování* a *využívání* (*exploration* vs *exploitation*). Aby agent získal co nejvyšší odměnu, tak musí preferovat akce, které již vyzkoušel a ukázaly se být efektivní. Pro objevení takových akcí musí otestovat i nové, nevyzkoušené akce, které mohou potenciálně být ještě efektivnější. Dilema tkví v tom, že ani objevování ani využívání nelze aplikovat exkluzivně. [4]

Kvůli výhodám uvedeným výše byl pro praktickou část zvolen tento model strojového učení.

2.2 Analýza dostupných 3D herních enginů

Do kategorie herních enginů je možné zahrnout široké spektrum softwaru. V této bakalářské práci budou analyzováni dva zástupci: *Unreal Engine* a *Unity*.

Výběr právě těchto dvou herních enginů není náhodný. Na internetové platformě Steam[5] je přes 50 % herních aplikací vyvíjeno v herním enginu Unity. Druhým nejpoužívanějším je Unreal Engine, který ohledně počtu vydaných her za Unity značně zaostává. U herního enginu však nejde jen o počet vydaných titulů. Prvenství Unreal Engine zastává na poli her ve vyšší cenové kategorii – 25 % her vyvíjených právě v UE má cenovku přes $29.99 (v porovnání s Unity s pohými 6 %) a hry s cenou přes $49.99 jsou vyvíjeny téměř exkluzivně v UE nebo specializovaných herních enginech. [10]

Oba tyto herní enginy nabízí širokou paletu nástrojů, které vývojářům usnadňují proces tvorby počítačových her. Jak bude čtenáři při čtení této kapitoly čím dál více jasné, tak jsou si tyto dva herní enginy v mnohém podobné. Text se tedy zaměří nejen na holý popis těchto dvou herních enginů, ale také na rozdíly mezi nimi.

2.2.1 Unity

Herní engine vydán roku 2004 malou firmou *Unity Technologies*[6] v Dánsku. Mezitím, co konkurentní herní enginy cílily na velké společnosti vyvíjející nejrozsáhlejší herní tituly na trhu, se nový projekt Unity zaměřil na nezávislé herní vývojáře s menším rozpočtem. S rostoucí popularitou se Unity vyvinulo v robustní herní engine, který má co nabídnout i větším firmám. [11]

Vývojové a běhové prostředí Unity lze využívat se 4 různými licenčními úrovněmi *tiers*. Softwarové řešení Unity tedy není univerzálně zdarma. Pokud však uživatel má nárok na osobní licenční úroveň *Unity Personal*, tak může využívat velkou část tohoto herního enginu bez omezení. Na osobní licenční úroveň má nárok každý (ať už fyzická nebo právnická osoba), komu nepřekročí příjmy či financování $100 000 amerických dolarů za předchozích 12 měsíců od data použití softwaru. Při překročení tohoto limitu musí osoba změnit licenci na vyšší úroveň se jménem *Unity Plus*. Ta má také obdobný finanční limit ve výši $200 000 a stojí $399/rok za jednoho člena týmu. Když se osoba, nyní již nejspíše právnická, přehoupne přes tento finanční limit, tak již musí využít neomezený plán *Unity Pro*, u kterého značně naroste cena na $2 040/rok za jednoho člena týmu. Unity Pro již nemá žádný limit, avšak pro korporace je v nabídce ještě licenční plán *Unity Enterprise*, který nemá veřejnou cenu – pouze po dohodě. [12], [13]

Dražší licence postupně umožní uživatelům nahlédnout do zdrojového kódu Unity, či jej dokonce editovat pro své potřeby. Již Unity Plus umožní vývojáři upravit výchozí obrazovku (tzv. *splash screen*). Vyšší úrovně licence také uživateli zajistí například lepší zákaznickou podporu, diagnostiky a některé pokročilé balíčky (Havoc Physics, Unity Mars). Vesměs nejde o nezbytně nutné součásti herního enginu, takže pro menší projekty by měly stačit levnější licence. Jedna ze zásadních věcí, která však u levnějších licencí chybí, je možnost nasazení aplikace na herní konzole. Vývojář se tedy může rozhodnout, jestli mu multiplatformnost zahrnující Nintendo Switch,

[5] Platforma zabývající se elektronickou distribucí her a softwaru.
[6] Do roku 2007 pod jménem *Over the Edge Entertainment*.

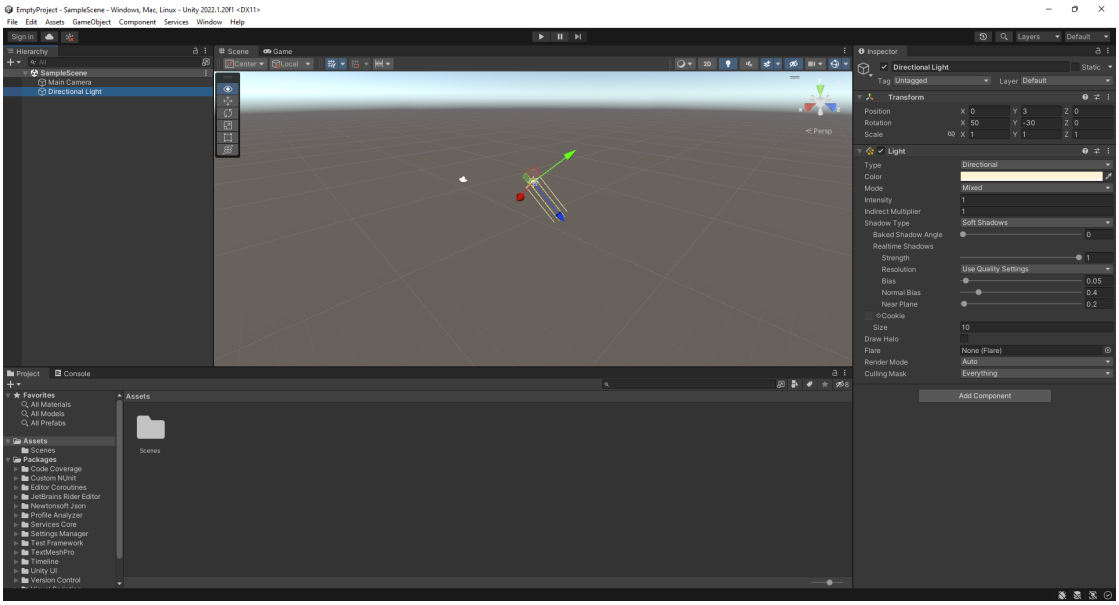
PlayStation a Xbox stojí za investici do Unity Pro. Ve všech licenčních verzích je dostupný *Unity Ads* a *In-App Purchase plug-in*, díky kterým může vývojář na své hře profitovat. Pro účely této práce bohatě postačuje licenční plán Unity Personal. [13]

Stažení a instalace základní verze Unity v plánu *Personal* je zcela zdarma. Samotné vývojové prostředí Unity se dá nainstalovat a spouštět jako samostatná aplikace nebo je možné využít aplikaci *Unity Hub*, která se automaticky stará o projekty, verze a licence Unity. Unity Hub navíc odkazuje na výukové materiály Unity Learn a obchod Asset Store. Při tvorbě této práce je použito Unity verze 2022.1.20f1 [14].

Při vytváření nového projektu je možno vybrat z několika šablon (*templatů*). Ve výběru jsou jak 2D tak 3D šablony. Dále se bude text zabývat hlavně tvorbou 3D her.

Základním stavebním prvkem herního enginu Unity jsou speciální objekty *gameobject*. Tyto gameobjecty jsou shlukovány a uspořádány ve *scénách*. Každý gameobject má svůj *transform* – pozici ve scéně. Jak se bude gameobject chovat určují *komponenty*, které jsou k němu připojené. Komponent je více druhů. Mezi základní patří například různé druhy světel, povrchů, kolizních detektorů a obalů, simulátorů fyzického chování, kamer a zdrojů zvuku. Tyto druhy jsou předdefinované a vývojář může měnit pouze hodnoty proměnných. Existuje však zvláštní druh komponenty zvaný *script*, který pomocí programovacího jazyka C# může vytvářet zcela nové druhy komponent, které jsou plně v režii vývojáře.

Obrázek 2.2 Uživatelské rozhraní Unity



Po vytvoření nového projektu uživatele uvítá rozhraní podobné jako na obrázku 2.2. Většinu plochy zabírá zobrazení 3D scény. Vlevo nahoře je vidět okno hierarchie – seznam jednotlivých gameobjectů v právě prohlížené scéně. Gameobjecty mohou být i zanořené, což hierarchie reflektuje. Při výběru nějakého gameobjectu (ať už v okně hierarchie nebo ve scéně) se napravo v *inspektorovi* zobrazí informace o vybraném gameobjectu. Na obrázku 2.2 je konkrétně vybráno směrové světlo. V inspektorovi je tedy vidět transform a komponenta *Light* s výchozími hodnotami, která gameobjectu přidává vlastnosti směrového světla. Komponent může mít jeden gameobject teoreticky neomezeně. V levém dolním rohu je okno projektu, které zobrazuje adresářovou strukturu projektu. Soubory s metadaty (*.meta*) Unity nezobrazuje.

Již v tomto stavu je hra přímo ve vývojovém prostředí spustitelná tlačítem se symbolem ▷ nahoře[7]. Přidání modelů je možné přesunutím souborů ve formátu *.fbx* nebo *.blend* přímo

[7] Ano, zatím jde pouze o statickou scénu.

do Unity vytvořeného adresáře *. . . /[jméno projektu]/Assets/*. Unity následně automaticky takto přesunuté soubory importuje do projektu.

Ekosystém Unity disponuje obchodem *Asset Store*. Na této internetové platformě jsou zavěšeny 2D i 3D modely, zvukové efekty, vizuální efekty, šablony a další obsah využitelný při vývoji her. Po zakoupení (některý je však zdarma) a stažení obsahu je možné jej importovat do projektu a ihned tento obsah použít ve vlastní hře.

Celá hra by mohla být vytvořena z objektů, které jsou předem naskládané ve scéně, někdy se však může hodit gameobjecty vytvářet za běhu hry. K těmto účelům v Unity slouží systém polotovarů (*prefabs*). Polotovar je gameobject, který není předem v konkrétní scéně. Za běhu hry může být poté *instancován* (i vícekrát) na skriptem určených místech ve scéně.

Finální export je z Unity možné provést na různé platformy. Při vhodném nastavení lze dosáhnout možnosti exportovat hru pro desktopové operační systémy i pro mobilní zařízení.

2.2.2 Unreal Engine

Vydán již roku 1998 (tedy 6 let před Unity) firmou *Epic Games* jako podpůrný software pro hru *Unreal*. Kromě vývoje samotnými vývojáři v Epic Games umožnil hráčům hry Unreal vytvářet vlastní úrovně, což do té doby nebylo vůbec běžné. Významným milníkem byl projekt Unreal Development Kit z roku 2004[8], který umožnil komukoli zdarma nekomerční vývoj v prostředí Unreal Engine. [15]

V době tvorby tohoto textu nabízí Epic Games tři licenční smlouvy: *Standard license*, *Enterprise program* a *Custom license*. Poslední uvedená – custom licence je individálně na míru a nemá uvedenou cenu. I pro využití korporátního enterprise programu je nutné kontaktovat Epic Games mailem, ale je zde uvedená cena $1 500/rok za jednoho člena týmu. Věta „Flexible terms.“ přímo na internetových stránkách Epic Games naznačuje, že i s uvedenou cenou nebo výhodami by se dalo hýbat. V základu uživatel s enterprise programem získá prémiovou podporu a možnosti soukromého tréninku. [16]

Pro tuto práci se jako nejzajímavější jeví standard license, se kterou je dostupné kompletní vývojové prostředí, propojení s knihovnou 3D scanů *Quixel Megascans* a všechny výukové materiály. S touto licencí je využití Unreal Enginu zpočátku zcela zdarma. Jakmile produkt za celou svou existenci překročí určenou hranici výdělku $1 000 000 amerických dolarů, tak je osoba povinna začít společnosti Epic Games vyplácet 5 % z výdělku[9]. Dosavadní výdělek před překročením této hranice není ovlivněn. Již s touto licencí má vývojář možnost nahlédnou do kompletního zdrojového kódu vývojového prostředí. Dokonce lze zdrojový kód upravovat a případně přispívat přes technologii *git* k vývoji záplat a novějších verzí Unreal Enginu. [17]

Stažení a správu verzí vývojového prostředí Unreal Engine zajišťuje aplikace *Epic Games Launcher* od stejné společnosti. Pokud by při vývoji vyvstala potřeba úpravy Unreal Enginu, tak je také možné naklonovat oficiální git repozitář, provést úpravy a zkompilovat vlastní samostatnou verzi. V tomto textu je předpokládáno využití neupraveného Unreal Enginu verze 5.1. [18]

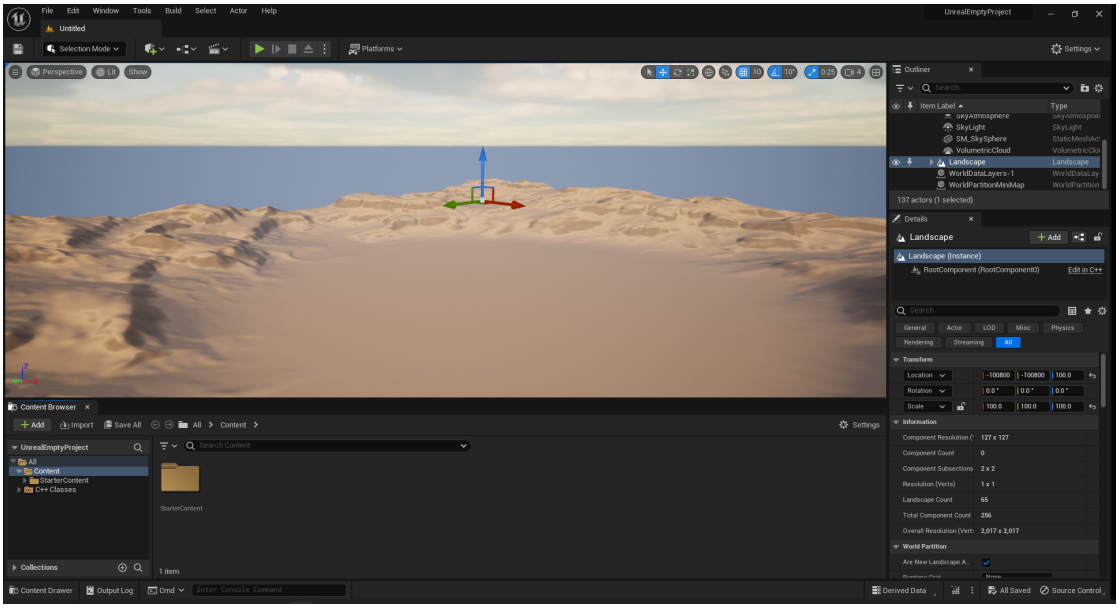
Při vytváření nového projektu je možné vybrat z několika šablon nebo nechat projekt prázdný (*blank*). Mezi výchozím nastavením lze spatřit i zaškrtávací box s popisem *Raytracing*, který umožní podporu trasování paprsků (viz definice 1.12) v reálném čase.

Uživatelské rozhraní vývojového prostředí Unreal Engine 5.1 je vidět na obrázku 2.3. Hlavní součástí je zobrazení vybrané 3D úrovně (*levelu*) v okně uprostřed zvaném *Level Viewport*. Jednotlivé levely se skládají ze základní jednotky typu *actor*. V pravém horním rohu je *Outliner* – hierarchický seznam právě se vyskytujících actorů v levelu. Zeleným tlačítkem nahoře se symbolem ▷ lze kdykoli úroveň spustit přímo v okně Level Viewportu. Pokud je některý actor v Outlineru nebo Level Viewportu vybrán, tak se v pravém dolním rohu v záložce *Details* zobrazí informace

[8] Tedy stejný rok, ve kterém na trh přišlo Unity. Náhoda?

[9] Tento limit se neaplikuje na takzvané *lineární* díla – neinteraktivní filmy, obrázky apod.

Obrázek 2.3 Uživatelské rozhraní Unreal Engine



o actorově pozici ve scéně (tedy jeho *transformu*) a *komponentách*, které jsou k tomuto actorovi připojeny.

V Unreal Enginu lze u actorů vytvářet i hierarchické uspořádání komponent. Komponenty přidávají actorům vlastnosti nebo chování. Může jít o univerzální předpřipravené komponenty například pro světla, povrch, kameru a podobné. Nebo může vývojář actorovi přidat vlastní komponentu v jazyce C++ nebo ve speciálním vizuálním programovacím systému *blueprint*. Vlastní komponenty mohou dědit i z předpřipravených komponent (i přímo z třídy Actor), které jsou taktéž v jazyce C++.

Importování modelů do projektu lze provést přesunutím souborů ve formátu *.fbx* nebo *.obj* do složky projektu *.../[jméno projektu]/Content/*. Formát *.fbx* je preferován, protože dokáže zachovat více informací o modelu. Vývojové prostředí následně uživatele upozorní na změny ve zdrojových souborech a umožní import modelů do projektu.

Uživatelské rozhraní Unreal Enginu je přímo propojené s databází *Quixel Megascans* skrze *Quixel Bridge*. Quixel Megascans je neustále se rozrůstající databáze 3D skenů prostředí, textur, modelů a dalšího obsahu. Pokud je obsah z Quixel Megascans využit v prostředí Unreal Enginu, tak je všechen kompletně zdarma [19]. Stačí obsah najít, stáhnout a přidat do projektu. Jako další možný zdroj obsahu může vývojářům sloužit internetový obchod *Unreal Marketplace*, kde jsou zavěšené nejen modely, ale také zásovné moduly, blueprinty a další zdroje určené konkrétně pro Unreal Engine.

V nastavení projektu lze vybrat z více cílových platforem zahrnujících PC, mobilní zařízení a konzole staré i nové generace.

2.2.3 Porovnání Unity a Unreal Engine

Herní enginy Unity a Unreal Engine, ač vyvíjeny nezávisle, mají některé prvky společné. A přestože se občas nazývají jinak, mají stejnou funkcionalitu. Například na obrázku 2.4 lze vidět porovnání uživatelského rozhraní. Okna s jejich respektivními názvy, která se starají o podobné části aplikace, jsou zvýrazněny stejnou barvou. Dále v textu budou porovnávány hlavní rozdíly mezi těmito dvěma vývojovými prostředími.

Obrázek 2.4 Porovnání uživatelského rozhraní Unity a UE [20]

Počínaje cenou má Unreal Engine benevolentnější licenční podmínky. Již ve verzi zdarma poskytuje téměř všechny nástroje. Teprve po překročení relativně vysoké hranice výdělku $1 000 000 je osoba povinna začít vyplácet 5 % z výdělku. Jde však o celkový výdělek za celou dobu existence projektu. Oproti tomu Unity po uživatelích požaduje přechod na vyšší licenční úroveň pokud jejich aplikace překročí $100 000 výdělku za posledních 12 měsíců. Pokud se tedy jedná o hru, která dlouhodobě generuje dostatečně malé množství výdělku, tak teoreticky může fungovat neomezeně dlouho zdarma. Procentuální honorář u Unreal Enginu má také výhodu, že dle výdělku projektu dobře škáluje, zatímco Unity s fixními cenami se může ve větších týmech prodražit. S Unity, pokud si vývojář nezaplatí alespoň $2 040/rok, tak se nedostane ke zdrojovému kódu vývojového prostředí. Unreal Engine zdrojový kód poskyje jakémukoli registrovanému uživateli *zdarma*. Základní verze Unity zdarma také neobsahuje některé vývojářské balíčky, které jsou přítomné ve vyšších verzích. Unreal Engine ve vyšších verzích poskytuje navíc pouze zákaznický servis a možnost soukromých tréninků. Pro velké vývojářské korporace jsou u obou softwarových řešení šity licenční podmínky na míru, takže v těchto případech nelze UE a Unity objektivně zhodnotit[10].

Instalace je v obou případech obdobná – Unreal Engine skrze Epic Games Launcher a Unity skrze Unity Hub. Pro Unity hovoří fakt, že lze stáhnout samostatnou aplikaci, kdežto Unreal Engine (pokud člověk nechce/nemůže využít Epic Games Launcher) lze pouze kompilovat ze zdrojových souborů.

Herní engine Unity také nabízí kompletnější podporu, návody a šablony pro tvorbu 2D her. Pro 3D prostředí jsou perfektně uzpůsobená obě vývojová prostředí. Výchozí nastavení v UE působí lepší grafickou kvalitou za cenu vyšších nároků na hardware vývojáře i hráče. Tato nastavení lze samozřejmě libovolně měnit, ale může to být nenápadnou nápovědou, pro jaké projekty jsou respektivní enginy určeny.

Základní pojmy se mezi enginy dají porovnat takto:

Tabulka 2.1 Porovnání pojmů Unity a Unreal Engine

Unity	Unreal Engine
Scene	Level
GameObject	Actor
Component	Component
Prefab	Blueprint Class

[10] Co se týče ceny.

Výrazný rozdíl mezi enginy je různý skriptovací jazyk komponent: C# pro Unity, C++ pro Unreal Engine. Výhodou UE je možnost dědění třídy Actor herního enginu. Lze tak přidat vlastnosti objektu napřímo. Unity toto s třídou GameObject neumožňuje. Další výhodou UE týkající se komponent je možnost jejich *hierarchického* uspořádání. To může u větších projektů značně zpřehlednit jednotlivé actory. [20]

Import modelů je v obou enginech preferován ze souborů *.fbx*. Unity nabízí import i souborů uložených softwarem Blender ve formátu *.blend*. Vnitřně však Unity stejně konvertuje tyto soubory do formátu *.fbx*.

Cílová platforma u obou enginů může být takřka libovolná. Pokud ovšem vývojář chce dostat svou hru na herní konzole typu PlayStation, Nintendo Switch nebo Xbox, tak si v případě Unity musí výrazně připlatit za vyšší úroveň licence.

Přesto, že předchozí text by mohl vyznívat, že Unreal Engine je téměř ve všem lepší, nelze při výběru zapomínat na těžko kvantifikovatelné a subjektivní faktory. Jedním z těchto faktorů je *komunita*. Unity vždy cílilo spíše na menší vývojářské týmy [11]. Tím si okolo sebe vytvořilo aktivní komunitu vývojářů, kteří na internetových fórech odpovídají na dotazy začátečníků. Implementační část této práce je provedena v herním enginu Unity. Rozhodujícím byla právě lepší komunitní podpora, která začátečníka lépe uvede do světa tvorby počítačových her a přehlednější oficiální dokumentace, která obsahuje praktické příklady. Nedostatky Unity oproti Unreal Enginu nejsou v tomto případě relevantní.

2.3 Analýza existujících her se strojovým učením

Nápad využití strojového učení ve světě her není revoluční. Dále budou uvedeny vybrané příklady her, u kterých bylo v nějaké formě využito strojové učení.

2.3.1 Šachy

Prvním zástupcem jsou šachy. Přestože informatikům dobře známý americký matematik John von Neumann pravil:

> *„Šachy nejsou hra. Partie šachů je přesně definovaná forma výpočtu. Možná nedokážeme přijít na všechny odpovědi, ale teoreticky musí existovat řešení, správný postup v jakékoli pozici.“* [21]

Šachy v tomto textu za hru pokládány budou. Hlavním rozdílem oproti velké většině ostatních her je absence elementu náhody v partii šachů. Historie zlepšování umělé inteligence v této prastaré hře je obsáhlá, tato práce rozebere fungování open-source šachového enginu *Leela Chess Zero*.

Leela Chess Zero (často také zkráceně *Lc0*) funguje na bázi prohledávání *stromu* herních *stavů*. Každý stav je jeden *uzel* v tomto stromu s odhadem jeho *hodnoty* a seznamu dalších možných tahů seřazených dle *priority*. Tradiční šachové enginy disponují člověkem precizně nastaveným systémem generování těchto priorit. Lc0 využívá ke generování priorit a hodnoty pozice hlubokou neuronovou síť trénovanou bez lidských vědomostí. Když je tah zahrán, jeho uzel se stane novým kořenem stromu. Předchozí kořen a ostatní větvě jsou smazány. Neuronová síť využívá algoritmus prohledávání stromu Monte Carlo. [22]

Lc0 má schopnost držet krok s konkurenčními konvenčními[11] šachovými enginy typu Stockfish, které se stabilně pohybují na nejvyšších příčkách žebříčku nejlepších šachových enginů. [23]

[11] Konvenčními – bez strojového učení.

2.3.2 StarCraft II

StarCraft II od společnosti Blizzard je pokračování dřívější hry StarCraft. Jedná se o strategické hry v reálném čase. Stejně jako u šachů je zde využíváno strojové učení za účelem vytvoření umělé inteligence, která bude simulovat hráče.

Společnost Blizzard spojila síly s firmou DeepMind a vytvořila ve hře integrované výukové prostředí pro umělou inteligenci (*StarCraft II Learning Environment*). V tomto prostředí se zvláště naskytuje příležitost trénovat posilovací model strojového učení. Každý hráč ovládá stovky jednotek, které musí spolupracovat, aby dosáhli společného cíle. Jde také o hru s nekompletní informací. Mapa je viditelná pouze na místech, kde hráč má jednotky. Aby zjistil stav protivníka, tak musí aktivně objevovat mapu. Počet možností, které hráč může provést, se v průběhu hry mění s technologickým postupem hráče. A v neposlední řadě hry trvají tisíce snímků a hráč musí dělat brzká rozhodnutí, která mohou mít viditelný dopad až mnohem později v průběhu hry. Veřejnosti dostupný je také dataset, který obsahuje přes milion záznamů lidských hráčů. [24]

Toto prostředí umožňuje vývojářům čerpat data přímo z herních proměnných[12]. Výstup algoritmu strojového učení lze přímo předávat hře v podobě volání funkcí rozhraní výukového prostředí.

Jednou z hlavních výzev pro vývojáře je nastavení *odměn* pro posilovací učení tak, aby umělá inteligence dokázala akeptovat *zpožděné* odměny – odměny, které se projeví až po časové prodlevě od provedené akce, která získání odměny způsobila.

2.3.3 Doom

První díl hry Doom je jedna z prvních třírozměrných FPS her. Zdrojový kód je v současnosti již veřejnosti dostupný, takže by vývojář umělé inteligence pro hru Doom mohl (stejně jako třeba u Starcraft II) do algoritmu posilovacího učení vstupní data čerpat přímo z proměnných upravené hry.

Existuje však výzkumná platforma pro posilovací učení *ViZDoom*, která vývojářům poskytuje pouze data vyrovnávací paměti obrazovky. Tým stojící za platformou ViZDoom pořádá i soutěže nejlepších algoritmů umělé inteligence pro tuto hru. [25]

Vítězem této soutěže v kategorii známých map pro rok 2016 se stala umělá inteligence se jménem *F1* [26].

F1 k poražení konkurenčních botů v bitvě všichni proti všem použila konvoluční neuronovou síť. Jako řešení problému zpožděné odměny byla zvolena strategie menších odměn v průběhu celé hry – pozitivní ohodnocení sbírání předmětů a negativní ohodnocení použití munice a ztráty životů. Pro zrychlení učení bylo také využito *curriculum learning*, které trénuje neuronovou síť na setu progresivně obtížnějších prostředí. [27]

V roce 2018 dokázal algoritmus *TSAIL* celou hru dohrát za méně než 26 minut [28]. Toto lze porovnat s běžným lidským průchodem hry, který činí okolo 5 hodin [29].

2.3.4 Galactic Arms Race

Aby čtenář této práce nenabyl dojmu, že strojové učení lze využívat v počítačových hrách pouze k simulovanému ovládání protivníka, je zde rozebrána hra *Galactic Arms Race* (dále GAR). Tato hra se odehrává ve vesmíru, kde spolu soupeří vesmírné lodě ve střeleckých bitvách.

Vývojářské studio *Evolutionary Games* se pro svou hru v roce 2009 rozhodlo využít strojové učení. Nevyužilo jej však k navigaci jednotek, ale ke *generování* herního obsahu. V průběhu hry jsou v reálném čase pro hráče generovány zcela nové zbraně na základě jeho preferencí skrze *neuroevoluci*. Neuronová síť geneticky kóduje a stará se o systém částicových zbraní. Postupem

[12] Samozřejmě jenom z těch, které jsou pro konkrétního hráče ve hře dostupné.

čaasu získává NS na komplexitě skrze dále popsanou metodu *cgNEAT*. Tímto samotný hráč určuje typ obsahu, který zaplní hru na základě jeho preferencí. [30]

Hlavní principy cgNEAT dle vývojářů GAR jsou:

- Každá unikátní zbraň je reprezentován jednou NS
- Fitness je NS přiřazena na základě doby používání zbraně
- Obsah je generován do herního světa, kde může být hráčem sebrán
- Evoluce probíhá následovně:
  1. Některé zbraně, které hráč již používá jsou vybrány jako *rodiče* (častěji používané mají vyšší pravděpodobnost být vybrány)
  2. Proběhne reprodukce zahrnující mutaci a křížení
  3. Šance na zvýšení komplexity NS
- Pro jakýkoli generovaný obsah existuje šance, že bude vybrán z předem evolvovaného obsahu, který se hráčům v minulosti líbil. [30]

Tímto může hra dosáhnout vysoké hodnoty opakovaného hraní, které se nestane repetetivní. Hráč také dostane herní zážitek, který se přizpůsobí jeho hernímu stylu, takže hra je svým způsobem „multifunkční“ a klesají náklady na ruční výrobu obsahu.

Kapitola 3
Praktická část

Praktické vypracování hry – návrh, implementace a testování.

3.1 Návrh prototypu hry

Základní myšlenka implementované hry je založena na snaze co nejnázorněji prezentovat postupné strojové učení. Je tedy třeba zvolit dostatečně rychlý algoritmus, který dokáže vstupy vyhodnocovat ve velmi krátkém čase (i vícekrát za snímek obrazovky). Viditelnost průběhu, zlepšování algoritmu a eliminace chyb je lépe vidět na současném běhu více strojově naučených algoritmů zároveň. Tyto algoritmy mohou mít v konečném důsledku velmi jednoduchý úkol, jako například vyhrát hru, porazit hráče nebo zničit konkrétní budovu. Jakým způsobem to provedou je v rámci možností hry jejich „svobodná“ – strojově naučená vůle. Tyto požadavky dobře nahrávají tomu, že by se takový algoritmus dal využít u početnějších jednotek nepřítele. Zároveň se tímto elegantně vyřeší postupné zvyšování náročnosti pro hráče. Na začátku hry jsou nepřátelské jednotky nenatrénované. Postupně se však začnou učit a dokáží dokonce lépe reagovat na hráčovy snahy o obranu.

Při návrhu je také zapotřebí myslet na to, aby se hra dala vhodně realizovat ve třírozměrném prostoru.

Jde tedy o velký počet nepřátel a strategické rozmisťování prvků, proti kterým mohou nepřátelé využívat možnosti strojového učení. Dobrým kandidátem je hra typu *tower defense*.

3.1.1 Konkretizace

Nyní lze tyto myšlenky konkretizovat do podoby návrhu prototypu hry. Hráč bude reprezentovat obrannou stranu s cílem přežít co nejdéle. K tomu mu pomohou spřávně rozmístěné věže, které různými způsoby budou likvidovat nepřátelské jednotky. Útočná nepřátelská strana bude na herní plochu nasazovat vlny jednotek, které se samostatně budou pohybovat podle svého strojově naučeného algoritmu. Pokud se některé nepřátelské jednotce podaří proniknout hráčovou obranou vrstvou a narazit do jeho báze, tak bude báze poškozena. Po určitém množtsví takového poškození bude báze zničena a hráč v této chvíli prohraje.

Je nutno podotknout, že hra je teoreticky nekonečná, protože obranná strana nemá šanci vyhrát. Výsledek je porovnatelný pouze podle délky času přežití – čím déle hráč přežije, tím lepší zvolil obrannou strategii umisťování věží.

Vzhledem k počtu nepřátel se bude hodit, pokud jejich reprezentace v 3D grafickém prostoru bude co nejefektivnější. S přihlédnutím k jednoduchosti implementace kolizí byl zvolen tvar kostky.

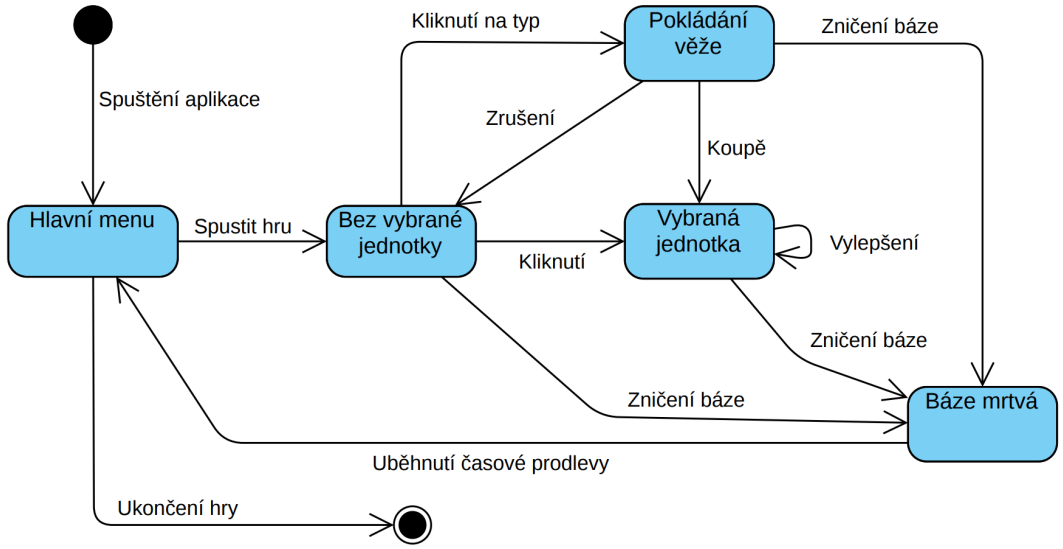
3.1.2 Stavový diagram

Z hlavního menu se hráč dostane do hlavní herní smyčky. Podle dosupných herních financí může na herní plochu stavět různé typy věží. Pokud na nějakou jednotku klikne, zobrazí se mu menu s informacemi o jednotce. Věže mohou navíc v tomto menu mít tlačítko, které věž vylepší.

Když vlny nepřátelských jednotek uberou bázi všechny životy, tak hra vypíše přežitý čas a vrátí hráče zpátky do hlavního menu.

Přechody mezi stavy aplikace se dají vyjádřit diagramem 3.1.

Obrázek 3.1 Stavový diagram



V diagramu nejsou nijak vyjádřeny vlny nepřátel, protože ty se objevují periodicky po celou dobu běhu hry.

Tento stavový model by měl být pro uživatele jednoduchý na pochopení. Samotný hráč samozřejmě se stavovým diagramem při hraní vůbec nepřijde do kontaktu a ani nemusí vědět, co takový stavový diagram znamená. Změny stavů by měly být pro uživatele intuitivní a snadno proveditelné.

3.1.3 Diagram tříd

Diagram tříd lépe ilustruje fungování hry vevnitř.

V diagramu jsou metody, které používá Unity (*Awake*, *Start*, *Update*, atd.), vynechány.

Celý diagram tříd je obsažen v příloze, zde budou rozebrány pouze jeho zajímavější části.

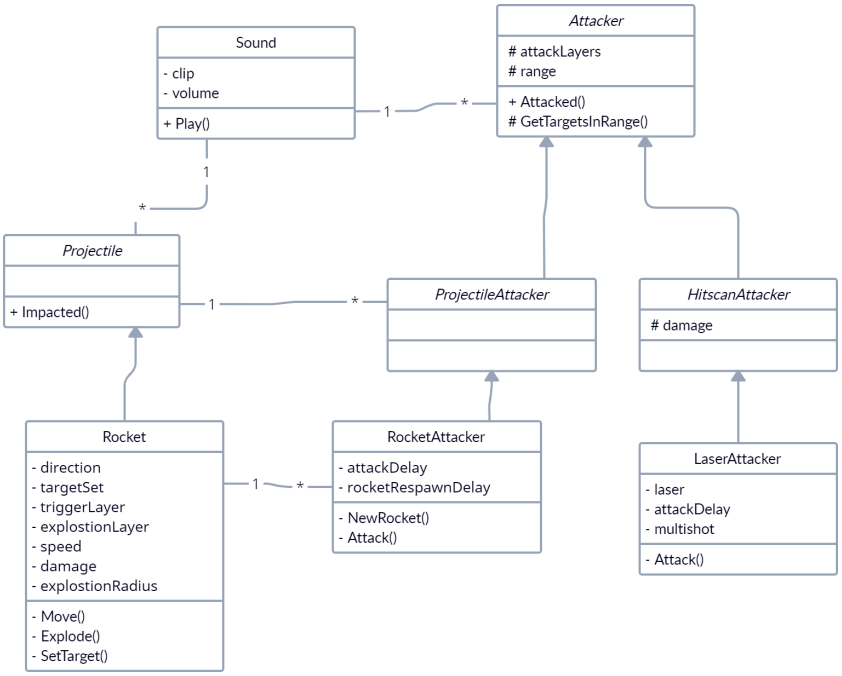
Třída *Attacker* je pouze virtuální, dědí z ní všechny třídy, které jsou následně přidány jako komponenty obranným věžím[1].

Přímo z třídy Attacker dědí virtuální třída *ProjectileAttacker*, která zastřešuje věže, které střílí projektily s nenulovou dobou letu. Asociací se k ní tedy řadí třída *Projectile*, která reprezentuje vystřelený projektil.

Již neabstraktní konkrétní dědic třídy ProjectileAttacker je *RocketAttacker*, který je komponentou obranné věže, která vrhá projektily *Rocket*. Tato věž střílí v intervalech vyjádřených

[1] Třída Attacker (útočník) u *obranných* věží se může zdát jako protimluv, avšak věže opravdu útočí na nepřátelské jednotky.

Obrázek 3.2 Diagram tříd – Attacker



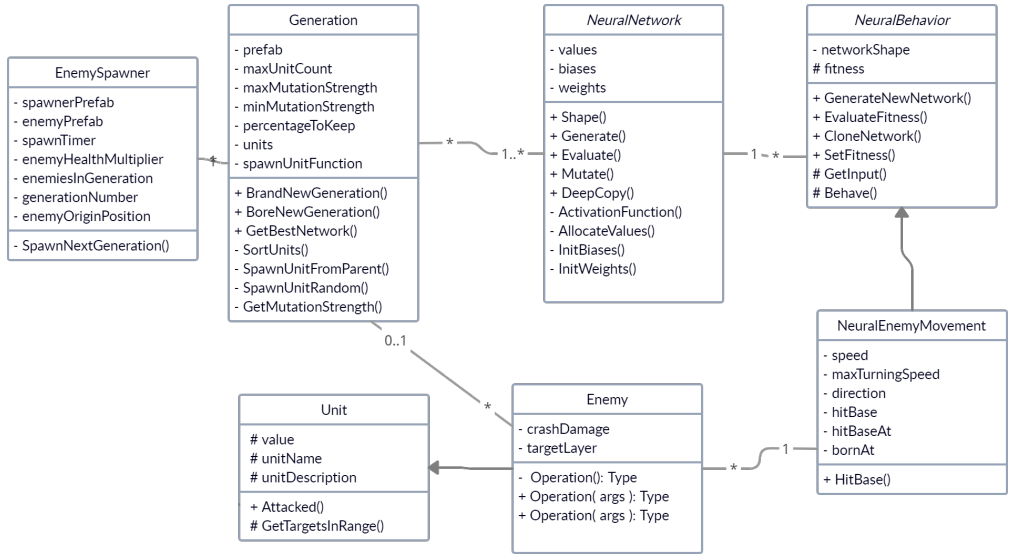
proměnnou *attackDelay*. Proměnná *damage*, která určuje hodnotu poškození, však nemá přímo věž, ale projektil Rocket, který také disponuje proměnnými *direction*, *speed* nebo *explosionRadius*.

Třída *HitscanAttacker* reprezentuje skupinu věží, které zasáhnou cíl ihned. Zde má tedy smysl proměnná poškození – *damage* přímo. Zástupce této skupiny je *LaserAttacker*, což je věž která generuje laserový paprsek. Může zasáhnout více nepřátel najednou dle proměnné *multishot*.

Zvukové efekty jednotlivých věží a projektilů jsou realizovány třídou *Sound*.

Struktura tříd je koncipována tak, aby mohla být jednodušše rozšiřitelná o další typy věží.

Obrázek 3.3 Diagram tříd – Strojové učení nepřátel



Na obrázku 3.3 je možné vidět diagram tříd použitý pro nepřátelské jednotky a jejich management.

Základem strojového učení je třída *NeuralNetwork*, jejíž instance reprezentuje jednu konkrétní neuronovou síť. Vytvoření nové neuronové sítě se dá provést dvěma způsoby: generováním úplně nové s náhodnými váhami pomocí metody *Generate* nebo vytvoření hluboké kopie již existující neuronové sítě pomocí metody *DeepCopy*. tato hluboká kopie se dá následně mutovat metodou *Mutate*. Pro vstup se provede výpočet výstupu metodou *Evaluate*.

Abstraktní třída *NeuralBehavior* je základem pro cílené využití neuronové sítě. Může tedy vyhodnocovat její výkon – *fitness*.

Implementovaným dědicem NeuralBehavior je *NeuralEnemyBehavior*, který již dokáže reagovat na konkrétní vstupy a dle nich vracet výstup, který řídí jednu nepřátelskou jednotku *Enemy*.

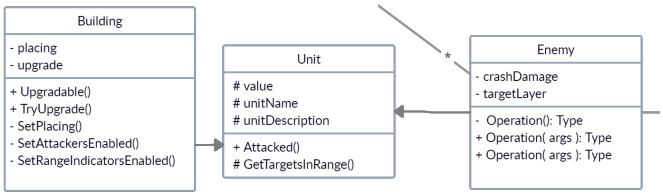
Sada nepřátelských jednotek tvoří jednu generaci – *Generation*. Pokud se nejedná o úplně první generaci, tak obsahuje pole nejlepších neuronových sítí z minulé generace, které náhodně mutuje a využívá ke generování jednotlivých nepřátel aktuální generace.

Třída *EnemySpawner* slouží jako manažer jednotlivých generací.

Tato struktura je s pár drobnými změnami aplikovatelná pro navigaci a chování téměř jakékoli jednotky ve hře. Bylo by možné mít například hráčovy obranné jednotky, které by se zlepšovaly strojovým učením.

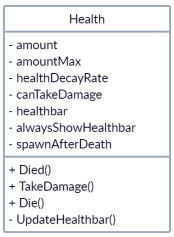
Jak je na obrázcích 3.3 a 3.4 vidět, třída Enemy dědí ze třídy *Unit*. Komponentu, která dědí z jednotky Unit obsahují téměř všechny herní jednotky. Proměnná *value* určuje peněžní hodnotu jednotky, *unitName* její název a *unitDescription* její popis. Společně tyto proměnné dávají ucelenou základní informaci o jednotce. Dědic třídy Unit hráčových věží se jmenuje *Building*.

Obrázek 3.4 Diagram tříd – Unit



Třída Building nese navíc informace o případném vylepšení a možnost zobrazení indikátoru dosahu.

Obrázek 3.5 Diagram tříd – Health



Významná je také třída životů *Health* (obrázek 3.5).

Třída nese kromě aktuální hodnoty životů také například informaci o tom, jestli zobrazovat ukazatel životů v uživatelském rozhraní nebo jestli se po její smrti má zrodit nějaká jiná jednotka.

Při návrhu všech tříd bylo myšleno na případné budoucí možnosti rozšíření.

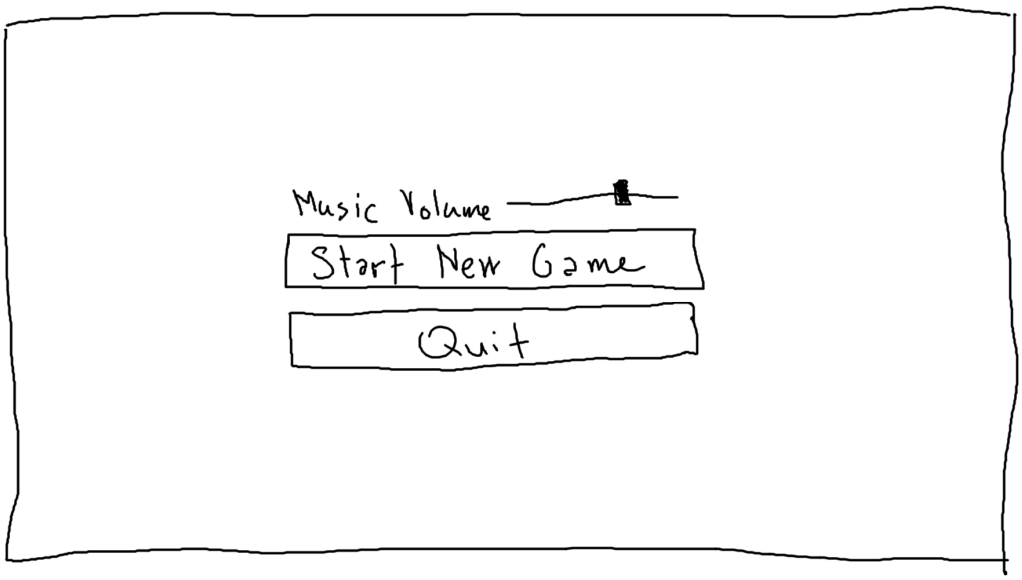
3.1.4 Uživatelské rozhraní

Uživatelské rozhraní by mělo být přehledné a snadno ovladatelné.

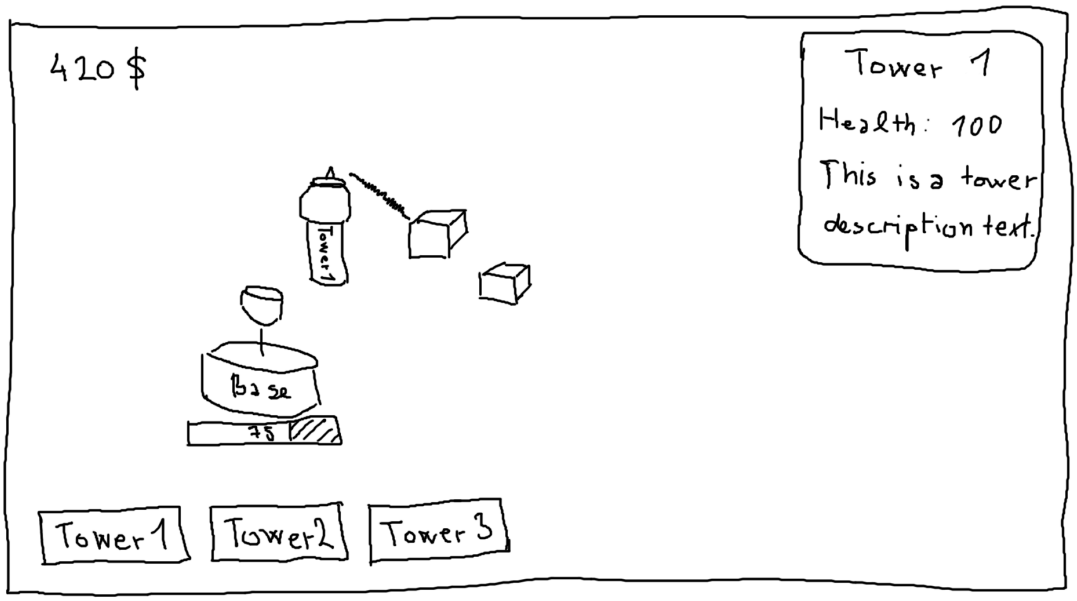
Hlavní menu (obr. 3.6) se teda skládá pouze z pár prvků. Jmenovitě posuvník *Music Volume* dovoluje hráči nastavení hlasitosti hudby, tlačítko *Start New Game* nastartuje novou hru a tlačítko *Quit* aplikaci vypne.

V návrhu uživatelského rozhraní běžící hry na obrázku 3.7 jsou v levém horním rohu jsou vyobrazeny hráčovy finance. V levém dolním rohu jsou tlačítka, pomocí kterých může hráč stavět

Obrázek 3.6 Návrh uživatelského rozhraní hlavního menu



Obrázek 3.7 Návrh uživatelského rozhraní hry



věže. Při výběru nějaké jednotky se v pravém horním rohu zobrazí informace o této jednotce. Hlavní hráčova báze pod sebou má ukazatel životů.

Uživatelské rozhraní je co nejjednodušší, aby hráč nebyl přehlcen různými menu, texty a tlačítky. Může se takto soustředit hlavně na herní plochu, kde se pohybují nepřátelské jednotky a jeho obranné věže po nich střílí.

3.2 Implementace hry

Implementace hry je provedena v herním enginu Unity. V tomto herním enginu je většina herních prvků (jednotky, kamery, světla, prostředí, atp.) reprezentována herním objektem *gameobject*. Ke gameobjectům se dají připojit *komponenty* – přednastavené algoritmy nebo vlastní skripty, které ovlivňují chování gameobjectů. Sady gameobjectů se shlukují do *scén*. Hra má dvě scény: hlavní menu a samotnou hru.

Hlavní menu slouží pro nastavení hlasitosti zvuku, startu nové hry a ukončení hry. Při započetí nové hry se hráči začnou počítat peníze v dolarech ($). Jde však spíš o intuitivní nápovědu, že jde o měnu. Na reálné dolary nemá žádnou vazbu.

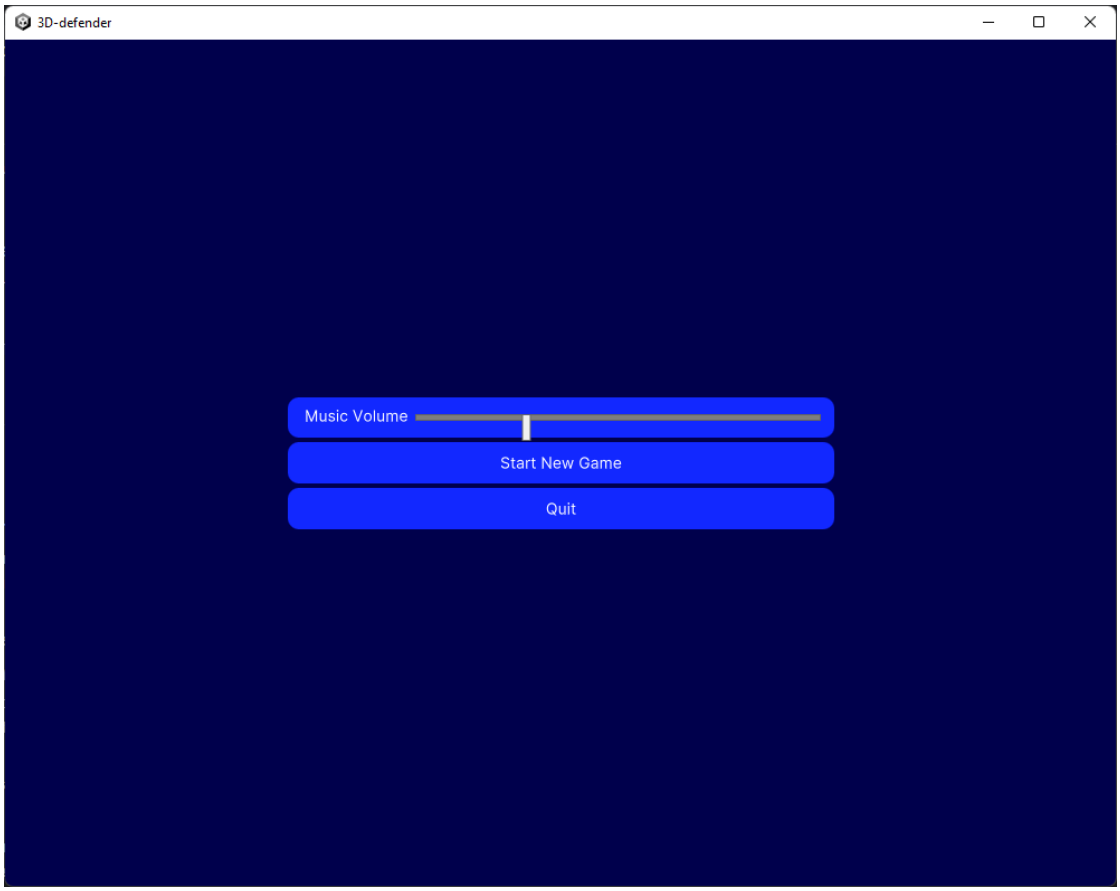
Hra je rozdělená do hlavních tří barev. Zelená znamená neutrální prostředí – herní plochu. V odstínech modré jsou uvedeny hráčovy obranné prvky a báze. Červená je rezervovaná pro nepřátelské jednotky. Díky tomuto rozdělení může hráč rychlým pohledem zjistit stav hry, protože nemusí zkoumat konkrétní typy jednotek, stačí barva.

3.2.1 Uživatelské rozhraní a ovládání

Uživatelské rozhraní je implementováno dle návrhu z kapitoly 3.1.4. Ovládání hry je v následující kapitole. Pro rozumné rozlišení (800x600 až 4k) se rozhraní chová responzivně. Okno lze zvětšovat a zmenšovat bez vlivu na hratelnost.

Hlavní menu je možné vidět na obrázku 3.8.

Obrázek 3.8 Hlavní menu



Stisknutím tlačítka *Start Game* se hráč dostane přímo do herní scény, kde začne novou instanci hry. Tlačítko *Quit* hru ukončí.

Obrázek 3.9 Výchozí nastavení hry a kamery v ní

Celkově se hráč na hru dívá z pohledu hlavní kamery. Kamera je perspektivní zobrazení. Tím, že se bližší objekty na obrazovce vykreslují ve větší velikosti, tak hra lépe vytváří dojem třírozměrného prostoru. Ortografická kamera je v počítačových strategických hrách také běžná, ale často takový způsob vykreslování dělá hru téměř dvourozměrnou (a některé počítačové strategie doopravdy 2D jsou). Z uvedených důvodů tedy byla zvolena kamera perspektivní, ale možnosti rozšíření jsou probrány v 3.2.9. Pokud hráč stiskne pravé tlačítko myši, tak se pohybem myši může rozhlížet do stran a nahoru – dolů. Posun kamery přes herní plochu se provede stisky kláves *W*, *A*, *S*, *D*. Do výšky se kamera posune stiskem mezerníku, dolů stiskem klávesy Shift. Rozložení kláves je udělané tak, aby hráč mohl ovládat všechny pohyby kamery myší a jednou rukou na klávesnici současně.

Kromě hlavního pohledu kamery jsou na obrazovce zobrazeny informační a ovládací prvky hry. V levém horním rohu je zobrazen hráčův finanční stav, který se automaticky aktualizuje na aktuální hodnotu.

V levém dolním rohu je menu věží. Pro každý základní typ věže je vygenerováno tlačítko, které nese název věže a její cenu. Základním typem věže jsou myšleny veže, které lze postavit rovnou – nejsou vylepšením nějaké jiné. Když hráč některé z těchto tlačítek stiskne, tak se mu pod kurzorem myši zobrazí poloprůhledný model odpovídající věže. Nyní ještě věž není postavená a je neaktivní, jde pouze o plánování stavby. Také se kolem věže při plánování stavby objeví poloprůhledný indikátor dosahu věže. Uživatel si může pohybem myši vybrat, kam by chtěl tuto

věž postavit. Když je s lokací věže na herním plánu spokojen, tak levým tlačítkem myši potvrdí stavbu. Pokud má na stavbu této věže peníze, tak se věž zneprůhlední a začne být aktivní. Pokud si hráč v průběhu plánování stavbu věže rozmyslí (například zjistí, že mu nevyhovuje dosah věže), tak může stavbu zrušit stisknutím pravého tlačítka myši.

Když uživatel zrovna není ve stavu plánování stavby, může stisknutím levého tlačítka myši vybrat libovolnou jednotku na herní ploše. Je vhodné poznamenat, že lze vybrat i nepřátelskou jednotku. V pravém horním rohu se zobrazí panel s informacemi o vybrané jednotce. Jde o název jednotky, její množství života a popis. Pokud hráč vybral věž, která má možnost vylepšení, tak se na panelu objeví i tlačítko *Upgrade* s cenou vylepšení. Jak je asi zřejmé, kliknutím na toto tlačítko dojde v vylepšení věže výměnou za určitý finanční obnos hráčových peněz.

Uprostřed obrazovky se při vybraných událostech objevují informativní textové hlášky. Události zahrnují například zrození nové vlny nepřátel nebo nedostatek finančních prostředků pro stavbu nebo vylepšení věží. Text po časové prodlevě zmizí.

Podstatnou součástí uživatelského rozhraní je také indikátor životů hlavní báze. Pokud má hráč v záběru hlavní kamery svou bázi, tak se pod bazí vykreslí ukazatel jejího aktuálního zdraví. Hráč by sice mohl získat detailní informace o zdraví báze na výběrovém panelu v pravém horním rohu, ale pro rychlou orientaci tento indikátor stačí.

Všechny textové prvky uživatelského rozhraní mají unifikovaný font. Vzhledem k tomu, že většina prvků má krátké textové popisky, byl zvolen bezpatkový font, který vypadá úhledněji, ale nehodil by se pro delší texty. Pokud by se v budoucnu ve hře vyskytovaly delší texty, tak by mohl být zvolen patkový font, který lépe vede oči po řádku.

Většina uživatelského rozhraní je v odstínech modré.

3.2.2 Logika hry

Hra začíná ve chvíli, kdy uživatel v hlavním menu stiskne tlačítko *Start New Game*. Na výchozí herní ploše je uprostřed umístěná báze s 500 jednotkami života a s vybraným rozestupem dva levitující nepřátelské talíře, pod kterými se ve vlnách rodí jednotlivé nepřátelské jednotky. Po zvolené časové prodlevě 20 sekund se pod každým nepřátelským talířem zrodí 50 nepřátelských jednotek.

Nepřátelské jednotky mají za úkol dostat se do středu hrací plochy a nabourat do hráčovi báze. Navigovány jsou algoritmem, který je popsán v 3.2.3. Mají 100 jednotek života.

V tomto stavu mu nepřátelské jednotky do pár desítek sekund (maximálně minut) bázi zničí a hráč prohraje. Na obranu proti nepřátelským jednotkám tedy hráč staví obranné věže, které na nepřátelské jednotky útočí.

Ve chvíli kdy je implementace odevzdávána, má hra 3 základních typy věží. Tyto typy jsou: *Tall Tower*, *Cubic Tower* a *Rocket Tower*. Všechny typy mají ještě svá vylepšení. Možnosti rozšíření v 3.2.9.

Pokud se nepřátelská jednotka dostane do dosahu obranné věže, tak obranná věž periodicky zahájí útok na tuto nepřátelskou jednotku.

Tall Tower neboli Vysoká věž stojí 100 $ a je specializovaná na vysoký dosah s velkou ničivou sílou. Střílí paprsek – laser, který zasáhne nepřátelskou jednotku ihned poté, co věž vystřelí. Její nevýhoda je relativně velká perioda útoku a malé množství zasažených nepřátel. Tall Tower má dvě postupné vylepšení *Taller Tower* a *Tallest Tower*, které mají větší dosah a silnější laser.

Cubic Tower neboli Kubická věž stojí 250 $ a také střílí laser. Může oproti Vysoké věži zasáhnout více jednotek najednou, ale má znatelně menší dosah, takže pokud se jí nepřátelské jednotky vyhnou, tak je hráčovi k ničemu. Postupné vylepšení věže *Magic Tower* a *Doom Tower* mají silnější laser a mohou zasáhnout ještě více nepřátel najednou.

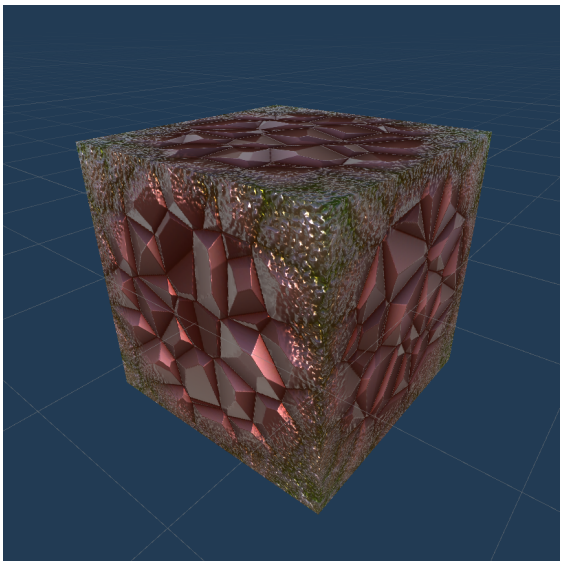
Rocket Tower neboli Raketová věž stojí 450 $ a střílí rakety. Rakety nezatáčejí a vybuchnou teprve po nárazu do nepřátelské jednotky nebo do země. Přesnost je u této obranné věže tedy menší, avšak výbuch rakety zasáhne všechny jednotky v určitém poloměru. Na masové ničení nepřátel vysoce efektivní, ale tomu odpovídá také cena věže. Jediné vylepšení je *Atom Tower*, které

vrhá atomové bomby. Atomové bomby jsou pomalejší, avšak jejich ničivá síla je bezkonkurenční.

3.2.3 Nepřátelské jednotky

Scéna hry obsahuje dvě komponenty třídy *EnemySpawner*, které pod přednastaveným modelem talíře na herní plán generuje vlny nepřátelských jednotek v podobě kostek (viz 3.10) – dvě nezávislé *generace*. Vlna se generuje vždy po jedné jednotce za snímek obrazovky, aby se výpočetní zátěž trochu rozložila. Pokud by se všechny nepřátelské jednotky vygenerovaly najednou, tak zvláště na slabších počítačích by bylo znatelné *trhnutí* – náhlý pokles ve snímcích za sekundu. Každá nepřátelská jednotka má svou vlastní kopii neuronové sítě implementované dle návrhu na obrázku 3.3.

Obrázek 3.10 Nepřátelská jednotka



Neuronová síť – třída *NeuralNetwork* vnitřně obsahuje tři souvislá pole: pro váhy (*weights*), odchylky (*biases*) a samotné hodnoty (*values*) jednotlivých neuronů. Díky tomu, že jsou pole souvislá, tak je přístup k nim velmi rychlý. Nad touto strukturou se provádí každý snímek tisíce operací, takže je snaha o co nejvyšší možnou optimalizaci. Neuronová síť může mít libovolný tvar a hloubku. Jediným omezením je samozřejmě minimálně jedna vstupní a jedna výstupní vrstva. Při vytvoření nové neuronové sítě je třeba inicializovat výše zmíněná tři pole. Pokud je vytvářena úplně nová neuronová síť (bez předka), tak jsou váhy a odchylky inicializovány generátorem náhodných čísel[2] v intervalu $\langle -0,5; 0,5 \rangle$.

Když neuronová síť předka má, tak se pomocí funkce *DeepCopy* předkova neuronová síť zkopíruje. Následně jsou jednotlivé váhy a odchylky *mutovány* – náhodně modifikovány. Síla mutace je určená parametrem. Volba tohoto parametru je popsána dále.

Dopředný chod neuronové sítě zajišťuje funkce *Evaluate*, která pro každou vrstvu vypočítá vrsvu následující počínaje vstupní vrstou, konče vrstvou výstupní. Výpočet pro jeden neuron N je proveden sečtením neuronů v předcházející vrstvě násobených jejich váhami pro tento neuron N. Poté je přičtena ještě odchylka. Závěr výpočtu sestává z aplikace *aktivační funkce*.

Jako aktivační funkce byla zvolena *Leaky ReLU* (dle 2.1). Vybrána byla především pro svou rychlost výpočtu. Přednost před klasickou ReLU dostala kvůli tomu, že na výsledek mají vliv i různě velké negativní hodnoty. Pro navigační systém je vhodné, pokud i záporné hodnoty (například souřadnic) mohou ovlivnit výsledek. Jako funkce pro výstupní vrstvu byla zvolena hyperbolická funkce tangens[3], protože výstup je později interpretován na intervalu $\langle -1; 1 \rangle$.

Výše popsaná třída NeuralNetwork zajišťuje pouze nízkoúrovňové výpočty neuronové sítě a je využitelná pro téměř jakoukoli aplikaci evoluční neuronové sítě. Obal na tuto třídu nazvaný *NeuralEnemyMovement* již dokáže zpracovávat vstupy, podle výstupů navigovat jednotku přes herní pole a vyhodnocovat *fitness* neuronové sítě. Má tedy specializované použití jako Unity komponenta pro navigaci jednotlivých nepřátelských jednotek. Na vstupu neuronové sítě jsou 4 složky:

- Aktuální pozice nepřátelské jednotky na ose X

[2] Generátor náhodných čísel poskytuje přímo herní engine Unity.
[3] Obor hodnot hyperbolického tangensu je interval $\langle -1; 1 \rangle$.

- Aktuální pozice nepřátelské jednotky na ose Y
- Úhel mezi vektorem pohybu nepřátelské jednotky a vektorem od nepřátelské jednotky k bázi
- Vzdálenost nepřátelské jednotky od báze

Aby všechny vstupy měly stejnou významnost, tak jsou vstupy normalizovány na interval $\langle -1; 1\rangle$[4]. Čistě pro sílu výpočtu navigace by se mohly do neuronové sítě zavést další vstupy (například vzdálenost od nejbližší obranné věže), ale při velkém množství jednotek by se výrazně zvyšovaly výpočetní nároky, protože vstupy do neuronové sítě se mění každý snímek pro každou nepřátelskou jednotku. Možnosti optimalizace jsou nastíněné v 3.2.9.

Ve výstupní vrstvě nepřátelských jednotek je pouze jediný neuron. Pokud označíme výstup neuronu jako N, tak úhel U, o který se změní směr nepřátelské jednotky se určí takto:

$$U = N \cdot T$$

Kde T značí konstantu maximálního možného zatočení. Pokud je tedy výstup N záporný, tak je výsledný úhel také záporný a jednotka zatočí na druhou stranu.

Výkon neuronové sítě je vypočítáván dvěma způsoby. Pokud se nepřátelské jednotce nepodařilo poškodit bázi, tak je proměnné *fitness* přířazena vzdálenost této jednotky od báze s obráceným znaménkem. Pokud jednotka do báze narazit dokázala, tak je *fitness* určena jako převrácená hodnota času pohybu. U hodnot *fitness* jde pouze o to, aby se daly navzájem porovnat. Díky výše zmíněnému platí, že pozitivnější hodnota je vždy lepší.

O uspořádání jednotek do generací a jejich generování se stará třída *Generation*. Každá generace – kromě první – si zachovává určité procento (dle proměnné) nejlepších neuronových sítí z generace předchozí. Pokud není kapacita generace zaplněna, tak je každý snímek generována nová nepřátelská jednotka, která obsahuje náhodnou neuronovou síť z výše zmíněného seznamu nejlepších neuronových sítí předchozí generace. Tato nová nepřátelská jednotka je následně mutována. Síla mutace je určena funkcí. Bylo zjištěno, že běžná konstantní funkce nestačí. Pokud je u konstantní funkce mutační síla příliš nízká, tak nepřátelské jednotky nedokáží dostatečně flexibilně reagovat na postavené obranné věže. Pokud je mutační síla příliš vysoká, tak nedojde k přesnému naučení dobré cesty[5]. Lineární funkce tento problém zmírňuje. Exponenciální se ukázala jako nejlepší řešení – nepřátelské jednotky se dokáží naučit přesnou cestu k cíly a zároveň jednotky ke konci generace,kde exponenciální funkce dosahuje nejvyšších hodnot, dokáží testovat nové cesty.

Metoda *BoreNewGeneration* třídy Generation vrací následující generaci. Nejdříve seřadí jednotlivé neuronové sítě dle evaluované *fitness*. Poté překopíruje 20 % nejlepších neuronových sítí do nové generace a tuto novou generaci vrátí.

Manažerská třída *EnemySpawner* se stará pouze o vytvoření první generace. Později již pouze obsahuje časovač, který značí odstranění aktuální generace a generování nové.

Čtenář si může všimnout, že využití neuronové sítě může být univerzální a s pár drobnými změnami lze neuronovou síť využít i u jiných jednotek. Naříklad by bylo možné, aby hráč také disponoval věží, který by využívala možností neuronových sítí. Tato univerzálnost je autorský záměr. Navigace nepřátelských jednotek je spíše důkaz funkčnosti algoritmu.

Také je možné přidat více různých druhů nepřátelských jednotek, omezením však zůstává nutnost učení těchto nových jednotek. To, že jeden typ navigace je vhodný pro jeden typ jednotky neznamená, že jiný typ jednotky bude schopný s touto navigací uspět.

3.2.4 Modely

Pro modelování jednotlivých objektů byl vybrán program *Blender* verze 3.3.0 [31]. Při tvorbě jednotlivých modelů bylo myšleno na udržení nízkého počtu polygonů, aby se příliš nezvyšo-

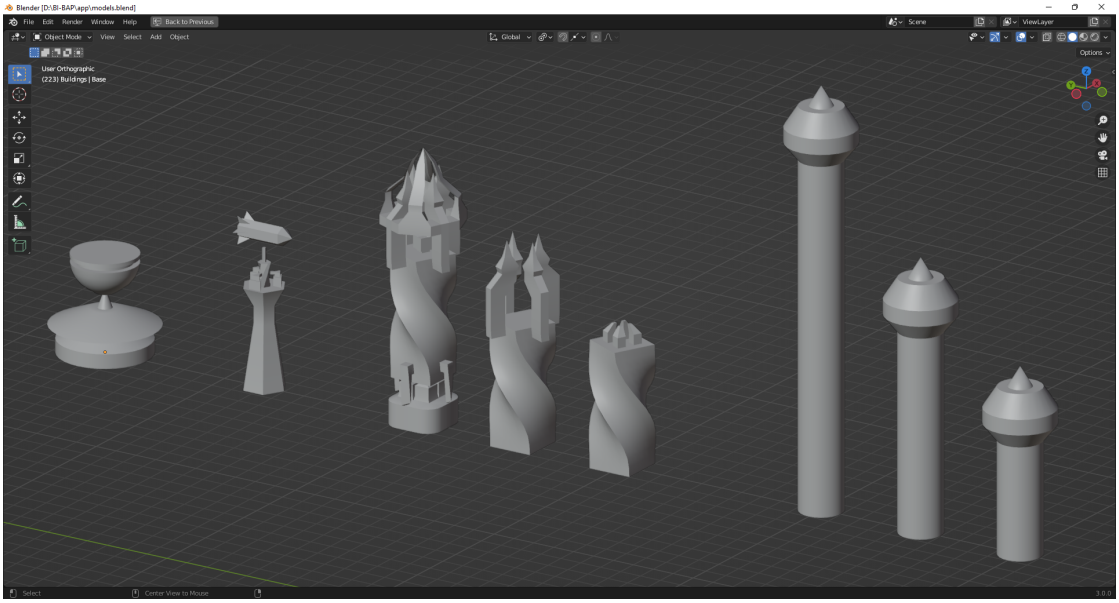
[4] Jak si milý čtenář asi domyslí, vzdálenost je normalizována na interval $\langle 0; 1\rangle$

[5] *Dobrá* dobrá cesta zde znamená taková, která vede k poškození hráčovy báze.

valy nároky na hardware uživatele. Vytvoření dojmu detailnější struktury objektu je přenecháno texturám.

Obrázek 3.11 Modely budov hráče

Zleva: báze, Rocket (a Atom) Tower, Doom Tower, Magic Tower, Cubic Tower, Tallest Tower, Taller Tower, Tall Tower

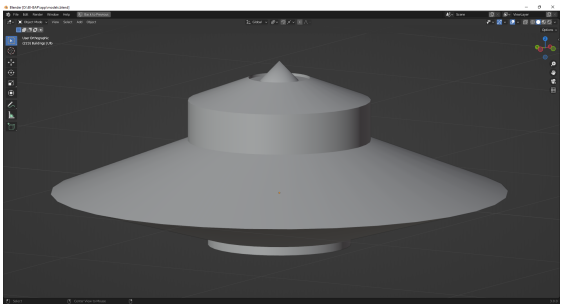


Na obrázku 3.11 jsou vidět modely budov, které hráč může v průběhu hry postavit. Každá vylepšovatelná skupina věží má vlastní styl: Tall Tower a její vylepšení jsou kulaté a zvyšují se, Cubic Tower a její vylepšení mají točité hranaté tělo a jejich vršek se vylepšováním stává komplexnějším. Toto hráči jasně komunikuje, která věž je kterého typu. Rocket Tower má ve hře své vylepšení Atom Tower, které je hráči signalizováno výrazně větším objektem vrhané rakety. Symetričnost věží také umožňuje později při nasazení ve hře použít jednoduché kolizní obaly (kvádr, koule, kapsle), které jsou znatelně rychlejší při detekci kolizí.

Báze má oproti ostatním modelům vyšší počet polygonů, protože ve scéně bude vždy jenom jedna. Věží bude hráč stavět více, měly by být tedy více „optimalizované“ – obsahují menší počet polygonů. Při nastavení stínování modelu v Blenderu na *auto smooth* se normály sousedících polygonů, které jsou od sebe skloněné do zvolené hodnoty úhlu, interpolují. I s jednodušší geometrií lze dosáhnout dojmu hladkého povrchu.

Nepřátelských jednotek se ve hře průměrně generuje ještě více než věží. Byla zde zvolena nejvyšší optimalizace geometrie na model kostky, která osahuje pouze 12 trojúhleníků (2 pro každou čtvercovou stěnu). Při použití tříbokého jehlanu by se dalo dostat až na hodnotu 4 trojúhleníků, avšak tímto získaná výhoda úspory výkonu by byla ztracena na výpočtu kolizního obalu, který by buď nebyl přesný nebo by byl výpočetně dražší.

Obrázek 3.12 Model talíře



Nepřátelský talíř viditelný na obrázku 3.12 pod sebou generuje nepřátelské jednotky. Zastává funkce báze nepřátelské armády, takže byl pro něj byl zvolen tvar podobného stylu

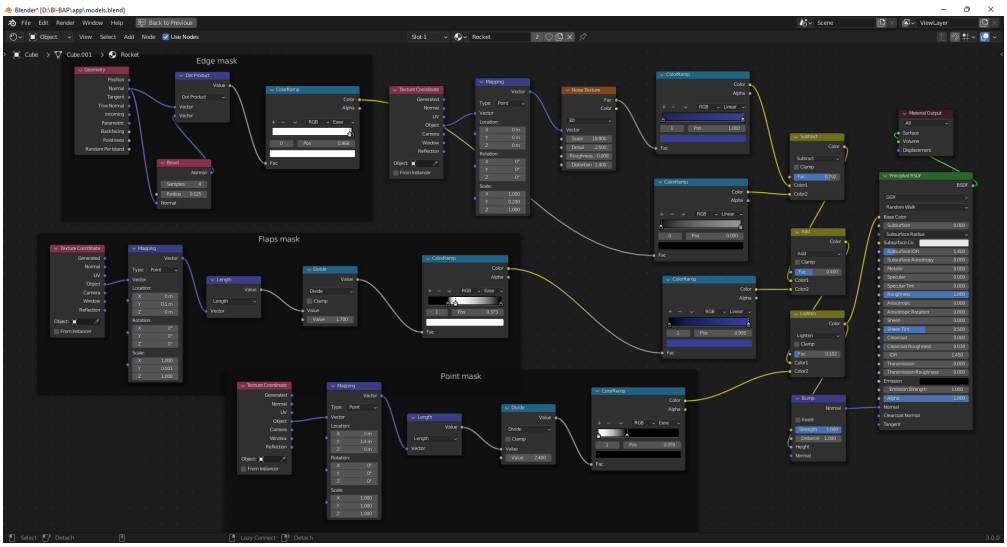
jako hráčova báze.

Zdrojové modely v programu Blender obsahují libovolné polygony. Při exportu do formátu *.fbx* byla provedena *triangulizace* – převedení všech polygonů na trojúhelníky. Při exportu je také nutno dbát na správný převod souřadné soustavy. Program Blender využívá souřadnou soustavu, ve které osa Z reprezentuje směr vzhůru, zatímco herní engine Unity pro směr vzůru využívá osu Y. Při nesprávném převodu mohou být do Unity importované modely otočené o 90°.

3.2.5 Textury

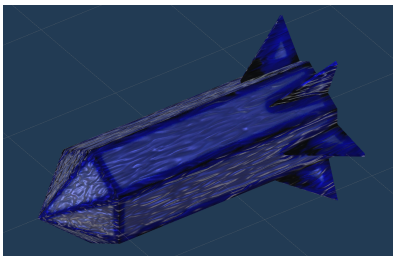
Textury jsou vytvořeny také v programu Blender [31]. Byla využita možnost *procedurálního* generování materiálů pomocí uzlů. Pomocí uzlů masek, parametricky generovaného šumu, vektorových a skalárních matematických operací, podpůrných uzlů a určité dávky trpělivosti lze vytvořit téměř libovolně vypadající materiál. Takto vytvořené materiály mají výhodu možnosti nedestruktivní editace pomocí parametrů. Je možné také uzly organizovat dle úlohy: některé se starají o vzor materiálu, jiné určují jeho barvu. Na obrázku 3.13 lze vidět příklad uzlové struktury využité pro generování materiálu rakety (pro Rocket a Atom Tower).

Obrázek 3.13 Procedurální materiál rakety



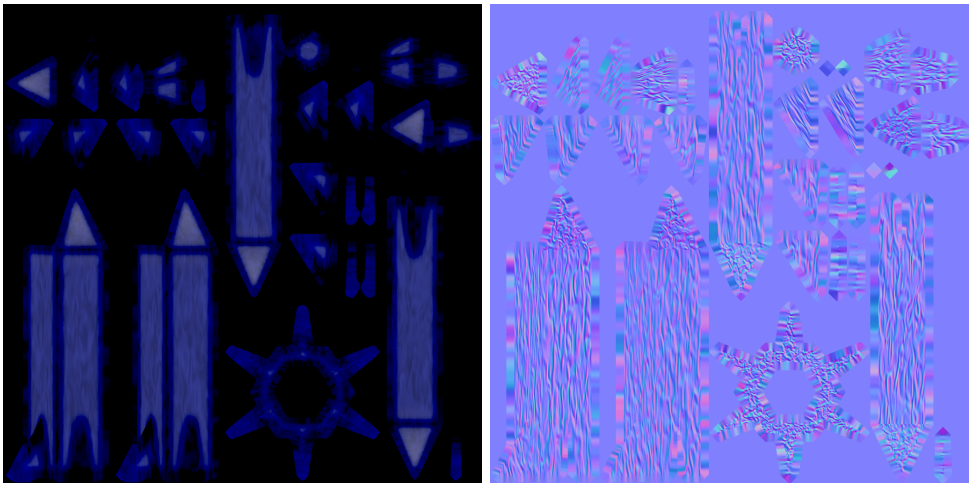
Základ povrchu rakety je tvořen perlinovým šumem, který moduluje jednolitou modrou barvu. Existují zde tři masky – *Edge mask* maskuje hrany, *Flaps mask* maskuje letky a *Point mask* maskuje špičku. Tyto masky určují místa v materiálu, která jsou vybraným způsobem upravena (například zesvětlena). Z výsledné barvy je také vypočítána *bump* mapa, která určuje vizuální výšku materiálu. Z uzlové struktury v Blenderu jsou na závěr *zapečeny* (*baked*) dva soubory textury: difuzní a normálová. Difuzní textura určuje barvu nenasvíceného modelu, normálová určuje orientaci normál geometrie.

Obrázek 3.15 Model rakety



Výsledný model importovaný do Unity s přidanými texturami vypadá jako na obrázku 3.15. Tímto způsobem je vytvořena většina textur pro modely, výjimku tvoří například model talíře, pro který byla textura namalována ručně.

Obrázek 3.14 Difuzní a normálová textura rakety



3.2.5.1 Textura pozadí

Pro texturu pozadí byl navíc použit freeware program Skypaint [32]. Tento program dokáže texturu pozadí namapovat do prostoru velmi podobně, jak nakonec vypadá v Unity. Po stisknutí pár tlačítek lze momentálně viditelný výřez ve vybraném editoru upravovat. Po uložení obrázku se prostředí aktualizuje. Takto lze v grafickém editoru namalovat celé pozadí.

Tímto způsobem jsou po celém obvodu namalovány jednoduše vystínované modré hory. Nad nimi jsou na obloze bílé mraky. Pozadí má působit jednoduchým dojmem, aby nerušilo hráče od hlavního dění na herní ploše.

Export z programu Skypaint je proveden do cylindrického tvaru, který je bez většího zkreslení upravitelný i později v klasickém editoru obrázků, což se hodilo při následném jemném posunu celého obrázku nahoru, aby obzor vypadal přirozeněji. Cylindrický tvar umí Unity importovat a vnitřně ho převádí na reprezentaci *cubemap*, což znamená pozadí ve tvaru kostky, kdy každá strana má svou texturu.

Obrázek 3.16 Textura pozadí

Na obrázku 3.16 je možné vidět výslednou texturu pozadí po všech úpravách.

3.2.6 Zvukové efekty

Na výrobu zvukových efektů byl využit online nástroj Chiptone [33]. Tento nástroj slouží jako syntetizér jednoduchých zvukových efektů pomocí přehledného uživatelského rozhraní (viz 3.17), ve kterém se metodou pokus-omyl vyzná i začátečník.

Obrázek 3.17 Uživatelské rozhraní programu Chiptone

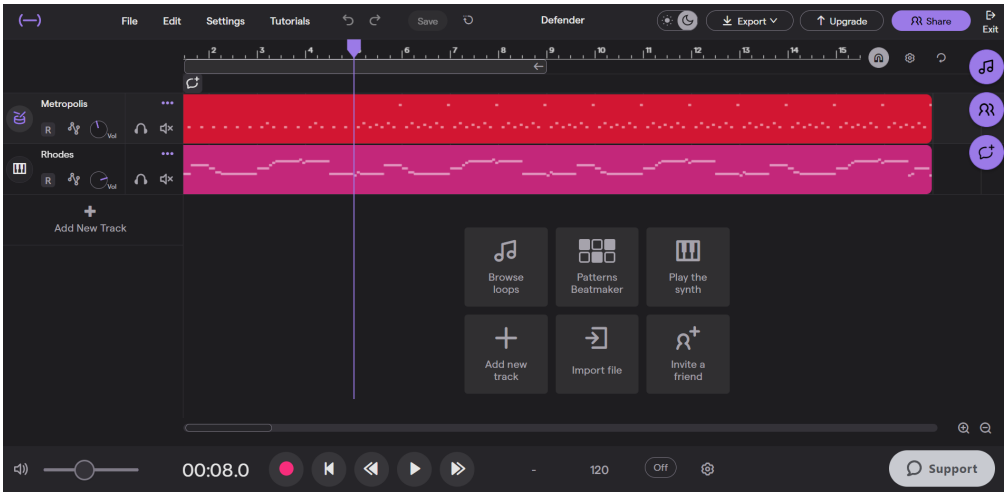
Nástroj dokáže exportovat zvuky do *.wav* souborů. Jsou takto ozvučené výstřely věží, exploze, zásahy jednotek a prohra.

Zvukové efekty, u kterých to má smysl, jsou zasazené do 3D prostoru, což jinými slovy znamená, že čím blíž je kamera ke zdroji zvuku, tím hlasitější zvuk je.

3.2.7 Hudba

Hudební smyčka je také vlastní, složena ve webové aplikaci *Soundtrap* (viz obrázek 3.18). [34]

Obrázek 3.18 Hudba v aplikaci Soundtrap



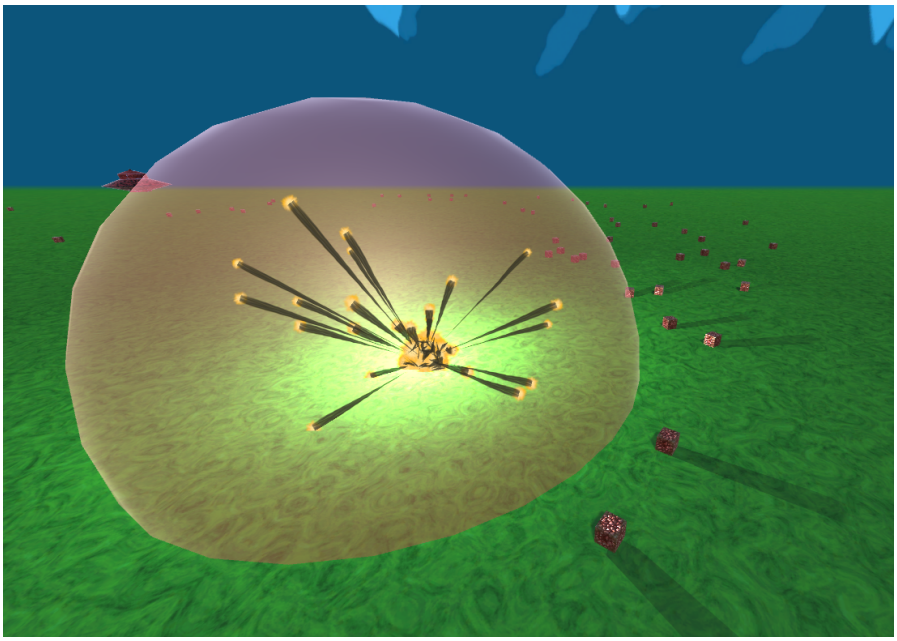
Herní hudba slouží k budování atmosféry v herním světě. Při jejím *loopování* (spouštění ve

smyčce) na sebe navazuje. Díky posuvníku v menu je hlasitost hudby nastavitelná uživatelem. Při poloze posuvníku v levo hudba nehraje, hráč má možnost si do pozadí pustit hudbu dle vlastního výběru.

3.2.8 Osvětlení

Hlavní osvětlení ve scéně zajišťuje směrový zdroj světla. Další osvětlení je možné spatřit při výbuchu rakety z Rocket Tower (obr. 3.19). Při výbuchu se do vzduchu rozletí kusy zplodin a u jednotek se na chvíli objeví stíny vedoucí směrem od výbuch, což dodává výbuchu na realističnosti. Aby hra mohla být hrána i na hardwarově slabších počítačích, tak je s množstvím světelných bodů ve scéně šetřeno.

Obrázek 3.19 Výbuch rakety



3.2.9 Možnosti rozšíření

Hra je naprogramovaná spíše jako ukázka možností strojového učení v reálném čase. Spousta věcí je na této hře rozšiřitelná. Následuje vyjmenování některých způsobů, kterými by se dala hra vylepšit, zpestřit nebo změnit. Logicky bude postupováno od obecnějších změn ke konkrétnějším.

Optimalizace hry by při případném přechodu na mobilní platformu nesla zásadní význam. Algoritmicky hra nabízí možnost paralelizace výpočtu neuronových sítí jednotlivých nepřátelských jednotek. Pokud by se posloupnost operací přizpůsobila, tak by některé vypočty mohly být prováděny i na GPU v compute shaderech. Jiný způsob optimalizace neuronové sítě je možnost rozložení výpočtu neuronové sítě do více snímků – navigace by fungovala podobně, i když by se jeden výpočet neuronové sítě aplikoval do více snímků za sebou.

Vzhledem k tomu, že je celá hra vytvořena v herním enginu Unity, tak se změnou pár nastavení Unity dá hra vyexportovat i na jiné platformy. Podporuje tedy rovnou Windows, Linux a Mac. Pokud by se lehce upravilo uživatelské rozhraní a ovládání (například větší tlačítka a pravý klik myši), tak by se dala hra spustit i na mobilních zařízeních. Specialitou je také možnost exportu do WebAssembly využívající WebGL, které s lehkou spouštěcí pomocí JavaScriptu rozběhne

aplikaci i ve webovém prohlížeči s téměř nativní rychlostí. Již teď je tedy hra téměř automaticky multiplatformní.

Hra samotná je rozšiřitelná například přidáním obtížností pro hráče, kteří hledají větší či naopak menší výzvu. Vyšší obtížnost by se dala řešit vícero způsoby. Jednoduché řešení je například zvýšení životů nepřátelských jednotek, snížení ceny za jednotlivé obranné věže, zrychlení nepřátelských jednotek, snížení životů báze nebo snížení poškození obranných věží. Některé způsoby by měli jisté další dopady. Například zvýšení počtu nepřátel v jedné vlně by nejen způsobilo náročnější ničení této vlny, ale také by se nepřátelské jednotky rychleji učili a při výrazném zvýšení počtu nepřátel by narůstala výpočetní náročnost všech neuronových sítí.

Pro některé hráče by mohla také být lákavá možnost detailnějšího nastavení hry. Většina proměnných uvedených výše a spousta dalších by mohla být dostupná samotným hráčům skrze přehledné uživatelské rozhraní, které by jim dovolovalo ve vybraných mezích nastavit libovolné hodnoty. Otevírala by se tak hráčům možnost experimentovat a pozorovat, jak na algoritmus strojového učení působí různá nastavení. Do nastavení by se daly zahrnout také změny ovládání nebo například volitelná perspektivní/ortografická hlavní kamera.

V téměř jakékoli počítačové hře se samozřejmě výrazně vyjímá její grafická podoba, která lze vylepšovat nepřeberným množstvím způsobů. Přehlednost uživatelského rozhraní je zásadní, ale nelze zapomenou ani na estetické vzezření modelů, textur, animací nebo třeba osvětlení. Při plánování grafických změn je ovšem možné narazit na problém subjektivního vnímání jednotlivých aspektů hry. Na rozdíl od většiny odvětví vývoje her je běžné, že na grafické prvky mají hráči i radikálně odlišné názory – testování je tedy obtížné.

Více přímočaré zlepšení je přidání více druhů obranných věží. Pokud by nekterému hráči určitý druh věže z jakéhokoli důvodu nevyhovoval, tak jej nemusí vůbec stavět. Přidání nových věží tedy může bez rizika pouze vylepšit. Pokud by se hra začala dostávat na e-sportové turnaje, tak by případné vývojáře hry čekal nelehký úkol precizního balancování jednotlivých věží tak, aby jeden typ věže nebyl přehnaně silný oproti ostatním typům věží.

Logickým by se mohlo jevit i přidání dalších typů nepřátelských jednotek, avšak zde je situace problematičtější. Zdůvodnění viz podsekce 3.2.3.

Téměř jistým zlepšením by také bylo ozvučení více herních událostí. Na mysl přijde například zvuk stavby věže, zrození vlny nepřátel nebo varování při nedostatku financí na stavbu/vylepšení obranné věže. Jednotlivé události by také mohly mít pár různých zvukových variací, které by se náhodně střídaly, aby zvuky nepůsobily repetetivním dojmem. Pod drobnohledem by se také dala měnit hlasitost jednotlivých zvukových efektů (relativně vůči ostatním zvukovým efektům).

S předchozím nepřímo související změnou by bylo zlepšení či přidání další hudby. Odlišná hudba pro hlavní menu a pro hru samotnou by nejspíše přišla jako první. Pokročilejší mechanismy přechodů mezi skladbami by mohly být implementovány později.

3.3 Testování hry

Testovaná verze aplikace byla sestavena pro operační systém Windows 64 bit. Testování proběhlo dvěma způsoby: kvantitativní sběr dat skrze dotazník a kvalitativní pozorování hráče a jeho interakce se hrou.

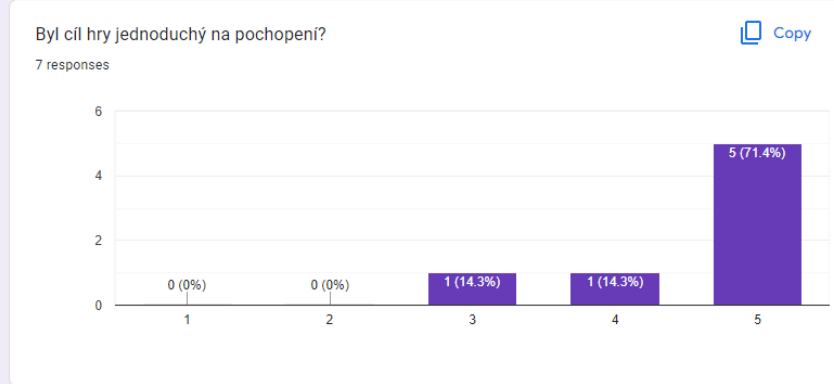
3.3.1 Dotazník

Po samostatném vyzkoušení hry vyplnilo dotazník celkem 7 respondentů. Hra byla respondentům rozeslána zabalená v *.zip* souboru. Ke spuštění stačilo rozbalit a poklepat na soubor *3D-defender.exe*. Jediné další informace o hře, které měli respondenti k dispozici, bylo ovládání hry (příloha B), které bylo přiložené k instrukcím instalace a vyplnění dotazníku.

Dotazník obsahoval celkem 5 otázek. Otázky byly zvoleny převážně uzavřené, aby šlo vyhodnocovat větší množství dotazníků. Pouze u poslední mohl respondent uvést jakékoli „jiné“

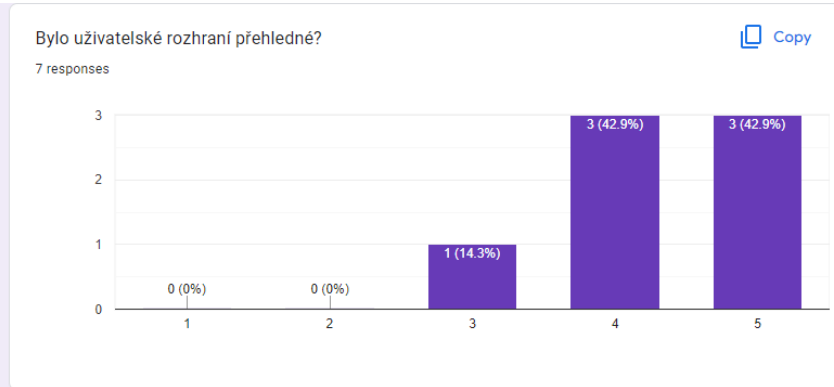
návrhy na vylepšení. Platforma pro zprostředkování tvorby a zobrazení výsledků dotazníku byla zvolena *Google Forms* (`https://docs.google.com/forms/`).

Obrázek 3.20 Dotazník: „Byl cíl hry jednoduchý na pochopení?“



První otázka dotazníku na obrázku 3.20 se týkala cíle hry. Hodnocení mělo číselnou škálu od 1 do 5 (1 = NE, 5 = ANO). Jak bylo předpokládáno, základní cíl a styl hry *tower defense* je jednoduchý na pochopení a ve většině případů – 5 ze 7 respondentů odpovědělo na otázku ano.

Obrázek 3.21 Dotazník: „Bylo uživatelské rozhraní přehledné?“



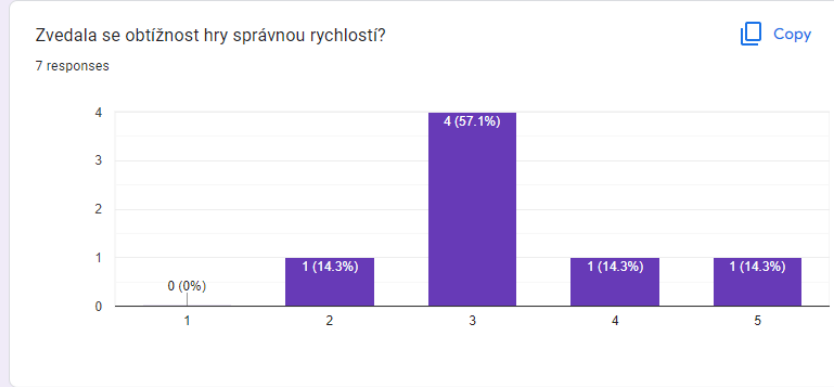
Druhá otázka na obrázku 3.21 dotazníku zkoumala přehlednost uživatelského rozhraní. Hodnocení měla stejné jako otázka předešlá. I zde se 6 ze 7 dotazovaných přiklonilo ke kladné odpovědi. Přestože výsledky nejsou tak jednoznačné jako u otázky první, lze předpokládat, že uživatelské rozhraní neruší herní zážitek.

Třetí otázka na rychlost zvedání obtížnosti na obrázku 3.22 měla jiné hodnocení: 1 = „obtížnost se zvedá příliš pomalu“, 5 = „obtížnost se zvedá moc rychle“. Hodnocení je zde více rozložené. Jak bylo nastíněno v sekci 3.2.9, existuje více možností řešení změny obtížnosti hry. Vzhledem k rozložení odpovědí by však bylo nejspíše pro hru přínosné, kdyby sám hráč mohl před začátkem hry zvolit například z několika přednastavených úrovní obtížnosti dle své preference.

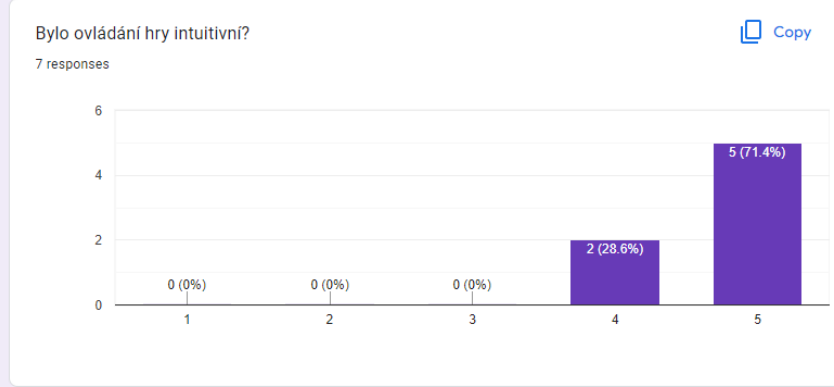
Čtvrtá otázka na obrázku 3.23 se zaměřovala na ovládání hry. Číselná škála hodnocení byla stejná jako u prvních dvou otázek. Z pouze pozitivních odpovědí lze soudit, že všichni respondenti z textového popisu ovládání, který jim byl zaslán, jednoduše pochopili ovládání hry.

Pátá otázka na obrázku 3.24 je poslední a zároveň informačně nejpřínosnější. Respondenti mohli vybrat jednu či více kategorií, ve kterých dle jejich názoru má hra největší potenciál na zlepšení. Na výběr bylo z několika přednastavených kategorií a kategorie *jiné*, ve které mohli vyjádřit své vlastní originální myšlenky a poznámky.

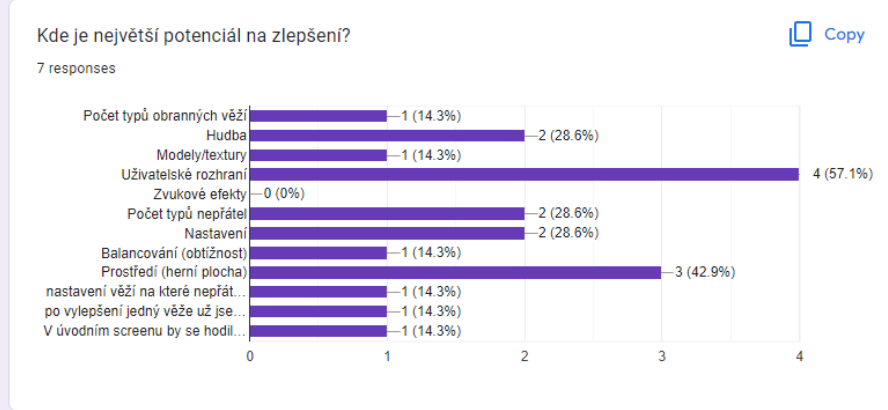
Obrázek 3.22 Dotazník: „Zvedala se obtížnost hry správnou rychlostí?“



Obrázek 3.23 Dotazník: „Bylo ovládání hry intuitivní?“



Obrázek 3.24 Dotazník: „Kde je největší potenciál na zlepšení?“



Nejčastěji vybíranou kategorií se stalo *uživatelské rozhraní*. Z otázky 3.21 je jasné, že respondentům nejde o přehlednost. Při dalším případném vývoji by však mělo být detailněji prozkoumáno estetické zpracování uživatelského rozhraní.

Následující nejčastěji volená byla kategorie *prostředí (herní plocha)*. Zde se jedná o téměř čistě estetický aspekt hry. Výzvou pro designéra zůstává vylepšení vzhledu prostředí bez ztráty přehlednosti. Výzvou pro programátora naopak zůstává zachování nízkých nároků na hardware.

Někteří respondenti také konkrétně vypsali své poznámky[6]. Prosba jednoho respondenta o možnost nastavení obranných věží na cílení nepřátelských jednotek, které jsou nejdál/nejblíž od obranné věže je implementačně jednodušše proveditelná, ale při velkém počtu jednotek výpočetně náročná. Jeden z respondentů zahrnul do kategorie *jiné* dokonce více různých nápadů na zlepšení. Bylo zahrnuto vylepšení uživatelského rozhraní o historii odehraných her s jejich výsledky (*high scores*) nebo možnost vypnutí zvukových efektů.

Celkově dotazník působí kladným dojmem. Obsahuje cenný podklad pro hodnocení a případný budoucí vývoj hry.

3.3.2 Pozorování

Pozorovaný hráč (dále v textu jako *Hráč*) dostal stejné informace o hře jako respondenti dotazníku. Šlo tedy o instrukce ke spuštění a ovládání hry. Navíc byl požádán o hlasový popis jeho přemýšlení.

Krátce po spuštění hry tlačítkem *Start New Game* popsal Hráč velmi přesně cíl hry. I zde (stejně jako u dotazníku) lze tedy říci, že hra jasně a srozumitelně komunikuje hlavní herní cíl.

Orientace v uživatelském rozhraní hry proběhla téměř okamžitě. Hráči se spojila souvislost mezi ukazatelem stavu peněz a cenou jednotlivých věží. Plánování a stavba věže se zdála také bez problémů, avšak po postavení první věže bylo Hráčem podotknuto, že teď už nelze bohužel věž přesunout. Nabízí se zde možnost přidání tlačítka uživatelského rozhraní vybrané obranné věže, která by za určitý finanční obnos umožnila uživateli tuto vybranou věž přesunout.

Při hře se také vyskytla estetická otázka Hráče: „Mají ti nepřátelé nějaký obličej?“

Případná personifikace nepřátelských jednotek by mohla pomoci vtáhnout uživatele více do hry. Pokud by šlo pouze o změnu textury, tak by dokonce toto vylepšení bylo bez navýšení výpočetní náročnosti.

Testování uživatelského rozhraní Hráčem odhalilo chybu implementace. Při specifickém pořadí kliknutí přestalo fungovat tlačítko vylepšení vybrané obranné věže. Tato chyba je v odevzdávané verzi aplikace opravena. Jde tedy o konkrétní příklad přínosu kvalitativního testování pozorováním.

Další konkrétní nápady na vylepšení, které vyplynuly z pozorování jsou dva: zobrazení popisu věže před položením (a zaplacením) a možnost zobrazení dosahu právě vybrané obranné věže i po položení. Tyto dvě změny také neznamenají zvýšenou hardwarovou náročnost hry a mohou uživateli přinést cenné informace, které může využít při strategickém plánování rozmístění obranných prvků.

[6] Za což jim patří vřelé poděkování.

Kapitola 4
Závěr

Závěrem lze s jistotou říci, že strojové učení je při vývoji her použitelné. Implementovaná hra využila možností evoluční neuronové sítě k postupném zpřesňování navigace jednotek ve funkční 3D tower defense hře *3D defender*.

V práci jsou v analytické části zmapovány současné možnosti využití nejpopulárnějších herních enginů Unity a Unreal Engine k vývoji her včetně jejich licenčních plánů pro týmy vývojářů různých velikostí. Jsou zde také analyzovány různé modely strojového učení a nápady na jejich využití v herním průmyslu. Posledním zanalyzovaným tématem jsou příklady již existujících her, které určitým způsobem při vývoji či běhu využívají strojové učení.

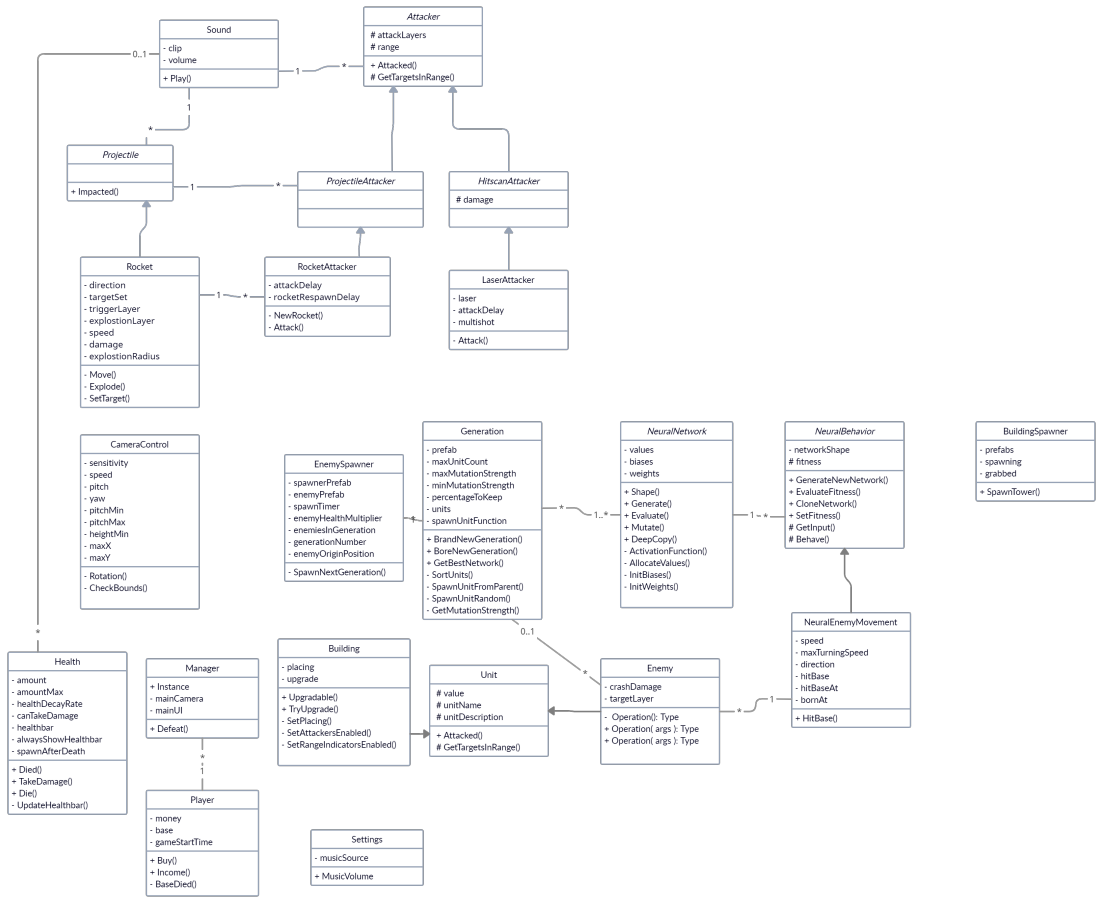
V praktické části je následně navržena originální nová hra, která využívá výhod modelu posilovacího učení jako důkazu, že je možné trénovat zpočátku zcela náhodnou umělou inteligenci přímo za běhu hry. Návrh obsahuje jak vnitřní fungování v podobě diagramu tříd, tak náčrt uživatelského rozhraní a stavový diagram. Před samotnou implementací proběhla výroba modelů převážně v nástroji pro 3D grafiku jménem Blender. Nechybí nastínění výhod a využití procedurálně generovaných textur pomocí systému uzlů. Speciální pozornost je věnována implementaci nepřátelských jednotek, které v reálném čase využívají neuronovou síť ke své navigaci a díky evolučnímu algoritmu se generace za generací zlepšují v proražení lidské obrany. Použití zvukových efektů a hudby vlastní výroby dokresluje atmosféru bitevního pole. Celá hra je sestavená ve dle analýzy vybraném herním enginu Unity pro operační systém Windows. Finální testování proběhlo formou dotazníku a kvalitativního pozorování. Odhalilo některé chyby a poukázalo na místa pro zlepšení na základě preferencí samotných hráčů. V práci jsou nápady na další případný vývoj. Implementovaná evoluční neuronová síť je generalizovatelná a lze ji využít i v jiných částech hry, které tím mohou získat další aspekt zájmu pro hráče.

Práce splnila všechny vytyčené cíle. Text je dle názoru autora psaný srozumitelně a i čtenář, který není expertem v oboru počítačových her, si z této práce odnese obohacující informace, myšlenky a nápady.

Příloha A

Kompletní diagram tříd

Obrázek A.1 Diagram tříd



Příloha B

Ovládání hry

Hra se ovládá klávesnicí a myší.

- Šipky / W, S, A, D → pohyb pohledu do stran
- Mezerník → pohyb pohledu nahoru
- Shift → pohyb pohledu dolů
- Držení pravého tlačítka myši → rozhlížení
- Levé tlačítko myši → pokládání věží

Bibliografie

1. DOBREV, Dimiter. A Definition of Artificial Intelligence. 2012. Dostupné z DOI: 10.48550/ARXIV.1210.1568. © arXiv.org perpetual, non-exclusive license.
2. IBM CLOUD EDUCATION. *Machine Learning* [online]. 2020-07-15. [cit. 2022-11-22]. Dostupné z: https://www.ibm.com/cz-en/cloud/learn/machine-learning.
3. BURKOV, Andriy. *Hundred-Page Machine Learning Book*. 2019. ISBN 9781999579500.
4. SUTTON, Richard; BARTO, Andrew. *Reinforcement Learning: An Introduction*. Springer Cham, 2020. ISBN 9780262039246. Dostupné také z: http://www.incompleteideas.net/book/RLbook2020.pdf. © 2018, 2020 Richard S. Sutton and Andrew G. Barto.
5. PEDDIE, Jon. *Ray Tracing: A Tool for All*. Westchester Publishing Services, 2019. ISBN 978-3-030-17490-3. Dostupné také z: https://link.springer.com/book/10.1007/978-3-030-17490-3. © Springer Nature Switzerland AG 2019.
6. UNITY TECHNOLOGIES. *Game Development Terms* [online]. [cit. 2022-11-25]. Dostupné z: https://unity.com/how-to/beginner/game-development-terms. © 2022 Unity Technologies.
7. COLLINS DICTIONARY. *Definition of 'game'* [online]. [cit. 2022-11-25]. Dostupné z: https://www.collinsdictionary.com/dictionary/english/game. © Collins 2022.
8. MASTERCLASS. *Tower Defense Game Genre: 6 Characteristics of TD Games* [online]. 2021-07-19. [cit. 2022-11-25]. Dostupné z: https://www.masterclass.com/articles/tower-defense-game-video-game-guide.
9. KUMAR, ANMOL. *MNIST digit classification - 99.5% accuracy* [online]. 2020-06-27. [cit. 2022-12-25]. Dostupné z: https://www.kaggle.com/code/anmolkumar/mnist-digit-classification-99-5-accuracy.
10. DOUCET, Lars; PECORELLA, Anthony. *Game engines on Steam: The definitive breakdown* [online]. 2021-09-02. [cit. 2022-12-23]. Dostupné z: https://www.gamedeveloper.com/business/game-engines-on-steam-the-definitive-breakdown.
11. PECKHAM, Eric. *How Unity built the world's most popular game engine* [online]. 2019-10-17. [cit. 2022-12-23]. Dostupné z: https://techcrunch.com/2019/10/17/how-unity-built-the-worlds-most-popular-game-engine/.
12. UNITY TECHNOLOGIES. *Unity Editor Software Terms* [online]. 2022-10-13. [cit. 2022-12-23]. Dostupné z: https://magazine.renderosity.com/article/5330/focus-unreal-engine-a-brief-history-of-unreal.
13. UNITY TECHNOLOGIES. *Choose the plan that is right for you* [online]. [cit. 2022-12-23]. Dostupné z: https://store.unity.com/compare-plans. © 2022 Unity Technologies.

14. UNITY TECHNOLOGIES. *Unity (verze 2022.1.20f1)* [online]. [B.r.]. [cit. 2022-12-24]. Dostupné z: https://unity.com/releases/editor/archive.
15. GROVE, Ricky. *Unreal Engine - A Brief History of Unreal* [online]. 2019-07-23. [cit. 2022-12-23]. Dostupné z: https://magazine.renderosity.com/article/5330/focus-unreal-engine-a-brief-history-of-unreal.
16. EPIC GAMES. *Licensing options* [online]. [cit. 2022-12-24]. Dostupné z: https://www.unrealengine.com/en-US/license. © 2004-2022, Epic Games, Inc. All rights reserved.
17. EPIC GAMES. *Frequently Asked Questions* [online]. [cit. 2022-12-24]. Dostupné z: https://www.unrealengine.com/en-US/faq. © 2004-2022, Epic Games, Inc. All rights reserved.
18. EPIC GAMES. *Unreal Engine (verze 5.1)* [online]. [B.r.]. [cit. 2022-12-24]. Dostupné z: https://www.unrealengine.com/en-US/download.
19. EPIC GAMES. *Unlimited Access for Unreal Engine* [online]. [cit. 2022-12-24]. Dostupné z: https://help.quixel.com/hc/en-us/sections/360000977797-Unlimited-Access-for-Unreal-Engine. © 2022 Epic Games, Inc. Quixel, Megascans, and Unreal Engine are trademarks or registered trademarks of Epic Games, Inc. in the USA and elsewhere.
20. EPIC GAMES. *Unreal Engine for Unity Developers* [online]. [cit. 2022-12-24]. Dostupné z: https://docs.unrealengine.com/5.0/en-US/unreal-engine-for-unity-developers/. © 2004-2022, Epic Games, Inc. All rights reserved.
21. MAHARAJ, Shiva; POLSON, Nick; TURK, Alex. Chess AI: Competing Paradigms for Machine Intelligence. *Entropy*. 2022, roč. 24, č. 4. ISSN 1099-4300. Dostupné z DOI: 10.3390/e24040550.
22. LINSCOTT, Gary. *Technical Explanation of Leela Chess Zeros* [online]. 2021-11-13. [cit. 2022-12-28]. Dostupné z: https://lczero.org/dev/wiki/technical-explanation-of-leela-chess-zero/.
23. PETE. *Lc0 Wins Computer Chess Championship, Makes History* [online]. 2019-12-06. [cit. 2022-12-28]. Dostupné z: https://www.chess.com/news/view/lc0-wins-computer-chess-championship-makes-history.
24. VINYALS, Oriol; EWALDS, Timo; BARTUNOV, Sergey; GEORGIEV, Petko; VEZHNEVETS, Alexander Sasha; YEO, Michelle; MAKHZANI, Alireza; KÜTTLER, Heinrich; AGAPIOU, John; SCHRITTWIESER, Julian; QUAN, John; GAFFNEY, Stephen; PETERSEN, Stig; SIMONYAN, Karen; SCHAUL, Tom; HASSELT, Hado van; SILVER, David; LILLICRAP, Timothy; CALDERONE, Kevin; KEET, Paul; BRUNASSO, Anthony; LAWRENCE, David; EKERMO, Anders; REPP, Jacob; TSING, Rodney. *StarCraft II: A New Challenge for Reinforcement Learning*. arXiv, 2017. Dostupné z DOI: 10.48550/ARXIV.1708.04782. © arXiv.org perpetual, non-exclusive license.
25. WYDMUCH, Marek; KEMPKA, Michał; JAŚKOWSKI, Wojciech; RUNC, Grzegorz; TOCZEK, Jakub. *ViZDoom* [online]. [cit. 2022-12-28]. Dostupné z: https://vizdoom.cs.put.edu.pl/. © VIZDOOM TEAM.
26. VIZDOOM TEAM. *VDAIC 2016 CIG Results* [online]. [cit. 2022-12-28]. Dostupné z: https://vizdoom.cs.put.edu.pl/competitions/vdaic-2016-cig/results. © VIZDOOM TEAM.
27. JUSTESEN, Niels; BONTRAGER, Philip; TOGELIUS, Julian; RISI, Sebastian. Deep Learning for Video Game Playing. *CoRR*. 2017, roč. abs/1708.07902. Dostupné z arXiv: 1708.07902.
28. VIZDOOM TEAM. *VDAIC 2018 CIG Results* [online]. [cit. 2022-12-28]. Dostupné z: https://vizdoom.cs.put.edu.pl/competitions/vdaic-2018-cig/results. © VIZDOOM TEAM.

29. HOWLONGTOBEAT. *Doom (1993)* [online]. [cit. 2022-12-28]. Dostupné z: https://howlongtobeat.com/game/2701. © 2011–2022 Ziff Davis, LLC, a Ziff Davis company. All Rights Reserved.
30. HASTINGS, Erin J.; GUHA, Ratan K.; STANLEY, Kenneth O. Evolving content in the Galactic Arms Race video game. In: *2009 IEEE Symposium on Computational Intelligence and Games*. 2009, s. 241–248. Dostupné z DOI: 10.1109/CIG.2009.5286468.
31. BLENDER FOUNDATION. *Blender (verze 3.3.0)* [online]. [B.r.]. [cit. 2022-12-04]. Dostupné z: https://www.blender.org/.
32. WASABI SOFTWARE. *Skypaint* [online]. [B.r.]. [cit. 2022-12-16]. Dostupné z: https://gamebanana.com/tools/6141. © Creative Commons Attribution-NonCommercial-NoDerivs 4.0 Unported License.
33. VIAN, Tom. *Chiptone (verze 0.5.1)* [online]. [B.r.]. [cit. 2022-12-03]. Dostupné z: https://sfbgames.itch.io/chiptone.
34. SPOTIFY. *Soundtrap* [online]. [B.r.]. [cit. 2022-12-03]. Dostupné z: https://www.soundtrap.com/. © 2022 Spotify USA Inc / Spotify AB.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src/`
  - `app/` ..... zdrojové kódy implementace hry
  - `text/` ..... zdrojová forma práce ve formátu LaTeX
- `out/`
  - `ctufit-thesis.pdf` ..... text práce ve formátu PDF
  - `app.zip` ..... archiv se spustitelnou formou implementace (Win 64 bit)