FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Diplomová práce

Hra založená na neuronových sítích a reinforcement learningu

Studijní program: N0613A140028 Informační technologie
Autor práce: **Bc. David Brož**
Vedoucí práce: Ing. Igor Kopetschke
Ústav nových technologií a aplikované informatiky

Liberec 2023

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání diplomové práce

Hra založená na neuronových sítích a reinforcement learningu

Jméno a příjmení:	**Bc. David Brož**
Osobní číslo:	M21000158
Studijní program:	N0613A140028 Informační technologie
Zadávající katedra:	Ústav nových technologií a aplikované informatiky
Akademický rok:	2022/2023

Zásady pro vypracování:

1. Seznamte se problematikou učení neuronových sítí pomocí reinforcement learning.
2. Navrhněte hru využívající neuronové sítě a reinforcement learning jako hlavní herní mechaniku.
3. Seznamte se s nástroji pro vývoj počítačových her a zvolte příslušný nástroj.
4. S pomocí zvolených nástrojů implementujte hru jako minimální životaschopný produkt.
5. Implementované řešení otestujte, získejte a vyhodnoťte zpětnou vazbu.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	40 – 50 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] PAN, Chao. Deep learning fundamentals: an introduction for beginners. [Wilmington, Delaware]: AI Sciences, 2018. ISBN 978-1-7212-3088-4.

[2] DEMUSH, Rostyslav. Reinforcement Learning Examples. Reinforcement Learning Applications: A Brief Guide on How to Get Business Value from RL. 2018. Dostupné také z: https://perfectial.com/blog/reinforcement-learning-applications.

[3] SZANDAŁA, Tomasz. Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. In: Bio-inspired Neurocomputing. Springer, 2021, s. 203–224.

Vedoucí práce:	Ing. Igor Kopetschke Ústav nových technologií a aplikované informatiky
Datum zadání práce:	12. října 2022
Předpokládaný termín odevzdání:	22. května 2023

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

Ing. Josef Novák, Ph.D.
vedoucí ústavu

V Liberci dne 19. října 2022

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Jsem si vědom toho, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má diplomová práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

22. května 2023 Bc. David Brož

Hra založená na neuronových sítích a reinforcement learningu

Abstrakt

Diplomová práce se zabývá návrhem a vývojem videohry využívající technologie umělých neuronových sítí a zpětnovazební učení jako hlavní herní mechaniku. Hráč ve hře dostane za úkol vycvičit agenty kontrolovaných neuronovou sítí tak, aby splnily předem daný úkol. K trénování těchto agentů hráč bude využívat nástroj pro tvoření překážkových drah a hodnotících systémů. Na těchto drahách se pak agenti budou pomocí metod zpětnovazebního učení trénovat. Cílem hry je navrhnout dráhy tak, aby agenti po jejich absolvování byly schopní splnit i daný úkol. Cílem práce je navrhnou hru, implementovat a otestovat minimální životaschopný produkt této hry.

Klíčová slova: videohra, Unity engine, Unity ML-Agents Toolkit, umělé neuronové sítě, PPO, zpětnovazebné učení

A game based on neural networks and reinforcement learning

Abstract

The diploma thesis deals with designing and developing a video game using technologies of artificial neural networks and reinforcement learning as its primary game mechanic. In the game, the player gets a task to train agents controlled by neural networks to complete a given task beforehand. To train these agents, the player uses a tool for creating obstacle courses and scoring systems. Using reinforcement learning methods the agents are then trained on these obstacle courses. The game's goal is to design the obstacle courses in a way that allows the agent to complete a given task after successfully finishing the obstacle course. The goals of this project are to design the game and implement and test a minimal viable product of this game.

Keywords: video game, Unity engine, Unity ML-Agents Toolkit, artificial neural network, PPO, reinforcement learning

Poděkování

Rád bych poděkoval panu Ing. Igoru Kopetschkemu za výborné vedení práce a za pomoc s řešení problémů při konzultacích, Ing. Romanu Staňkovi a Dr. Ing. Sanderu Cornelusovi Johannesovi Bakkesovi za pomoc při rešerši problematik strojového učení a herního designu, Bc. Tomáši Erlebachovi za jeho roli jako motivační partner při psaní diplomové práce a všem, kteří obětovali svůj čas aby si mojí hru mohli zahrát a poskytnou mi svůj názor. Dále děkuji Ing. Lucii Hambálkové a mojí mamince za pomoc s korekturami, a oběma mým rodičům za neochvějnou podporu během celého studia.

Obsah

Seznam zkratek

API	Applicacion Programing Interface (rozhraní pro programování aplikací)
GAIL	Generative Adversarial Imitation Learning
GEQ	Game Experience Questionnaire (dotazník herních zážitků)
GUI	Graphical User Interface (grafické uživatelské rozhraní)
MVP	Minimal Viable Product (minimální životaschopný produkt)
PPO	Proximal Policy Optimization
SAC	Soft Actor-Critic
SDK	Software Development Kit (sada vývojových nástrojů)
SGD	Stochastic Gradient Descent (Stochastický gradientní sestup)
VF	Value Fuction (užitková funkce)

Seznam obrázků

Seznam algoritmů

Seznam tabulek

Úvod

Koncem roku 2022 pojem umělá inteligence získal světovou pozornost s nástupem produktů jako je jazykový model ChatGPT od společnosti OpenAI a modely generující obrázky z textových vstupů Midjourney a Stable Diffusion. Tento rok byl přelomový moment, kdy se obor strojového učení dostal ze zákoutí obývaného pouze akademickými a průmyslovými experty na veřejnou scénu, do zájmů zvědavých amatérů a do chytlavých a ne vždy příliš věrohodných příběhů šířících se po nejrůznějších internetových portálech. Tyto nové technologie vzbuzují mezi lidmi jak nadšení tak zděšení. Výroky o tom, jak nové technologie změní ten a ten průmysl, zruší ty a ty pracovní pozice, anebo zahubí svět tím a tím způsobem jsou první polovinou roku 2023 každodenní chléb. Tyto fakty nebyly důvodem, proč jsem si zvolil tento projekt, jsou však důkazem, že je na něj teď pravděpodobně dobrý čas.

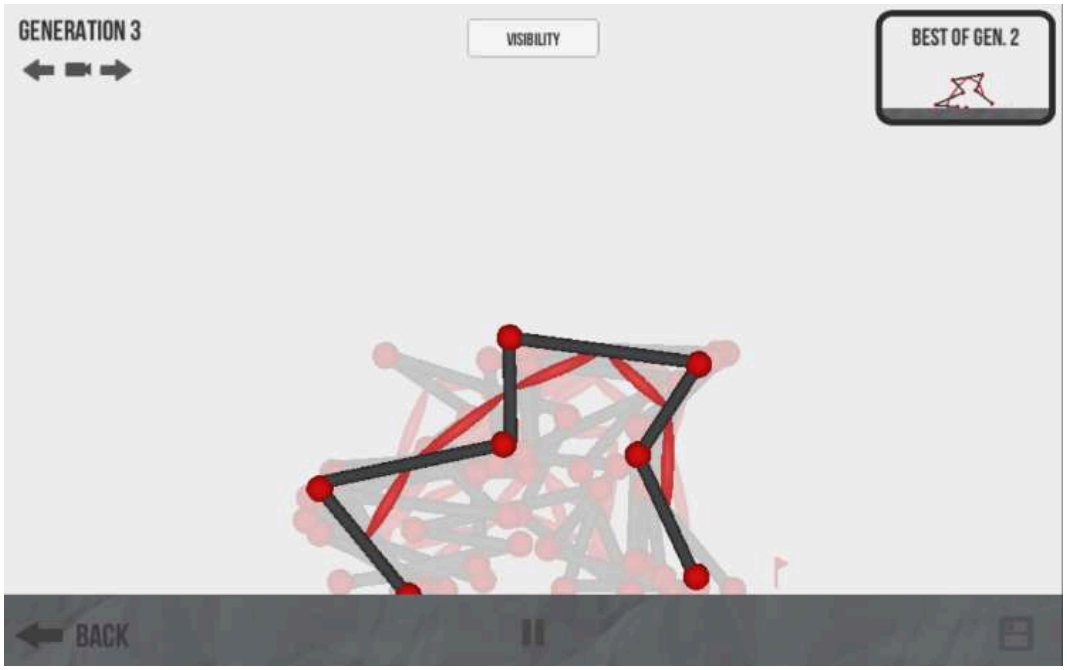
Ještě předtím, než proběhlo zpopularizování těchto technologií, jsem projevil zvědavost o hrách využívající neuronové sítě. Když jsem se ponořil do hledání takovýchto her, zjistil jsem, že žádné takové hry vlastně nejsou. Tedy abych byl konkrétní, žádné takové zajímavé hry nikde nejsou. Většina takových her buď využívají neuronové sítě jako oponenta - což z pohledu hráče vypadá více méně stejně jako již dlouho známé rozhodovací stromy, anebo jsou to hry, které fungují stylem nastav a sleduj. Hry, které fungují spíše jako demo předvádějící, jak neuronové sítě a strojové učení funguje, než jako interaktivní zážitek. Hledal jsem hru, kde cílem je učit neuronovou síť a řešit zajímavé problémy. Zároveň si myslím, že v blízké době bude zájem o takovou hru, která jednoduše představí problematiku strojového učení nováčkům mimo obor. Jelikož jsem žádnou takovou nenašel, rozhodl jsem se ji sám vytvořit.

V první kapitole této práce se nejdříve seznámíme s existujícími hrami a jejich nedostatky, které mě motivovaly k vytvoření nové hry. V druhé kapitole definuji, jaké jsou cíle vytvářené hry a jaké jsou cíle minimálního životaschopného produktu, který je jedním z cílů této práce. Ve třetí kapitole se seznámíme s problematikou učení neuronových sítí pomocí zpětnovazebného učení (Reinforcement Learning) Ve čtvrté kapitole uvedu nástroje, jejichž použití jsem zvažoval pro tvorbu minimálního životaschopného produktu. V páté kapitole rozeberu návrh tvořené hry a šestá kapitola se zabývá implementací minimálního životaschopného produktu. V kapitole sedmé popíši metodiku testování a minimálního životaschopného produktu, vyhodnotím výsledky testování a doporučím změny pro další verze hry.

1 Existující hry

Hry využívající neuronové sítě jako herní mechaniku již existují. V této kapitole krátce představím některé z nich, a analyzuji nedostatky těchto her, které mě motivovaly k vytvoření hry, a jaké poznatky si z těchto her odnáším pro tvorbu mojí hry.

1.1 Evolution



Obrázek 1.1: Snímek obrazovky ze hry Evolution

1.1.1 Popis

Evolution je velice jednoduchá hra a jedná se spíše o prototyp. Cílem hry je vytvořit pomocí kolen, kostí a svalů zvířátko, které se pak trénuje genetickým algoritmem na jednu z několika úloh, jako je běhání, skákání, šplhání, skok, létání nebo překážkový běh. Zvířátka jsou velice abstraktní (viz obrázek 1.1) a hráč má jistou limitovanou kontrolu nad trénováním nastavováním některých parametrů.

Mezi parametry trénování, nad kterými hráč má kontrolu, patří: populace každé generace, doba trvání jedné generace, batch size, parametry neuronové sítě, počet vrstev, počet neuronů v jednotlivých vrstvách, výběrový algoritmus - 4 předdefinované možnosti, rekombinační algoritmus, mutační algoritmus, pravděpodobnost mutace a zda nejlepší jedince z minulé generace zachovat i v nové generaci.[1]

Tato hra prezentuje podobnou kategorii her, které jsou o trénování neuronových sítí, nedávají však hráči mnoho možností, jak do trénování zasáhnout. Neuronová síť se trénuje sama pomocí předdefinovaného algoritmu, který lze občas zaměnit za jiný, ve finále však ne zásadně rozdílný algoritmus. Pokud uživatel chce naučit neuronovou síť jiné chování nebo kreativní řešení problému, hra mu to neumožní. Do této kategorie bych také zařadil hry jako Football Evo - fotbalová simulace, kde se agenti učí hrát fotbal.

1.1.2 Analýza

Hra je velice hezká ukázka toho, jak evoluční algoritmus funguje a čeho je schopen, ovšem jako zábavná hra nefunguje příliš dobře. Jedná se spíše o malý simulátor, který trénuje vaše zvířátko, nežli o interaktivní záležitost. Většinu času hráč vidí své vizuálně nepříliš zajímavé zvířátko, jak dělá nesmyslné pohyby a selhává v pokusu o běh. Celá interakce hry spočívá ve stvoření zvířátka a nastavení několika parametrů trénování, jejichž dopad na výsledek není nijak zřejmý. Celou záležitost navíc zásadně zhoršuje skutečnost, že hra nemá možnost zrychlení simulace.

1.1.3 Poznatky

Z této hry si odnáším následující poznatky:

1. Proces trénování musí být rychlý. Sledování chování neuronové sítě je zajímavé pouze pokud se děje něco neočekávaného nebo vtipného (což se neděje tak často). Cílem je proces trénování urychlit, popřípadě skrýt, a jen hráči ukázat klíčové momenty trénování.

2. Nastavení parametrů musí být srozumitelné na první pohled. Hráč by neměl být nucen nejprve plně porozumět procesu trénování neuronových sítí, aby mohl nastavit parametry trénování.

3. Herní agenti a jejich akce by měly být vizuálně zajímavé.

4. Hráč musí mít představu o následcích jeho akcí. Ve hře Evolution není příliš jasné, jaký vliv daná konstrukce zvířátka má na jeho schopnost se hýbat.

5. Hra by měla mít zajímavé problémy k řešení.

6. Hra by měla mít prostor pro kreativní prostředí.

1.2 Forza Motosport

1.2.1 Popis hry

Forza Motosport je simulační závodní série videoher využívající systém Drivatar, modelující chování hráče hned od svého vzniku jejího prvního titulu na herním systému Xbox vydané v roce 2005. Tato série her je známá pro její přesnou simulaci fyziky na závodních drahách. Umělá inteligence se učí během hraní hry a ukládá se na disk konzole, každý hráč má tak vlastního unikátního AI závodníka, proti kterému hráč pak může závodit. Využívá Bayesovské umělé neuronové sítě.[2]

Tato hra reprezentuje kategorii her, které využívají neuronové sítě a strojové učení k vytváření oponentů do hry, avšak nejsou o samotném trénování.

1.2.2 Analýza

Tato hra má velice působivé oponenty řízené umělou inteligencí. Dokáží se poměrně věrohodně naučit lidské chování ve hře a dokonce se i přizpůsobit stylu konkrétního hráče. Problém je, že toto je pouze modernější řešení starého problému umělých oponentů ve hrách. Z pohledu hráče se to jeví poměrně stejným způsobem, jako například rozhodovací stromy. Cílem hry není trénovat tyto oponenty, ale závodit ve virtuálních autech. Neuronové sítě zde jsou jen proto, aby bylo proti čemu závodit.

1.2.3 Poznatky

Z této hry si odnáším následující poznatky:

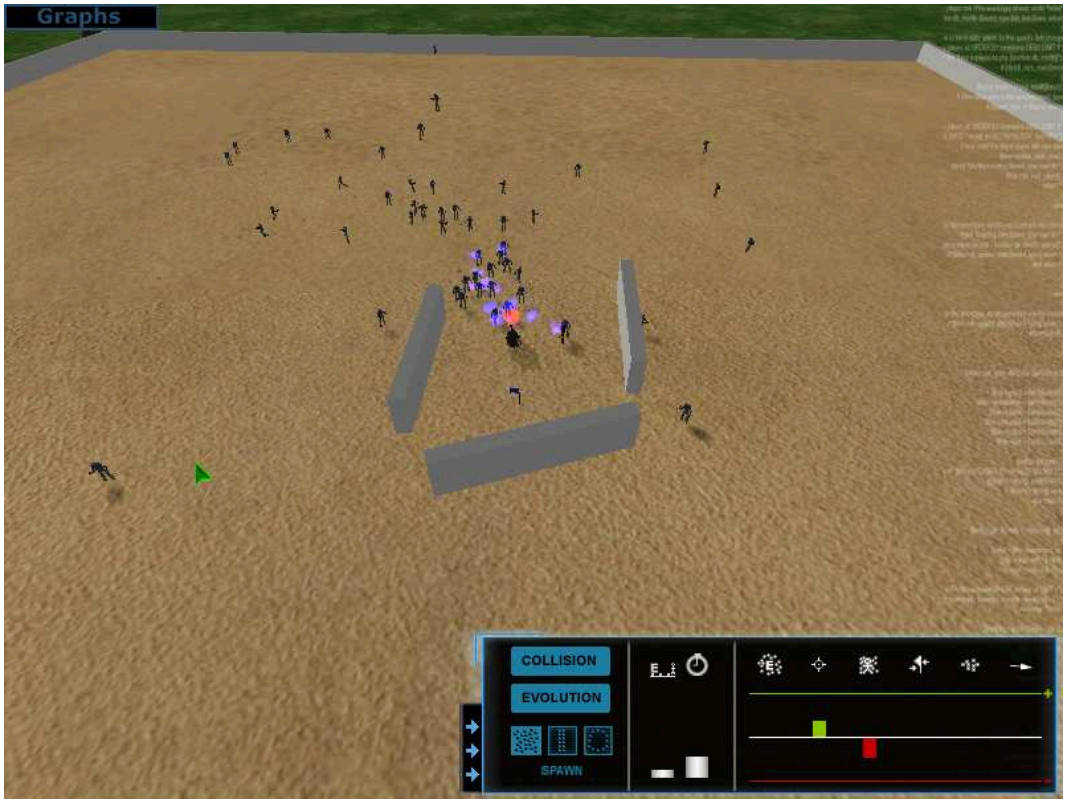
1. Cílem hry je trénovat umělou inteligenci a řešit s ní problémy.

1.3 Nero

1.3.1 Popis

NERO (Neuroevolving Robotic Operatives) je hra z roku 2005, ve které má hráč za úkol vycvičit tým robotických vojáků (dále agenti) na souboj s ostatními robotickými vojáky. Hra k tomu využívá algoritmus rtNEAT (real-time Neuroevolution of Augmenting Topologies) - zajímavou adaptaci klasického evolučního algoritmu do učení v reálném čase - a hráč tak může sledovat, jak se agenti učí v reálném čase. K cvičení agentů hráči slouží nástroj umožňující usazovat objekty do prostředí a uživatelské rozhraní umožňující pomocí posuvníku nastavovat za jaké akce a do jaké míry jsou agenti hodnoceni. (viz obrázek 1.2). Hráč má možnost agentům přidělovat hodnocení za: přiblížení se k nepříteli, zasažení cíle, blízkost k vlajce, za shlukování a za držení pozice. Do prostředí může usazovat stěny, statické či pohyblivé nepřátelské cíle nebo již předtrénovaný tým agentů.[3]

NERO reprezentuje kategorii her, které jsem při mém původním průzkumu hledal, a jejichž nedostatek mě motivoval k vytvoření nové hry. Tyto hry mají za cíl



Obrázek 1.2: Snímek obrazovky ze hry NERO

trénovat agenty a poskytují hráči možnost do trénování zasahovat. Toto je jediná hra, která jsem v této kategorii našel.

1.3.2 Analýza

Hra obsahuje několik elementů, které jsem hledal. Umožňuje hráči zasahovat do trénování tím, že jednak tvoří prostředí, ve kterém se agenti pohybují, a jednak tím, že agentům uděluje hodnocení za žádané nebo odebírá za nežádané chování. Toto dává možnost hráči učit agenty dle své ideální představy. Ačkoli přesně tento styl hry jsem hledal, tato hra stále nemá vše, co bych od ní chtěl. Má pouze 6 obecných možností, za co je možné agenty hodnotit, a cíl je neustále stejný. Pokud by chtěl hráč vycvičit vojáky tak, aby útočili na nepřítele nějakým konkrétním kreativním způsobem, nebo udělali otočku před každým výstřelem, hra mu to neumožňuje. Myslím si, že jde vytvořit hru, kde je mnohem větší kontrola nad chováním agenta, a o to bych se chtěl pokusit.

1.3.3 Poznatky

1. Předdefinovaná specifická hodnotící pravidla jsou příliš neflexibilní pro podporu kreativity.

2 Definice cílů

S pomocí poznatků získaných v předchozí kapitole jsem si definoval následující cíle pro konečnou hru, kterou vytvářím, a pro minimální životaschopný produkt (MVP) implementovaný v rámci této práce.

2.1 Cíle hry

Cíle pro konečný produkt hry, kterých se hra snaží dosáhnout, jsou:

1. Umožnit hráči učit agenty dle své libosti
2. Podporovat u hráče tvorbu kreativních řešení
3. Poskytnout zajímavé problémy pro agenty k vyřešení

2.2 Cíle minimálního životaschopného produktu

Minimální životaschopný (MVP) je produkt s nejmenší možnou funkcionalitou, a v této práci je jeho cílem ověřit, zda použité technologie fungují a hra má potenciál zábavy. Cílem není hra se všemi očekávanými funkcemi, ani hra bez chyb, ale hra, která:

1. Lze spustit a hrát
2. Umožňuje trénování agentů
3. Umožňuje hráči ovlivňovat trénování agentů
4. Poskytuje alespoň jeden zajímavý problém k vyřešení, který ve hře lze úspěšně vyřešit
5. Lze použít pro testování s hráči
6. Lze použít pro další vývoj

3 Zpětnovazebné učení

Zpětnovazebné učení (český překlad Reinforcement Learning) je odvětví strojového učení zabývající se sekvencemi rozhodnutí.

3.1 Princip

Zpětnovazebné učení využívá agenta simulovaného uvnitř prostředí, který v každém kroku simulace provádí nějaké *akce* a z prostředí dostává *pozorování* a *odměny*. Algoritmus zpětnovazebního učení se na základě pozorování obdržené agentem během interakce agenta s prostředím snaží zvolit akce agenta tak, aby maximalizoval dosažené odměny.[4] Ve velice zjednodušené představě je zpětnovazební učení učení stylem pokus-omyl. V rámci této práce budu používat následující terminologii:

Pozorování Pozorování je měření dat z prostředí, které nějakým způsobem reprezentuje stav v prostředí. Pozorováním může být například obraz kamery, pozice agenta, rychlost agenta, vzdálenost objektů od agenta a podobně.

Akce Je činnost, kterou agent provádí v prostředí, například chůze, skok nebo pohyb rukou.

Odměna Odměna je měření výkonu agenta v relaci k cíli učení. Obvykle se jedná o reálné číslo poskytované v každém kroku simulace. Odměny pozdějších kroků v simulaci se obvykle promítají do odměn předchozích kroků s určitým diskontním faktorem. Myšlenka promítání odměn je taková, že pokud akce v jednom kroku nevede k okamžité odměně, ale umožnila dosažení odměny v dalších krocích, tak akce byla výhodná. Diskontní faktor je zde pro rozlišení dvou sekvencí akcí, které nakonec dostanou stejnou odměnu, ale jedna ze sekvencí je výhodnější. Je výhodnější se do cíle dostat rovnou, než kolem něj mnohokrát kroužit a teprve poté se do něj dostat. Bez diskontního faktoru by tyto dvě sekvence měly identické odměny a z pohledu agenta byly stejně výhodné.

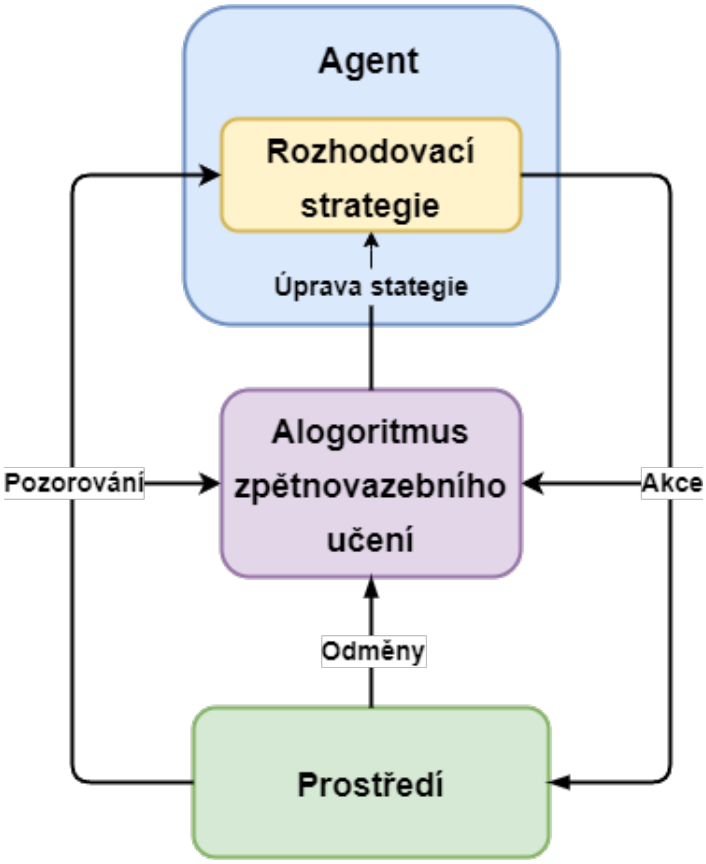
Rozhodovací strategie (volný překlad Decision policy) je produkt algoritmu zpětnovazebního učení a jedná se o mapování stávajících pozorování z prostředí na akce. Dá se říci, že jde o libovolný mechanismus určující agentovi akce. Častou implementací rozhodovací strategie je neuronová síť, která dostává pozorování od agenta jako

vstup, a výstup odpovídá jednotlivým akcím, které agent může provést. Pokud se mluví o *stochastické rozhodovací strategii*, myslí se tím rozhodovací strategie, která nemá deterministické mapování, ale mapuje pozorování z prostředí na pravděpodobnostní rozložení přes všechny akce.

Zkušenost Je záznam sesbíraných pozorování, akcí a odměny agenta v jednom kroku (momentu) simulace.

Epizoda Je série zkušeností, které spolu souvisí a odpovídají jednomu pokusu agenta o splnění cíle.

Celý proces lze analogicky přirovnat k loutkovému představení. Agent je loutka, prostředí je scéna, rozhodovací strategie je loutkoherec, akce jsou pohyby loutky, pozorování je to, co loutkoherec vidí, odměna je reakce publika na představení, zkušenost je paměť loutkoherce v daném okamžiku, a epizoda je jedno loutkové představení. Zpětnovazební algoritmus je způsob, kterým se loutkoherec učí ze svých představení. Proces zpětnovazebního učení je vyobrazen na obrázku 3.1



Obrázek 3.1: Diagram zpětnovazebního učení - upraveno [5]

3.2 Proximal Policy Optimization

Proximal Policy Optimalization (dále PPO) je jedna z rodin algoritmů zpětnovazebního učení využívající algoritmy strojového učení s učitelem, jako jsou SGD a ADAM pro optimalizaci náhražkové funkce.[6] PPO střídá fáze sběru zkušeností (dat) z prostředí a optimalizaci rozhodovací strategie. Jednou z hlavních myšlenek PPO je zaručit, aby se rozhodovací strategie příliš nezměnila během jedné aktualizace, což zaručuje náhražková funkce, která je definovaná ve vzorci 3.1.

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{3.1}$$

$$r_t(\theta) = \frac{\pi_\theta(s_t|a_t)}{\pi_{\theta old}(s_t|a_t)} \tag{3.2}$$

Kde $\hat{E}_t[...]$ je střední hodnota indikující empirický průměr várky dat, $\pi_\theta(s_t|a_t)$ je pravděpodobnost zvolení akce a ve stavu s v kroku t podle rozhodovací strategie π s parametry θ, $_{old}$ jsou parametry rozhodovací strategie před aktualizací parametrů, $r_t(\theta)$ je poměr pravděpodobností zvolení akce a ve stavu s v kroku t před a po aktualizaci parametrů (viz vzorec 3.2), ϵ je parametr přijatelného prahu divergence mezi starou a novou rozhodovací strategií během aktualizace gradientu a $\hat{A}_t$ je odhad výhody akce.

Odhad výhody akce $\hat{A}_t$ lze vypočítat různými způsoby. Pravděpodobně nejpoužívanější způsob je metoda využívající kritika (Actor-Critic Method) podle rovnice 3.3, kde hodnota $V_{\theta_{VF}}(s_t)$ je odhad dosažených odměn ve stavu s_t vypočítaný neuronovou sítí s parametry θ_{VF} optimalizovanou vůči kriteriální funkci L^{VF} definovanou v rovnici 3.5 . V rovnicích 3.3 a 3.4 je γ diskontní faktor, λ volný parametr a T je délka epizody. V rovnici 3.5 je $V_t^{targ})^2$ odměna skutečně obdržena v kroku t podle pozorování z prostředí.

$$\hat{A}_t = \delta_t + (\gamma\lambda)^1\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-t} \tag{3.3}$$

$$\delta_t = r_t + \gamma V_{\theta_{VF}}(s_{t+1}) - V_{\theta_{VF}}(s_t) \tag{3.4}$$

$$L^{VF} = \hat{E}_t[(V_{\theta_{VF}}(s_t) - V_t^{targ})^2] \tag{3.5}$$

Rovnice jsou přejaty z [6] a upraveny pro potřeby této práce. Původní rovnice se zabývají variantou PPO, ve které kritik i optimalizovaná rozhodovací strategie sdílejí prarametry θ (platí $\theta = \theta_{VF}$). V mé práci používám variantu PPO, ve které rozhodovací strategie používá prarametry θ a kritik používá vlastní parametry θ_{VF}. V původní rovnice také používají kriteriální funkci, která je kombinací L^{VF} a L^{CLIP}. Použitá varianta PPO je popsána následujícím pseudokódem:

Algoritmus 1: Proximal Policy Optimization s kritikem

```
for iterace = 1, 2, ... do
    for agent = 1, 2, ..., N do
        Spusť rozhodovací strategii π_{θ_old} v prostředí po T kroků
        Vypočítej odhady výhod Â_1, Â_2, ..., Â_T
    for epocha = 1, 2, ..., K do
        Maximalizuj funkci L^{CLIP} vzhledem k parametrům θ,
         s velikostí várky M kde M ≤ NT
        Minimalizuj funkci L^{VF} vzhledem k parametrům θ_{VF}
    Aktualizuj původní parametry: θ_{old} ← θ
```

4 Nástroje pro tvorbu her

V dnešní době je standard používat pro vývoj her herní engine. Jedná se o sadu nástrojů, knihoven a frameworků, které umožňují vývojářům vytvářet různé typy her nebo animací a animovaných filmů. Herní engine obvykle poskytuje nástroje pro tvorbu grafického rozhraní, scén, herních objektů, nástroje pro simulaci fyziky, zvukové efekty, vykreslování grafiky v reálném čase, animace, sběr a interpretaci vstupů od uživatele a další běžně potřebné nástroje pro všechny druhy her.

Pro tento projekt jsem zvažoval použití tří herních enginů: Unity, Unreal a Torque.

4.1 Torque engine

Obrázek 4.1: Logo herního enginu Torque

Torque je C++ engine postavené na dlouhé historii vývoje. Torque enginy jsou od roku 2012 open-source s MIT licencí a vytvořily si širokou komunitu podporující tento engine. Torque je ve dvou variantách Torque3D a Torque2D pro vývoj 2D a 3D her.[7] Pomocí Turque eningu byly vytvořeny hry jako jsou Ultimate Duck Hunting, Wildlife Tycoon: Venture Africa nebo NERO (viz sekce1.3).

4.2 Unreal engine

Obrázek 4.2: Logo herního enginu Unreal

Unreal je jeden z nejznámějších herních enginů na světě. S jeho pomocí byly vytvořeny hry jako jsou Fortnite, Gears of War, BioShock nebo XCom-2. Je známý zejména pro svoji schopnost vykreslovat fotorealistickou grafiku ve vysoké kvalitě ve vysoké rychlosti.

Společnost Epic Games vyvýjející herní engine Unreal také nově oznámila nové nástroje pro trénování agentů v Unreal enginu pod názvem Learning Agents, využívající algoritmy strojového učení v jazyce Python a frameworku PyTorch. [8]

4.3 Unity engine

Obrázek 4.3: Logo herního enginu Unity

Unity je multiplatformní vývojářský software pro tvorbu her, pomocí které byly vytvořeny hry jako Hearstone, Cuphead nebo Pokémon Go. Obsahuje nástroje pro renderování, simulaci fyziky a grafické uživatelské rozhraní Unity Editor. Unity je známé svojí historickou přívětivostí pro nováčky, širokou a aktivní komunitou vývojářů, širokou podporu platforem a svojí licencí, která umožňuje jeho využití pro vývoj her zdarma pro jednotlivce a společnosti s ročním prodejem nebo financováním pod 100 000$. Unity je kromě oblasti videoher používáno v oborech architektury, strojírenství, v auto průmyslu a ve filmovém průmyslu. Unity je také proslulý svojí rychlostí při prototypování her, což je velice relevantní vlastnost pro tento projekt. Unity využívá jazyk C# a je postaven na implementaci frameworku .NET pro platformu Mono.[9]

Zároveň je engine Unity doplněn nástroji pro strojové učení The Unity Machine Learning Agents Toolkit (dále jen ML-Agents), které poskytují implementaci algoritmů strojového učení založené na frameworku PyTorch. [9]

5 Návrh hry

Hru jsem navrhl jako plošinovou skákačku lehce připomínající ikonickou Super Mario Bros. hru od společnosti Nintendo. V této hře hráč ovládá pomocí klávesnice hlavní postavu italského instalatéra Maria a snaží se sním překonat úroveň tvořenou plošinami, nepřáteli, trubkami a jinými objekty, které mají určité efekty podle pravidel hry. Postava Mario má schopnost běžet na levou nebo pravou stranu, skočit a skrčit se.

Pro moji hru jsem si vzal tento základní koncept a provedl na něm tři zásadní změny:

1. Akce postavy neovládá hráč, ale neuronová síť
2. V úrovni může být více než jedna hlavní postava
3. Schopnosti postavy si hráč může přizpůsobit pro každou postavu

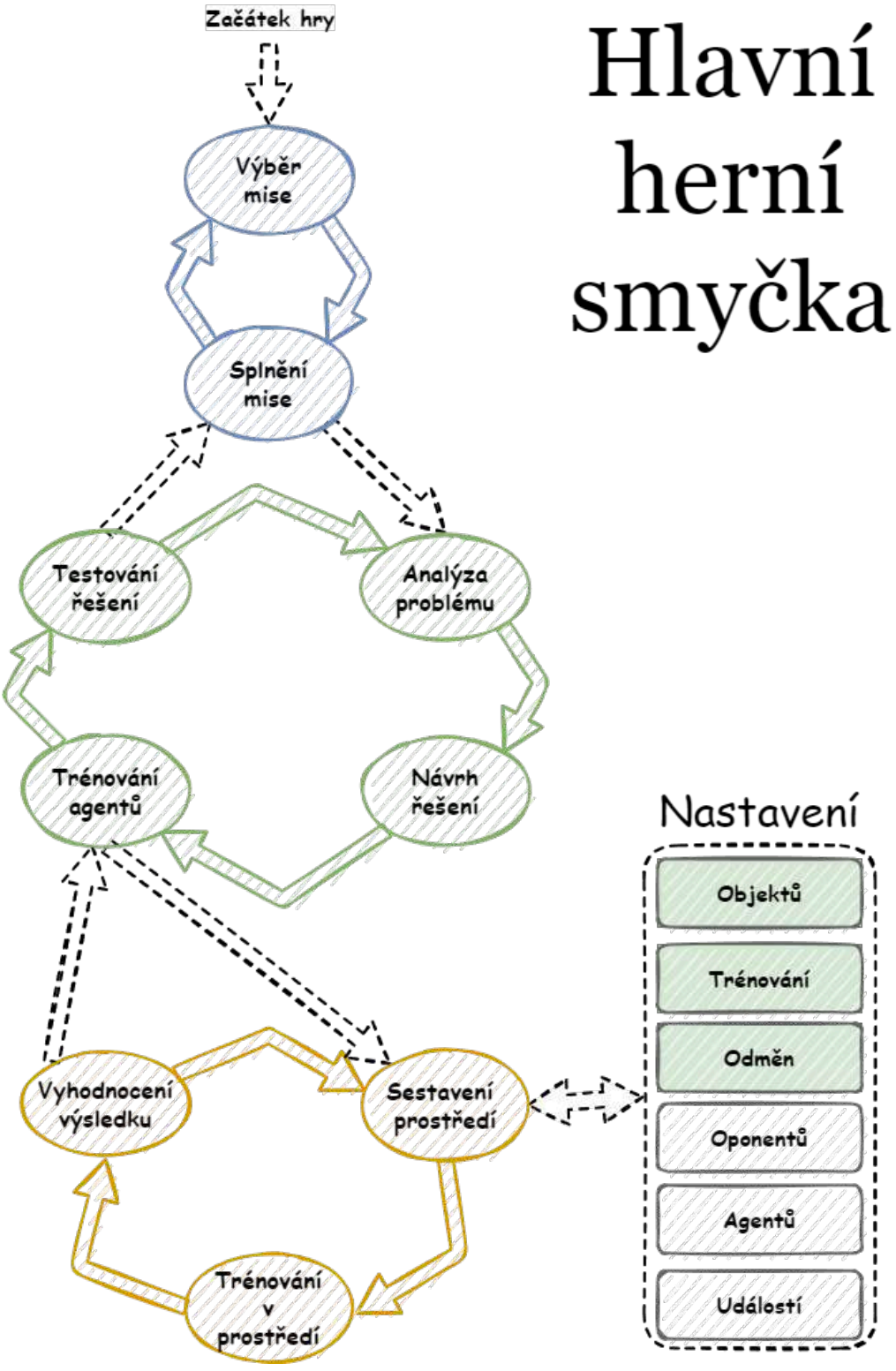
Úkolem hráče tak není přímo ovládat hlavní postavu a navigovat s ní skrz úroveň, ale pomocí nástrojů hry vytvořit agenty, kteří jsou schopni sami překonat úroveň. Jelikož agentů může být více, toto dává možnost pro vytváření úrovní, které vyžadují kooperaci více postav s různými schopnostmi. V této části práce budu mluvit o schopnostech a dovednostech - tato dvě slova v rámci této práce nejsou zaměnitelná synonyma, mají však specifický význam, který zde definuji:

Schopnost Je akce nebo činnost, kterou je postava agenta schopna vykonat, jako je například běh nebo skok. Schopnosti jsou postavě přiděleny kódem hry a neuronová síť se je neučí.

Dovednost Je dovednost neuronové sítě využívat schopnosti postavy k dosažení cíle. Dovednosti se musí neuronová síť naučit.

5.1 Hlavní herní smyčka

Hlavní herní smyčka je koncept, který v oboru herního designu popisuje jaké akce, v jakém pořadí hráč provádí během hraní hry. Jde o popis toho, co hráč ve hře dělá. Na obrázku 5.1 jsem vyobrazil můj návrh herní smyčky pro tuto hru.



Obrázek 5.1: Hlavní herní smyčka hry

5.1.1 Výběr mise

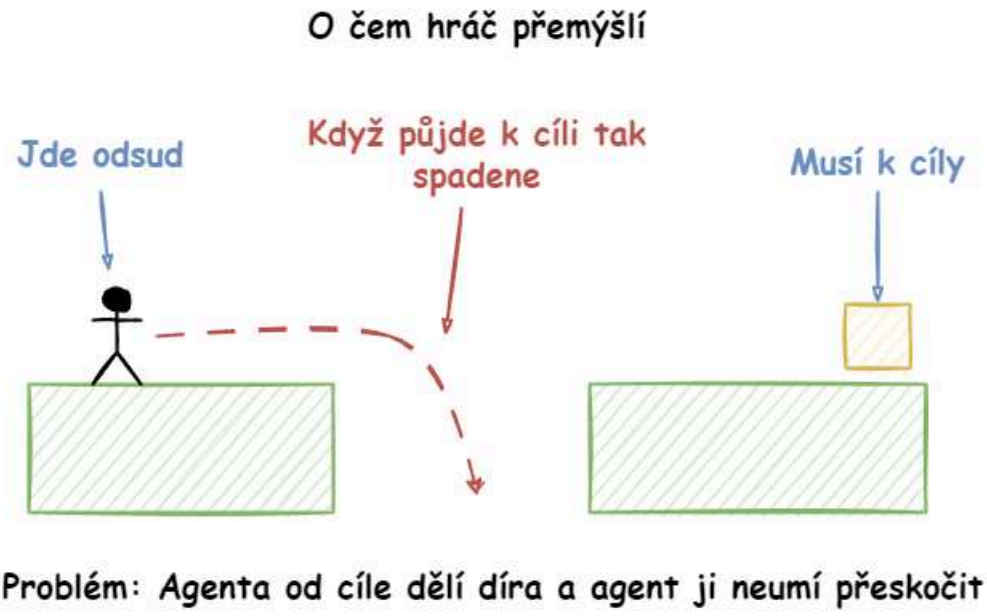
Prvním krokem je výběr mise, kde si hráč vybere jeden z dostupných problémů, který chce řešit. K tomu hráči slouží grafické rozhraní, které hráči zároveň komunikuje stav mise. Návrh grafického rozhraní je dále popsán v sekci 5.3.5.

5.1.2 Splnění mise

Po výběru mise se hráč pokusí misi splnit. Splnění mise bude standardně probíhat ve čtyrech částech. Hráč nejdříve analyzuje, co brání splnění mise (sekce 5.1.3), následně navrhne způsob, jak by mohl tento problém pomocí trénování agentů vyřešit (sekce 5.1.4), natrénuje agenty podle návrhu (sekce 5.1.5) a nakonec vytrénované agenty vyšle do mise, aby se ji pokusily splnit(sekce 5.1.6). Pokud se agentům povede splnit misi, hráč přechází na další misi, v opačném případě se vrací do kroku analýzy problému.

5.1.3 Analýza problému

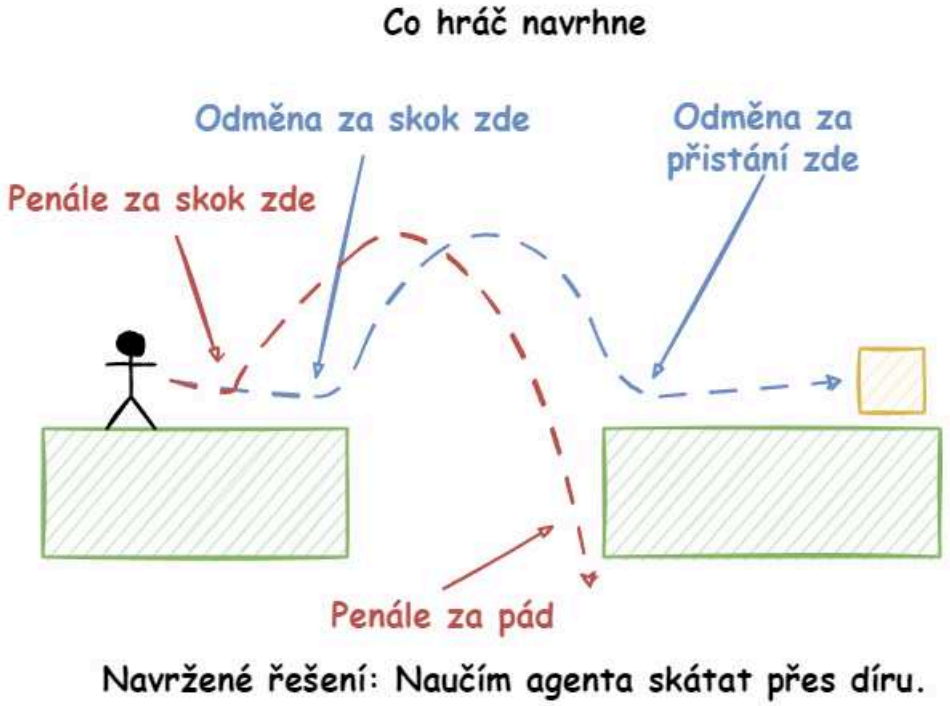
V tomto kroku se hráč podívá na misi, kterou si vybral ke splnění, a analyzuje, proč jeho agenti nejsou schopni tuto misi splnit. Na obrázku 5.2 je vyobrazen možný myšlenkový pochod hráče v tomto kroku při plnění jednoduché mise, která dělí agenta a cíl dírou v podlaze.



Obrázek 5.2: Myšlenkový pochod hráče při analýze mise

5.1.4 Návrh řešení

Když hráč zjistí, jaký problém agentovi brání ve splnění mise, navrhne řešení které agentovi nebo agentům umožní misi splnit. Navržené řešení je nějaká dovednost, kterou agenti potřebují k splnění mise. Na obrázku 5.3 je vyobrazen možný návrh řešení problému popsaného v předchozí sekci.



Obrázek 5.3: Návrh řešení, se kterým hráč může přijít

5.1.5 Trénování agentů

V tomto kroku se hráč pokusí pomocí nástrojů hry a zpětnovazebního učení naučit agenty dovednosti, které si hráč navrhl ve fázi návrhu řešení (viz 5.1.4). Nejdříve sestaví prostředí, ve kterém se agenti budou trénovat (sekce 5.1.7), poté nechá agentu v prostředí trénovat po potřebnou dobu (sekce 5.1.8) a nakonec vyhodnotí, zda trénovací proces proběhl podle jeho představ (sekce 5.1.9). Pokud se hráč rozhodne, že trénování agentů neproběhlo uspokojivě, začne trénování agentů od znovu. V opačném případě se přesouvá k testování řešení.)

5.1.6 Testovaní řešení

V této fázi hra otestuje, zda agenti mají potřebné dovednosti pro splnění mise. Vytrénovaní agenti se vypustí do prostředí mise, a pokud dokáží dosáhnout cíle v požadovaném časovém limitu, agenti a hráč misi splnili a mohou se přesunout na řešení další mise. Pokud agenti nemají dovednosti potřebné k dosažení cílů mise, hráč se přesouvá do fáze analýzy problému a pokouší se splnit misi dále.

5.1.7 Sestavení prostředí

V této fázi hráč sestavuje prostředí, ve kterém se agenti mohou naučit dovednosti z fáze návrhu řešení. Několik různých nástrojů, které mu umožňují nastavovat následující parametry:

Objekty Objekty jsou elementy prostředí, se kterými postava agenta může interagovat. Hráč pomocí grafického rozhraní do prostředí umisťuje objekty jako jsou například terén, cíl, brána, páka, bodák, žebřík a další.

Trénování V rámci trénování hráč může nastavovat jací agenti (postavy a neuronové sítě) se v trénovacím procesu použijí, v jakých prostředích se trénovací proces bude dít, kdy se má trénovací proces zastavit. Agenti se můžou trénovat buďto od začátku s náhodně inicializovanou neuronovou sítí, nebo použít již trénovanou neuronovou síť jako počáteční bod.

Odměny V rámci tvoření trénovacího prostředí hráč také pomocí hodnotících pravidel nastaví, za co se agentovi v rámci trénovacího procesu budou udělovat odměny a penalizace. Hráč tak může specifikovat akce, které jsou od agenta žádoucí a nežádoucí, a agent se při trénovacím procesu snaží maximalizovat odměny. Hodnotící pravidla mohou kontrolovat některé skutečnosti během trénovaní, jako jsou například: pozice agenta, zda se agent dotýká nějakého prvku prostředí, vzdálenost agenta od určitého bodu, jakou akci provádí nebo prováděl, jak dlouho agent jednal a podobně.

Oponenti V prostředí se také budou moci vyskytovat oponenti, kteří budou jednat proti zájmům agentů. Hráč bude moci nastavovat logiku těchto oponentů pomocí jednoduchých rozhodovacích stromů nebo jako oponenty dosadit jiné vytrénované agenty.

Nastavení agentů Agenti budou modifikovatelní. U každého agenta bude možnost změnit neuronovou síť, která ho řídí, a schopnosti, kterými postava agenta disponuje. V rámci neuronové sítě bude moci hráč nastavit velikost neuronové sítě a typy vrstev, které se použijí (lineární nebo rekurentní). V rámci schopností postavy agenta, bude mít hráč možnost přidávat a odebírat schopnosti, jako jsou například: vidění, slyšení, skákání, lezení po žebříku, střílení ze zbraně, udělat ze sebe stoličku pro ostatní agenty, vykřikování signálů, rozpoznávání vykřikovaných signálů a další.

Události Během interakce agenta s prostřením se můžou provádět události, které prostředí určitým způsobem modifikují. Hráč bude mít možnost definovat, jaké události se stanou a za jakých podmínek. Události prostředí mohou být například: změna terénu, začne pršet, začnou padat kameny, zhroutí se most a další.

Nutno podotknout, že toto je pouze návrh celé hry a nikoliv výčet všech funkcí, které se vyskytují v MVP. Některé z výše zmíněných parametrů nelze v rámci

MVP nastavovat. Parametry nastavitelné v sestavování prostředí rámci v MVP jsou zvýrazněny na obrázku 5.1 v části označené Nastavení se zeleným rámečkem.

5.1.8 Trénování v prostředí

V této fázi hráč pošle agenty do vytvořeného trénovacího prostředí a čeká, než agenti ukončí svůj trénovací proces. Hráč se může dívat na zrychlené jednání agentů v prostředí a sledovat metriky trénovacího prostředí, jako je například průměrná odměna za pokus.

5.1.9 Vyhodnocení výsledku

Poté, co agenti ukončí svůj trénovací proces, hráč vyhodnotí, zda trénovací proces proběhl podle očekávání. Pomáhají mu k tomu metriky sbírané z prostředí a náhled do prostředí během trénovacího procesu. Pokud hráč není spokojený s tím, jak se agenti natrénovali, může proces začít znovu, nebo na proběhlý trénovací proces navázat dalším.

5.2 Návrh agenta

Agent řízený neuronovou sítí bude volit diskrétní akce. To znamená, že každá akce má konečně mnoho možností, jak ji provést. Příkladem diskrétní akce je běh doleva a její nediskrétní ekvivalent by bylo nastavení rychlosti směrem doleva. Takto bude možné implementovat ovládání agenta pro hráče pomocí klávesnice tak, aby přesně odpovídalo možnostem agenta. Neuronová síť se vlastně bude učit, jaké klávesy má stisknout.

Diskrétní akce poskytují mnohem menší prostor možných akcí, který se agent musí během trénování naučit a důsledkem je rychlejší trénování.

5.2.1 Schopnosti postavy

Akce Agent bude mít k dispozici různé skupiny akcí, které bude moci v prostředí provádět. Z každé skupiny akcí může agent provést pouze jednu možnost, kterou pak postava agenta vykoná. Způsob vykonání každé akce bude jasně definován v kódu hry. Každý agent bude mít skupinu akcí běhu s možnostmi běžet doleva, běžet doprava nebo stát, a skupinu akcí skoku s možnostmi skočit nebo neskočit.

Pozorování Agent bude mít skupiny senzorů, které budou neuronové síti poskytovat pozorování, na jejichž základě se může rozhodovat. Základním senzorem bude zrak, kterým bude agent schopen detekovat objekty před sebou v určitém úhlu v určité vzdálenost.

5.2.2 Modifikace agenta

Schopnosti Kromě základních schopností definovaných v 5.2.1 bude hráč mít možnost přidávat jednotlivým agentům schopnosti dle potřeby. Tyto schopnosti budou předem definované a bude se jednat o nové skupiny akcí nebo o nové senzory. Například hráč bude moci agentovi dát schopnost provádět akci plazení nebo výkřiku, a senzor na poslouchání výkřiků nebo na sledování průběhu času.

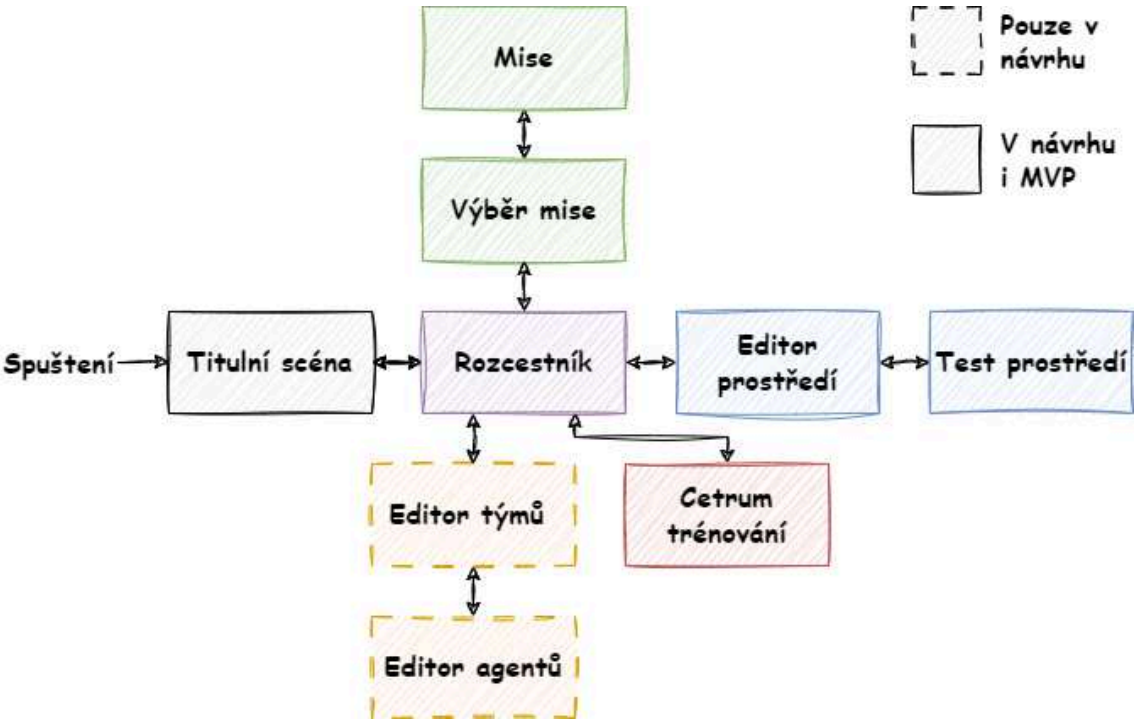
Neuronová síť Další možností bude modifikovat topologii neuronové sítě, která agenta řídí.K dispozici bude možnost přidat další vrstvy, zvýšit počet neuronů ve vrstvách nebo přidat rekurentní vrstvu, která agentovi poskytne schopnost paměti.

5.2.3 Týmy agentů

Agenti budou moci jednat buďto sami nebo v rámci týmu s ostatními agenty a každá mise bude mít maximální počet agentů. Týmy agentů si bude moci hráč vytvářet z již natrénovaných agentů a pak je dále trénovat buď samostatně nebo dohromady.

5.3 Scény ve hře

V této části představím návrh uživatelského rozhraní ve hře. Rozhraní hry je rozloženo na několik obrazovek/scén, které jsou vyobrazeny společně s přechody mezi scénami na obrázku 5.4. Tyto scény jsem navrhoval s pomocí aplikace pro návrh rozhraní Figma.



Obrázek 5.4: Mapa scén a přechodů mezi scénami

5.3.1 Editor prostředí

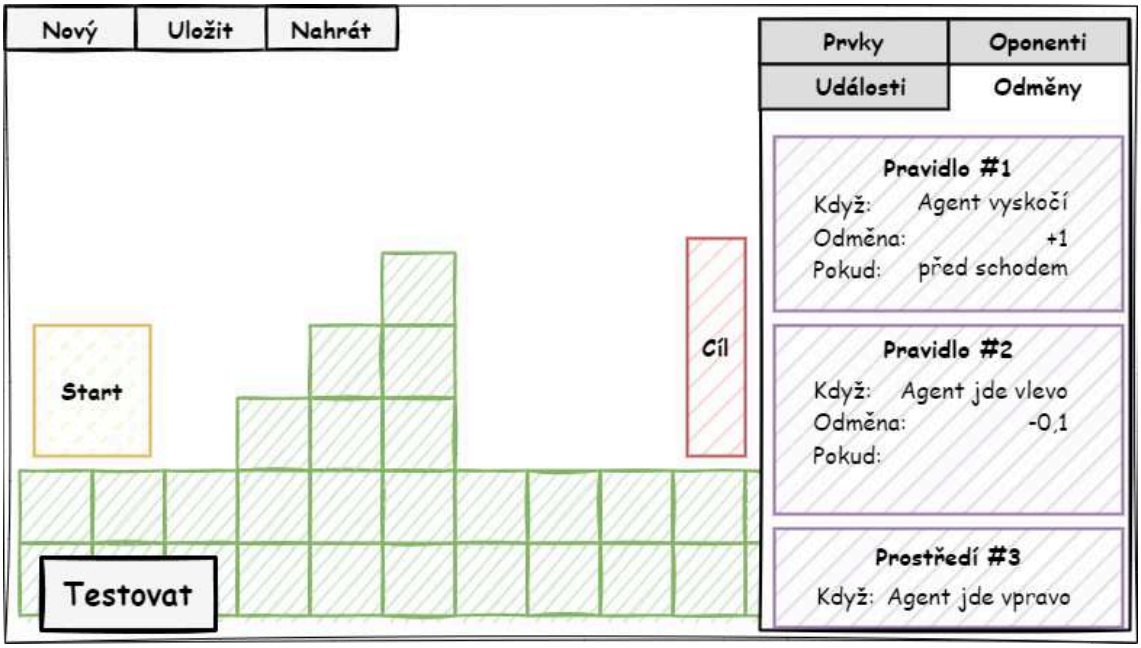
Editor prostředí je nejdůležitější část hry. Zde hráč bude trávit nejvíce času, protože zde tvoří prostředí a hodnotící pravidla, která se následně použijí pro trénovací proces. Tato scéna se skládá ze dvou hlavních částí. Na levé straně je volný prostor, ve kterém se bude nacházet tvořené prostředí, a na pravé straně je panel obsahující nástroje pro úpravu prostředí. Panel obsahuje několik záložek, které poskytují nástroje pro úpravu v různých dimenzích.

Záložka prvky Zde bude seznam prvků, které hráč může umístit do prostředí. Prvek si kliknutím na jeho ikonu může zvolit a pak na levé straně obrazovky umisťovat pomocí levého tlačítka myši nebo mazat pomocí tlačítka pravého. Prvky můžou být například bloky terénu, cíle k dosažení, pasti, zdi, páky, brány nebo dveře.

Záložka hodnotící systém V záložce hodnotícího systému bude seznam pravidel, které určují kdy, jaké a za jakých podmínek agent dostane odměny. Každé pravidlo bude vždy reagovat na určitou událost v prostředí a bude obsahovat sadu podmínek, které budou chování pravidla modifikovat.

Záložka oponentů V této záložce hráč bude moci nastavovat chování oponentů. Agenti se budou moc chovat různými způsoby. Jeden způsob je podle stanoveného plánu definovaného hráčem, jako například, že oponent poběží jedním směrem dokud nenarazí a poté se obrátí a poběží na druhou stranu. Druhý způsob je použití již natrénovaných agentů jako oponentů pro jiné agenty. Hráč zde bude mít rozhraní pro stanovení plánů a pro výběr agentů-oponentů.

Záložka událostí Zde bude hráč nastavovat dynamické chování prostředí. Budou zde nástroje pro stanovení změn v prostředí během trénování. Události se budou dít podle hráčem stanovených pravidel a efekt události bude také pod kontrolou hráče. Příklady událostí, které hráč může nastavit jsou: každé dvě sekundy se otevře nebo zavře brána, po poražení oponenta se do prostředí přidají dva další oponenti, po uplynutí tří sekund zmizí kus podlahy nebo po uplynutí deseti sekund, pokud je agent není ve stanovené zóně, tak se epizoda ukončí a agent dostane negativní odměnu.

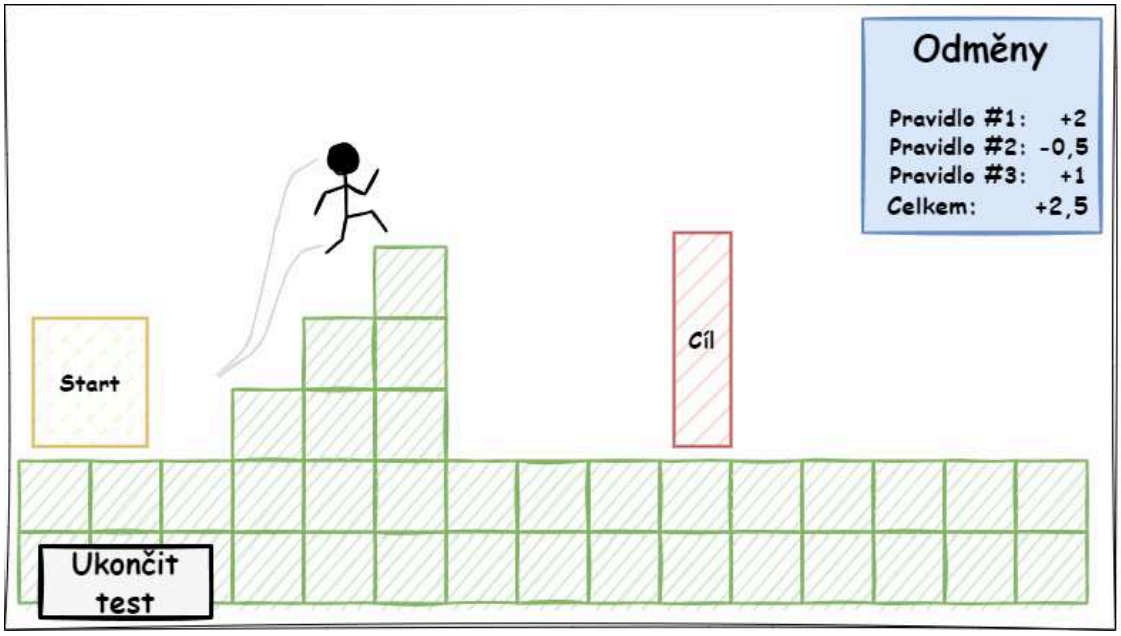


Obrázek 5.5: Návrh rozhraní pro editor prostředí

5.3.2 Test prostředí

Poté, co si hráč postavil trénovací prostředí v Editoru prostředí, má možnost si prostředí vlastnoručně otestovat v této scéně. Tato scéna navazuje přímo na editor

prostředí a hráč v ní ovládá agenta pomocí klávesnice. Toto umožňuje hráči vyzkoušet, jak hodnotící pravidla vyhodnocují akce agenta (k tomu mu slouží čítač odměn), a zároveň může ověřit schopnosti agenta pohybovat se po prostředí. Hráč může kdykoliv test ukončit a vrátit se do editoru prostředí pro další změny.



Obrázek 5.6: Návrh testovací scény prostředí

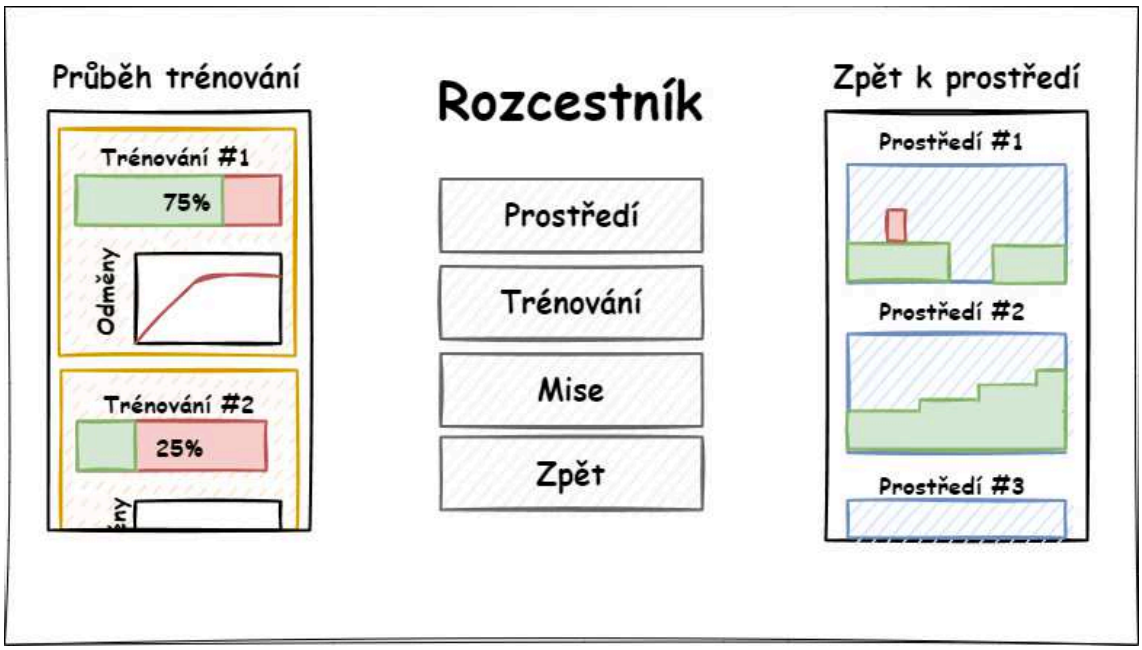
5.3.3 Titulní scéna

Titulní scéna je zde pro upoutání hráče hned, jakmile spustí hru a zachování dobrého prvního dojmu. Scéna je doplněna grafikou, která sděluje hráči, proč a jak je hra pro něj zajímavá. Účel je především upoutat pozornost a posluhovat jako základní menu. Titulní scéna je hráči očekávaný standard.

Obrázek 5.7: Návrh titulní scény hry

5.3.4 Rozcestník

Účel rozcestníku je nasměrovat hráče do jednotlivých částí hry. Jedná se o jednoduchou scénu, která slouží jako domovská obrazovka. Zároveň také slouží jako hlavní přehled všeho, co se děje - jaké scény má hráč zrovna rozdělané, jaké trénování zrovna probíhá, jací agenti se v nich trénují a jak daleko je trénovací proces od konce.

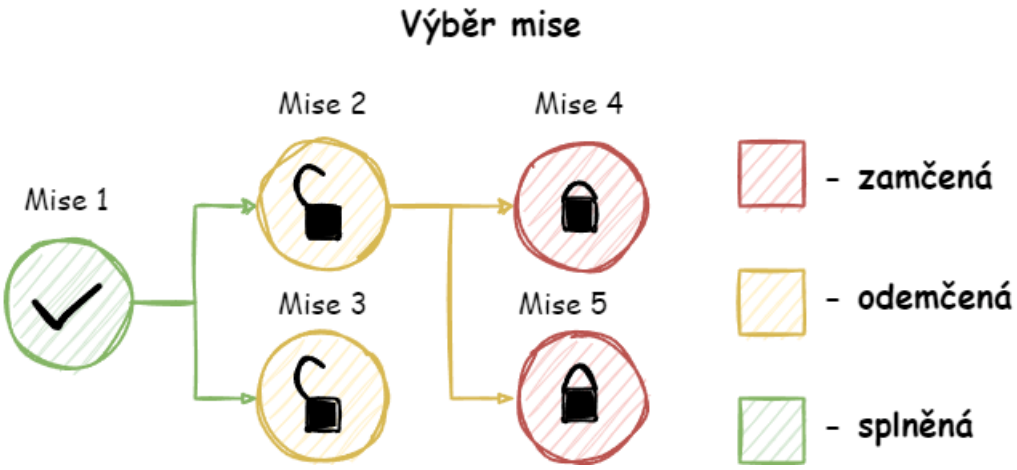


Obrázek 5.8: Návrh scény rozcestníku

5.3.5 Výběr mise

V této scéně si hráč vybírá, kterou misi chce zkusit splnit. Slouží mu k tomu graf (viz 5.9), který má na každém vrcholu jedno tlačítko odpovídající jedné misi ve hře, které zároveň komunikuje stav mise.

Stav mise může nabývat tří hodnot: zamčeno, odemčeno a splněno. Hráč se nemůže pokusit splnit zamčenou misi, hráč se pokouší splnit jednu z odemčených misí a splněné mise jsou takové, ve kterých hráč již alespoň jednou splnil cíl mise jakýmkoliv způsobem. Mise na sebe navazují, každá mise (mimo úvodních) má jednu nebo více předchozích misí. Pokud některá z předchozích misí není splněná, mise je zamčena. Hráč může spustit misi kliknutím na tlačítko požadované mise.

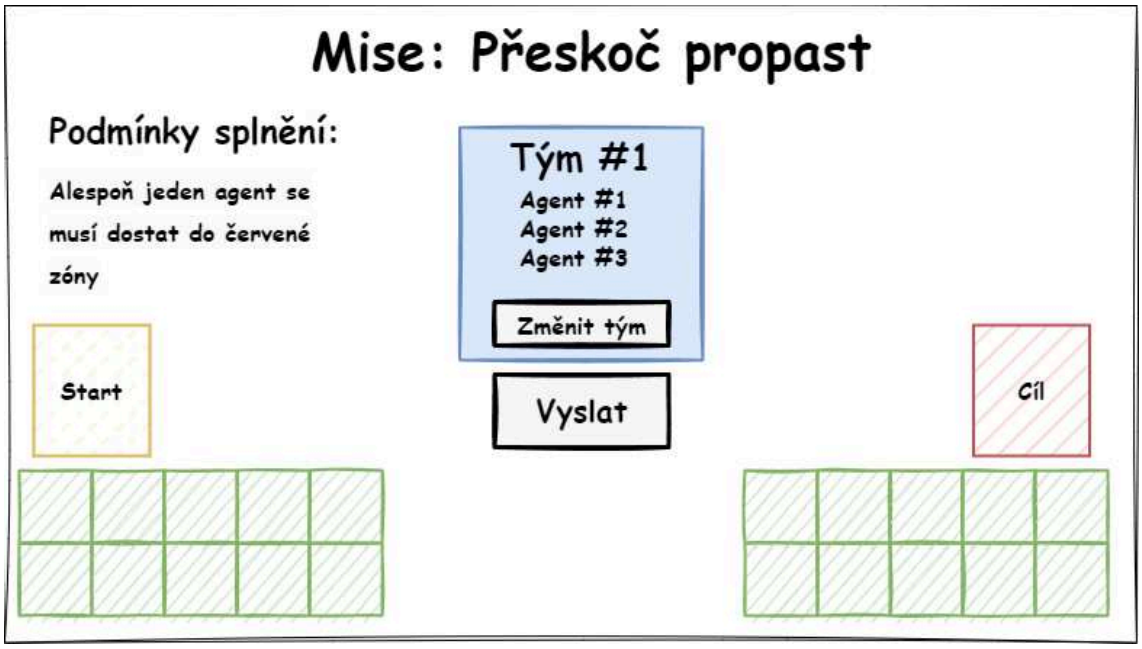


Obrázek 5.9: Návrh rozhraní pro výběr mise

5.3.6 Mise

V této scéně hráč vidí problém, který jeho trénovaní agenti musí překonat. V misi je prostředí podobné trénovacímu prostředí, které hráč vytvářel v Editoru prostředí (viz 5.3.1).

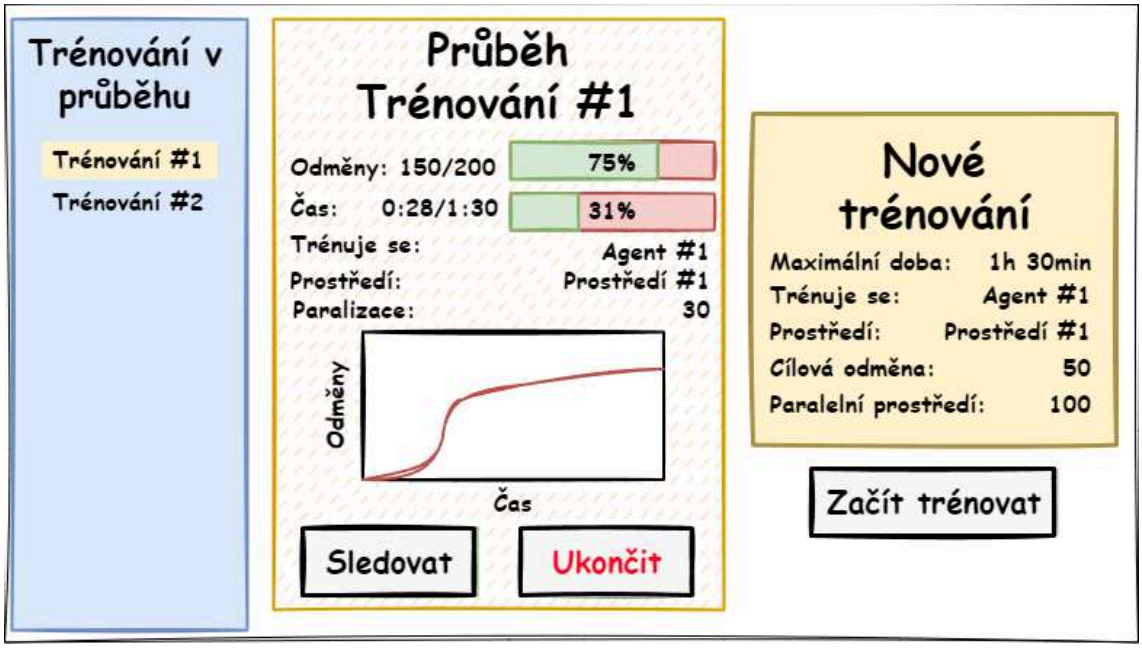
Je zde tlačítko, kterým může hráč vyslat své agenty do prostředí a GUI pro výběr agentů. GUI také obsahuje informace o tom, jaké jsou podmínky pro splnění mise, jaký je limit agentů a jméno mise, které může sloužit jako nápověda pro hráče. Mise indikuje místo, kde se agenti po jejím zahájení v prostředí objeví.



Obrázek 5.10: Návrh scény mise

5.3.7 Centrum trénování

V této scéně hráč nastavuje trénovací proces. Má zde možnost nastavit, jací agenti se trénují na jacích prostředích, jak dlouho se má trénovat, jaká je míra paralelizace, jaké jsou pravidla pro předčasné zastavení a zároveň tato scéna slouží k poskytnutí bližších informací o průběhu trénování.

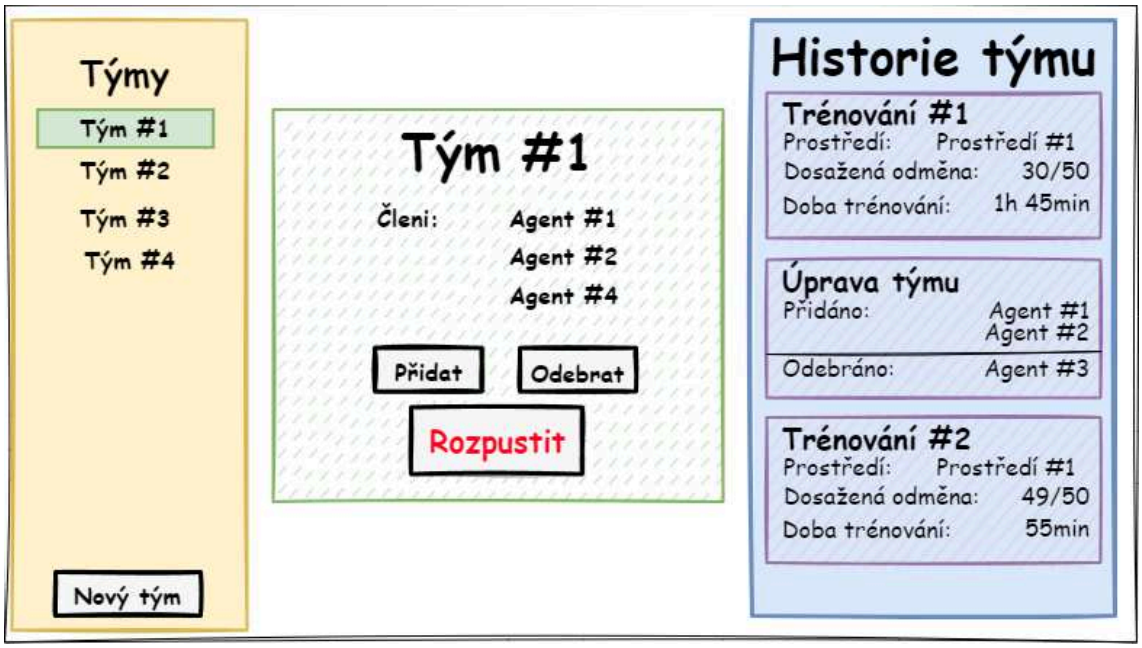


Obrázek 5.11: Návrh scény trénovacího centra trénování

5.3.8 Editor týmů

V této scéně hráč sestavuje týmy agentů pro trénování a plnění misí. Týmy se můžou skládat z libovolné kombinace agentů a týmů agentů. Tato scéna také může hráče přesměrovat na editor agentů.

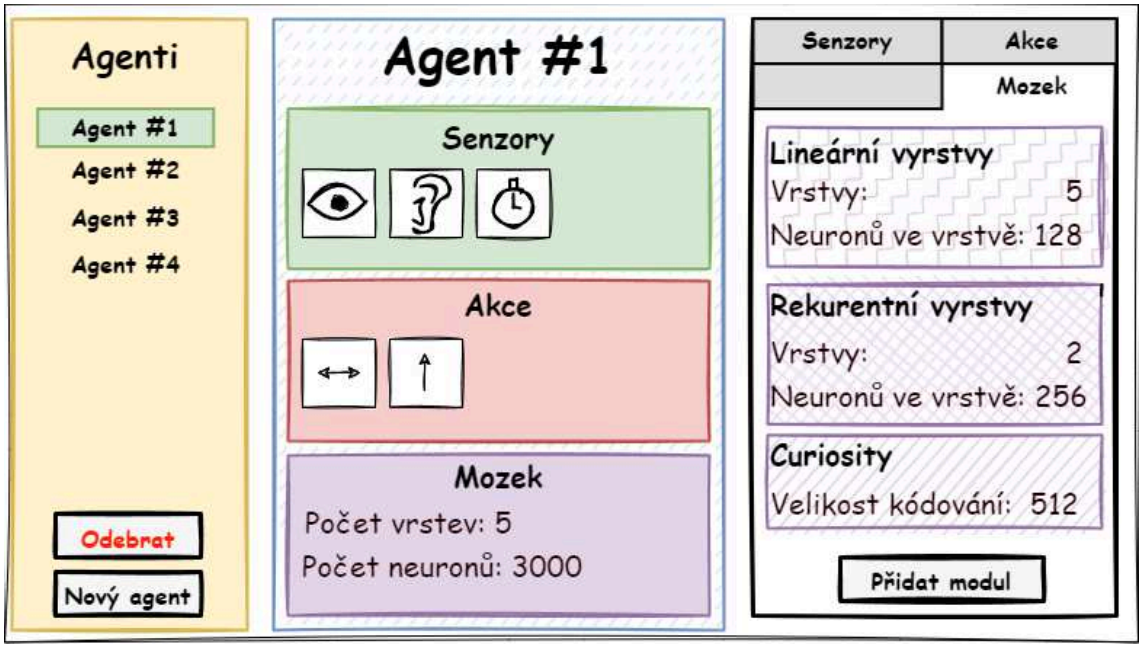
Hráč zde také vidí historii vybraného týmu agentů, která nese informace o tom, jak dlouho, na jakých trénovacích prostředích a s jakými výsledky byl tým trénován.



Obrázek 5.12: Návrh scény editoru týmů

5.3.9 Editor agentů

V této scéně hráč upravuje schopnosti agentů. Má možnost jednotlivým agentům přidávat nebo odebírat senzory a akce, které je postava agenta schopna provést. Zároveň se zde nastavuje, jakou topologii neuronové sítě agent využívá.



Obrázek 5.13: Návrh scény editoru agentů

5.4 Návrh minimálního životaschopného produktu

Návrh hry je příliš velký a složitý na to, aby se implementoval celý najednou. Navíc je pouze teoretický a pracuje s neověřenými předpoklady o tom, co je nezbytné, užitečné a jak hráči hru budou hrát. Z toho důvodu je nutné nejdříve implementovat MVP, které otestuje některé z těchto předpokladů. Očekávám, že návrh hry se po otestování MVP do jisté míry změní, obzvlášť v detailních rozložení jednotlivých funkcí a systémů. Rozhodl jsem se proto velké množství návrhu v MVP neimplementovat, a soustředit se pouze na nejnutnější funkce potřebné k otestování. Konkrétní scény, které jsem se rozhodl z MVP vypustit, jsou na obrázku 5.4 znázorněny čárkovaným rámečkem.

5.4.1 Agent

Z návrhu agenta jsem se rozhodl vypustit jakoukoliv modifikaci. Modifikace agenta pouze zvyšuje komplexitu a hloubku hry, není však nutná k hraní, a proto není nutná ani v MVP. Agent bude schopen vykonávat pouze základní pohyby běhu a skoku a bude používat pouze jeden senzor zraku. Topologie neuronové sítě ovládající agenta bude pevně daná bez možnosti modifikace. Z tohoto důvodu jsem z MVP vypustil celou scénu **Editor agentů**, protože bez možnosti měnit schopnosti agenta je scéna bezvýznamná a zbytečná.

Ze stejných důvodů jsem vypustil také podporu více agentů a scénu **Editor týmů**. V každé misi a v každém trénování bude pouze jeden agent. Nemá příliš smysl implementovat podporu více agentů v momentě, kdy ještě s jistotou nevím, zda hra funguje alespoň pro jedno agenta.

5.4.2 Scény

Ne všechny funkce v návrhu jsou nutné, v této podkapitole u každé scény uvádím, jaké funkce jsem se rozhodl zahrnout do MVP.

Titulní scéna Zanechal jsem v MVP jako zástupný symbol pro zachování struktury hry. Největší nárok na implementaci je vytvoření zajímavého grafického zpracování, které však z MVP vypouštím, a tak implementace této scény nezabírá žádný významný obnos času. Scéna pouze poskytuje název hry a možnosti pokračovat nebo odejít.

Rozcestník Zanechávám možnosti navigace mezi scénami, vypustil jsem však sledování informací o hře.

Výběr mise Zanechal jsem v MVP se všemi funkcemi v návrhu s jednoduchým grafickým zpracováním.

Mise Zanechal jsem tlačítko na vyslání agenta a název mise. Hra MVP bude mít pouze jednoho agenta, takže informace o limitu agentů je bezvýznamná a stejné platí pro cíl. Každá mise bude mít pouze jeden cíl, a to aby se agent dotkl bedny.

Editor prostředí V panelu pro umisťování prvků prostředí budou pouze tři naprosto nutné prvky. Blok terénu, bedna, která slouží jako cíl agenta, a startovní pozice agenta.

V panelu hodnotícího systému implementuji přidávání a editaci hodnotících pravidel s minimální sadou spouští a podmínek, které umožní dostatečnou kontrolu nad trénováním agenta pro splnění mise a limitovanou kreativitu. Zahrnu tři spouštěče pravidel - uplynutí času, kolizi s prvkem prostředí a akci agenta, a tři podmínky - kontrola typu akce, kontrola prvku, kdo kterého se narazilo, a kontrola dotyku prvku mimo událost kolize.

Zároveň vynechávám funkce ukládání a nahrávání více prostředí, hra bude mít pouze jedno prostředí uložené. Toto zjednoduší předávání dat mezi hrou a trénovacím procesem a ulehčí implementaci GUI. Více prostředí není naprosto nutné pro fungování hry.

Test prostředí V testu prostředí bude pouze ukazatel odměny za jednu epizodu a ovládání agenta po prostředí. Délka epizody bude stanovena pevně.

Centrum trénování Hráč bude mít možnost zde spustit trénování, nastavit délku trénování a restartovat neuronovou síť agenta do výchozího nastavení. Délka epizody během trénování bude stanovena pevně, trénovaný agent bude pevně nastaven na jednu možnost a trénovací prostředí bude vždy to naposledy uložené.

6 Implementace

V této části se zabývám implementací MVP. Přiblížím funkce zvolených nástrojů, struktury neuronové sítě a postavy agenta.

6.1 Unity Engine

Unity je engine, který jsem si zvolil pro tvorbu hry, ve verzi Unity 2021.3.18f1 Personal. Hlavními důvody pro volbu tohoto nástroje jsou moje předchozí zkušenosti s enginem, výše zmíněné nástroje ML-Agents a jeho poměrná jednoduchost. Tyto vlastnosti zásadně urychlí vývoj hry a žádný z ostatních prozkoumaných nástrojů neposkytuje nic, co by tuto výhodu překonalo.

Unreal také nově poskytuje nástroje pro trénování agentů, které ale byly oznámeny až v průběhu tohoto projektu. Engine Unreal je známý zejména pro grafickou kvalitu svých her, což není cíl tohoto projektu, a navíc s enginem Unreal nemám předchozí zkuženosti narozdíl od Unity. Torque engine jsem zvažoval hlavně kvůli hře NERO, která se snažila o velice podobný cíl jako tento projekt a je vytvořená v pomocí enginu Torque.

6.1.1 Struktura Unity

Unity lze použít pro tvorbu celé řady různých produktů, ale v následující části budu uvažovat použití Unity v případě tvoření her. Projekt v Unity se skládá z kolekce objektů *Asset*, které obvykle odpovídají souborům v projektu. Mezi důležité typy těchto objektů patří typ *Scene*, *GameObject* a *Component*.

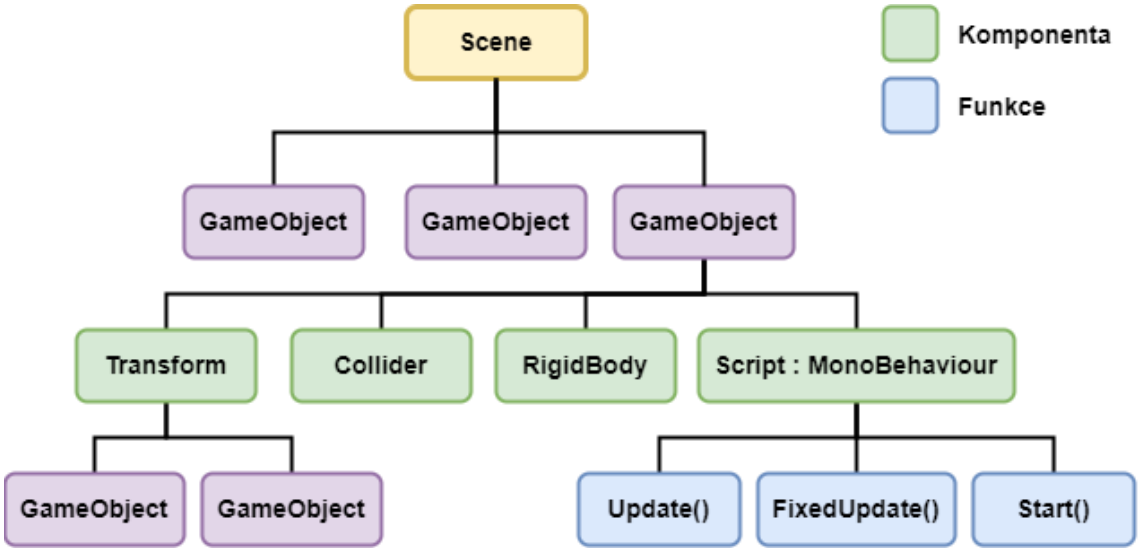
Scene Dále jako scéna, odpovídá prostředí hry a je nosič objektů *GameObject* a jde o prostor do kterého lze vykreslovat grafika. Ve scénách se odehrává celá hra.

GameObject Dále jako herní objekt, reprezentuje určitý objekt ve scéně a je nosič objektů *Component*.

Component Dále jako komponenta, reprezentuje jednotlivé části určitého objektu ve hře. Komponenta je obecný typ, který má mnoho implementací. Mezi nejrelevantnější patří komponenty *Transform*, který je u každého *GameObject* a určuje pozici objektu ve scéně, jeho rotaci, velikost a hierarchickou strukturu s ostatními herními objekty, *Collider2D*, který určuje detekci kolize v rámci fyziky, *RigidBody*,

které určuje chování objektu v simulaci fyziky hry, a *Script*, který je uživatelem definovaný pomocí jazyka C#.

Vztahy mezi jednotlivými typy jsou vyobrazeny na obrázku 6.1

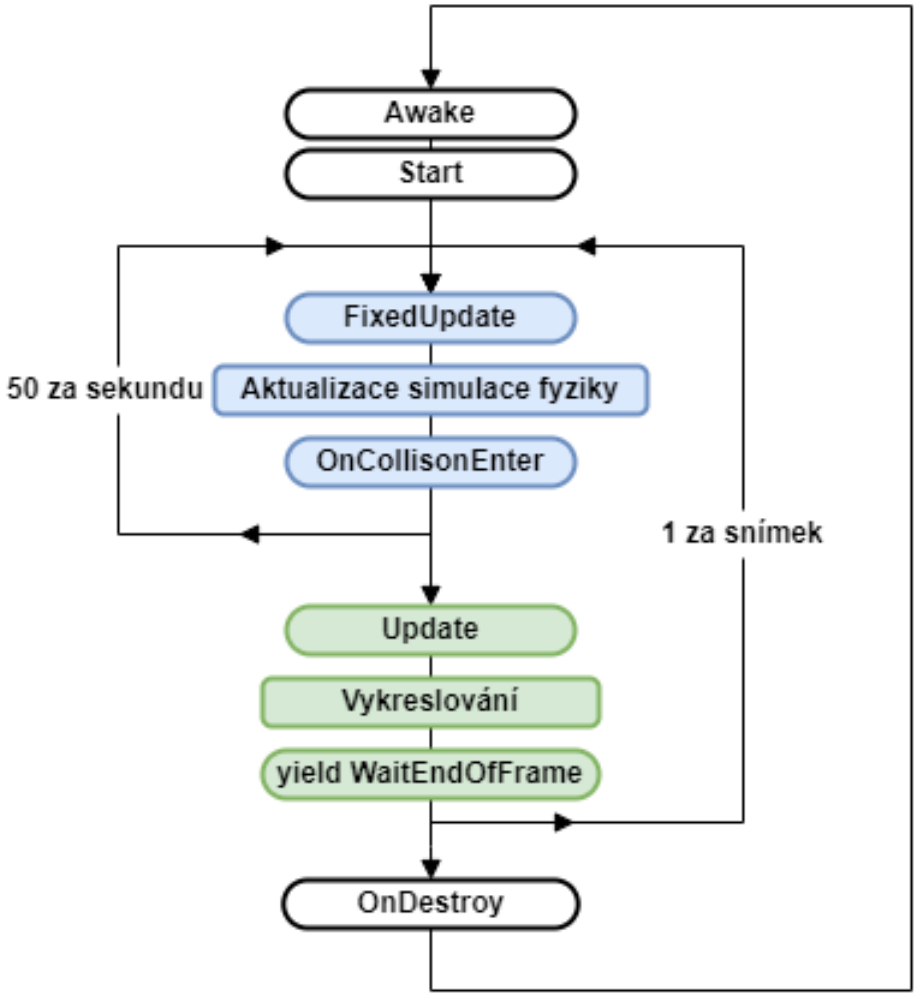


Obrázek 6.1: Struktura objektů v rámci scény v Unity

Script *Script* musí dědit od třídy *MonoBehaviour*, aby mohl být připojen jako komponenta k hernímu objektu, a je psaný uživatelem. Třída *MonoBehaviour* poskytuje několik funkcí, které jsou během provozu hry pravidelně Unity volány, pokud je komponenta povolena. Každá komponenta má booleovský paramert *enabled*, který určuje, zda komponenta bude zavolána Unity. Mezi často využívané funkce této třídy patří:

- **Awake()** volaná jednou za životní cyklus při vytvoření nebo nahrání komponenty bez ohledu na to, zda je komponenta povolena
- **Start()** volaná na začátku životního cyklu při vytvoření nebo nahrání komponenty po funkci Awake()
- **FixedUpdate()** volaná před každou aktualizací fyzikální simulace. V základním nastavení jde o frekvenci 50 za sekundu
- **OnCollisonEnter()** volaná pokud v poslední aktualizaci fyzické simulace došlo ke kolizi
- **Update()** volaná jednou za snímek

Výše zmíněný seznam neobsahuje plný výčet všech funkci MonoBehaviour. Uživatel může definovat chování skriptu přepsáním těchto metod. Vztah těchto funkcí je znázorněn na obrázku 6.2

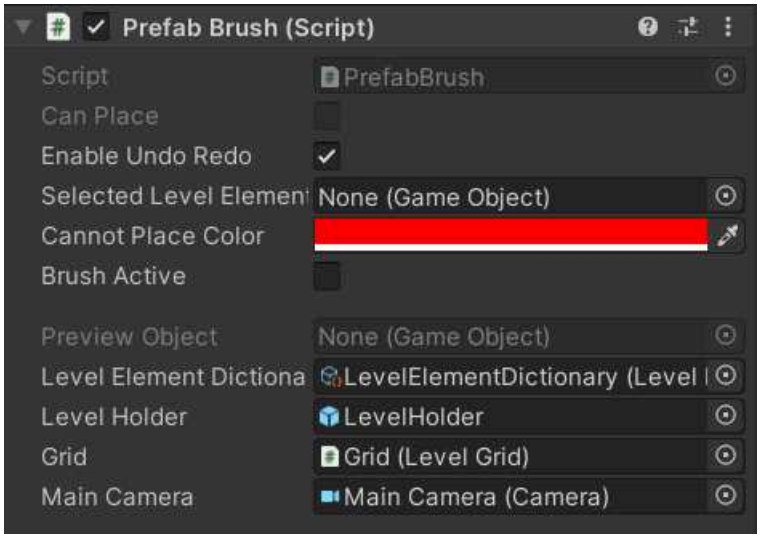


Obrázek 6.2: Zjednodušená smyčka volání funkcí Unity - upraveno [10]

Serializace Velkou částí práce s Unity je proces serializace - automatická transformace datových struktur a herních objektů do formátu, které Unity v pozdější době využívá pro rekonstrukci těchto objektů do scény. Pokud jsou hodnoty pole ve *Script* komponentě označené jako public nebo atributem [SerializedField] nebo jde o třídu s [Serializable] atributem, pak jsou viditelné uvnitř Unity Editoru a jejich hodnoty lze nastavit pomocí GUI (viz obrázek 6.3).

6.1.2 ML-Agents

ML-Agents Toolkit je sada nástrojů s otevřeným zdrojovým kódem, která umožňuje vytvořit prostředí v Unity Editoru a interagovat s ním pomocí Python API. Obsahuje ML-Agents SDK, které obsahuje všechny potřebné funkcionality pro definici prostředí uvnitř Unity Editoru společně se sadou ukázkových prostředí, zpětnovazebné algoritmy SAC a PPO a další nástroje pro strojové učení.[9].



Obrázek 6.3: Snímek obrazovky ukazující část GUI Unity Editoru, ve kterém lze nastavit serializované pole třídy *PrebabBrush*.

Python API Definuje, jakým způsobem učící algoritmy zapsané v jazyce Python komunikují s prostředím vytvořeném v Unity. ML-Agents Toolkit poskytuje implementaci několika algoritmů (PPO, SAC, GAIL a další) využívající toto API, ale uživatel má možnost doplnit vlastní implementaci libovolného algoritmu, pokud využije toto API. Na obrázku 6.4 je implementace těchto algoritmů znázorněna v bloku Python trenér. Výstupem Python trenéra je neuronová síť, kterou lze následně vložit zpět do hry. Python API také poskytuje rozhraní *Gym*, které představuje standardní způsob komunikace mezi algoritmem zpětnovazebního učení a učícím prostředím a je často využívané v oboru zpětnovazebního učení.[4]

Tyto nástroje jsou poskytované pomocí balíků Python *mlagents* (použitá verze v0.30.0) a *mlagents-env* (použitá verze v0.30.0). Zároveň je k provozu těchto balíčků potřeba instalace programovacího jazyka Python (použitá verze 3.9.7) a knihovna Pytorch (použitá verze 1.7.1)

ML-Agents SDK Obsahuje všechny funkcionality nutné k definování učícího prostředí uvnitř Unity. Patří mezi ně komponenta *Agent*, která indikuje, že herní objekt s touto komponentou je agent, a tudíž je schopen sbírat pozorování, provádět akce a dostávat odměny. Každý agent je spojen s rozhodovací strategií pomocí názvu rozhodovací strategie. Více agentů může mít stejnou rozhodovací strategii, což vede mimo jiné k tomu, že zkušenosti všech agentů s touto strategií se použijí pro optimalizaci této strategie.

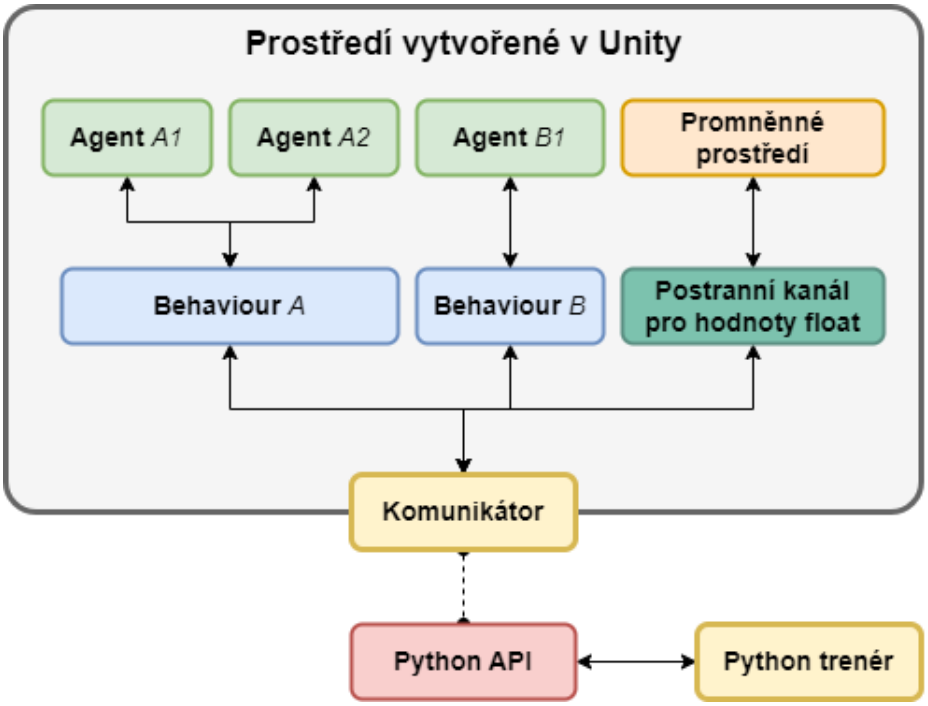
Na obrázku 6.4 je znázorněn vztah mezi rozhodovací strategií označenou jako blok Behaviour a komponentou Agent znázorněnou jako blok Agent. V jedné scéně může být použit libovolný počet rozhodovacích strategií. Rozhodovací strategie může být buďto Python trenér komunikující s prostředím přes komunikátor nebo již vytvořená vložená neuronová síť, která zrovna není optimalizovaná, nebo pevně zakódovaný kód, jako je například rozhodovací strom nebo ovládání pomocí klávesnice.

Tyto alternativy rozhodovací funkce by v obrázku 6.4 nahradily blok Komunikátor. Agenti během simulace hry mohou požádat o akci od rozhodovací strategie buď to v pravidelném intervalu nebo v uživatelem specifikovaných momentech definovaných komponentou *Script*

Další částí ML-Agents SDK je *Academy* - třída, která sleduje kroky agentů, spravuje agenty ve scéně a poskytuje další možnost komunikace mezi Python trenérem a učícím prostředím v podobě postranního kanálu přenášející hodnoty typu float, pomocí kterých lze upravit parametry prostředí jako například síla gravitace nebo rychlost objektů ve scéně. (viz obrázek 6.4)[9]

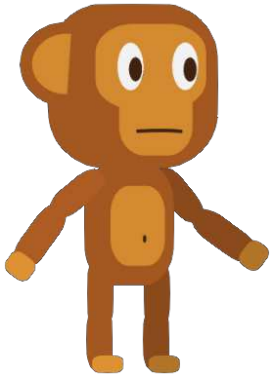
Tyto nástroje lze do projektu Unity stáhnout přes integrovaný balíčkovací systém v balíčku s názvem *ML Agents*, který jsem použil pro vytvoření MVP ve verzi 2.0.1.

mlagents-learn Ml-Agents Toolkit také poskytuje koncový bod pro spuštění trénovacího procesu v podobě *mlagents-learn*. Pomocí tohoto příkazu spustitelného v Python je možné nastartovat komunikaci mezi Python trenérem a trénovacím prostředím buďto přímo v Unity Editoru, nebo na sestaveném trénovacím prostředí. Parametry trénovaní lze pak nastavit pomocí konfiguračního souboru ve formátu *yaml*, který definuje jak informace o použité rozhodovací strategii, tak i o délce trénovaní, záchytných bodech a další.



Obrázek 6.4: Struktura částí ML-Agents - upraveno [9]

6.2 Agent

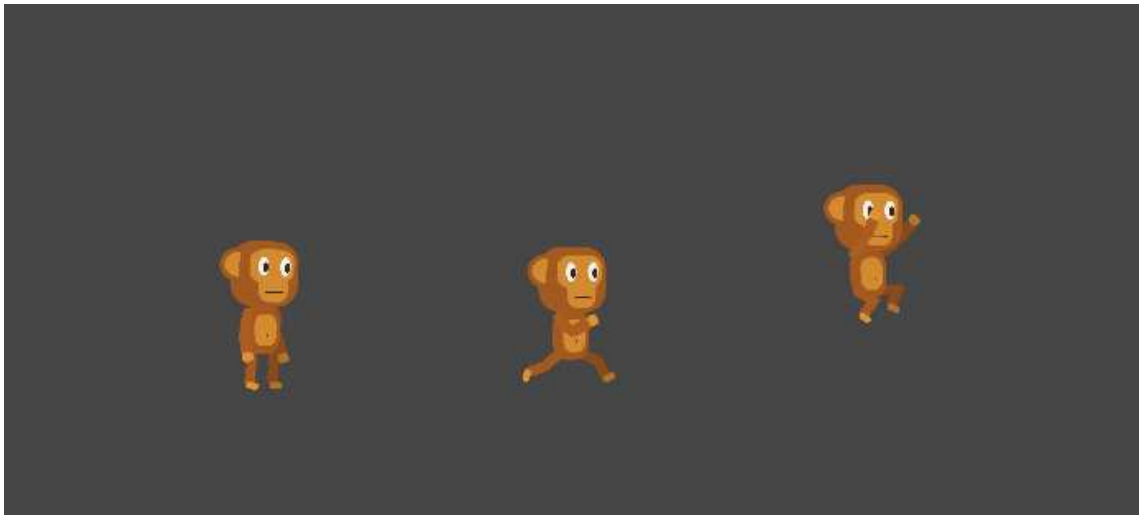


Obrázek 6.5: Agent ovládaný neuronovou sítí

6.2.1 Implementace v Unity

Postava agenta je *GameObject* obsahující následující komponenty: *Ridigbody2D* pro simulaci fyziky, *BoxCollider2D* pro detekci kolizí, *Animator* pro spouštění animací postavy, *MonkeyController* definující akce proveditelné postavou agenta, *MonkeyAgent* řídící proces trénování a předávání pokynů neuronové sítě komponentě *MonkeyController*, *BehaviourParameters* definující strukturu neuronové sítě používanou agentem a *DecisionRequester*, který zajišťuje průběžné předávání akcí od neuronové sítě komponentě *MonkeyAgent*. Některé tyto komponenty jsou poskytované přímo od Unity a ML-Agents a zbylé popisuji dále:

MonkeyController Tento skript ovládá postavu agenta. Obsluhuje logiku akcí a vykonává akce agenta. Definuje akce běhu a skákání, rozhoduje, kdy postava může skočit (kdy je na zemi) a poskytuje parametry pro ladění rychlost běhu, zrychlení a zpomalení, výšku skoku, z čeho může postava skákat. Dále rozhoduje, zda na jakou stranu je agent otočen a posílá instrukce komponentě *Animator* pro spouštění animací dle potřeby. Animace jsem vytvořil pomocí funkcí Unity pro animaci a kostry vytvořené pomocí rastrového grafického editoru Krita. Snímky akcí a jejich animací agenta jsou viditelné na obrázku 6.6.

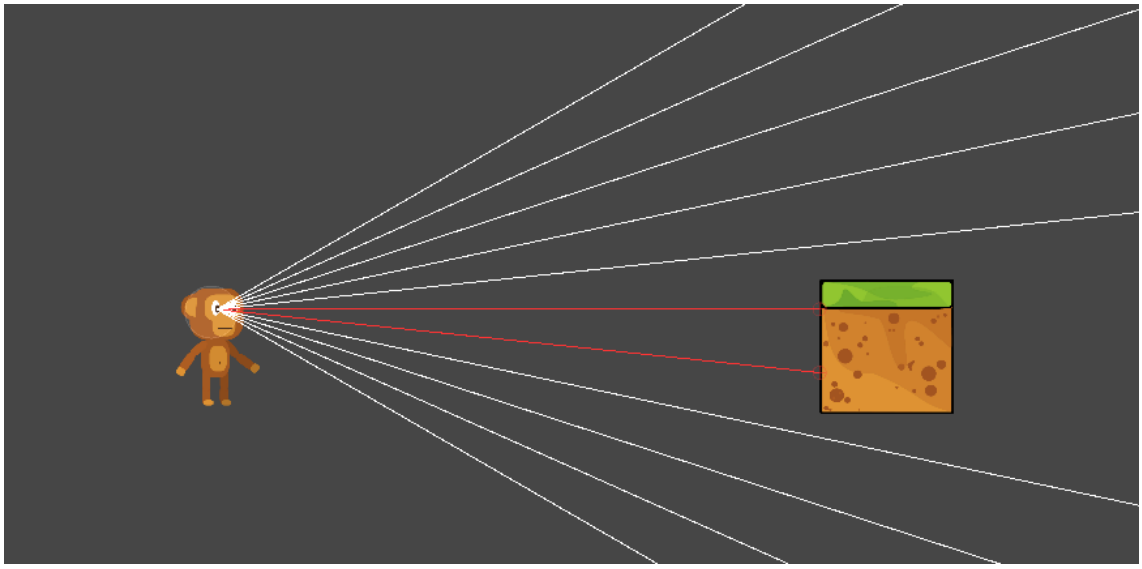


Obrázek 6.6: Akce prováděné agentem agenta - stání na místě, běh a skok

MonkeyAgent Je komponenta pro obsluhu trénovacího procesu. Je zde definováno, kdy začíná a končí epizoda, shromažďuje odměny a pozorování od ostatních komponent a předává je Python trenéru, předává instrukce k provedení akcí komponentě *MonkeyController*, zajišťuje resetování agenta na začátku nové epizody, sleduje, zda agent nespadl mimo prostředí, a informuje hodnotící systém o akcích, které agent provádí. Tato třída dědí od třídy *Agent* (poskytovaná od ML-Agents), která poskytuje metodu pro přidání odměn agentu *AddReward*.

Senzory Využívají komponentu poskytovanou od ML-Agents *RayPerceptionSenzor2D*, která poskytuje agentovi možnost detekovat objekty v prostředí pomocí paprsků detekující kolizi. Tento senzor rozpoznává prvky cíle a terénu a předává neuronové sítí informace o tom, zda paprsek detekoval rozpoznávaný prvek, o který se jedná a v jaké vzdáleností je. Funkce toho senzoru je viditelná na obrázku 6.7

Kromě senzorů zraku je agentovi poskytována komponentou *MonkeyController* informace o tom, zda je otočen doleva nebo doprava.



Obrázek 6.7: Zrakové senzory agenta detekující blok terénu

6.2.2 Neuronová síť

Topologie Neuronová síť ovládající chování agenta se skládá z 45 neuronů vstupní vrstvy, dvouch skrytých vrstev o 128 neuronech a výstupní vrstvy pěti neuronů, které se pak překládají na dvě diskrétní akce. Všechny vrstvy jsou dopředné, plně propojené. Neuronová síť zároveň využívá masku akcí, která rozděluje akce na skupiny, které se pak klasifikují do dvou tříd (skočit nebo neskočit) pro jednu skupinu, a do tří tříd (stůj, běž doleva, běž doprava) pro druhou skupinu. Topologie neuronové sítě je možno vidět na obrázku v příloze A.2 vytvořeného pomocí aplikace Netron.[11]

Vstupní vrstva odpovídá vrstva dvěma typům prvků prostředí, pro které je 11 detekčních paprsků, který každý poskytuje dvě informace - zda paprsek narazil do daného typu a jak daleko byl náraz zaznamenán. Toto dohromady činní 44 vstupů a jeden vstup indikuje, zda je agent otočen doleva nebo doprava.

Curiosity Neuronová síť agenta je také doplněná o modul *Curiosity*, který poskytuje agentovi vlastní vnitřní odměny za to, že se dostává do nových stavů. Pomáhá tak agentovi dostat se z lokálních minim a dosáhnou cíle v prostředích, kde se odměny objevují zřídka.

Modul obsahuje dva modely neuronových sítí, které se trénují paralelně s agentem. První z nich bere dvě za sebou jdoucí pozorování, zakóduje je a z kódování se snaží predikovat, kterou akci agent provedl aby z prvního pozorování dostal druhé. Druhý model se na základe tohoto kódování a stávající akci agenta pokouší predikovat následující kódování. Hodnota kriteriální funkce (loss) se pak přiděluje agentovi jako odměna.

Tato metoda má za následek, že pokud se agent pohybuje stále ve stejných

stavech, hodnota kriteriální funkce (a odměna) jde k nule, jelikož se modely byly schopny naučit predikovat přechody mezi těmito stavy. Pokud agent navštěvuje nové stavy, hodnota kriteriální funkce roste a agent tak dostává větší odměny. Výsledkem je zvědavý agent, který je motivován nacházet nové stavy.[12]

Pro modul *Curiosity* používám neuronovou síť s dvěma skrytími vrstvami o 128 neuronech, velikost kódování pozorování (encoding size) 128, *learning rate* = 0.0003, $\gamma = 0.99$ a sílu signálu (strength) 0.02. Síla signálu je míra toho, jak velký efekt má výsledek kriteriální funkce na odměnu agenta.

Algoritmus K trénování neuronové sítě agenta se používá algoritmus PPO (viz podkapitola 3.2), který se spustí po sesbírání 100 zkušeností (100 akcí a 100 pozorování) do zásobníku, po tři epochy. Používám parametry $\epsilon = 0.2$, $\lambda = 0.99$, $\gamma = 0.99$, *learning rate* = 0.0003 a *batch size* = 10 (velikost dávky). Celkový počet kroků provedených během trénovacího kroku je definován uživatelem se základním nastavením 10 000 kroků.

6.3 Hodnotící systém

Hodnotící systém sleduje průběh trénovacího procesu a přiděluje agentovi odměny na základě skutečností v prostředí. Jak a kdy se odměny přidělují specifikuje hráč. Hodnotící systém se skládá z následujících tříd:

EnviromentEvent Je třída přenášející informace o událostech uvnitř trénovacího prostředí, na které hodnotící systém může reagovat. *EnviromentEvent* obsahuje informace o typu události, kterými můžou být kolize, uplynutí času nebo akce agenta, a relevantní data k této události, jako jsou agent, kterého se událost týká, s čím proběhla kolize, nebo kterou akci agent provedl.

AbstractEnviromentEventGenerator Jejím účelem je sledovat události uvnitř trénovacího prostředí, a reagovat na specifické události vygenerováním a rozesláním instance třídy *EnviromentEvent*.

Pro každý typ *EnviromentEvent* exituje vlastní implementace této třídy, ve které je definováno, jak a kdy se má instance *EnviromentEvent* vytvořit. Po vytvoření se *EnviromentEvent* předá příslušné instanci *EnviromentEventSystem*, která dále reaguje na tuto událost. Pro lepší čitelnost je *AbstractEnviromentEventGenerator* na obrázku A.3 zapsán pouze jako EventGenerator.

EnviromentEventSystem Zodpovídá za vytvoření a správu relevantních generátorů událostí *AbstractEnviromentEventGenerator* a předávání událostí *EnviromentEvent* správnému hodnotícímu systému *ScoringSystem*. Generátory událostí vytvoří na základě instrukcí obdržených od *ScoringSystem* po zavolání metody *Subscribe*. Instrukce je reprezentována třídou *EventGeneratorInstruction*. *EventGeneratorInstruction* drží seznam herních objektů ve scéně a typ události, který mají herní objekty v seznamu generovat.

ScoringSystem Je základní nosič hodnotícího systému. Obsahuje metodu *HandleEvent* jejímž zavoláním dojde k výpočtu odměn pro agenta pro událost *EnviromentEvent* předanou argumentu .*ScoringSystem*, obsahuje seznam hodnotících pravidel *ScoringRule*, které jsou schopné reagovat na příchozí *EnviromentEvent* a vypočítat skóre pro příslušného agenta. Kterému agentovi se má skóre udělit rozhodne na základě dat v *EnviromentEvent* a poté zavolá metodu *AddReward* u komponenty *MonkeyAgent* připevněné k hernímu objektu agenta.

Dalšími důležitými metodami jsou *GetData*, metoda generující všechna potřebná data pro uložení, *LoadData*, která umí z nahraných dat rekonstruovat stejný hodnotící systém, a metoda *Initialize*, která vygeneruje instrukce pro vytvoření potřebných generátorů událostí *EnviromentEvent* a předá je třídě *EnviromentEventSystem*, která se stará o jejich správu a vytvoření.

ScoringRule Je jedno hodnotící pravidlo, které určuje, jaká odměna se má přidělit za obdrženou událost *EnviromentEvent*. Pravidlo si drží informace o tom, na který *EnviromentEvent* se vztahuje a jaká je základní odměna přidělená za spuštění tohot pravidla. Pokud do *ScoringSystem* předá požadovaný *EnviromentEvent*, vrátí se základní odměna modifikovaná podmínkami *ICondition* držené třídou v seznamu.

Také má metody *GetData* a *LoadData*, které podobně jako u *ScoringSystem* slouží k pozdější rekonstrukci pravidala.

ICondition Podmínka je třída implementující rozhraní *ICondition* s metodou *CalculateModifier*, která podle dat předaného *EnviromentEvent* rozhodne, jakým způsobem se základní odměna modifikuje. Podmínky mají jeden modifikátor při nesplnění podmínky a druhý při nesplnění. Hodnotu modifikátoru nastavuje hráč, ale v základním nastavení je pro splnění modifikátor 1 a pro neplnění 0. Modifikátorem se násobí základní odměna.

Kdy se udělí modifikátor za splnění a kdy za nesplnění je různé pro každou implementaci. Mezi současné implmentace podmínky patří *AgentCollison* - ověřuje kolizi se specifikovanými prvku prostředí, *AgentDecisionType* - ověřuje typ akce provedené agentem, *AgentTouching* - ověřuje prvky, kterých se během vyhodnocování podmínky dotýká, a *DefaultCondition*, která vždy vrací modifikátor 1.

Také má metody *GetData* a *LoadData*, které podobně jako u *ScoringSystem* slouží k pozdější rekonstrukci podmínky, metodu *GetEventGenerators*, která vrací herní objekty, které by měly podle potřeb této podmínky generovat události *EnviromentEvent*, a *GetApplicableEventTypes*, která vrací, pro které typy *EnviromentEvent* je podmínka použitelná.

Jak probíhá celý proces přidělovaní odměn agentovi je znázorněno v příloze na komunikačním diagramu A.3.

6.4 Editor Prostředí

Editor prostředí je scéna sloužící k tvorbě úpravě prostředí, ve kterém se agent bude

trénovat, kde hráč tráví nejvíce času a skládá se z dvou hlavních částí.

6.4.1 Plátno

Plátno je levá část obrazovky, na které hráč může umisťovat prvky prostředí zvolené v panelu na levé straně obrazovky.

PrefabBrush O umisťování objektů na plátno se stará třída *PrefabBrush*(prefab je *GameObject* uložený jako Asset, který se využívá pro vytváření kopií na scéně). *PrefabBrush* má aktivní a neaktivní stav. Pokud si hráč vybere prvek z postranního panelu prvků, *PrefabBrush* přejde do aktivního stavu a vytvoří kopii zvoleného prvku, ke kterému si drží referenci v proměnné *previewObject*. Tato kopie (dále jako *previewObject*) slouží jako kurzor ukazující, kde se po kliknutí levým tlačítkem myši vytvoří nová kopie zvoleného prvku pomocí metody *PlacePrefab*, nebo po kliknutí pravým odstraní jiný existující prvek. *PreviewObject* sleduje kurzor myši a pohybuje se po mřížce a svojí barvou indikuje, zda lze prvek do dané pozice umístit.

Kromě vytváření podle kurzoru myši zároveň poskytuje metodu *PlacePrefabs* pro umístění více různých prvků najednou podle přiložených dat. Tato metoda se používá při nahrávání uloženého prostředí. Dále poskytuje funkce *undo* a *redo* podle návrhového vzoru Memento.

LevelElement Jednotlivé prvky umisťované do prostředí jsou herní objekty, které mají komponentu *LevelElement*. Tuto komponentu jsem vytvořil za účelem ukládání a správy jednotlivých prvků prostředí. Komponenta si drží následující proměnné *levelElementKey* - řetězec znaků identifikující, který *GameObject* se má vytvořit pro tento prvek prostředí, *levelElementId* - celé číslo unikátní mezi všemi ostatními *LevelElement* v daném prostředí jednoznačně identifikující daný prvek, *position* - vektor pozice v prostoru, a *selectButtonImage* - obrázek, který se má zobrazit na postranním panelu jako reprezentace prvku.

Jedná se o informace, které jsou nutné k jednoznačné rekonstrukci prvku v prostředí, a při ukládání prostředí se uloží do souboru ve formátu JSON. Při nahrávání uloženého prostředí se tyto informace předají třídě *PrefabBrush*, která je následně umístí správné kopie prvků na správné místo. *LevelElementId* se použije pro rekonstrukci odkazů na správné prvky prostředí v hodnotícím systému.

LevelElementDictionary Mapování mezi klíčem *levelElementKey* a správným prvkem je drženo ve třídě *LevelElementDictionary* - slovníku prvků prostředí. Tato třída dědí od *ScriptableObject*, což umožňuje vytváření souborů Asset této třídy a tak zachovat mapování trvale v paměti hry. *LevelElementDictionary* tak slouží jako překladač mezi *levelElementKey* a herním objektem (*GameObject*), který odpovídá danému klíči.

6.4.2 Prvky prostředí

V MVP jsou implementované tři prvky, které hráč může umisťovat do prostředí.

Všechny tyto prvky jsou herní objekty obsahující komponenty *LevelElement* pro práci s editorem prostředí, *SpriteRenderer* pro vykreslování grafiky a *Highlight-Conroller*, který jsem vytvořil pro práci se zvýrazňováním prvků v prostředí pro komunikaci mezi rozhraním a uživatelem.

Terén Nejčastěji používaným prvkem je terén, který hráč umisťuje po blocích. Jedná se o jednoduchý herní objet, který kromě společných komponent s ostatními prvky také používá *Collider2D* pro detekci kolizí, a *SpriteRandomizer*, který se stará o změnu vykreslované grafiky v závislosti na tom, zda je blok terénu pod jiným blokem a na náhodě. Tuto komponentu jsem vytvořil pro rozbití vizuální monotónnosti prostředí. Tento prvek tvoří většinu trénovacího prostředí. Prvek je viditelný pro agenta.

Startovní pozice Aby hráč mohl určit, kde se má agent v prostředí při zahájení epizody trénovacího procesu objevit, vytvořil jsem prvek startovní pozice. Prvek vizuálně vypadá jako jako průhledný agent a obsahuje komponentu *MonkeySpawner*, která slouží identifikaci prvku, jako startovní pozice, a poskytuje přesnou kontrolu nad pozicí, kde se má agent objevit. Pokud se v prostředí objevuje více startovních pozic, agent se vždy objeví na náhodné z nich při začátku každé epizody. Prvek nekoliduje s agentem a agent jej při simulaci nedetekuje.

Bedna Posledním implementovaným prvkem je cíl s grafickou representací bedny s banánem. Prvek obsahuje komponentu *Collider2D* a po detekci kolize s agentem vyšle signál komponentě *MonkeyAgent* pro ukončení epizody a přiřazení odměny za splnění cíle.

6.4.3 Postranní panel

Postranní panel se skládá ze dvou karet mezi kterými se hráč může přepínat a obě tyto karty jsou zobrazeny na obrázku v příloze A.1.

Karta prvků V této kartě jsou umístěny tlačítka pro výběr prvků. Každé tlačítko odkazuje na jeden druh prvků identifikovaný podle *levelElementKey* v třídě *LevelElement*. Po stisknutí tlačítka se *levelElementKey* předá třídě *PrefabBrush*, které z *LevelElementDictionary* dostane příslušný herní objekt, ze kterého pak může vytvářet kopie.

Tlačítka v této kartě jsou generována automaticky podle záznamů v *LevelElementDictionary*.

Karta hodnotícího systémů Tato karta poskytuje uživatelské rozhraní, pomocí kterého hráč může tvořit a upravovat hodnotící systém. Obsahuje tlačítko pro přidání pravidla do hodnotícího systému prostředí, které v panelu vygeneruje GUI element pro úpravu tohoto pravidla.

Tento element GUI se skládá z několika částí zanořených do několika úrovní. Tyto úrovně se skládají podle momentálních potřeb pravidla. První část je pro třídu *ScoringRule*, která obsahuje textové pole pro zadání jména pravidla a základní odměny, rozbalovací seznam pro spouště pravidla a seznam podmínek s tlačítkem pro přidání nové podmínky. Do tohoto seznamu se po stisknutí tlačítka pro přidání nové podmínky přidá nový element GUI pro podmínku.

Jelikož podmínka není jedna třída s jasně definovanými parametry, ale celá skupina tříd, tento element GUI je univerzální pro všechny podmínky a obsahuje pouze rozbalovací seznam podmínek, které lze k dané spoušti hodnotícího pravidla přiřadit. Po zvolení některé z podmínek rozbalovacího seznamu se k tomuto elementu GUI přidá speciální element GUI, který poskytuje potřebné rozhraní pro nastavení té specifické podmínky.

Tyto elementy GUI pro specifické podmínky většinou obsahují textové pole pro zadání modifikátoru odměny při splnění a nesplnění podmínky a některé z nich mají i tlačítko pro vybrání prvku prostředí a tlačítko pro zobrazení již vybraných prvků prostředí.

6.5 Trénování

Zahájení trénovacího procesu začíná zavoláním příkazu *mlagents-learn*(obsažený v balíčku *mlagents*), který spustí trénovací proces podle přiloženého konfiguračního souboru ve značkovacím jazyce YAML. Pokud se nepředá konfigurační soubor, použije se výchozí nastavení. Trénovací proces začne buďto v sestaveném trénovacím prostředí (sestavená hra z Unity, kde první scéna obsahuje agenta k učení), ke kterému se cesta musí přidat do konfiguračnímu souboru, nebo Python program spuštěný příkazem *mlagents-learn* čeká na stisknutí tlačítka Play v Unity Editoru a poté spustí trénovací proces přímo v editoru.

Tento krok představoval pro můj projekt drobný problém. Jelikož cílem má být hratelná hra, nemohu použít trénování v editoru, a zároveň nemohu předat trénovací prostředí v sestavené hře, protože v době sestavení ještě neexistuje. Trénovací prostředí je stvořené až hráčem dávno po sestavení hry. Trénování na prostředí, které vytvoří hráč není během hraní předpokládané využití nástrojů ML-Agents a tudíž ani přímo podporované.

K tomu, abych trénovací proces z návrhu hry zprovoznil, musel jsem do hry přidat trénovací scénu, která sama sestrojí požadované prostředí za běhu, a automaticky scénu načíst, pokud je hra spuštěna s cílem trénovat agenta. Abych docílil nahrání správné scény při spuštění hry, využil jsem atribut poskytovaný Unity *[RuntimeInitializeOnLoadMethod]*, který zajišťuje zavolání metody hned po spuštění hry, pro implementaci metody *GameInitialization*, která přečte vstupní argumenty předané z příkazové řádky, a při předání předání argumentu *–training_mode* načte trénovací scénu. V trénovací scéně poté třída *TrainingInit* v metodě zavolané při nahrání scénu *Start* přečte data z vlastního konfiguračního souboru napsaného ve formátu JSON, který obsahuje informace o tom, jak má nastavit objekt agenta a odkud načíst prvky prostředí.

Prostředí je poté staveno stejným způsobem v Editoru prostředí třídou PrefabBrush při nahrávání (viz 6.4)

6.5.1 Konfigurace

Python trenér K úpravě konfiguračního souboru pro Python trenéra jsem využil knihovnu s otevřeným zdrojovým kódem *YamlDotNet* ve verzi 12.3.1, která poskytuje nástroje pro čtení a úpravu souboru YAML.[13] Do konfiguračního souboru zapíši délku trénování podle údajů od hráče a změním *run_id*, které jednoznačně identifikuje jeden proces trénování a data, které se procesu týkají (uložený model neuronové sítě, statistiky, záchytné body, informace o průběhu procesu, atd.). Pokud hráč zaškrtne přepínač s názvem *Reset Brain*, nastaví se *run_id* na hodnotu *ppo_run_0* a data předchozího trénování se začnou přepisovat. Pokud hráč nezaškrtne tento přepínač, *run_id* se nastaví na další číselnou hodnotu a hodnota *initialize_from* v konfiguračním souboru se nastaví na předchozí *run_id*.

Toto přepisování je povolené booleovským parametrem konfiguračního souboru *force*, které je v případě použití v MVP vždy nastaven na jako pravdivý.

Prostředí Kromě konfigurace Python trenéra je také nutné nakonfigurovat trénovací prostředí pro úspěšné sestavení na začátku trénovacího procesu. Pro tuto konfiguraci používám soubor *training_set_up.json*, který obsahuje jméno trénované rozhodovací strategie, které je třeba předat objektu agenta, a název souboru, ve kterém se nachází data uloženého prostředí ve formátu JSON.

6.5.2 Integrace natrénovaného modelu

Kromě problému se startem trénovacího procesu se také objevil problém s integrací natrénovaného modelu zpět do hry. Podle standardního použití ML-Agents se model ve formátu .onnx přidá do složky projektu *Assets*, pro který pak Unity Editor vytvoří reprezentaci připravenou pro hru, které se dá pomocí GUI Unity Editoru nebo metody *Resources.Load* přiřadit agentovi.[14]. Tuto reprezentaci však vytváří Unity Editor, jehož kód není ve sestavené hře, a tak nelze využít při běhu hry.

Problém jsem řešil nahráním souboru *.onnx* a jeho převedením do formy interpretovatelné Unity jinými způsoby, než poskytuje standardní cesta využívající knihovny Unity Editor. Pro nahrání modelu jsem použil metod knihovny *Unity.Barracuda*. Jedná se o rozhraní pro práci s neuronovými sítěmi, na kterém byl Unity ML-Agents Toolkit postaven a je dostupná pod licencí Unity Companion License. Má znatelně méně přehlednou dokumentaci, než zbytek nástrojů Unity. Tato knihovna také obsahuje třídu *ONNXModelImporter*, která se stará mimo jiné o konverzi *.onnx* souboru při nahrávání do Unity Editoru pomocí metody *OnImportAsset*. Tato metoda nebyla přímo použitelná v mém kontextu, ale posloužila jako návod k řešení.

6.6 Ostatní

Kromě výše uvedených funkcí jsem do hry implementoval čtyři mise viditelné na obrázcích v příloze B.1. Mise obsahují tlačítko, které vyšle agenta s natrénovanou neuronovou sítí do prostředí na jednu epizodu, nebo ukončí epizodu předčasně, pokud již agent je v prostředí. Na konci této epizody komponenta *MissionManager* řídící celý proces průběhu mise zkontroluje odměny agenta, a pokud dosáhne dostatečně mnoho odměn, tak prohlásí misi za splněnou a odemkne tak další mise. Splněné mise hráče se ukládají pomocí třídy *PlayerPrefs* do registrů platformy (počítače) hráče. Nejedná se o bezpečný systém, ale pro testování je dostačující

Poslední částí hry je její instalace. Hráči jsem poskytl redukovanou verzi návodu od tvůrců ML-Agents. Z návodu jsem vypustil vše, co není potřebné pro spuštění hry. Nepředpokládám, že toto bude trvalé řešení, ale považuji ho za dostatečné pro potřeby MVP.

7 Testování

Hlavním účelem MVP bylo otestovat, zda hra funguje a má zábavní potenciál. Měření zážitku hráče ve hře je klíčové k pochopení dopadů hraní hry, navrhování a dalšímu vývoji. V této kapitole rozebírám, jakým způsobem jsem hru testoval a jaké závěry jsem z výsledků testu vyvodil.

7.1 Metodika

Existuje velké množství dostupných dotazníků vytvořených pro měření zážitků hráče a neexistuje žádný přesně stanovený standard.[15] Dotazník, který jsem se rozhodl pro sběr zpětné vazby použít, je Game Experience Questionnaire (GEQ).[16] Jedná se o jeden z více používaných dotazníků obsahující soubor otázek s odpovědmi na škále 0 až 4 (vůbec až extrémně) rozřazených do 7 komponent: pozitivní vliv, negativní vliv, výzva, kompetence, ponoření do hry, plynutí - stav hlubokého soustředění v angličtině známí jako *flow*[17], a napětí.

Tento dotazník v konfiguraci jádrového modulu[16] jsem vlastními silami přeložil do češtiny a doplnil jsem o otázky relevantní vůči mému projektu, přidal do testovacího balíčku pro hráče, a rozeslal hráčům. Z části GEQ jsem vynechal dvě otázky z komponenty ponoření do hry, jelikož jsem je posoudil jako irelevantní vůči hře. Hráče jsem kontaktoval individuálně a vybíral jsem je z okruhu osob mně blízkých. Testovací balíček obsahoval sestavenou hru, odkaz na dotazník zprostředkovaný pomocí Formuláře Google, doplňující manuál ke hře obsahující dodatkové informace pro hraní předaný odkazem na sdílený dokument Google a soubor *README.md* obsahující instrukce pro zprovoznění hry.

Je vhodné podotknout, že použitá metoda výběru hráčů způsobuje významné zkreslení dat a při vyhodnocování metrik je důležité toto mít na mysli. Skupina vybraných hráčů mě osobně zná, a tak lze předpokládat jisté kognitivní zkreslení dat obzvláště v případě komponent pozitivní vliv a negativní vliv.

Kromě rozesílání dotazníků v testovacím balíčků jsem byl osobně přítomen u třech testů hry, kde jsem mohl pozorovat oblasti herního zážitku a fungování hry neprozkoumávané dotazníkem.

7.2 Vyhodnocení metrik

Po rozeslání testovacích balíčků se mi podařilo získat zpětnou vazbu pomocí dotaz-

níku od 7 hráčů. Rozsah obdržené zpětné vazby je příliš malý, aby se na něm dala provést významná statistická analýza dat, přesto jsou data informativní a užitečná pro další vývoj hry. Obdržená zpětná vazba je přiložená sekci příloh C.1

Hodnoty jednotlivých komponent GEQ se počítají jako průměr přes všechny otázky vztažené k této komponentě.[16] V tabulce 7.1 se nachází výsledné hodnoty těchto komponent zaokrouhlené dvě desetinná místa. Každá komponenta je doplněna čísly otázek, které se k této komponentě vztahují a jejich odpovědi byly použité pro výpočet.

Tabulka 7.1: Komponenty dotazníku GEQ

Komponenta	**Hodnota**	**Otázky**
Pozitivní vliv	2,69	1, 3, 6, 15 ,32
Negativní vliv	0,89	4, 7, 18, 21
Výzva	2,09	11, 16, 25, 26, 29
Kompetence	1,71	2, 5, 8, 19, 30
Ponoření do hry	2,43	9, 17, 27, 31
Plynutí	2,03	20, 22, 23, 24, 28
Napětí	1,62	10, 13, 14

Při vyhodnocování těchto hodnot si nejsem jistý, jak užitečné a informativní jsou. Z rozdílů mezi komponentami pozitivní a negativní vliv vyplývá, že hráči se hru spíše užívali a bavila je. Usuzuji, že změny těchto metrik, může být užitečné sledovat mezi jednotlivými verzemi hry, ale jejich momentální hodnotu nevím jak užitečně vyhodnotit. Nabízí se hledání korelací mezi vlastnostmi hráče, jako je zkušenost se strojovým učením, a metrikami GEQ, ale usuzuji, že na to je vzorek odpovědí příliš malý a příliš zkreslený. Využití GEQ se hodí spíše pro větší testy s rozmanitějším publikem.

Jako více užitečné informace považuji odpovědi na určité otázky z dotazníku. Otázky ohledně frustrace ukazuje a obtížnost naznačují, že většina hráčů se dostala do situace, kde se snažili dlouho překonat misi neúspěšně. Zajímavý poznatek pro mne je, že byť hráči uváděli ve většině případů velkou frustraci, žádný z hráčů neuvedl, že by mu hra způsobila špatnou náladu.

Jako důležitější metriky z dotazníku považuji čas, po který hráči hru hráli. Hráči se z pohledu času stráveného ve hře rozdělili do dvou skupinek. První skupina hrála hru jednu až dvě hodiny a druhá tři až čtyři hodiny. V obou případech hráči hru hráli déle, než jsem očekával. Moje očekávání bylo, že někteří hráči hru vyzkouší, rychle ji opustí a ve hře stráví pouze desítky minut. Hra je velice limitovaná, a skutečnost, že i tak hráči dobrovolně hráli hru i tři a více hodin znamená, hra má zábavní potenciál. Někteří hráči měli potíže zprovoznit hru, ale všem se ji podařilo zprovoznit a splnili alespoň první misi (viz tabulka C.1). Toto dokazuje, že hra z technické stránky funguje.

Největší přínos však jednoznačně přinesla osobní přítomnost u testů. Při pozorování hráčů a rozhovorech s hráči jsem zaznamenal mnoho poznatků a problémů, které se v dotazníku nezachytily. Zde jsem zaznamenal, že přechody mezi scénami

jsou zbytečně striktní. Hráč musel do jednotlivých částí navigovat přes scénu rozcestníku, když většinou chtěl jít z editoru prostředí rovnou do centrum trénování, z centrum trénování do poslední mise a z mise do editoru prostředí. Tento problém měl největší dopad při cestě z centrum trénování do poslední mise, protože hra si nepamatuje poslední zvolenou misi a hráč tak musí navigovat přes výběr mise.

Dále výběr prvků pro podmínky hodnotícího systému v editoru prostředí se nechoval podle zvyků hráču. Často se stávalo, že hráči nepotvrdili výběr a někdy omylem začali vybírat prvky pro další podmínku bez ukončení předchozího výběru, což mělo za následek přiřazení prvků ke špatné podmínce.

7.3 Vyvozené závěry

Hra funguje, některé hráče velice bavila a má zábavní potenciál. Hru se podařilo zprovoznit všem hráčům, nicméně objevila se téměř jednoznačná shoda mezi hráči na tom, že instalace je těžká a nepříjemná.

Grafické uživatelské prostředí v editoru prostředí převážně funguje, hráči byli schopni jej použít pro dokončení mise, není však intuitivní, nekomunikuje dobře jeho použití ani funkce a neodpovídá intuitivnímu myšlení hráče. V mnoha případech je zbytečně složité a obsahuje prvky, které hráč využívá jen vzácně. Někteří hráči projevili nespokojenost s chybějící možností mazání pravidel.

Dále hra je příliš těžká, obzvláště třetí mise, která poskytuje příliš malou toleranci. Kromě malé tolerance v třetí misi, hru zásadně ztěžuje její neprůhlednost. Hra nevysvětluje dobře svoje pravidla a to hráčům ztěžuje řešení problémů. Přiložený manuál si mnoho hráčů ze začátku nepřečetlo.

7.4 Návrh další iterace

Je mnoho prvků, které by bylo potřeba ve hře upravit a do hry přidat. Některé jsou však důležitější, než ostatní. Na základě zpětné vazby jsem určil čtyři změny, jako ty s nejvyšší prioritou pro další vývoj hry. Další test hry bude hned po provedení těchto změn.

Instalace Instalace hry je příliš obtížná a není nutné, aby tomu tak bylo. Momentální řešení vždy bylo myšleno jako dočasné, ale test ukazuje, že toto řešení má velký dopad jak na počet potenciálních hráčů účastnících se testů, tak i na jejich ochotu a zážitek ze hry. Zjednodušení instalace hry se jeví jako momentálně nejlepší investice z pohledu časové náročnosti vůči dopadu.

GUI pro hodnotící systém Způsob myšlení a práce s hodnotícím systémem neodpovídá tomu, jak s ním chtějí hráči pracovat. Koncept modifikátorů u podmínek se ukazuje jako užitečný v některých případech, ale ve většině případu nepoužitý. Z pozorování interakce hráčů s editorem jsem vypozoroval, že hráči chtějí vytvářet

velice jednoduchá pravidla typu „Pokud vyskočí tak ztratí 1 bod.“ a v další iteraci se pokusím GUI co jevíce přiblížit této myšlence.

Tutoriál Ke hře byl přiložen manuál, který si pouze někteří hráči přečetli. Manuál byl myšlen jako dočasné řešení, ale ukazuje se, že je nedostačující. Je potřeba ve hře vytvořit tutoriál, který hráče hrou provede, a tato funkce nemůže být odložena do pozdějších iterací hry, ale měla by být součástí hry hned od začátku.

Lehčí mise Třetí mise, kde hráč má za úkol naučit agenta skákat přes díru se ukázala, jako neočekávaně těžká. Hlavním důvodem je to, že prostor, odkud může agent skočit a díru překonat, je velice úzký. Obtížnost třetí mise byla výrazně vyšší, než obtížnost druhé mise, a to byl hlavní důvod frustrace hráčů.

Závěr

V této práci jsem nejdříve prozkoumal existující hry využívající neuronové sítě a rozdělil jsem je na do tří kategorií: hry poskytující limitovanou kontrolu nad trénováním, hry využívající neuronové sítě, které však nejsou o trénování neuronových sítí a hry poskytující širokou kontrolu nad trénováním.

Definoval jsem cíle vytvářené hry a cíle minimálního životaschopného produktu vytvářeného v rámci této práce. Uvedl jsem problematiku zpětnovazebného učení a popsal jsem algoritmus PPO používaný v této práci. Krátce jsem představil dostupné nástroje pro tvorbu her a zdůvodnil jsem volbu enginu Unity.

Dále jsem představil návrh využívající neuronové sítě a zpětnovazebné učení jako hlavní herní mechaniku a popsal jsem implementaci minimálního životaschopného produktu. Bylo zde popsáno, jak je sestaven hodnotící systém, z čeho se skládá agent využívaný ve hře a jaké může provádět akce. V poslední části práce jsem uvedl metodiku testování minimálního životaschopného produktu a vyhodnotil jsem zpětnou vazbu od hráčů.

Tento produkt splnil své cíle. Hráči byli schopni jej spustit a hrát, umožnil hráčům trénovat agenty a zásadním způsobem zasahovat do trénování agentů. Také poskytl zajímavé problémy pro hráče k řešení, především v třetí misi. Minimální životaschopný produkt byl použit k testování, poskytl užitečné informace a lze jej použít pro další vývoj. Všechny body zadání diplomové práce byly splněny.

Zmíněný produkt také demonstroval, že hra je zajímavá alespoň pro malý počet hráčů a zaslouží si další vývoj. Hra má zábavní a edukativní potenciál. Testování však odhalilo mnoho nedostatků, které je potřeba v další verzi hry vyřešit. Hra je příliš obtížná na instalaci pro běžného hráče, potřebuje integrovaný tutoriál, je příliš těžká příliš rychle a grafické uživatelské rozhraní není v momentálním stavu intuitivní.

Dalšími kroky projektu budou zjednodušení instalace hry, úprava obtížnosti hry, tvorba nového uživatelského rozhraní pro tvorbu a úpravu hodnotících pravidel a integrace tutoriálu. Pokud další test prokáže, že zmíněné problémy jsou dostatečně ošetřeny, může začít implementace ostatních částí návrhu.

Na závěr bych rád podotkl, že konečný produkt by potenciálně mohl posloužit jako platforma pro objev nových metod a postupů v oboru zpětnovazebného učení s pomocí velkého počtu hráčů, kteří by mohli poskytnout crowdsourcing řešení určitých problémů.

Použitá literatura

[1] KEIWAN DONYAGARD. *Evolution* [https://keiwan.itch.io/evolution]. 2016. Přístup získán 7. 5. 2023.

[2] TOMMY THOMPSON. *How Forza's Drivatar Actually Works* [https://www.gamedeveloper.com/design/how-forza-s-drivatar-actually-works]. 2021. Přístup získán 7. 5. 2023.

[3] STANLEY, K.O., B.D. BRYANT a R. MIIKKULAINEN. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation*. 2005, roč. 9, č. 6, s. 653–668. Dostupné z DOI: 10.1109/TEVC.2005.856210.

[4] BROCKMAN, Greg et al. *OpenAI Gym*. 2016. Dostupné z arXiv: 1606.01540 [cs.LG].

[5] *What Is Reinforcement Learning?* [https://www.mathworks.com/help/reinforcement-learning/ug/what-is-reinforcement-learning.html]. The MathWorks, Inc., [b.r.]. Přístup získán 12. 5. 2023.

[6] SCHULMAN, John et al. *Proximal Policy Optimization Algorithms*. 2017. Dostupné z arXiv: 1707.06347 [cs.LG].

[7] *Torque 3D engine* [https://torque3d.org/torque3d/]. [B.r.]. Přístup získán 7. 5. 2023.

[8] *Learning Agents Introduction* [https://dev.epicgames.com/community/learning/tutorials/8OWY/unreal-engine-learning-agents-introduction]. Epic Games, 2023. Přístup získán 7. 5. 2023.

[9] JULIANI, Arthur et al. *Unity: A General Platform for Intelligent Agents*. 2020. Dostupné z arXiv: 1809.02627 [cs.LG].

[10] *Order of execution for event function* [https://docs.unity3d.com/Manual/ExecutionOrder.html]. Unity Technologies, [b.r.]. Version 2021.3.

[11] LUTZ, Roeder. *Netron, Visualizer for neural network, deep learning, and machine learning models*. 2017. Dostupné z DOI: 10.5281/zenodo.5854962.

[12] PATHAK, Deepak et al. *Curiosity-driven Exploration by Self-supervised Prediction*. 2017. Dostupné z arXiv: 1705.05363 [cs.LG].

[13] AUBRY, Antoine. *YamlDotNet* [https://github.com/aaubry/YamlDotNet]. 2023. Přístup získán 18. 5. 2023.

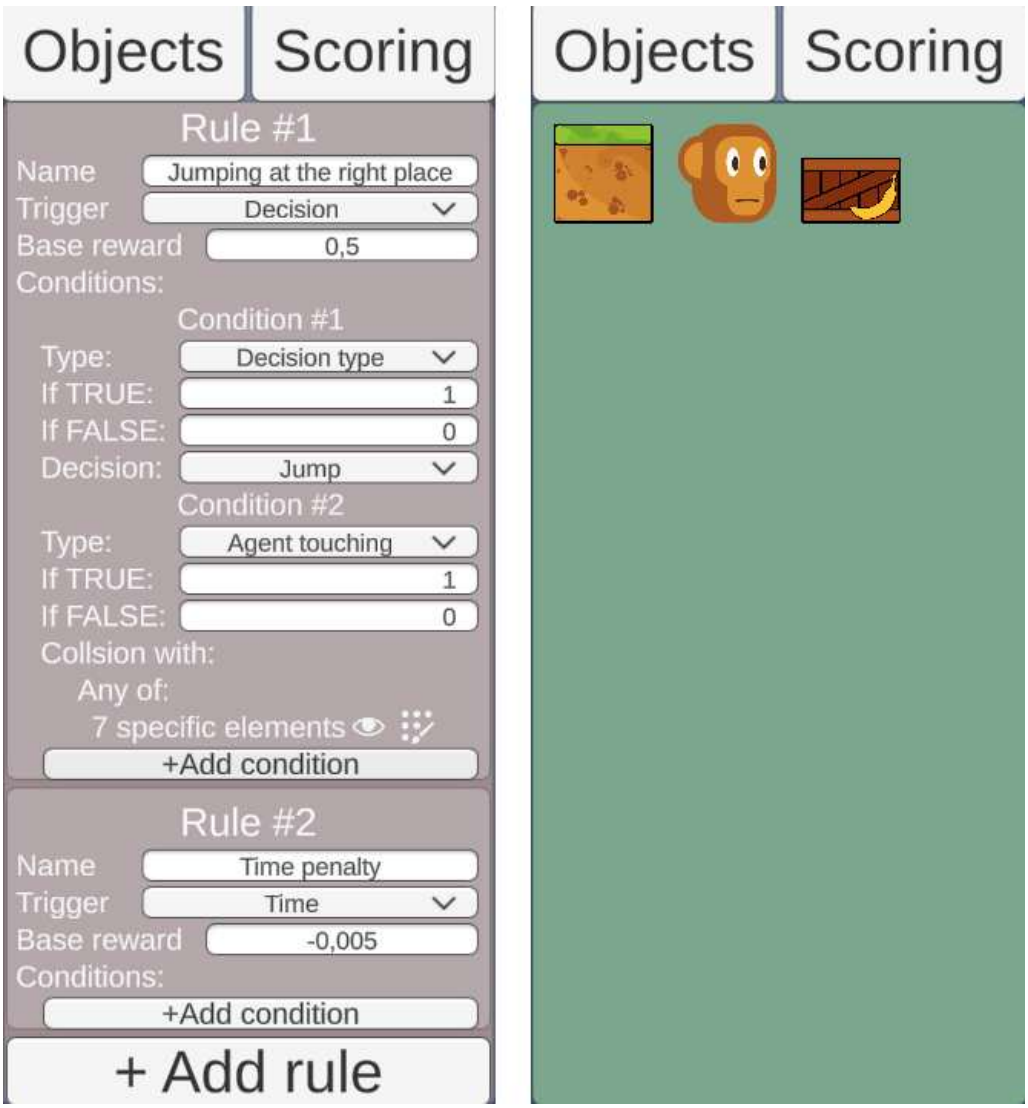
[14] *Importing assets* [https://docs.unity3d.com/Manual/ImportingAssets.html]. Unity Technologies, [b.r.]. Version 2021.3.

[15] JOHNSON, Daniel, M. John GARDNER a Ryan PERRY. Validation of two game experience scales: The Player Experience of Need Satisfaction (PENS) and Game Experience Questionnaire (GEQ). *International Journal of Human--Computer Studies*. 2018, roč. 118, s. 38–46. ISSN 1071-5819. Dostupné z DOI: https://doi.org/10.1016/j.ijhcs.2018.05.003.

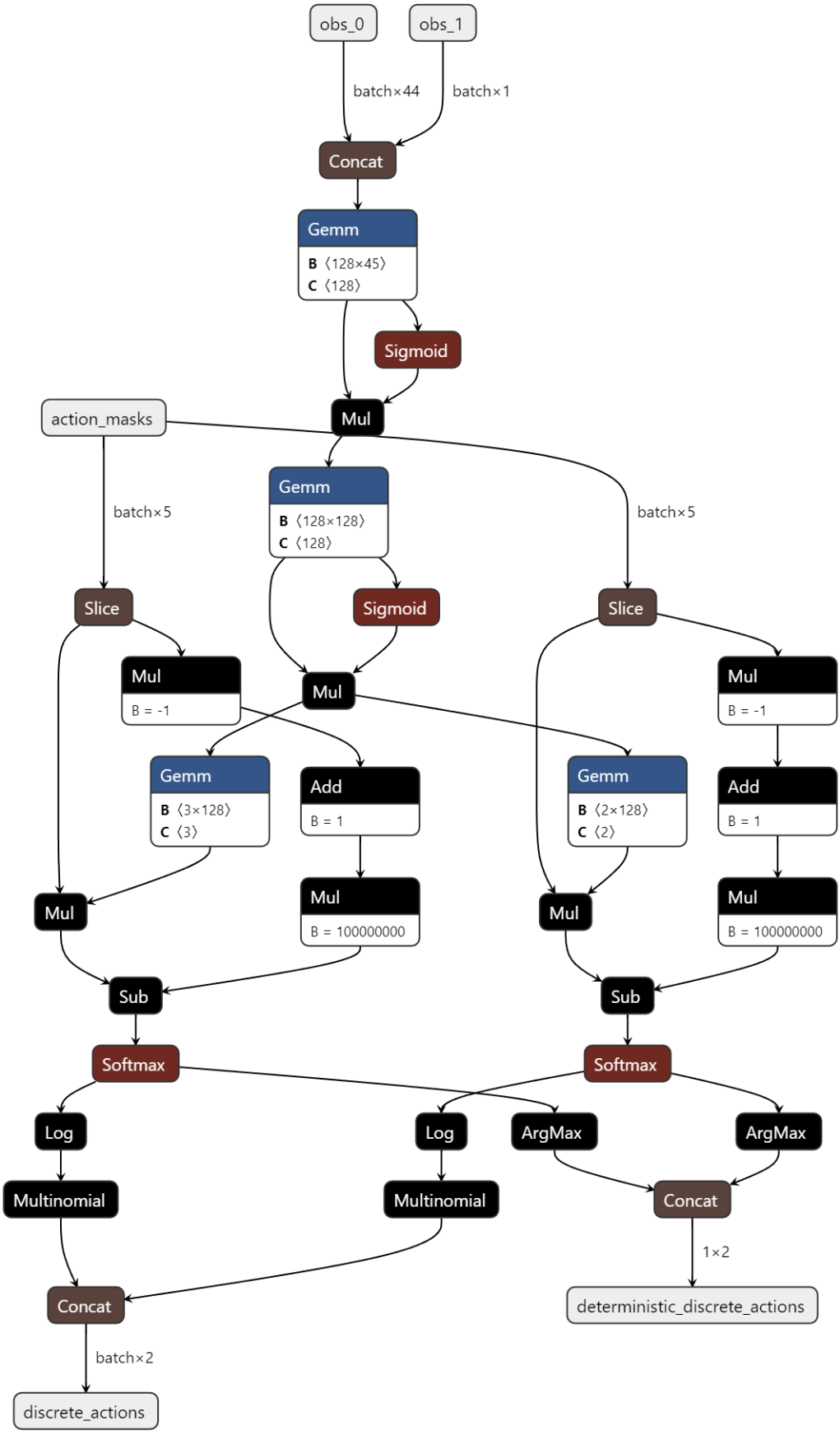
[16] IJSSELSTEIJN, W.A., Y.A.W. DE KORT a K. POELS. *The Game Experience Questionnaire*. Technische Universiteit Eindhoven, 2013.

[17] HARMAT, Laszlo et al. (ed.). *Flow experience*. 1. vyd. Cham, Switzerland: Springer International Publishing, 2016.

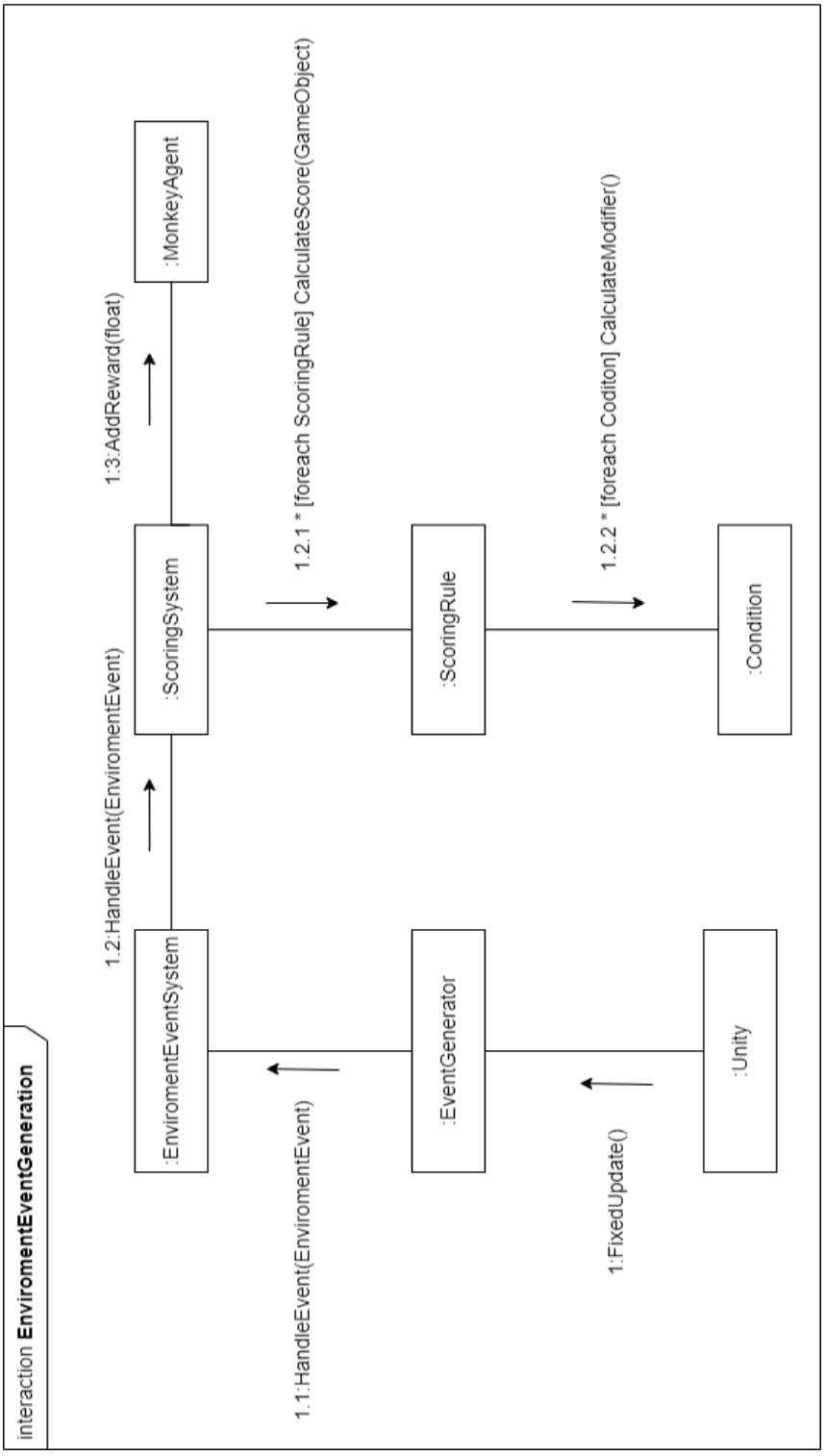
A Přílohy



Obrázek A.1: Postranní panel editoru prostředí karta hodnotícího systému (nalevo) a karta prvků (napravo)



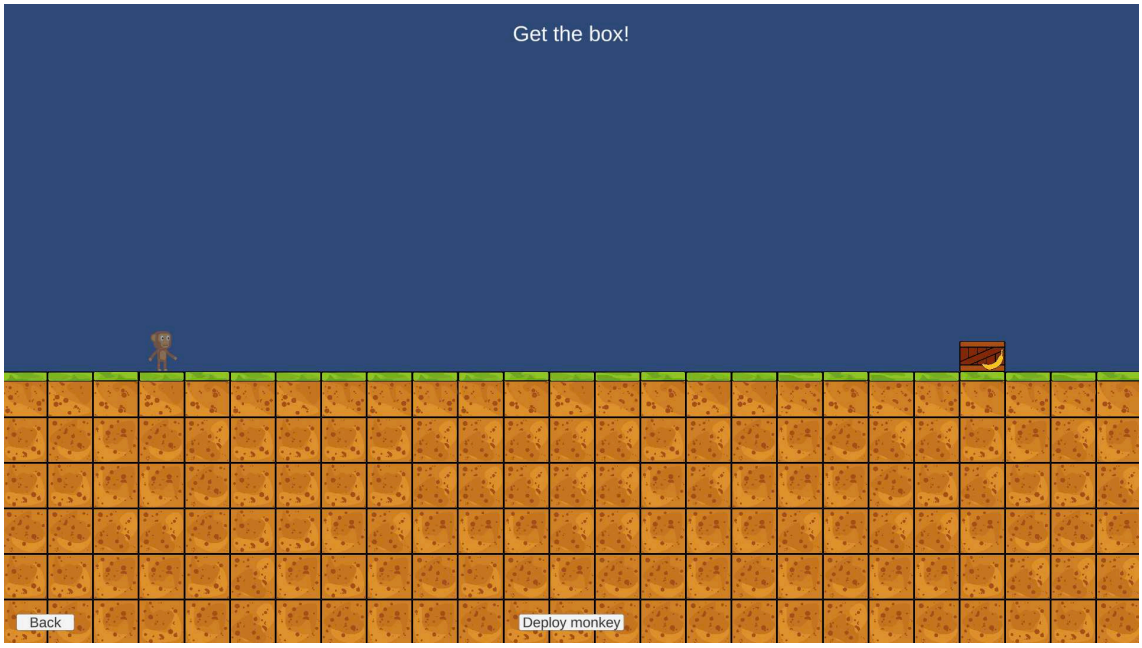
Obrázek A.2: Topologie neuronové sítě agenta[11]



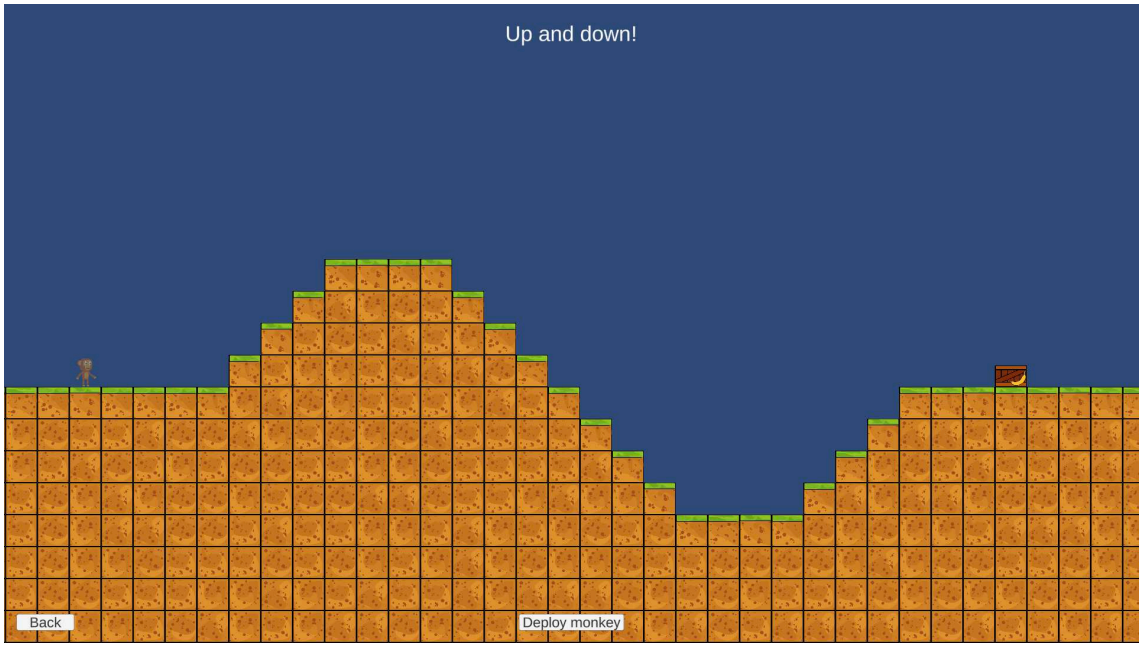
Obrázek A.3: Komunikace tříd při vzniku události prostředí

B Přílohy

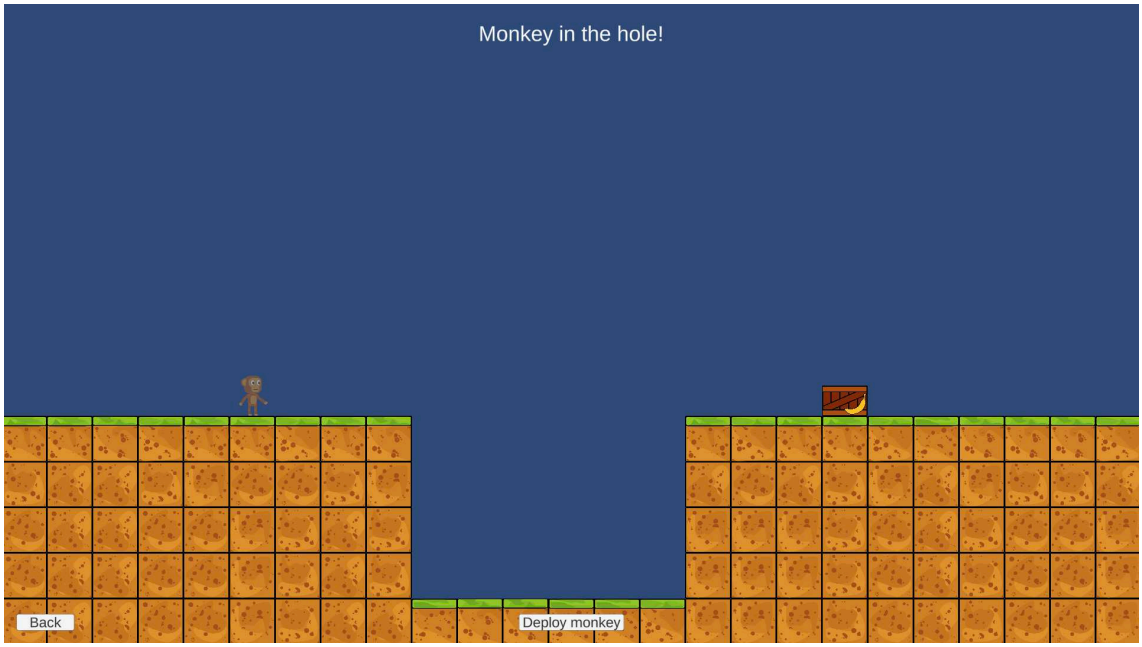
B.1 Mise hry



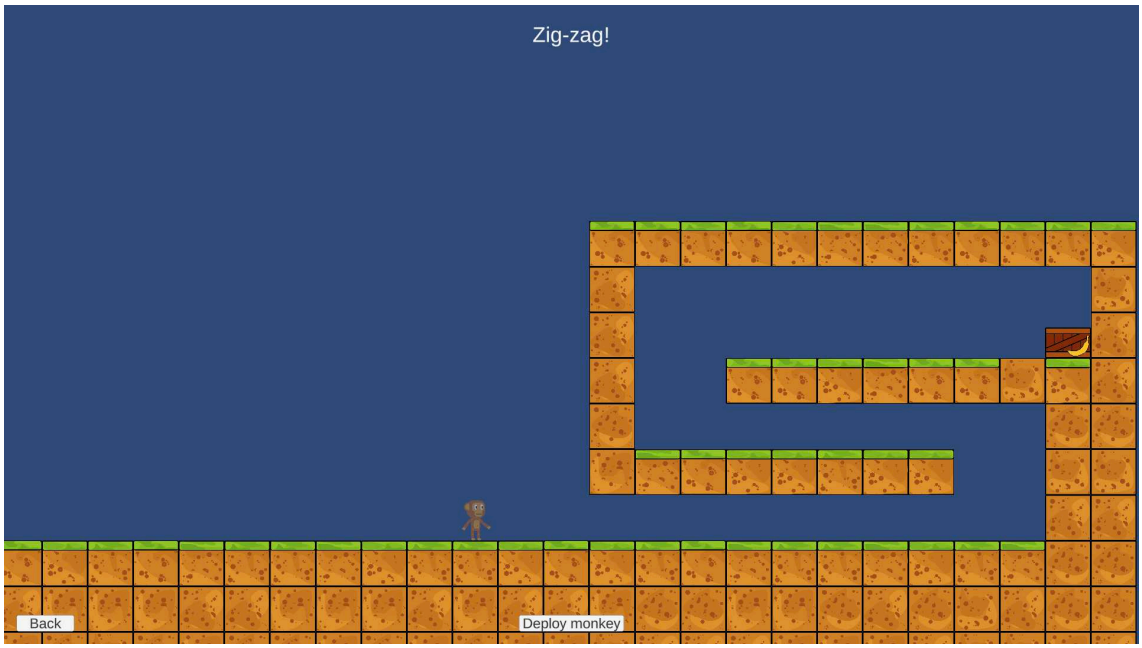
Obrázek B.1: První implementovaná mise



Obrázek B.2: Druhá implementovaná mise



Obrázek B.3: Třetí implementovaná mise



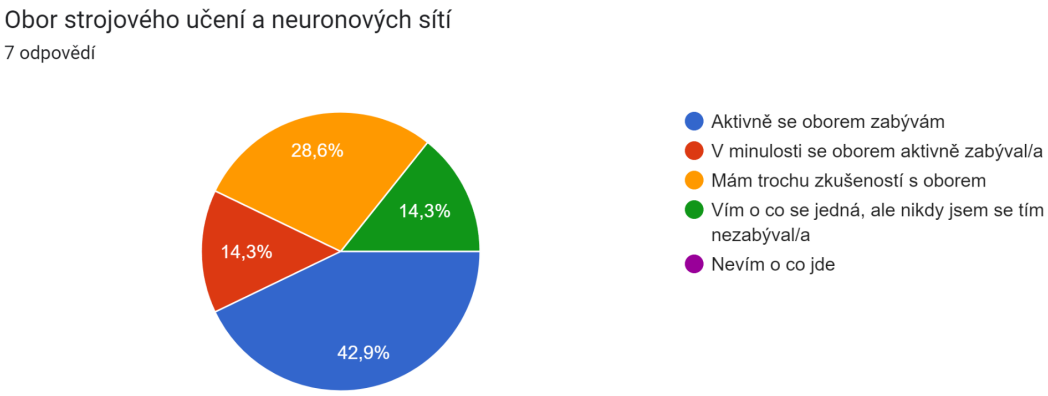
Obrázek B.4: Čtvrtá implementovaná mise

C Přílohy

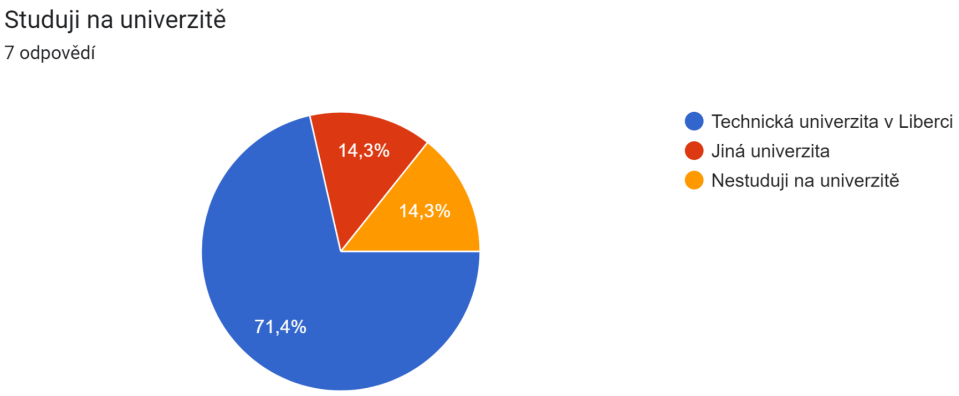
C.1 Odpovědi na dotazník

Tabulka C.1: Vlastní doplněné otázky na škále 0 až 4

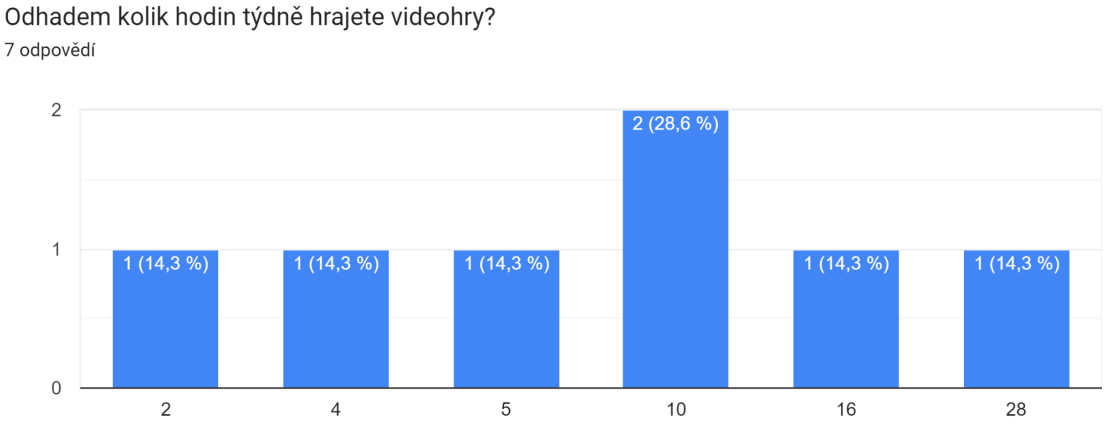
Otázka	**0**	**1**	**2**	**3**	**4**,	$\bar{x}$
Kolik misí jste dokázali splnit?	-	2	1	4	-	2,29
Jak obtížné bylo pro Vás hru zprovoznit?	1	1	2	3	-	2,00
Máte zájem o budoucí vývoj hry?	-	-	1	4	2	3,14



Obrázek C.1: Graf zkušeností hráčů s oborem neuronových sítí

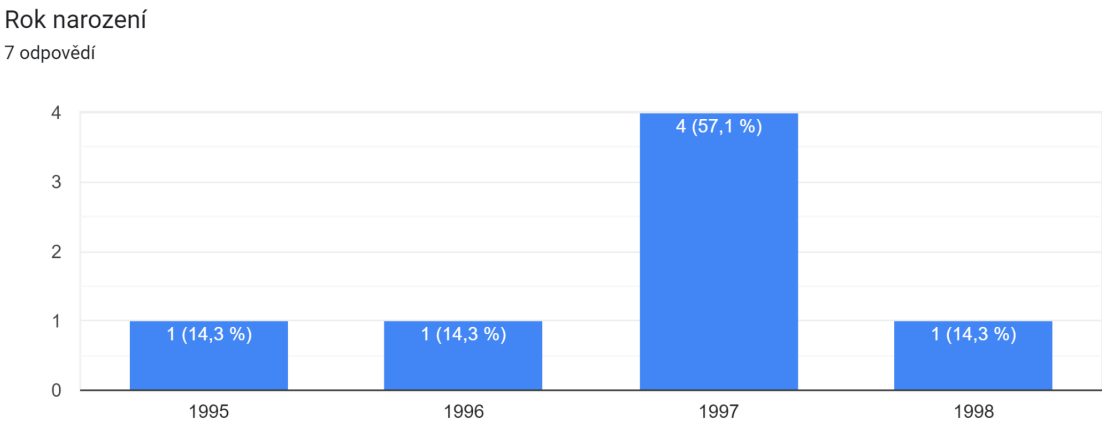


Obrázek C.2: Graf univerzit hráčů



Obrázek C.3: Histogram hodin hráčů strávených hraním videoher týdně

Obrázek C.4: Doba strávená hráči ve hře



Obrázek C.5: Rok narození hráčů

Tabulka C.2: Počty odpovědí na otázky GEQ na škále 0 až 4

#	**Cítil/a jsem se**	**0**	**1**	**2**	**3**	**4**	$\bar{x}$
1	Spokojeně	-	-	2	5	-	2,71
2	Kompetentně	1	2	2	1	1	1,86
3	Šťastně	-	-	6	-	1	2,29
4	Unaveně	2	3	-	2	-	1,29
5	Dovedně	1	1	3	2	-	1,86
6	Dobře	-	-	2	5	-	2,71
7	Znuděně	4	2	1	-	-	0,57
8	Úspěšně	1	-	4	2	-	2,00
9	Nápaditě	-	2	2	3	-	2,14
10	Otráveně	2	1	3	1	-	1,43
11	Pod tlakem	5	-	2	-	-	0,57
12	Soutěživě	-	1	2	2	2	2,71
13	Podrážděně	3	1	3	-	-	1,00
14	Frustrovaně	-	2	-	5	-	2,43
	O hře jsem si myslel/a, že:						
15	Je zábavná	-	-	3	4	-	2,57
16	Je těžká	-	-	1	6	-	2,86
17	Je působivá	-	2	1	3	1	2,43
18	Způsobila mi špatnou náladu	7	-	-	-	-	0,00
19	Jsem v ní dobrý/á	1	1	5	-	-	1,57
20	Plně mě zaneprázdnila	-	1	1	5	-	2,57
	Při hraní jsem						
21	Přemýšlel/a o jiných věcech	1	3	-	3	-	1,71
22	Zapomněl/a na vše okolo mě	2	1	2	2	-	1,57
23	Ztratil/a pojem o čase	1	-	1	4	1	2,57
24	Se hluboce na hru koncentroval/a	-	2	2	3	-	2,14
25	Cítil/a výzvu	-	-	-	4	3	3,43
26	Cítil/a časový nátlak	2	3	2	-	-	1,00
27	Mohl/a objevovat nové věci	-	-	4	3	-	2,43
28	Ztratil/a propojení s okolním světem	1	3	3	-	-	1,29
29	Musel/a se hodně snažit.	-	-	4	2	1	2,57
30	Byl rychlý/byla rychlá v dosažení cíle.	1	3	3	-	-	1,29
	Jak moc jsou následující tvrzení pravdivé						
31	Připadalo mi to jako bohatý zážitek	-	-	2	5	-	2,71
32	Užil/a jsem si to.	-	-	1	4	2	3,14