BAKALÁŘSKÁ PRÁCE

Přemysl Bašta

Vesmírná hra s umělou inteligencí

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Martin Pilát, Ph.D.

Studijní program: Informatika

Studijní obor: Obecná informatika

Praha 2020

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Tímto bych chtěl poděkovat svému vedoucímu, který mě v průběhu práce vedl a poskytoval mi podporu vědeckého, praktického i lidského charakteru. Dále bych chtěl poděkovat svojí rodině, mému bratrovi za projevený zájem o problematiku, kterou jsem se zabýval, mé matce za pomoc s jazykovou stránkou práce a mé přítelkyni za každodenní trpělivou podporu.

Název práce: Vesmírná hra s umělou inteligencí

Autor: Přemysl Bašta

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Martin Pilát, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Součástí této práce byla implementace mé vlastní, jednoduché, vesmírné hry, která slouží jako experimentální prostředí pro testování různých přístupů umělé inteligence. Nad stavy a akcemi hry byly vytvořeny abstrakce ve formě senzorických metod a akčních plánů, které umožňují jednoduše přecházet z informací nízké úrovně do informací vyšší úrovně a tak pomáhají algoritmům umělé inteligence jednodušeji manipulovat s agenty, kteří se ve hře pohybují. Jako algoritmy umělé inteligence byly pro hledání inteligentních agentů zvoleny Genetické programování a Hluboké Q-učení. Závěrečná část se soustředí na popsání chování nalezených agentů a vzájemné porovnání výsledků z provedených experimentů.

Klíčová slova: Vesmírná hra Umělá inteligence Genetické programování Hluboké Q-učení

Title: Space game with artificial intelligence

Author: Přemysl Bašta

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Martin Pilát, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Part of this thesis consists of the implementation of my own simple space game which serves as an experimenting environment for testing different aproaches of artificial inteligence. There have been created abstractions in a form of sensoric methods and action plans as a transition between low level and high level information about game state and actions. These abstractions help algorithms of aritifical inteligence with game agent manipulation. As far as algorithms are considered I chose Genetic programming and Deep Q-learning as main aproachces for inteligent agent development. Final part contains description of behaviour of developed agents and comparison of performed experiments.

Keywords: Space game Artificial Inteligence Genetic programming Deep Q-learning

Obsah

Úvod

Počítačové hry tvoří v současnosti velkou část zabavního průmyslu. S vývojem výkonějších výpočetních technologií se i rozšiřují možnosti, čeho všeho lze ve hrách dosáhnout. Proto v dnešní době lze ve hrách dělat téměř vše na co si vzpomeneme, existují hry sportovní, strategické, střílecí, závodící a mnohé další.

Kromě zábavního prožitku ze samotného hraní mají hry i další zajímavou vlastnost. Ve většině her tvoří herní prostředí samotný, uzavřený svět se svými vlastními zákonitostmi. Agent v prostředí dané hry obvykle ví, v jakém stavu se nachází, má na výběr konečný počet akcí, které může provést a cíl, kterého chce dosáhnout. Přestože herní prostředí může být velmi komplikované, v porovnání s prostředím skutečného světa se vždy jedná o mnohem jednodušší model.

Právě tato jednoduchost a uzavřenost prostředí počítačových her nám otvírá možnosti v experimentování s umělou inteligencí.

V rámci této práce nejprve navrhneme a implemetujeme jednoduchou vesmírnou hru, kterou využijeme pro experimentování s přístupy umělé inteligence. Po vytvoření hry navrhneme metody, jak přistupovat ke stavům a akcím hry tak, aby se mohl agent ve hře lépe orientovat. A následně se seznámíme s některými konkrétními algoritmy umělé inteligence. Vysvětlíme si jejich základní principy a ukážeme si, kde se tyto algoritmy dají využít. V závěrečné fázi budeme tyto algoritmy v různých experimentech aplikovat na herní prostředí naší vesmírné hry a naučit tak agenty inteligentnímu chování.

Hlavními cíli této bakalářské práce tedy budou:

1. Vyvinutí jednoduché vesmírné hry za úmyslem vytvoření herního prostředí pro experimentování.

2. Navrhnutí abstrakcí ke stavům a akcím hry pro intuitivnější přístup k hernímu prostředí.

3. Seznámení se s Genetickým programování a Hlubokým Q-učením a aplikováním těchto algoritmů v našem herním prostředí.

4. Nalezení agentů se zajímavým chováním. Zajímat nás budou jak agenti, kteří ve hře dosahují velmi dobrých výsledků, tak i agenti, jejichž chování je velmi pestré.

1. Navržená hra - Asteroidy

V této úvodní kapitole se seznámíme se základním chováním a cílem hry. Vysvětlíme zde základní zákonitosti, jaké objekty se ve hře vysktují, jaký mají význam a jak s nimi manipulovat.

1.1 Herní logika

Jedná se o hru dvou hráčů. Každý z hráčů ovládá svou vesmírnou loď. Prostředí hry má představovat zjednodušený vesmírný prostor, kde neplatí gravitační ani odporové síly. To má tedy za následek, že když se vesmírná loď rozletí v nějakém směru, tak v tomto směru letí i nadále i bez dalšího akcelerování.

Vesmírný prostor je v této hře nekonečný a dalo by se říct jistým způsobem cyklický, pokud vesmírná loď proletí dolní hranicí herního prostoru, tak nezmizí, ani nenaboural, ale objeví se na stejné pozici jen na horní hranici a opačně. Analogicky to platí i s bočními hranicemi. Vesmírné lodě sebou mohou prolétávat a nedochází ke srážce. Nejsou zde žádné statické překážky, kterým by bylo třeba se vyhnout. Co ale může způsobit srážku s vesmírnou lodí jsou asteroidy.

Asteroidy se generují na náhodných místech a s náhodným směrem i rychlostí letu. Nové asteroidy během hry vznikají tím častěji, čím déle hra trvá. V boji s asteroidy má hráč v zásadě dvě možnosti. Buď se může pokusit danému asteroidu vyhnout, tím že s lodí pohne mimo trajektorii asteroidu, anebo může asteroid sestřelit. Každý hráč má omezený počet životů a každá srážka lodě s asteroidem ubere hráči část jeho životů.

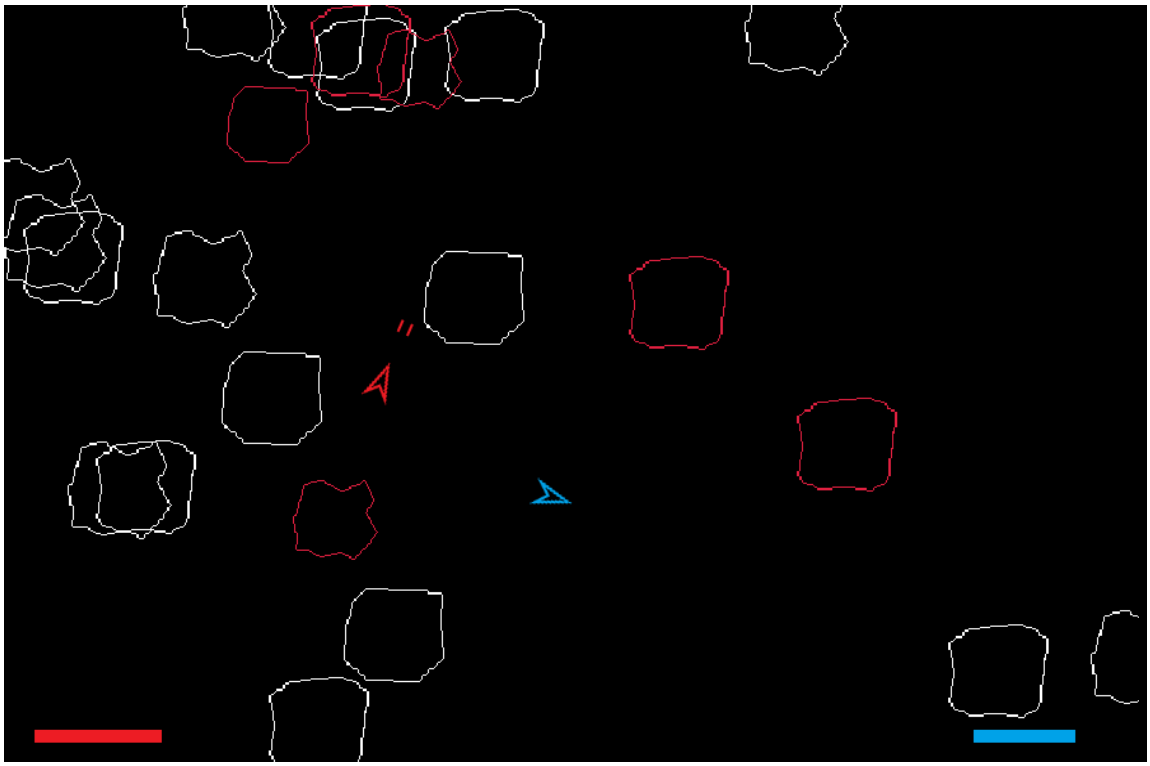
Hráč má k dispozici dva typy střel, obyčejnou a rozdvojovací. Vystřelená střela má značně vyšší rychlost než vesmírné lodě i než kolem letící asteroidy. Střely nejsou primárně určeny k přímému zasažení lodě protihráče, vesmírné lodě jsou k nepřátelským střelám imunní. Smyslem střel je sestřelování letících asteroidů, pomocí kterých teprve může k zásahu nepřítele dojít. Asteroidy mohou mít tři velikosti. Náhodně vytvořený asteroid je vždy největší. Každým rozstřelením asteroidu vznikají asteroidy o stupeň menší velikosti. Nově vytvořené asteroidy vznikají na základě typu rozstřelení na místě původního sestřeleného asteroidu. Typ rozstřelení závisí na druhu střely, jakou byl asteroid sestřelen.

V případě střely obyčejné vznikne namísto původního asteroidu jeden menší, který letí stejným směrem jako střela, která ho zasáhla. V případě střely rozdvojovací se původní asteroid rozstřelí na dva menší, kde každý z nich je oproti směru střely vychýlen o 15°po a proti směru hodinových ručiček. Pokud je zasažen asteroid nejmenší velikosti, tak již žádné další asteroidy nevznikají. Rychlost asteroidů je nepřímo závislá na jejich velikosti, čím je asteroid menší, tím vyšší rychlost má.

Asteroidy vzniklé rozstřelením se v jistém pojetí stávají projektily daného hráče, přičemž hráč nemůže být zasažen takto vytvořenými vlastními projektily.

1.2 Cíl hry

Během hry vzniká postupně více a více asteroidů, čímž je postupně stále obtížnější se všem vyhýbat, nebo je sestřelit. Hráč nemůže zranit nepřítele střelou přímo, může se ale snažit rozstřelit nějaký z kolem letících asteroidů tak, aby pomocí nově vzniklých asteroidů zasáhl nepřítele. Cílem hráče je ovládat svou loď takovým způsobem, aby nepříteli došly životy dříve než jemu samému. Hra končí pokud libovolnému z hráčů dojdou životy, v takovém případě se druhý hráč stává vítězem.



Obrázek 1.1: Screenshot ze hry

2. Architektura hry

V úvodní kapitole jsme se seznámili s fungováním hry z uživatelského pohledu. Zde se pro změnu podíváme jak je hra navržena interně, jaké stavební kameny obsahuje, jak jsou reprezentovány a jaký je jejich význam.

2.1 Vesmírné objekty

Všechny vesmírné objekty mají některá data společná. Každý vesmírný objekt má souřadnice své současné polohy a vektor rychlosti.

2.1.1 Asteroidy

Asteroidy mají navíc informace o tom, jaké jsou velikosti a zdali byly vytvořené nějakým z hráčů, tedy jsou projektily, anebo byly vytvořeny jako asteroidy neutrální. Na základě těchto dvou informací je asteroidu při vytvoření přiřazen obrázek, pomocí kterého je po dobu své existence vykreslován.

2.1.2 Střely

Vystřelené střely neletí věčně, ale mají omezenou životnost kolik kroků hry budou existovat. Tato hodnota se nastavuje z konfiguračního souboru z položky *BULLET_LIFE_COUNT*. V každém kroku hry se střele její životnost sníží o jedna a pokud se dostane na nulu, tak střela bude zničena. Střele se při vytvoření nastaví úhel, pod kterým poletí. Tento úhel je roven úhlu natočení vesmírné lodi, který měla při vystřelení. Přirozeně, stejně jako u asteroidů, i u střely musíme evidovat, kterému z hráčů patří, toto je řešeno odkazem na objekt vesmírné lodi, která střelu vystřelila. Jak již bylo zmíněno v předchozí kapitole, střely jsou dvojího druhu. Příznakem *split* se určuje zda se jedná o střelu obyčejnou nebo rozdvojovací

2.1.3 Vesmírná loď

Vesmírná loď má základní polohové informace rozšířené o úhel. Ten se s každou rotací lodě zvětší nebo zmenší o 12°. Akcelerace je realizována pomocí vektorového sčítání. K současnému vektoru rychlosti se přičte vektor odpovídající současnému natočení lodi. Maximální rychlost vesmírné lodi je omezená. V případě že akcelerací vznikne vektor rychlosti, jehož délka je větší než hodnota maximální rychlosti, tak dojde k jeho zkrácení. Směr vektoru se zachová, ale je délka bude zkrácena na maximální možnou délku.

2.2 Prostředí

Hra běží v cyklu diskrétních kroků, které dohromady simulují plynulý pohyb hry. Herní prostředí je inspirováno projektem *open ai gym* od Google (Brockman a kol. (2016)). Jedním rozdílem je však přístup k vykreslování hry. V případě *open ai gym* se prostředí vykresluje zavoláním metody *render()* na instanci prostředí zvenku. Já jsem zvolil přístup jiný. V případě, že chceme hru graficky zobrazovat, předáme v konstruktoru prostředí grafický modul, který vykreslování vesmírných objektů implementuje. A prostředí už poté objekty graficky vykresluje interně samo. Rozhodnutí, že se má grafický modul volitelně injektovat v konstruktoru a nemá být natvrdo svázán s prostředím, jsem učinil z důvodu větší nezávislosti modulů. Při další práci s knihovnami pro evoluční algoritmy se ukázalo být pevné svázání herního prostředí s grafickým modulem problematické.

Herní prostředí se stará o manipulaci všech vesmírných objektů a akcí s nimi spojenými. V každém kroku dostává od hráčů akce, které chtějí provést, a prostředí na to odpovídajícím způsobem reaguje. Akce každého hráče jsou reprezentovány polem, které obsahuje elementární možné akce:

- Rotace vlevo
- Rotace vpravo
- Akcelerace
- Obyčejná střela
- Rozdvojovací střela
- Prázdná akce

Hráč může provádět více akcí najednou. Na základě přítomných elementárních akcí se provádí dané reakce. Prostředí se stará o vesmírné objekty přímo. V případě elementárních akcí, které mění rychlost nebo orientaci vesmírné lodi, prostředí zavolá funkce, které požadované změny na vesmírné lodi provede. A v případě elementárních akcí střel se na základě polohy a orientace dané vesmírné lodi vytvoří nová střela, kterou opět bude mít ve správě právě prostředí.

Hra, jak již bylo řečeno, má být konečná, toho je docíleno narůstajícím počtem asteroidů. Toto inkrementální generování asteroidů je také zodpovědností herního prostředí. Herní prostředí si pamatuje počet kroků, který uběhl od posledního vytvoření asteroidu. Pokud tento počet překročí danou mez, tak prostředí vytvoří nový asteroid. Postupného nárůstu nových asteroidů je docíleno inkrementálním snižováním této meze.

Další důležitou funkcí herního prostředí je kontrola srážek. Všechny objekty jsou prostorově reprezentovány jako kruhy s danými poloměry. Postupně se prochází všechny objekty, u kterých nás zajímají srážky a Euklidovskou metrikou se kontroluje, zda od sebe nejsou vzdáleny méně, než je součet jejich poloměrů. V případě srážky se prostředí postará o správnou reakci - zničení nebo změnu sražených objektů a případně vytvoření nových objektů vzniklých srážkou.

V rámci srážek se upravují také odměny jednotlivých hráčů. Odměna je hodnota, která vyjadřuje, jak úspěšný byl tento krok pro každého z hráčů. V každém kroku, který hráč přežil, obdrží odměnu hodnoty 1. Existují ale

konkrétní srážky objektů, které hodnotu odměny mohou změnit. V případě, že hráč sestřelil nepřátelský asteroid, nebo svým asteroidem srazil nepřátelskou loď, se výše odměny zvýší. Naopak, pokud byla jeho vesmírná loď zasažena nepřátelským asteroidem, je hodnota odměny snížena. Koncept odměn nijak neovlivňuje samotný běh hry, ale bude se nám hodit v dalších kapitolách v umělé inteligenci.

Nezmínili jsme zatím pohyb objektů. I ten přirozeně spadá do logiky herního prostředí. Zde se prochází seznamy všech vesmírných objektů a pohyb se provede přičtením jejich vektoru rychlosti k současné poloze. Po provedení pohybu se u všech objektů provede kontrola, zda se náhodou nevyskytují mimo herní prostor. Pokud toto nastane, tak jsou příslušné objekty vráceny zpět do prostoru na své odpovídající místo.

Pokud byl při vytvoření prostředí předán grafický modul, tak se prostředí postará i o vykreslení vesmírných objektů. Pro všechny vesmírné objekty se na grafickém modulu zavolá příslušná metoda pro jejich vykreslení. Způsob implementace vykreslení jednotlivých objektů již není odpovědností herního prostředí, ale o to se stará grafický modul.

Poslední zatím nezmíněnou funkcí herního prostředí je kontrola, zda hra neskončila. Na konci kroku herní prostředí kontroluje, zda mají oba hráči kladný počet životů a případě hru ukončí a informuje agenty o konečném stavu.

Jedna instance prostředí odpovídá jedné hře. Herní prostředí má dvě základní metody pro řízení hry.
Metoda *reset()* inicializuje hru do počátečního stavu a tento stav vrátí. Tato metoda se musí zavolat před začátkem hry.
A druhá metoda *next_step(actions_one, actions_two)*, která na na základě akcí hráčů, převede hru do následného stavu. Právě v této metodě je schovaná celá logika manipulace s vesmírnými objekty popsána výše.

```
def next_step(actions_one, actions_two):
    handle_actions(actions_one, actions_two)
    generate_asteroid()
    check_collisions()
    move_objects()
    draw_objects()
    check_end()
    return step_count, game_over, state, reward
```

2.2.1 Stav prostředí

Herní prostředí vrací po každém kroku současný stav hry. Stav hry se skládá ze seznamů všech vesmírných objektů včetně kompletních informací o nich. Pravděpodobně by bylo možné vracet i méně obsáhlou informaci o současném stavu hry. Avšak motivací pro mě bylo předávat kompletní informace o všech objektech a následně až v jednotlivých experimentech volit pro rozhodování jen omezené, či z těchto kompletních informací vyextrahované, informace o stavu. Mou snahou bylo, aby herní prostředí neomezovalo agenty v jejich rozhodování a poskytovalo jim kompletní informace.

2.3 Agent

Agent je ústřední postavou celé hry a obzvláště v dalších kapitolách pro nás bude nejzajímavějším předmětem zájmu. Je to právě toto místo, kde budeme později mluvit o umělé inteligenci. V případě agenta, který je ovládán lidským hráčem, se agent, respektive člověk, který jej ovládá, neřídí datovou reprezentací stavu, tak jak jej obdržel od herního prostředí. Ale rozhoduje se na základě toho, jak hráč vizuálně vnímá herní průběh a příkazy k provedení jednotlivých akcí udává ovládáním kláves na klávesnici. Lidský hráč pro nás ale nebude primárním předmětem zájmu, my se budeme spíše soustředit na agenty strojové.

Každý agent musí implementovat jedinou metodu *choose_actions(state)*, která musí vracet akce, které chce agent v současném stavu hry vykonat. Úkolem agenta je, na základě obdrženého stavu, zvolit akci, kterou chce provést. A právě tento rozhodovací problém pro nás bude v dalších kapitolách předmětem experimentování s různými abstrakcemi a přístupy umělé inteligence.

2.4 Grafické prostředí

Pro grafické zobrazování hry jsem zvolil python knihovnu pygame Shinners (2011), která, jak si lze z názvu domyslet, slouží k programování jednodušších her v pythonu. Tato knihovna nabízí kromě různých grafických funkcí, také podporu pro manipulaci s herními objekty. Mimo jiné je v této knihovně zabudovaná podpora pro kontrolu srážek herních objektů. Mou přirozenou snahou bylo této funkcionality využít, ale toto se později ukázalo být nevhodné. Prvním problémem bylo, že herní objekty jsou v rámci této knihovny reprezentovány pomocí čtverce, nikoliv jako kruhy a kontrola srážek je tedy realizována jako dotaz, zda se dva odpovídající čtverce pronikají. Tato kontrola srážky je výpočetně náročnější než kontrola v případě objektů, které jsou reprezentovány kruhem. Závažnějším problémem se ukázala být integrace pygame modulu do herního prostředí. Při pokusu o paralelizaci více běhů hry se objevily technické problémy, které se mi ani po značném úsilí nepodařilo vyřešit. Proto jsem se rozhodl, že pygame modul budu využívat pouze pro grafické vykreslování hry a kontrolu srážek si naimplementuju sám. V tomto rozsahu pro mě knihovna pygame byla zcela dostačující. Vytvořil jsem si sadu obrázků reprezentující jednotlivé vesmírné objekty a pomocí knihovny pygame je mohu vykreslovat potřebným způsobem.

2.5 Hlavní herní cyklus

Vysvětlili jsme tedy všechny základní prvky, které v této hře potřebujeme a nyní je propojíme dohromady. K běhu hry potřebujeme inicializovat instanci herního prostředí a dva agenty, kteří budou představovat naše hráče. a následně můžeme začít v cyklu simulovat hru. V každém kroku cyklu agenti na zvolí své akce a ty předají zpět hernímu prostředí. Takto hra běží, dokud herní prostředí neoznámí, že daným krokem hra skončila. Kostra herní simulace pak vypadá následovně:

```
env = Enviroment()
agent_one = Some_agent()
agent_two = Some_agent()
state = env.reset()
game_over = False

while not game_over:
    actions_one = agent_one.choose_actions(state)
    actions_two = agent_two.choose_actions(state)

    game_over, state = env.next_step(actions_one, actions_two)
```

2.6 Instalace a spuštění hry

Instrukce pro instalaci a spuštění hry jsou popsány v souboru README.

3. Senzory a akční plány

V této kapitole se již přesouváme od hry samotné ke způsobu, jak na daný stav hry nahlížet chytřeji a také, jak na něj lépe reagovat. Na nejnižší úrovni herní prostředí vrací v každém kroku stav hry, který obsahuje seznamy všech vesmírných objektů a úkolem agentů je na něj reagovat vybráním elementární akce. V této kapitole se zaměříme na abstrakce, které z obsáhlých a detailních informací nízké úrovně získají menší objemy zajímavějších informací vyšší úrovně. Podobně jako abstrakce nad herním stavem budeme chtít vytvořit i abstrakce nad akcemi agentů, jejich cílem bude namísto volby elementárních akcí nízké úrovně volit akce vyšší úrovně. A právě tyto abstrakce realizujeme pomocí senzorů a akčních plánů.

3.1 Senzor

Senzorem nazveme metodu, která z detailního stavu reprezentovaného seznamem vesmírných objektů extrahuje informaci vyšší úrovně, která není ve stavu explicitně zadána. Informace získané z těchto senzorů, respektive senzorických metod, můžeme využít v rozhodovacím problému vybírání akcí. Většina senzorických metod v rámci svého výpočtu využívá simulování hry. Simulace na základě současného rozpoložení všech vesmírných objektů ve stavu pokračuje v pohybu všech objektů tak, jak by hra probíhala obvyklým způsobem. V simulaci žádný z hráčů neprovádí žádné akce, kromě těch, na jejichž dopad se v dané simulaci dotazujeme. Ze simulace můžeme zjistit jaké konkrétní události nastanou v nejbližších krocích hry. Všechny simulace pokračují ve hrání hry jen omezený počet kroků. Tento počet kroků je roven konstantě *IMPACT_RADIUS*. Pro tuto hodnotu se mi ukázal být vhodný počet 25. Je to hodnota dostatečně vysoká na to, aby senzorické metody včas zaznamenaly potřebné informace a zároveň je dostatečně nízká, aby nebyly výpočetně příliš náročné. Kromě senzorů, které využívají pro výpočet simulování hry, existují i senzory, jejichž výpočet probíhá nad současným stavem staticky bez nutnosti simulace.

3.1.1 Příklady senzorů

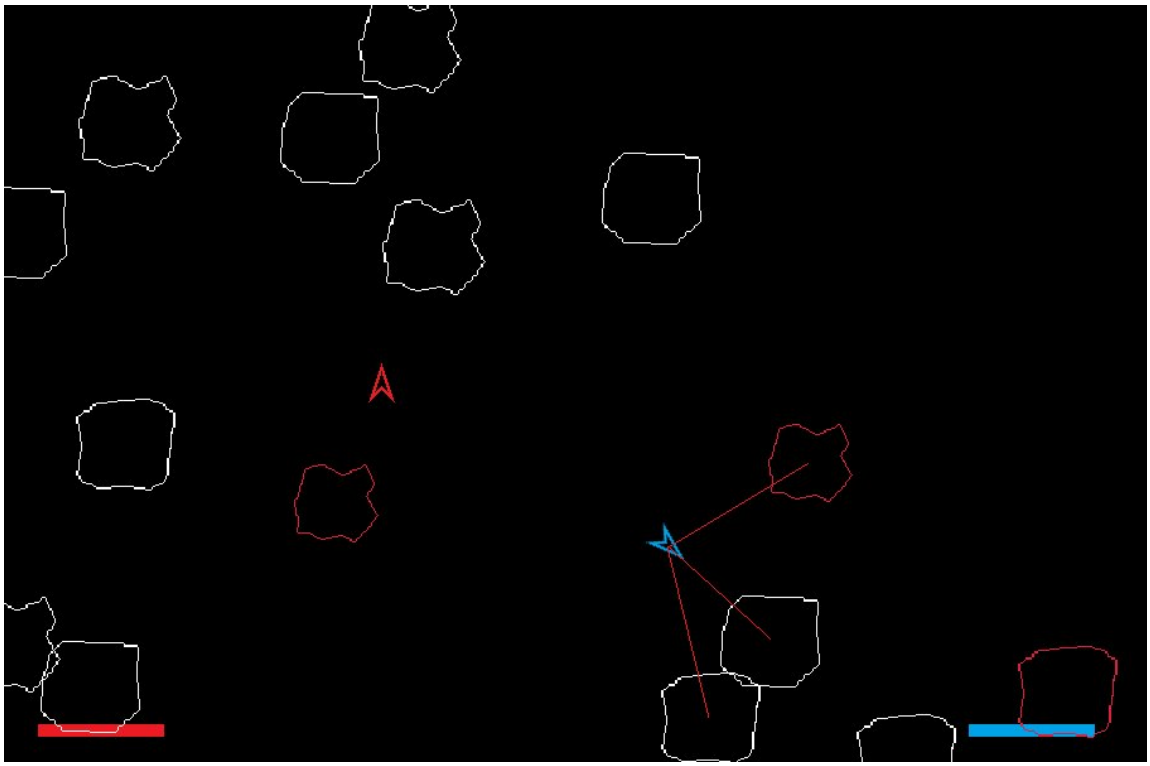
- První sražený neutrální asteroid - Zde se simuluje pohyb vesmírné lodi, střel a neutrálních asteroidů. V případě srážky vesmírné lodi s neutrálním asteroidem vrací tento senzor daný neutrální asteroid a počet kroků, po kterém ke srážce došlo. Do simulace se zahrnují i vlastní střely hráče. V případě sestřelení asteroidu střelou, jsou střela i asteroid z následné simulace odstraněny.

- První sražený nepřátelský asteroid - Jde o téměř identický senzor. Jediným rozdílem je zde, že se senzor nesoustředí na asteroidy neutrální, ale nepřátelské.

- Asteroid zasáhne nepřátelskou loď - V této simulaci se pohybujou pouze konkrétním asteroidem a nepřátelskou lodí. Senzor vrací informaci zda, a

případně v kolika krocích simulace se asteroid střetl s nepřátelskou lodí. V simulaci nepřátelská loď neprovádí žádné reakce.

- Střela zasáhne konkrétní asteroid - Jde o simulaci podobnou předchozí. Simuluje se pohyb konkrétního asteroidu a konkrétní střely.
- Střela zasáhne libovolný asteroid - Simuluje se pohyb konkrétní střely a všech neutrálních a nepřátelských asteroidů. Tato senzorická metoda vrací zda střela zasáhla sestřelila asteroid, daný asteroid a počet kroků, po kterém k sestřelení došlo.
- Vzdálenost dvou bodů - Vrátí vzdálenost dvou bodů v Euklidovské metrice.
- Přepočítání nejbližší polohy asteroidu od vesmírné lodi - Vesmírný prostor je v jistém pohledu cyklický, proto se pro získání nejkratší vzdálenosti asteroidu od vesmírné lodi nemůžeme spolehnout na vzdálenost jejich současných poloh, ale musíme vzít v úvahu všechny čtyři polohy asteroidu. Ty získáme postupným posunutím asteroidu o šířku a délku prostoru. Jinými slovy vzdalenost vesmírné lodi od asteroidu posunutého přes hranici prostoru může být kratší než přímá vzdálenot k původní polohy asteroidu.
- N nejbližších asteroidů od vesmírné lodi - Tento senzor spočítá nejkratší vzdálenosti vesmírné lodi od všech asteroidů ve hře a vrátí relativní polohu N nejbližších z nich.



Obrázek 3.1: Ukázka senzoru "N nejbližších asteroidů od vesmírné lodi"pro N=3

3.2 Akční plán

Druhou zmíněnou abstrakcí jsou akční plány, jejich cílem je podobně jako u senzoru pracovat namísto akcí nízké úrovně s akcemi vyšší úrovně. V případě akcí toto znamená namísto volby jednotlivých elementárních akcí volit akční plány, který mají komplexnější cíl. Akční plán reprezentuje posloupnost elementárních akcí, které má agent vykonávat v následujících krocích hry. Cíl akčního plánu se liší podle toho, o jaký druh se jedná.

Podobně jako u senzorických metod i zde hledání většiny akčních plánů probíhá simulací hry.

3.2.1 Jednotlivé plány

- Útočný plán - Tento akční plán hledá posloupnost rotací následovaných střelou tak, aby byl sestřelen asteroid, který poté zasáhne nepřátelskou loď. Simulace inkrementálně prochází všechny možné otočení a následné střely. Vesmírná loď se v rámci simulace otočí, vystřelí se střela a následně se sleduje, zda tato střela zasáhne libovolný velký nebo střední asteroid. Nad tímto asteroidem se následně zkouší, zda-li asteroidy vzniklé rozstřelením zasáhnou nepřátelskou loď. Rozstřelení asteroidu se zkouší pro oba typy střel.

  V případě nesestřelení žádného asteoridu, nebo minutí nepřátelské lodi rozstřeleným asteroidem, se v simulaci vesmírná loď pokusí provést další rotaci a celý pokus o střelbu opakovat. Pokud v simulaci nastalo úspěšné sestřelení, tak se vrací útočný akční plán, který obsahuje přílušný počet rotací následovaný střelou.

- Obranný plán - Pro hledání obraného plánu je nejprve potřeba vědět, před kterým asteroidem je potřeba vesmírnou loď bránit. Zde využijeme připravené senzory. Posloupnost obraného plánu obsahuje dvě části. V První části se vesmírná loď rotuje tak, aby mířila na asteroid, před kterým je potřeba se bránit. A v druhé části probíhá samotná střelba, ta spočívá v jediné elementární akci střelby. Pro vyhodnocení, zda vystřelením střely sestřelíme konkrétní asteroid, využijeme připravené senzorické metody.

  Nalezení potřebné rotace pro nasměrování vesmírné lodi k nebezpečnému asteroidu probíhá statickým výpočtem bez použití simulace. Nejprve zjistíme přesnou polohu, kde k potenciální srážce dojde. V případě, že vesmírná loď, anebo asteroid stojí staticky na místě, tak ke srážce dojde na současné poloze vesmírné lodi. V případě pohybu obou objektů se poloha střetu nachází na průniku jejich trajektorií. Tuto polohu střetu musíme vzít v úvahu při nasměrovávání vesmírné rakety k asteroidu. Pokud by se vesmírná loď otočila pouze vzhledem k současné poloze asteroidu, tak by vystřelením mohla letící asteroid minout. Proto namísto současné polohy asteroidu musí vesmírná loď mířit na cílovou polohu. Cílová poloha pro nás bude bod, který leží na úsečce spojující současnou polohu asteroidu a polohu střetu objektů a to ve vzdálenosti 15% jejich vzdálenosti blíže k poloze asteroidu. Tímto způsobem bude vesmírná loď mířit mírně před asteroid do jeho trajektorie. Empiricky jsem vyzkoušel, že tato vzdálenost v

praxi funguje velmi dobře. Po vypočtení cílové polohy stačí spočítat rozdíl současného úhlu vesmírné lodi od úhlu směřujícímu k cílové poloze. A z tohoto rozdílu snadno získáme potřebný počet rotací. Ty budou tvořit výsledný obranný plán.

- Úhybný plán - Jedná se také o defenzivní plán, ale namísto přímého sestřelení nebezpečného asteroidu se tento akční plán snaží asteroidu vyhnouot. Podobně jako u obranného plánu potřebujeme vědět před jakým asteroidem je potřeba se vyhnout. Tento asteroid je stejně jako v předchozím obranném plánu získaný senzorem. Po nalezení nebezpečného asteroidu simulace postupně prochází všechny možné otočení a následnou akceleraci. Počet potřebných akcí akcelerace probíhá také inkrementálně. Nejprve se vyzkouší provedení jedné akce akcelerace a následně se simuluje pohyb vesmírné lodi a asteroidu bez dalšího akcelerování. Při srážce vesmírné lodi s asteroidem se oba objekty vrátí na původní místo a vyzkouší se o jednu akci akcelerace více než v předešlém případě. V případě úspěšného vyhnutí bude úhybný akční plán obsahovat posloupnost akcí rotace následované posloupností správného počtu akcí akcelerace.

- Zastavovací plán - Tento plán jsem vytvořil na základě sledování chování agenta využívajícího úhybný plán. Úhybný plán vždy vrací nejkratší možný plán, který stačí na vyhnutí se srážce. Jako důsledek proto obvykle plán obsahuje minimální počet rotací, který je dostačující. To mělo v praxi za následek, že agent, který se řídí pouze úhybními plány používá akceleraci mnohem častěji než rotace a lítá proto obrovskou rychlostí přibližně stejným směrem napříč prostorem. Proto mě napadlo vytvořit akční plán, který vesmírnou loď uvede do klidu.

  Zastavovací plán má přímočarou myšlenku. Nejdříve se vesmírná loď otočí tak, aby byla nasměrována proti směru svého pohybu a následně provede potřebný počet akcelerací, aby zpomalila až do úplného zastavení. V tomto akčním plánu není potřeba provádět simulaci, výpočet potřebného počtu rotací i následných akcelerací lze spočítat statickým výpočtem.

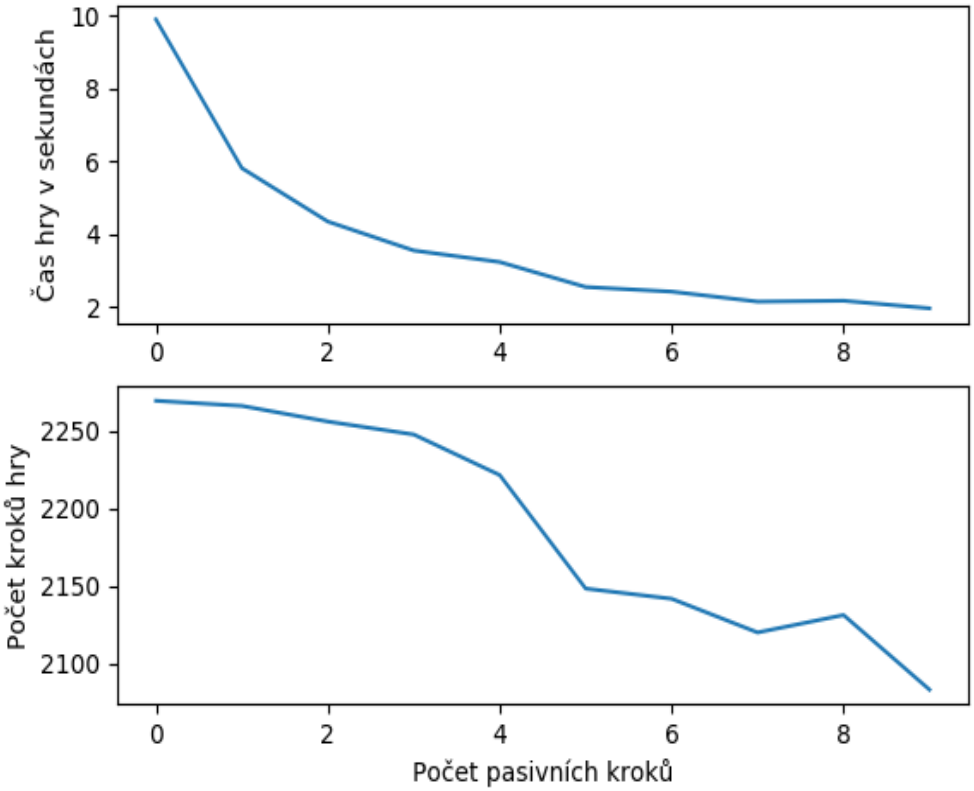
  V kombinaci s úhybným plánem se agent chová v jistém smyslu klidněji. Namísto zběsilého letu napříč prostorem se vesmírná loď po vyhnutí asteroidu zastaví.

  Tento akční plán jsem vytvořil na základě svého lidského instinktu, jak bych očekával od inteligentního agenta, že by se mohl chovat. Zda bude mít tento akční plán v praxi reálný přínos uvidíme v pozdější kapitole, kde budeme s akčními plány experimentovat.

3.2.2 Přepočítávání akčních plánů

Získávání akčních plánů je výpočetně náročné, proto jsem se pokusil jejich přepočítávání vhodně omezit. Akční plány nám vracejí posloupnost akcí na více kroků dopředu a proto není vždy potřeba je přepočítávat v každém kroku. Změny mezi dvěmi po sobě jdoucími stavy hry nejsou veliké, ale mohou být občas dostatečné na to, aby se přepočítaný plán lišil od toho spočítaného v předchozím kroku. V každém kroku hry se proto rozhoduje, zda se daný plán přepočítá, nebo

se bude následovat plán původní. Plán se bude přepočítávat ve dvou možných případech, těmi jsou uplynutí daného počtu pasivních kroků, ve kterých jsme pouze pokračovali v již vytvořeném původním plánu, a nebo jeho dokončení. Počet neaktivních kroků, po kterých dochází k přepočítávání plánu, se rovná konstaně *INACTIVE_STEPS_LIMIT*. S vyšším počtem pasivních kroků se velmi výrazně snižuje výpočetní čas potřebný k odehrání hry, ale počet kroků, který hra trvala, se sníží jen minimálně (viz obr03). Je zde vidět, že kvalita hráčů se mírně snižuje, ale v porovnání kolik se ušetří výpočetního času, je tato ztráta kvality zanedbatelná. V příštích kapitolách se nám bude pro učení agentů hodit odehrát velké množství her, proto si dovolíme nastavit vyšší počet pasivních kroků.



Obrázek 3.2: Srovnání počtu neaktivních kroků k průměrné délce hry. Čísla byla získána průměrováním 40 her, kde proti sobě hráli agenti využívající pouze obranných plánů.

3.2.3 Použití akčních plánů

Metody, které vypočítávají akční plán vracejí kromě plánů samotných i jejich délku. V případě nenalezení akčního plánsu se místo jeho délky vrací konstanta *NOT_FOUND_STEPS_COUNT* vysoké hodnoty, ta reprezentuje, že akční plán neexistuje. Díky tomu lze tyto metody použí také jako senzorické metody.

4. Algoritmy umělé inteligence

V této kapitole se seznámíme se dvěma přístupy, které spadají do odvětví umělé inteligence. Představíme jejich základní myšlenku. Zmíníme, kde se tyto algoritmy dají použít a v následující kapitole je i využijeme v našem herním prostředí pro trénování inteligentních agentů.

4.1 Genetické programování

4.1.1 Základní princip

Poli a kol. (2008) Genetické programování je evoluční technika, která vytváří počítačové programy. Cílem genetického programování je vyřešit co nejlépe zadaný problém, neklademe však žádné požadavky na to, jakým způsobem je potřeba ho vyřešit.

Genetické programování spadá do kategorie evolučních algoritmů, s těmi se pojí jistá terminilogie. Každý program budeme nazývat jedincem a jejich množinu populací. Algoritmus poběží iterativně v generacích. V každé iteraci se provede výběr některých jedinců z populace, ti se pomocí genetických operátorů a případných mutací upraví, a na závěr se rozhodne, kteří z nich přežijí do další generace. Jedince, kteří se v generaci vysktují na začátku iterace, nazývame rodiče. A podobně jedince, kteří byli na konci iterace vybráni do další generace, se nazývají potomci. Každý program představuje jedno konkrétní řešení daného problému. Kvalitu tohoto řešení ohodnocujeme takzvanou fitness funkcí. Čím vyšší hodnotu tato funkce jedinci přidělí, tím je lepším řešením daného problému.

Reprezentace

Programy jsou v genetickém programování obvykle reprezentované syntaktickými stromy. Stromy ve vnitřních uzlech obsahují funkce (neterminály) a v listech terminály. Vyhodnocení stromu následně probíhá od listů ke kořeni a výsledek programu je roven hodnotě v kořeni.

Inicializace populace

Na začátku evoluce potřebujeme inicializovat populaci, ta je na počátku tvořena náhodně vytvořenými jedinci. K vytváření náhodných jedinců se používá kombinace dvou přístupů. Prvním z nich je vytváření jedinců s pevně danou hloubkou stromu, kde v listech jsou vždy pouze terminály. V druhém stavění stromů probíhá náhodně z předem určeného počtu neterminálů. Po vyčerpání počtu neterminálů se opět pouze doplní terminály jako listy. Tato metoda vytváří jedince různých velikostí a tvarů. Častou praxí je počáteční populaci vytvořit tak, že každá z metod vytvoří polovinu jedinců.

Selekce

V každé iteraci chceme vybrat několik jedinců, nad kterými budeme provádět různé genetické úpravy. Snahou selekce je volit jedince s vyšší hodnotou fitness

funkce. Asi nejpoužívanější metodou selekce je turnajová selekce. V té se náhodně zvolí daný počet jedinců, kteří se mezi sebou porovnají na základě jejich hodnoty fitness funkce a nejlepší z nich je poté zvolen dso výběru. Dalším z mnoha metod (Jebari (2013)) selekce je ruletová selekce. Zde je pravděpodobnost zvolení jedince do výběru přímo úměrná jeho hodnotě fitness funkce. Pravděpodobnost výběru i-tého jedince je

$$p(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)}.$$

Ruletová selekce má jednoduchou implementaci a zároveň je rychlá na výpočet. Jejím problémem je ale předčasná konvergence k lokálnímu optimu.

Genetické operátory mohou občas upravit nejlepšího jedince tak, že se jeho hodnota fitness funkce výrazně zhorší. Z tohoto důvodu se často používá technika zvaná elitismus, která automaticky do další generace vybere pár nejlepších současných jedinců. Tímto způsobem máme garantováno, že neztracíme nejlepší současné jedince.

`https://www.researchgate.net/publication/259461147_Selection_Methods_for_Genetic_Algorithms`

Genetické operátory

Genetické operátory jsou dvojího druhu, křížení a mutace. Myšlenkou křížení je ze dvou jedinců, nazývejme je rodiče, vytvořit nového potomka, který bude tvořen kombinací obou z jeho rodičů. V genetickém programování pracujeme s jedinci reprezentovanými stromy, proto křížení jedinců je realizováno křížením jejich stromů. V každém z rodičů se pro křížení zvolí jeden uzel stejného typu. Výsledný potomek má strukturu podobnou prvnímu rodiči, jen na původním místě vybraného uzlu bude nyní podstrom, který je zavěšený pod vybraným uzlem druhého rodiče.

Druhým genetickým operátorem je mutace, ta již nepotřebuje mít dva rodiče, ale úprava se provede nad jedincem samotným. Mutovat můžeme v jedinci buď jediný bod, nebo celý podstrom. V případě mutace podstromu se namísto vybraného podstromu vygeneruje zcela nový podstrom. Toto v zásadě představuje křížení s novým náhodně vytvořeným jedincem.
Mutace jediného bodu změní náhodně jediný uzel ve stromě. V případě terminálu se může vybrat libovolný jiný terminál. A v případě vnitřního uzlu může být vybraný libovolný jiný neterminál, který je stejného typu.

Silná a volná typovanost

Ve volně typovaném genetickém programování nezadáváme typy funkcí ani terminálů. Jediné co musíme u funkcí určit je jejich arita. Na základě arity se následně generují a mutují jedinci korektně. Obvykle se nám ale více bude hodit silně typované genetické programování, zde určujeme u všech funkcí nejen jejich aritu, ale také typ každého z argumentů dané funkce a také typ návratové hodnoty. Podobně musíme určit i hodnotové typy terminálů. Na závěr určíme hodnotové typy celého problému a algoritmus už se postará o to, aby byly typové podmínky pro všechny jedince dodrženy. `https://deap.readthedocs.io/en/master/tutorials/advanced/gp.html`

4.1.2 Využití

Genetické programování je využitelné ve všech problémech, kde jsme schopní vymyslet způsob, jak jedince představujícího řešení problému reprezentovat a jak počítat fitness funkci, která dané řešení ohodnotí. To tedy znamená, že možnosti využití jsou téměř nekonečné.

volně přeloženo a zkráceno z původního textu str. 126 Obecně se genetické programování ukazuje být vhodné ve všech problémech, které splňují některou, z následujících podmínek:

- Vzájemné vztahy zkoumaných proměnných nejsou dobře známy, nebo je podezření, že jejich současné porozumění může být mylné.
- Nalezení velikosti a tvaru hledaného řešení je částí řešeného problému.
- Existují simulátory pro testování vhodnosti zadaného řešení, ale neexistují metody pro přímé získání dobrých řešení.
- Obvyklé metody matematické analýzy nedávají, nebo ani nemohou být použity pro získání analytického řešení.
- Přibližné řešení je zcela postačující.

Symbolická regrese

V mnoha problémech je naším cílem nalézt funkci, jejíž hodnota splňuje nějakou požadovanou vlastnost. Toto známe pod názvem symbolická regrese. Obyčejná regrese má obvykle za cíl nalézt koeficienty předem zadané funkce, tak aby co nejlépe odpovídala daným datům. Zde je problém, že pokud potřebná funkce nemá stejnou strukturu, jako zadaná funkce, tak dobré koeficienty nenalezneme nikdy a musíme zkusit hledat funkci jiné struktury. Tento problém může vyřešit právě symbolická regrese. Ta hledá vhodnou funkci, aniž by na začátku měla očekávání o její struktuře. Uvedeme triviální příklad. Řekněme, že hledáme výraz, jehož hodnoty odpovídají polynomu x^2+x+1 na intervalu $[-1,1]$. Budeme hledat funkci jedné proměnné x, proto x přidáme jako terminál. Dále přidáme jako terminály číselné konstanty (například -1,1,2,5,10), které budou sloužit pro hledání koeficientů. Pro náš případ bude stačit, když si jako aritmetické funkce přidáme ty základní, tedy sčítání, odečítání, násobení a dělení. Fitness funkci můžeme zvolit jako součet absolutních hodnot rozdílů výrazu jedince a hledaného výrazu $x^2 + x + 1$. Toto stačí pro spuštění evolučního algoritmu. V takto triviálním případě se hledané řešení nalezne pravděpodobně vezmi brzo a bude mít jednu z podobu stromu reprezentujícího daný výraz (viz strom výrazu 4.1)

4.2 Hluboké Q-učení

Než se dostaneme k samotnému fungování algoritmu hlubokého Q-učení, musíme nejprve vybudovat jeho stavební kameny.

4.2.1 Neuronová síť

První důležitý koncept, který v rámci hlubokého Q-učení budeme potřebovat, jsou neuronové sítě. I zde se nejprve budeme muset seznámit se základní jednotkou - perceptronem, než budeme schopni říct, co neuronové sítě jsou.

Perceptron je algoritmus, který má na vstupu několik hodnot x_i a jeden výstup. S každou vstupní hodnotou je spojena jedna váha w_i. Kromě vah vstupů obsahuje perceptron ještě tzv. práh. Perceptron spočítá vážený součet vstupů a porovná výsledek s prahem. Pokud je výsledek větší než práh, tak perceptron vrací hodnotu 1, jinak 0. Můžeme zde ale využít triku pro zbavení se prahu. K prahu můžeme přistupovat jako k další váze s konstantním vstupem -1. V takovém případě pak perceptron provádí porovnání váženého součtu vstupů s hodnotou 0. Matematicky zapsáno:

$$f(\sum_{i=0}^{n} w_i f_i)$$

, kde f vrací 1 pro $x > 0$ a 0 jinak.

Trénování daného perceptronu probíhá pomocí předkládání dvojic vstupů a výstupů (x,y) z trénovací množiny a upravování vah následujícím způsobem:

$$w_i = w_i + r(y - f(x_i)x_i$$

, kde r je parametr učení.

Rozhodovací hranice perceptronu představuje nadrovinu ve vstupním prostoru. Lze ukázat, že trénování perceptronu zkonverguje, pokud jsou třídy v datech lineárně separabilní.

Většinou ale řešíme problémy, kde třídy v datech lineárně separabilní nejsou, v takových případech nám jeden perceptron nestačí a potřebujeme jich využít víc. Spojením více perceptronů získáváme dopřednou neuronovou síť. Ta se skládá z vrstev perceptronů, kde vstupy perceptronů první vrstvy jsou samotná data x a vstupy perceptronů v dalších vrstvách jsou rovny výstupům perceptronů z vrstvy předchozí.

Pro trénování více vrstvých perceptronů se používá gradientní metoda. Z toho důvodů se využívají jiné funkce f, než ta zmíněná nahoře, ta má totiž gradient ve většině případů roven 0. Typickým příkladem používané funkce je sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}.$$

Následně musíme zvolit chybovou funkci $L(x,y|w)$ neuronové sítě. Zde se typicky využívá střední kvadratická chyba (ang. Mean squared error). Chybová funkce se derivuje podle vah v síti a následně se jednotlivé váhy upraví.

$$w_i = w_i + \alpha \frac{\partial L(x,y|w)}{\partial w_i}$$

`https://ocw.mit.edu/courses/health-sciences-and-technology/hst-947-medical-lecture-notes/ch7_mach3.pdf` strana 9 až 13

4.2.2 Zpětnovazební učení

Podobně jako v předchozí sekci, také zde musíme nejprve zavést nové pojemy a terminologii. Popišme si nejprve, co je to markovský rozhodovací prostor. Markovský rozhodovací prostor popisuje prostředí a je definován čtveřicí (S,A,P,R), kde S je množina stavů, A je množina všech akcí (případně A_s představuje množinu akcí, které mohou být provedeny ve stavu s), $P : S \times A \times S \rightarrow [0,1]$ představuje přechodovou funkci, kde $P_a(s,s')$ vrací pravděpodobnost, že se aplikováním akce a ve stavu s dostaneme do stavu s', a $R : S \times A \times S \rightarrow \mathbb{R}$ představuje funkci odměn $R_a(s,s')$, která vrací odměnu, kterou agent obdrží, pokud ve stavu s provede akci a a dostane se tak do stavu s'. Přechodová funkce i funkce odměn musí navíc splňovat podmínku, že musí být jejich hodnoty nezávislé na předchozích stavech.

`https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-410-principles-lecture-notes/MIT16_410F10_lec22.pdf`

Chování agenta v prostředí můžeme popsat pomocí strategie $\pi : S \times A \rightarrow [0,1]$, ta určuje pravděpodobnost, že se agent ve stavu s rozhodne pro akci a. Pro agenta v prostředí ještě definujme jeho celkovou odměnu jako

$$\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t,s_{t+1})$$

, kde $\gamma < 1$ je diskontní faktor, díky kterému je suma konečná a a_t je akce agenta vybraná v kroku t. Cílem zpětnovazebního učení je nalézt optimální strategii $\pi^\star$, kde $a_t = \pi^\star(s_t)$, takovou, že její celková odměna je maximální.

Hodnotu stavu s při použití strategie π lze definovat jako

$$V^\pi(s) = \mathbf{E}[\sum_{t=o}^{\infty} \gamma^t r_t | s_0 = s]$$

, kde r_t značí odměnu získanou v kroku t. Podobně také můžeme definovat hodnotu $Q^\pi(s,a)$ akce a provedené ve stavu s při následování strategie π.

$$Q^\pi(s,a) = \sum_{s'} P_a(s,s')[R_a(s,s') + \gamma \sum_{a'} \pi(s',a') Q^\pi(s',a')]$$

Z Bellmanovy rovnice pro optimální strategie platí:

$$Q^*(s,a) = R_a(s,s') + \gamma \max_{a'} Q_k(s',a')$$

Agent ke zlepšování své strategie může využívat přechodové funkce a funkce odměn. Často ale agent hodnoty jednotlivých stavů předem nezná a musí se je učit za běhu. Zároveň musí volit mezi explorací tj. prohledáváním prostoru a exploatací tj. využíváním známého. Zde využijeme ϵ-hladového (ang. ϵ-greedy) přístupu, kdy s pravděpodobností ϵ vybere náhodou akci a s pravděpodobností $1 - \epsilon$ vybere nejlepší známou akci.

Nyní se už dostáváme ke Q-učení. Q je reprezentována jako matice zpočátku inicializovaná samými nulami. Agent následně ve stavu s_t vybírá například ϵ-hladovým přístupem akci a_t, získá od prostředí odměnu r_t a přesune se do stavu s'. Na základě těchto informací se provede aktualizace matice následovně:

$$Q(s_t,a_t) \leftarrow (1 - \alpha) Q(s_t,a_t) + \alpha(r_t + \gamma(\max_a(Q(s_{t+1},a))))$$

4.2.3 Hluboké Q-učení

Po seznámí se se základy neuronových sítí a zpětnovazebního učení můžeme přejít k samotnému hlubokému Q-učení. Hluboké Q-učení následuje stejnou myšlenku jako obyčejné Q-učení, také chceme nalézt odměnu vybrané akce v současném stavu. Hlavním rozdílem oproti normálnímu Q-učení je způsob, jak se Q hodnoty reprezentují. V normálním Q-učení jsou Q hodnoty uložené v matici, ta může být v případě velkých prostorů příliš obrovská a Q-učení pak může probíhat velmi pomalu, nebo dokonce vůbec. V Hlubokém Q-učení bude Q reprezentováno pomocí neuronové sítě.

Trénování se provádí pomocí porovnávání rozdílu aktuální odměny $R_a(s,s')$ prostředí od odměny spočítané pomocí Bellmanovy rovnice z Q. Cílem je tedy minimalizovat rozdíl mezi

$$Q(s,a) \quad \text{a} \quad R_a(s,s') + \gamma \max_{a'} Q_\theta(s',a')$$

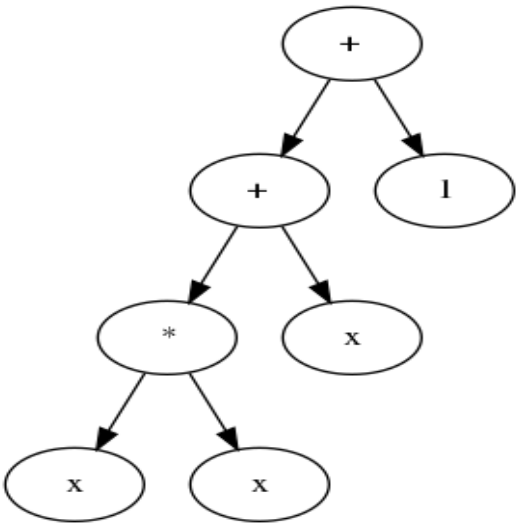
, kde Q_θ jsou parametry neuronové sítě reprezentující matici Q. Chybovou funkcí pro trénování neuronové sítě pak může být střední čtvercová chyba tohoto rozdílu.

4.2.4 Využití

V praxi bylo hluboké Q-učení použito například pro naučení se hraní atari her. `https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf`

Namísto ručně zpracovaných informacích o stavu hry zde byly jako vstupy využity přímo vizuální výstupy hry. Q-síť je proto reprezentována konvoluční neuronovou sítí, která na vstupu bere vektor pixelů obrázku hry a na výstupu vrací odhad budoucích odměn pro každou z možných akcí. Modelu nebylo řečeno nic o principu fungování hry. Model se učil pouze na základě vizuálního výstupu, odměn, které dostával od prostředí, konečných stavů a množiny možných akcí, tedy podobně, jak by se hru učil hrát člověk.

Stejná neuronová síť byla použitá na sedmi různých atari hrách. Na šesti z nich překonala všechny stávající přístupy, které využívaly algoritmy zpětnovazebního učení a na třech z nich překonala výsledky nejlepších lidských hráčů.



Obrázek 4.1: Strom výrazu

5. Provedené experimenty

V předchozí kapitole jsme se seznámili s algoritmy, které jsou aplikovatelné i v našem prostředí. V této kapitole se pomocí různých experimentů pokusíme nalézt agenty, kteří se budou v jistém pojetí chovat inteligentně.

Abychom mohli výsledky experimentů vyhodnocovat, případně vzájemně mezi sebou porovnávat, potřebujeme mít na výsledky konkrétní kritéria. Při testování obranného akčního plánu se ukázalo, že agent, který využívá pouze obranného akčního plánu k sestřelení nejbližšího asteroidu, se kterým vesmírné lodi hrozí srážka, dokázal díky dobré obraně zdatně přežívat ve hře mnoho kroků. A zároveň je tento obranný agent vzhledem k využívání pouze jednoho akčního plánu dostatečně neinteligentní na to, abychom ho mohli použít jako referenčního agenta.

Jako kritérium pro hodnocení výsledků experimentů tedy bude sloužit souboj mezi výsledným agentem experimentu a obranným agentem. První z nich, kterému se podaří zvítězit desetkrát, bude označen za vítěze. Výsledky těchto soubojů pak budou sloužit jako porovnání mezi jednotlivými provedenými experimenty.

Kromě primárního cíle nalézt agenty s inteligentním chováním máme i cíl sekundární, a to nalézt agenty, kteří se budou chovat zajímavě ve smyslu pestrosti akcí, akčních plánů, nebo také zajímavě v tom smyslu, že bude jejich chování působit jako chování lidského hráče.

5.1 Genetické programování

V prvních experimentech využijeme již zmíněného genetického programování. Nezbytným požadavkem pro jeho využití je existence reprezentace jedince a fitness funkce, která ho ohodnotí. Obojí dokážeme jednoduše vyřešit. Jedinec bude představovat rozhodovací funkci, která se na základě vstupních argumentů rozhodne, který akční plán bude vybrán.

V našem případě máme hru, kde spolu dva hráči soupeří a hra končí výhrou jednoho z hráčů. Přesně tohoto můžeme ve fitness funkci jedince využít. Pro využití výsledku hry, jak jedinec ve hře dopadl, musíme nejprve zvolit proti jakému hráči bude jedinec, který je předmětem našeho zájmu, hrát.

Pro experimentování s genetickým programováním jsem zvolil knihovnu deap pro python. Zde lze jednoduše konfigurovat evoluční algoritmus na konkrétní řešený problém. Stačí popsat jak reprezentovat jedince a jak se vypočítá jeho fitness funkce a zbytek knihovna vyřeší za nás.

Reprezentace jedince

Ve 3. kapitole jsme si vybudovali abstrakce v podobě senzorů a akčních plánů a těch zde budeme chtít využít. Jedince budeme podobně jako u symbolické regrese reprezentovat stromem. Strom jedince budeme budovat prvkami z následující množiny terminálů a neterminálů.

- Terminály:

  - Vstupní argumenty rozhodovací funkce
  - Celočíselné konstanty -1, 1, 3, 5, 10, 100
  - Nulární funkce vracející hodnoty reprezentující zvolený akční plán

Jako argumenty funkce jsem zvolil následující hodnoty: délky všech čtyř akčních plánů a počet kroků před srážkou vesmírné lodi s asteroidem. Délky akčních plánů se pohybují v intervalu (1,100), proto jsou číselné konstanty zvoleny tak, aby se jejich sčítáním a násobením lehce dosáhlo dalších hodnot z tohoto intervalu.

- Neterminály:
  - Aritmetické operace sčítání a násobení
  - Funkce *compare*
  - Funkce *if_then_else*

Z aritmetický operací nám stačí sčítání a násobení. Operaci odčítání získáme pomocí sčítání a násobení konstantou -1. Hodnoty z intervalu (1,100) jednoduše získáme také pomocí sčítáním a násobením potřebných konstant, proto pro operaci dělení není důvod. Všechny aritmetické operace jsou typu $([int,int], int)$. Funkce *compare* je typu $([int,int], Bool)$, ta vrací zda je první argument větší než druhý. Poslední použitá funkce *if_then_else* je typu $([Bool, ActionPlanEnum, ActionPlanEnum], ActionPlanEnum)$. Tato funkce dostává jako argumenty výraz typu bool a následně dvě hodnoty reprezentující akční plány. Na základě pravdivosti výrazu vrací funkce první nebo druhou z hodnot akčních plánů.

Takto popsaná reprezentace jedince bude použita ve všech následujících experimentech. To, v čem se budou experimenty lišit, je způsob výpočtu fitness funkce a průběh evolučního algoritmu.

5.1.1 Experiment 1: Soupeření s obranným agentem

Cílem tohoto experimentu bylo vyvinout agenta, který bude lepší než obranný agent. Výpočet fitness funkce proto zahrnuje zahrání šesti her současného jedince s obranným agentem a výsledek je průměr z hodnot každé z her. Hra je pokaždé velmi náhodná, tedy zahrání jedné hry by mělo nízkou vypovídající hodnotu. Proto jsem pro přesnější informaci zvolil zahrání šesti her. Hodnota zahrané hry se skládá z více částí.

- Počet kroků trvání hry
  Myšlenkou je zde, obzvláště v počátku evoluce, upřednostňovat takové jedince, kteří dokáží vydržet ve hře co nejdéle, tedy nejsou ve hře okamžitě poraženi. Pro představu, délky her se pohybují přibližně v intervalu $(900, 2900)$ kroků.

- Penalizace za nevyužití některého z plánů
  Během hry se udržuje historie, kolikrát se agent rozhodl pro každý z akčních plánů. Za každý ze čtyř akčních plánů, který agent ani jednou během hry nezvolil bude přičtena penalizace -500. Cílem těchto penalizací je upřednostňovat takové jedince, kteří používají všechny akční plány a tím pádem mají pestřejší chování.

- Bonus (penalizace) za výhru (prohru)
  Toto je asi nejdůležitější část. Pro zdůraznění rozdílu mezi vyhranými a prohranými hrami se v případě výhry přičte k výsledku hodnota 2000 a v případě prohry se 2000 odečte. Motivací mohou být následující dvě situace. Řekněme, že v jedné hře se podařilo jedinci dlouho bránit a dokázal vydržet 2500 kroků hry a poté prohrál. A v další situaci jedinec porazil soupeře v rychlých 1200 krocích. Bez bonusu za vyhranou hru, by prohraná hra získala jedinci daleko vyšší hodnotu, než situace z druhé hry, kterou vyhrál.

Algoritmus byl spuštěn s následujícími parametry:

- Velikost populace: 30
- Pravděpodobnost křížení: 60%
- Pravděpodobnost mutace: 20%
- Počet generací: 100
- Metoda selekce: turnajová selekce

Výsledný nejlepší jedinec bohužel nesplnil naše očekávání a nedokázal obranného agenta porazit. V souboji jedinec nejen nedokázal konkurovat obrannému agentovi, ale ani se neřídil příliš pestrou strategií.

V 95% volil, stejně jako obranný agent, obranný akční plán a ve zbylých pár procentech volil všechny zbylé akční plány (viz 5.1). Vyžívání všech akčních plánů bylo pravděpodobně dosaženo právě skrze vysokou penalizaci při nepoužití libovolného z nich, ale vidíme, že agent je volil spíše právě z tohoto důvodu, než že by je chtěl aktivně využívat v rámci své strategie.

5.1.2 Experiment 2: Postupné zaměňování úspěšnějšího jedince

V tomto experimentu nebylo cílem porazit konkrétního, stálého agenta jako v předchozím případě. Zde bylo cílem postupně vybudovat nejzdatnějšího jedince. Stejně jako v předchozím experimentu i zde fitness funkce spočívá v zahrání šesti her, avšak zde nebudeme hrám přiřazovat žádnou číselnou hodnotu, ale spokojíme se s jednoduchou informací, který z agentů v dané hře zvítězil.

V průběhu evoluce si budeme pamatovat současného nejlepšího jedince. Na začátku bude tento jedinec vybrán zcela náhodně. Obvyklým způsobem vytvoříme počáteční populaci a započneme evoluci. Fitness funkce jedince bude počítat poměr, kolik ze šesti zahraných her jedinec vyhrál v souboji se současně nejlepším nalezeným jedincem. Evoluce hledá řešení, která budou proti současnému nejlepšímu jedinci co nejúspěšnější. V každé 3. generaci se následně kontroluje, zda již náhodou nebyl v populaci nalezen jedinec, který, pro současnou situaci, nejlepšího jedince porazil alespoň v pěti ze šesti hrách. Pokud ano, tak takový jedinec bude nově zvolen jako nejlepší a evoluce bude pokračovat stejným způsobem dál.

Po výměně nejlepšího jedince musíme nově přepočítat fitness funkci všech stávajících jedinců v populaci, protože jejich současná hodnota se vztahovala k původnímu soupeři. Rovněž musíme, ze stejného důvodu, smazat všechny jedince ze síně slávy (ang. Hall of fame), kde se průběžně ukládají nejlepší jedinci spolu s hodnotou jejich fitness funkce.

Všechny tyto změny už nelze v knihovně deap nakonfigurovat přímočarým způsobem jako v předchozím experimentu, ale bylo zapotřebí upravit samotnou kostru evolučního algoritmu.

Algoritmus byl spuštěn s následujícími parametry

- Velikost populace: 10
- Pravděpodobnost křížení: 60%
- Pravděpodobnost mutace: 20%
- Počet generací: 450
- Metoda selekce: turnajová selekce

Výsledného agenta jsme nechali zahrát souboj s obranným agentem a tentokrát přinesl experiment daleko lepší výsledky. Nalezený agent se oproti předchozímu experimentu dokázal naučit lépe utočit, volil útočný akční plán téměř ve 40% případech a díky tomu dosáhl našeho primárního cíle. V Souboji porazil obranného agenta se skóre 10:0. Nicméně ani tentokrát se agent nenaučil nic jiného než obranu a útok (viz 5.2).

5.1.3 Experiment 3: Postupné zaměňování úspěšnějšího jedince bez obranného akčního plánu

V předchozích experimentech se nám v obou případech podařilo vytvořit agenty, kteří v drtivé většině stavů rozhodují jen mezi obranným a útočným plánem. To má za následek, že se agenti po celou dobu hry pouze otáčejí a střílejí, ale zůstavají při tom na jednom stejném místě. V tomto experimentu tomuto problému zkusíme předejít, tím, že donutíme agenta bránit se uhýbáním namísto sestřelování nebezpečných asteroidů.

Experiment probíhá stejným způsobem jako v předchozím případě, jen s tím rozdílem, že agentovi zakážeme používání obranného plánu. Z argumentů rozhodovací funkce odstraníme informaci o obranném plánu. A z množiny terminálů používaných při tvorbě programů odstraníme nulární funkci reprezentující obranný akční plán.

To je vše co je potřeba změnit a zbylá logika může zůstat stejná jako v předešlém experimentu.

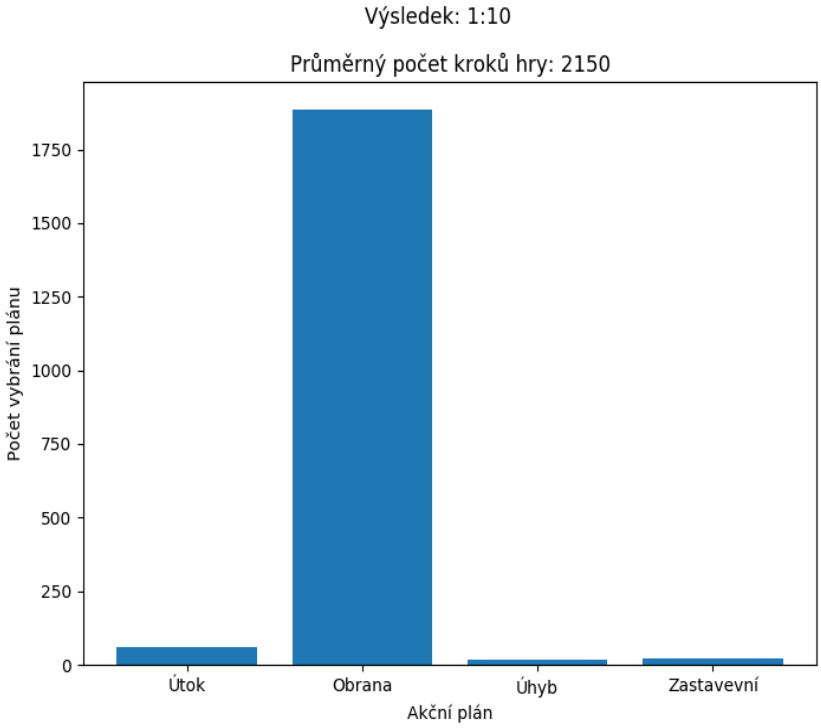
Algoritmus byl spuštěn s následujícími parametry

- Velikost populace: 10
- Pravděpodobnost křížení: 60%
- Pravděpodobnost mutace: 20%
- Počet generací: 2000
- Metoda selekce: turnajová selekce

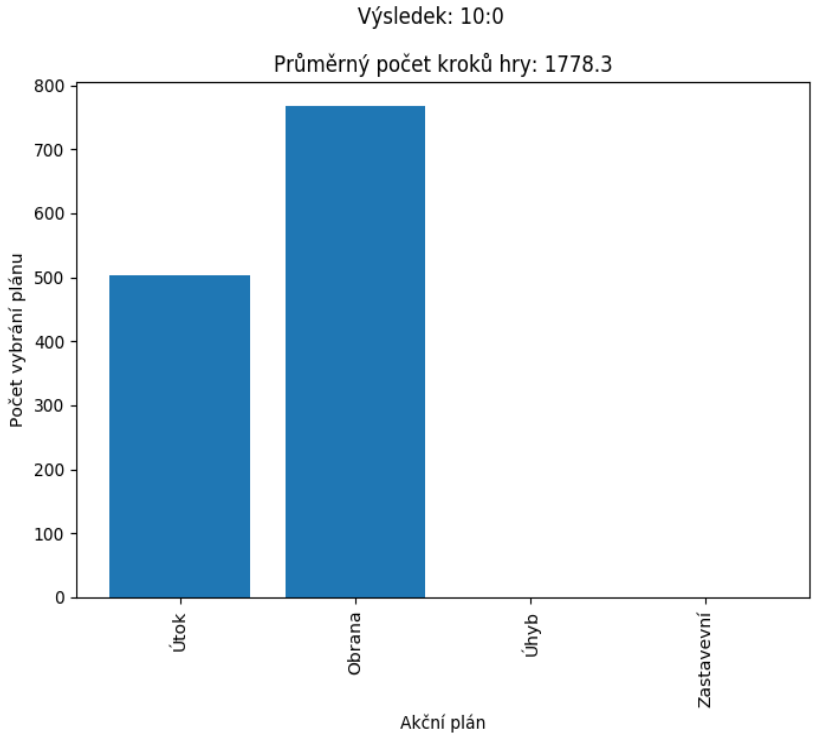
S výsledným agentem jsme opět provedli souboj s obranným agentem. První čeho si můžeme všimnout je, že náš agent v souboji prohrál s výsledkem 2:10, tedy bez obranného akčního plánu nebyl schopný tak úspěšně konkurovat obrannému agentovi. Druhá skutečnost, která stojí za povšimnutí je průměrná délka hry, ta byla v průměru přibližně o 500 kroků kratší než v předchozím experimentu. Z toho vyplývá, že využívání úhybného akčního plánu k přežívání není tak účinné, jako bránění se pomocí obranného akčního plánu. To ale není nijak překvapivé. Pro agenta je prostředí tím víc nebezepečné, čím více je v něm nebezpečných asteroidů. Používání úhybného akčního plánu vede k vyhnutí vesmírné lodi před nebezepčným asteroidem, né před jeho zničením, jako je to u obranného akčního plánu. To má za následek, že v případě úhybného akčního plánu agent neredukuje počet nebezepčných asteroidů a mnohem dříve se dostane do stavu, kdy je pro agenta příliš obtížné se roji asteroidů vyhnout.

Zajímavým výsledkem experimentu je také to, že, přestože agent používá v rámci úhybného akčního plánu akceleraci pro obranu velmi často, ani tentokrát nepoužívá zastavovací akční plán (viz 5.3). To ukazuje, že zastavování letu není pro získání lepších výsledků stěžejní.

Výsledný agent v souboji jednoznačně dosáhl špatných výsledků, ale pokud se na hru podíváme z lidského pohledu, tak oproti agentům, kteří zůstávají po celou dobu hry na místě, působí agentovo chování mnohem zajímavěji.



Obrázek 5.1: Výsledek experimentu 1



Obrázek 5.2: Výsledek experimentu 2

5.2 Hluboké Q-učení

Herní prostředí nám v každém kroku vrací odměnu, kterou oba z hráčů za jejich akci obdrželi. Tuto informaci jsme v experimentech provedených v rámci genetického programování nevyužili, ale zde budou mít zásadní roli. Za každý krok, kdy hra ještě neskončila, získávají agenti automaticky odměnu 1. Na konci hry agent obdrží vysokou odměnu 2000 v případě výhry a v případě prohry naopak získá penalizaci v podobě odměny vysoké záporné hodnoty -1000. Odměnu za výhru, nebo prohru získá agent až na úplném konci hry, to může ztěžovat učící proces. Proto prostředí dává agentům i průběžné menší odměny, pro lepší možnost učení se.

Konkrétně to jsou následující odměny:

- Sestřelení asteroidu
  Za každý sestřelený asteroid získává agent odměnu hodnoty 5.

- Zasažení nepřátelské vesmírné lodi asteroidem
  Zranění nepřítele je právě to, co agent potřebuje pro přiblížení se vítězství, proto za každé takové zasažení získává od prostředí odměnu v hodnotě 20.

- Zasažení vlastní vesmírné lodi asteroidem
  Takový stav je pro agenta znevýhodňující a cílem je se mu vyvarovat, proto za takovýto stav agent od prostředí dostává penalizaci v hodnotě -10.

V rámci učení agentů budeme využívat ϵ-hladového (ang. ϵ-greedy) přístupu. V každém kroku hry vygenerujeme náhodnou hodnotou z intervalu (0,1) a pokud je tato hodnota menší než hodnota ϵ, tak provedeme volbu akce náhodně, v opačném případě volíme nejlepší akce dle Q-sítě. Hodnota ϵ se na počátku inicializuje na hodnotu 1 a po každé zahrané hře se sníží vynásobením koeficientem menším než 1. Průběžným snižování hodnoty ϵ způsobíme, že z počátku učení se budou zkoušet náhodné akce a v průběhu přejde z prohledávání nových akcí ke zkoušení již osvědčených akcí.

Při trénování se nám stává, že měníme funkci, která odhaduje Q a tím je ovlivněno i chování agenta a odhady. K zachování větší stability trénování využijeme konceptu přehrávání zkušeností (ang. Experience replay). Při hraní hry si v každém kroku uložíme do paměti pětici současného stavu, provedené akce, obdržené odměny, stavu, do kterého jsme se dostali a informace zda hra neskončila. Po konci zahrání hry následně náhodně vybereme tyto pětice z paměti a trénování provedeme na nich.

5.2.1 Experiment 4: Soupeření s obranným agentem

V 1. experimentu jsme za pomocí genetického programování hledali agenta, který je úspěšný v souboji s obranným jedincem. Pro určení jak agent v souboji obstál jsme využívali fitness funkci. Zde, pomocí hlubokého q učení, budeme také učit agenta vzájemnými souboji s obranným agentem, ale budeme namísto fitness funkce pro trénování využívat odměny.

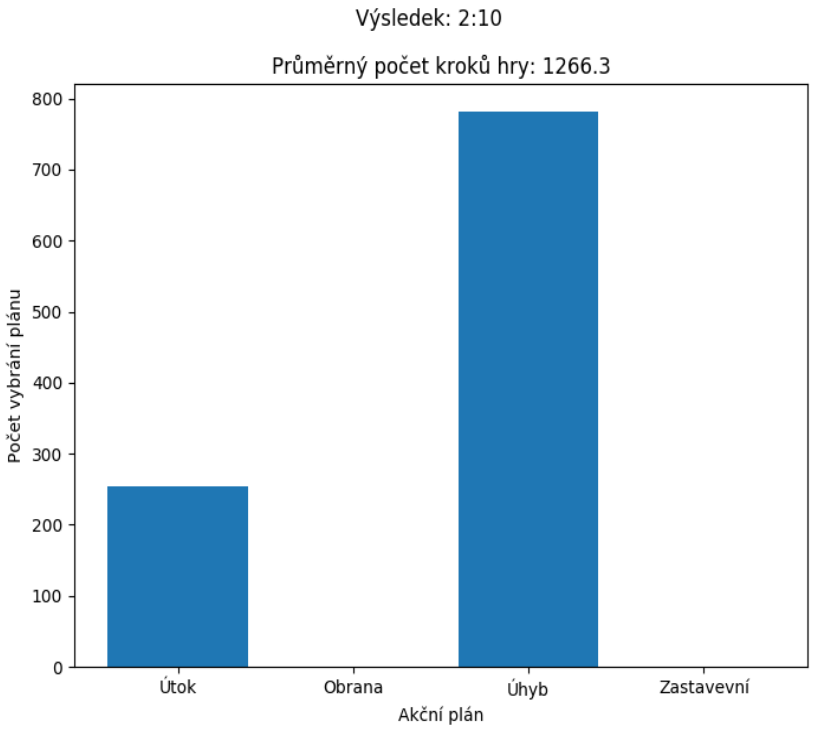
S 1. experimentem zde bude také stejný přístup ke vstupům a výstupům. Na vstupu budou opět délky všech čtyř akčních plánů a počet kroků před srážkou vesmírné lodi s asteroidem. A na výstupu čtyři hodnoty reprezentující výběr konkrétního akčního plánu.

Q učení spočívá v učení se rozhodování akcí. Akce zde v tomto pojetí však nebudou představovat elementární akce, nýbrž akční plány. Q-síť bude tedy volit akční plány a proto zde budeme muset provádět mezikrok pro přechod od akčních plánů k akcím. Nejprve vždy zvolíme akční plán a následně pro pokračování v simulaci hry z vybraného akčního plánu vybereme první akci. V tomto experimentu budeme využívat přehrávání zkušeností, tj. budeme průběžně ukládat pětice informací o přechodech do dalších stavů. I zde pro pamatování si zkušenosti platí, že akcí budeme rozumět akční plán.

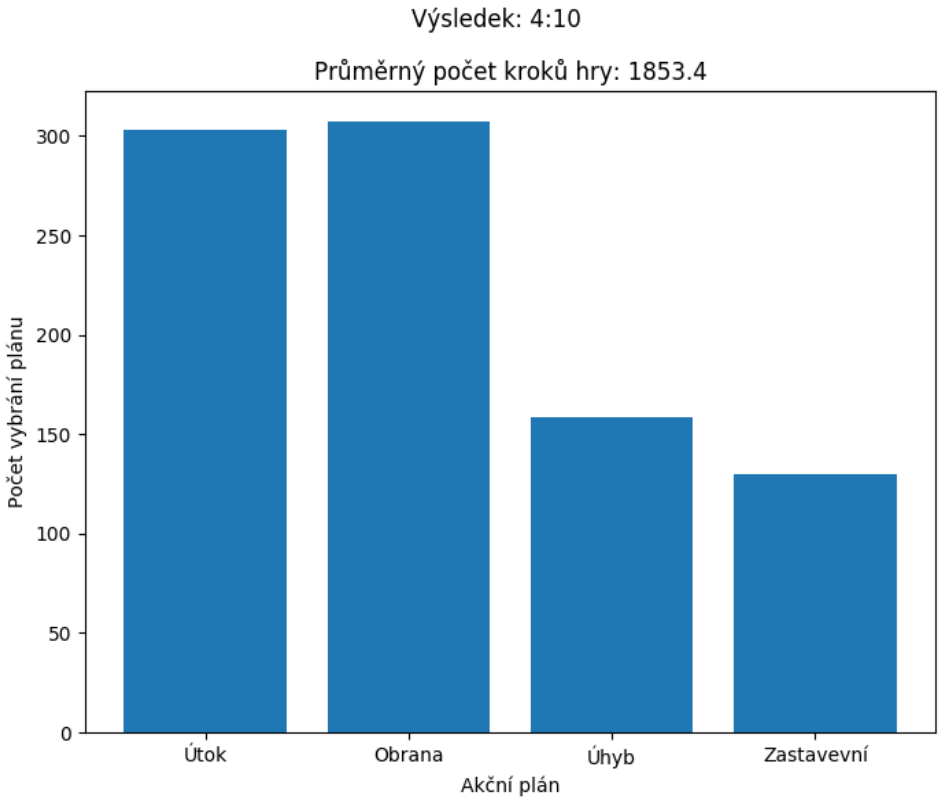
Parametry experimentu:

- Q-síť je hustá neuronová síť s pěti vstupy, čtyřmi výstupy a jednou skrytou vrstvou.
- Během učení bude zahráno 1500 her.
- Konstanta pro snižování ϵ je nastavena na 0.998. To znamená, že například po zahrání 1400 her se bude v další hře volit akce náhodně jen v 6% případů.

V souboji s obranným agentem se výslednému agentovi podařilo zvítězit pouze ve čtyřech hrách. Nepodařilo se nám tedy sice nalézt agenta, který by stabilně porážel obranného agenta, ale dosáhli jsme jiného zajímavého výsledku. Velkým přínosem tohoto experimentu je pestrá strategie nalezeného agenta. Výsledný agent ve velkém zastoupení používá všechny akční plány (viz 5.4). Výsledkem je agent, který se brání nejen sestřelováním nepřátelských asteroidů, ale i vyhýbáním se. A díky tomu se agent také pohybuje a nezůstává jen staticky stát na stejném místě po celou dobu hry. Toho se nám také podařilo dosáhnout ve 3. experimentu, ale ve srovnání s agentem získaným ze 3. experimentu je tento agent daleko více obranyschopný.



Obrázek 5.3: Výsledek experimentu 3



Obrázek 5.4: Výsledek experimentu 4

5.2.2 Experiment 5: Soupeření s obranným agentem - Rozšířeno

V předchozím experimentu jsme dosáhli zajímavého chování agenta, ale nepodařilo se nám stabilně vyhrávat nad obranným agentem. Zkusíme proto předchozí experiment rozšířit. V tomto experimentu zkusíme přidat další vstupní argumenty, které by mohli agentovi pomoct v rozhodování.

Přidané parametry:

- Dvojice počtu zbývajících životů obou agentů
- Počet nebezepčných asteroidů v blízké vzdálenosti od agenta
- Celkový počet nebezpečných asteroidů v celé hře

Snaha všech přidaných argumentů je rozšířit agentovi poznání o současném stavu hry a díky tomu mu dát možnost se komplexněji rozhodovat pro akční plány.

Parametry experimentu:

- Q-síť je stejná síť jako v předchozím případě, jen namísto pěti vstupních argumentů, bude nyní přijímat vstupů devět.
- V tomto experimentu zkusíme kvůli rozšíření vstupních argumentů také prodloužit trénování sítě, proto bude v rámci trénování zahráno 3000 her.
- Adekvátně ke zvýšení počtu zahraných her také zvětšíme konstantu pro snižování ϵ z hodnoty 0.998 na 0.9989. Díky tomu bude stejné pravděpodobnosti 6% pro volbu náhodné akce dosaženo přibližně po zahrání 2550 her.

Výsledný agent dopadlo velmi úspěšně. Z průběhu trénování vidíme, že agent se velmi dobře učil a od přibližně 2300. hry (viz 5.5) už začal vyhrávat ve větší části her. Rozšířením vstupních argumentů a přidání trénovacích her se nám podařilo zlepšit výsledek z předchozícho experimentu. Agent sice ztratil pestrost akčních plánů, ale za to se významně zlepšil ve vyhrávání. Z výsledku 4:10 z předchozícho experimentu se zlepšil na 10:3. Zajímavé na nalezeném agentovi je také jeho agresivita. Agent používá útočný akční plán přibližně dvakrát tak často jako obranný plán.

Při testování výsledného agenta jsem si všimnul, že zahrání jedné hry je časově značně náročné, přičemž to co při simulaci trvalo netriviální objem času bylo samotné dotazování Q-sítě na akční plán. Při snaze tento problém vyřešit jsem zjistil, že v některých stavech hry jsou všechny akční plány prázdné. Toto může nastat v případě kdy agent stojí na místě, není ohrožený žádným asteroidem a zároveň nenalezl žádný asteroid, kterým by mohl přímo ohrozit nepřítele. V takovém stavu nemá velký smysl rozhodovat o volbě konkrétního akčního plánu. Proto jsem nastavil, že v takových případech agent rozhodování provádět nebude.

Podobně jsem také vypozoroval, že během jedné hry často nastane situace, že právě jeden z akčních plánů je neprázdný. Překvapením pro mě bylo, že Q-síť někdy v takových případech volila jiný prázdný plán před tímto neprázdným. Proto jsem nastavil vyjímku i pro tyto případy a v současnou chvíli platí, že když agent má k dispozici právě jeden neprázdný akční plán, tak ho volí automaticky

Obrázek 5.5: Průběh trénování v experimentu 5 - Výsledné odměny jsou součtem počtu kroků hry a odměny (resp. penalizace) za výhru (resp. prohru). Samotné tyto odměny tvoří rozdíl v hodnotě 3000, díky tomu je z obrázku zřetelné, ve kterých hrách agent vyhrál.

bez dotazování se Q-sítě. Těmito opatřeními bylo dosaženo lepší časové náročnosti hraní hry a také byl agent v souboji úspěšnější. Zlepšení agenta si vysvětluji právě tím, že volba jakéhokoliv neprázdného plánu před prázdným je vždy výhodnější.

5.2.3 Experiment 6: Elementární agent proti obrannému agentovi

V tomto experimentu zkusíme sestoupit od abstrakcí v podobě akčních plánů k elementárním akcím. Tentokrát nebudeme Q-síť používat k volbě akčního plánu, ale přímo k volbě elementární akce. Výsledný agent bude volit vždy jen jednu akci, proto nebudeme moci využít koncept přepočítávání akčních plánů a agent se bude muset rozhodovat v každém kroku. Opět budeme k trénování využívat soubojů s obranným agentem a učit se na základě odměn získaných od herního prostředí.

K pěti elementárním akcím, pro které se bude agent rozhodovat, přidáme navíc také možnost prázdné akce. Nebudeme zde volit akční plány, proto ani nemá dobrý smysl používat jejich délky jako argumenty pro rozhodování. Proto zde můžeme zvolit zcela jiný přístup. Samotné simulace pro získání akčních plánů jsou výpočetně velmi náročné, a tedy díky tomu, že zde volíme jednodušší přístup, budeme schopni, oproti předchozím experimentům, zahrát v rámci trénování větší množství her.

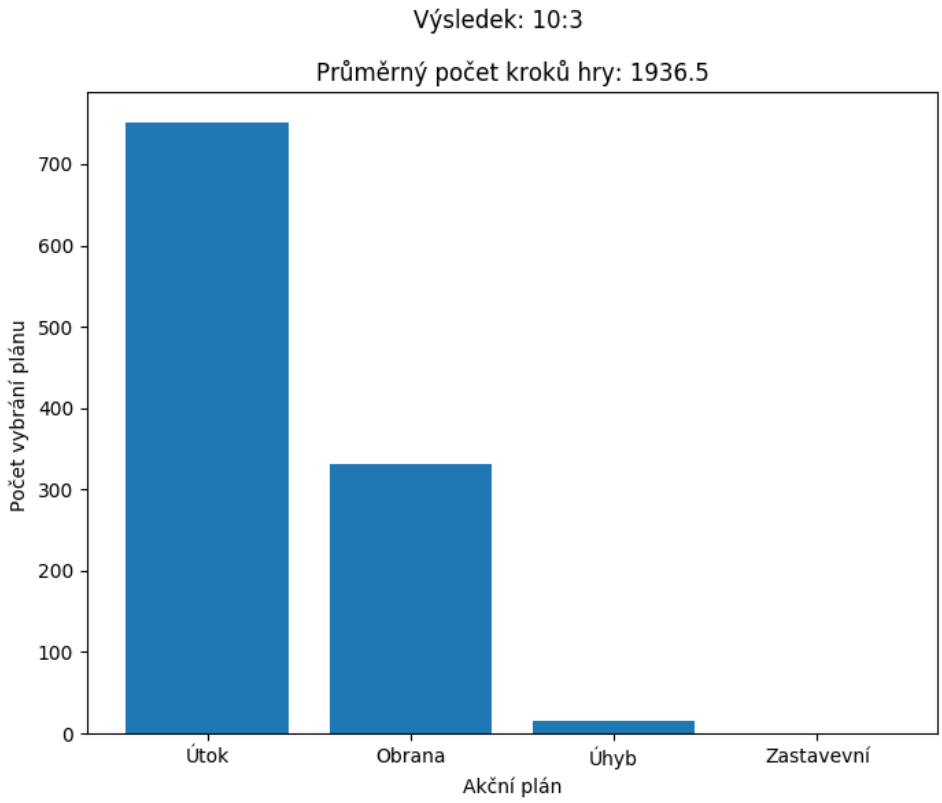
Jako vstupní argumenty jsem zvolil následující hodnoty:

- Vektor současného pohybu vesmírné lodi
- Úhel natočení vesmírné lodi
- Počet uplynulých kroků od posledního výstřelu
- Relativní poloha nepřátelské lodi
- Relativní polohy tří nejbližších nebezpečných asteroidů

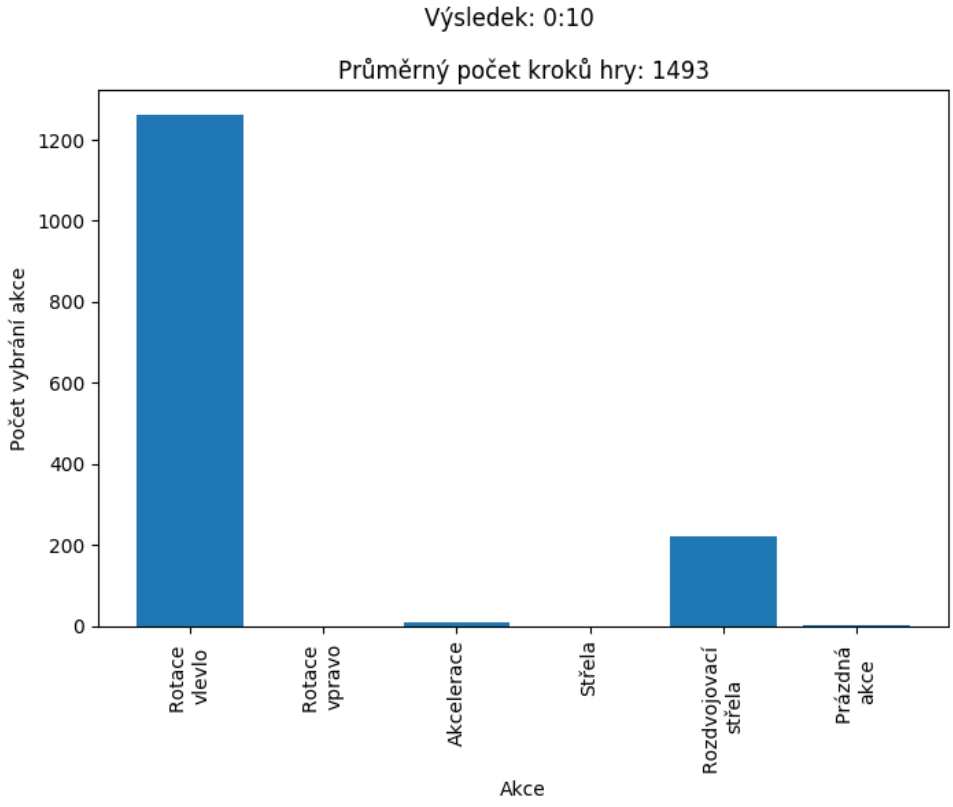
Parametry experimentu:

- Q-síť je hustá neuronová síť s dvěmi skrytými vstvami, čtrnácti vstupními a šesti výstupními hodnotami.
- Díky nevyužívání akčních plánů bude hraní her rychlejší, proto pro trénování zahrajeme 10000 her.
- Konstanta pro snižoání ϵ je nastavena na hodnotu 0.9997

Výsledný agent proti obrannému agentovi nedopadl úspěšně. V souboji byl jednoznačně poražen se skóre 0:10. A z přehledu používaných akcí během souboje můžeme i vypozorovat proč takto dopadl. Z elementárních operací se rozhodoval v drtivé většině pro rotaci vlevo a rozdvojovací střelu. To v praxi znamená, že se agent naučil točit dokola a kdykoliv může, tak vystřelit. Tato strategie skutečně přináší nějaké výsledky. Touto kombinací rotace a střelby se agent dokáže ubránit před srážkou s některými asteroidy, které by ho jinak zasáhly. Zároveň tímto způsobem sestřeluje netriviální množství asteroidů, které se kolem něho nacházejí a tím potenciálně staví nepřítele do ohrožení. Avšak pro toto chování se agent rozhoduje bezmyšlenkovitě. Nemíří na žádné konkrétní asteroidy, ani na nepřátelskou loď. Největší slabinou je, že agent se zde prakticky vůbec nenaučil bránit. Veškeré asteroidy, před kterými se agent ubrání, zasáhne díky náhodě.



Obrázek 5.6: Výsledek experimentu 5



Obrázek 5.7: Výsledek experimentu 6

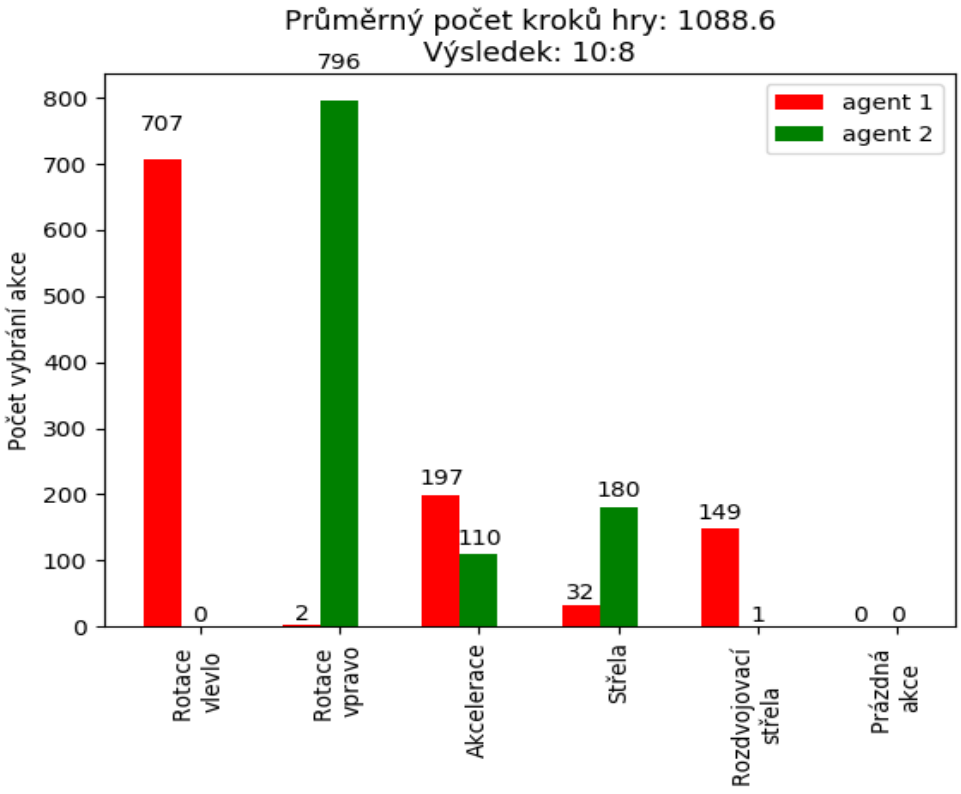
5.2.4 Experiment 7: Dva elementární agenti

Tento experiment rozšiřuje experiment předchozí. Budeme opět pracovat s elementárním agentem reprezentovaným neuronovou sítí stejného formátu jako v předhozím experimentu. Tentokrát ale nebudeme při trénování hrát hry proti obrannému agentovi, nýbrž proti dalšímu elementárnímu agentovi, který bude také zároveň trénován. Budeme tedy provádět dvojí Q-učení simultánně. Cílem je zde dosáhnout vzájemného adaptivního učení, kde se každý z agentů snaží zlepšovat proti svému nepříteli a postupně tak oba agenty zlepšovat.

Parametry experimentu:

- Každý agent bude reprezentován vlastní Q-sítí stejného formátu jako v předchozím experimentu.
- Tím, že pro souboj nebudeme používat obranného agenta, ale dalšího elementárního agenta, ušetříme čas na výpočtu obranného agenta. V rámci trénování tedy zahrajeme 20000 her.
- Konstanta pro snižoání ϵ je nastavena na hodnotu 0.9998

Výsledný souboj jsme výjmečně neprovedli proti obrannému agentovi, ale mezi vzniklými agenty mezi sebou. Z přehledu souboje je vidět, že se agenti od předchozího experimentu nijak zásadně nezlepšili. Oba agenti se v drtivé většině stavů jen točí na jednu stranu. Vidíme, že každý agent volí exklusivně pouze jednu stranu, na kterou se rotuje. Agent 1 se z přehledu souboje zdá být mírně pestřejší, kombinuje oba typy střel a navíc v téměř pětině stavů volil akceleraci. Když jsem však vizuálně sledoval souboj agentů, tak žádný z agentů nejevil známky komplexnějšího chování.



Obrázek 5.8: Výsledek experimentu 7

Závěr

V této práci se nám úspěšně podařilo vymyslet a naimplementovat abstrakce v podobě senzorů a akčních plánů do jednoduché vesmírné hry. Seznámili jsme se s algoritmy Genetického programování a Hlubokého Q-učení. Tyto algoritmy jsme aplikovali v našem herním prostředí a provedli s nimi sérii experimentů.

Stanovené cíle z úvodu

Co nebylo doděláno a může být do budoucna rozšířeno.

V závěru co jsem udělal V Závěru tyto cíle zhodnotit Jak se to dá rozšířit

Seznam použité literatury

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. a Zaremba, W. (2016). Openai gym. `https://arxiv.org/pdf/1606.01540.pdf`.

Jebari, K. (2013). Selection methods for genetic algorithms. *International Journal of Emerging Sciences*, **3**, 333–344.

Poli, R., Langdon, W. B. a McPhee, N. F. (2008). *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`. (With contributions by J. R. Koza).

Shinners, P. (2011). Pygame. `http://pygame.org/`.

Seznam obrázků