ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Neuronové sítě pro řízení počítačové hry
Student:	Ondřej Černý
Vedoucí:	Ing. Zdeněk Buk, Ph.D.
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce zimního semestru 2018/19

Pokyny pro vypracování

Implementujte jednoduchou počítačovou hru typu “Asteroids”, kde hráč má k dispozici vesmírnou loď a 4 akce - otáčení vlevo/vpravo, ovládání motoru a střelbu. Loď se pohybuje v poli asteroidů a cílem je co nejdelší přežití bez kolize. Prozkoumejte vhodné typy neuronových sítí pro ovládání této hry. Vybranou síť včetně evolučního učicího algoritmu implementujte a otestujte. Neuronová síť bude mít jako vstup informace o okolních objektech. Konkrétní reprezentaci vstupu včetně velikosti okolí zvolte na základě experimentů.

Seznam odborné literatury

[1] Michal Bureš: Implementace počítačové hry využívající prvky umělé inteligence, bakalářská práce, ČVUT, FEL, 2012

[2] R. J. Williams and D. Zipser. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks, 1989.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 18. února 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Teoretické Informatiky

Bakalářská práce

Neuronové sítě pro řízení počítačové hry

Ondřej Černý

Vedoucí práce: Ing. Zdeněk Buk, Ph.D.

15. května 2017

Poděkování

Děkuji vedoucímu Ing. Zdeňkovi Bukovi, Ph.D., za veškerou podporu a za možnost zpracování zajímavého tématu. Dále bych chtěl poděkovat své rodině za trpělivost, kterou se mnou měla během psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2017 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Černý, Ondřej. *Neuronové sítě pro řízení počítačové hry.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato práce se zabývá použitím neuronových sítí pro hraní počítačové hry Asteroids namísto lidského hráče. Diskutuje nasazení různých typů umělé inteligence do této hry. V práci je implementován klon zmíněné hry a plně rekurentní neuronová síť hrající hru z hráčovy perspektivy. K učení neuronové sítě používá práce diferenciální evoluci, podmnožinu evolučních algoritmů. Dále se v práci experimentuje s nastavením diferenciální evoluce a neuronovou sítí. Výsledky těchto experimentů jsou porovnávány s lidským chováním.

Klíčová slova Umělá inteligence, Neuronová síť, Diferenciální evoluce, Hra Asteroids, Evoluční algoritmus

Abstract

This thesis deals with the topic of using neural networks in place of human player in playing computer game Asteroids. It describes how to use different types of artificial intelligence in this game. A clone of the game and fully recurrent neural network playing the game from human perspective are made in this work. Differential evolution, a subset of evolutionary algorithms, was

used for learning purposes of the neural network. Furthermore this thesis experiments with different settings of the evolution and neural network. Results of said experimentation are compared with human behavior.

Keywords Artificial intelligence, Neural network, Differential evolution, Asteroids game, Evolutionary algorithm

Obsah

Seznam obrázků

Úvod

Počítačové hry se v moderní době staly populárním fenoménem. Rozšířily se už i mezi starší generace a na přenosná zařízení. Existuje řada her, které v sobě nesou určitý aspekt reality. Díky tomu můžeme část poznatků získaných ze hry aplikovat v běžném životě, nejenom z kategorie edukačních her, kde se člověk učí cizí jazyk nebo procvičuje násobilku. Strategické hry rozvíjejí schopnosti taktizovat nad rozvinutými scénáři, logické hry trénují dedukci, akční hry procvičují postřeh a koordinaci.

Hry nás mohou učit ale i jiné věci. Můžeme se například v zápletce příběhu vžít do situací, které konfrontují etický problém, a zkusit si jej „cvičně“ vyřešit. Jinde můžeme z modelu herního světa dedukovat pravidla, jež by se reálně dala aplikovat v jiných situacích. Získání těchto poznatků vyžaduje však hodně času a energie. Pokud bychom dokázali simulovat průběh dané hry a jen pozorovat celek (nebo jenom vybrané části), byly by tyto poznatky jednodušeji vstřebatelné.

Asteroids je hra, ve které hráč ovládá vesmírnou loď a snaží se přežít co nejdéle. Je v ní obklopen asteroidy a nepřáteli, kteří se mu snaží toto překazit.

Tato bakalářská práce se zabývá možnostmi jedné části umělé inteligence - neuronových sítí. Konkrétně se zaměřuje na implementaci plně rekurentní neuronové sítě do zmíněné hry. V mé práci byla vyvinuta kopie této hry pro potřeby propojení umělé inteligence s informacemi o prostředí hry a ovládání lodi.

V kapitole 1 se zabývám pojmy a vlastnostmi hry Asteroids. V další kapitole 2 uvádím různé možnosti, kterými je umělá inteligence schopná hrát počítačové hry. Kapitola 3 obsahuje experimenty s neuronovou sítí. V poslední kapitole 4 se diskutují výsledky těchto experimentů.

Kapitola 1

Hra Asteroids

1.1 Top Down Shooter

Hra Asteroids se řadí mezi počítačové hry žánru střílečka s pohledem shora dolů. Někteří označují tento žánr pojmem „Shoot 'em up“. Ve hře této kategorie se hlavní hrdina ocitá proti velkému množství nepřátel. Tyto nepřátelé na něj střílí zpět či jej ohrožují jiným způsobem. Častokrát se hráč dívá na herní scénu shora či ze strany, odpadává tedy ze hry třetí dimenze a hraje se pouze se šířkou a výškou. Hra bývá často rychlá a akční, chyby hráče jsou penalizovány, ale ne tolik, aby se nedaly opravit. Hráč tedy musí mít rychlý postřeh a správně reagovat na nebezpečí. Častokrát hře není cizí přidávat za dobré výkony další pokusy (označované jako životy).

Dají se zde ještě rozlišit různé podžánry, každý s jiným zaměřením na typ hry. Mezi tyto kategorie patří například:

- **bullet hell** (kulkové peklo) – typický je pro tento žánr velký počet (desítky až stovky) projektilů na jedné obrazovce, hráč se musí těmto kulkám vyhýbat.

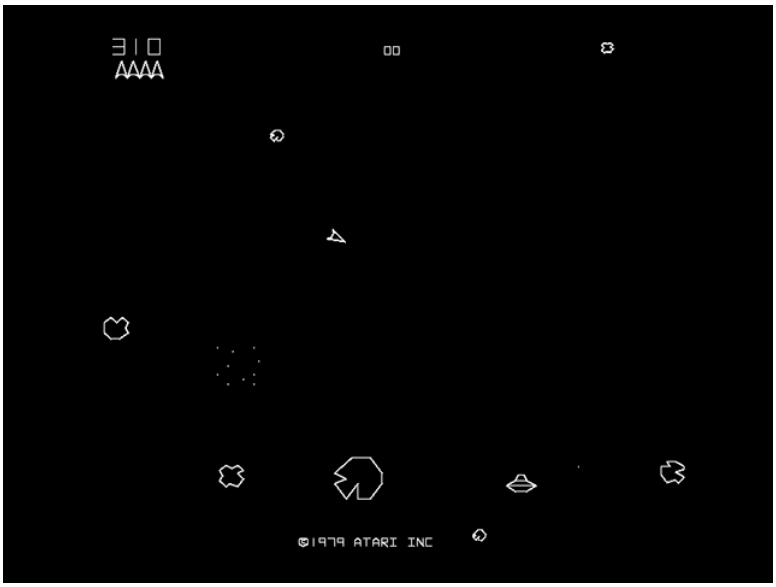
- **twin-stick shooter** (dvojpáková střílečka) – hráči se povoluje střílet i jiným směrem, než je jeho otočení. Pro tyto hry je častokrát doporučovaný ovladač se dvěma analogovými páčkami.

- **run and gun** (běž a střílej) – střílečka s pohledem z boku, hlavní hrdina běží (často zleva doprava) a střílí před sebe. Také název již naznačuje zde aplikovanou mechaniku gravitace a skákání.

Je možné tyto žánry kombinovat, hra se nemusí nutně upnout jen na jeden z nich. Těmto kategoriím se nebudeme ve zbytku práce věnovat a vystačíme si se střílečkou z pohledu shora.

1.2 Originální hra

Asteroids je střílečka ve dvou dimenzích. Hráč v ní ovládá trojúhelníkovou vesmírnou loď a může se s ní pohybovat otáčením doleva nebo doprava a zažehnutím trysek. Těmi se začne loď pohybovat v jejím směru otočení. Pokud je loď v pohybu, tak postupně zpomaluje, dokud úplně nezastaví nebo hráč nezačne používat motor lodi v opačném směru, než se loď pohybuje. Další možností ovládání pohybu lodi je nouzový teleport. Touto schopností se loď okamžitě přenese na jiné místo na obrazovce. Může se však stát, že se loď přemístí přímo do asteroidu a v tu chvíli má hráč o život méně. V poslední řadě může loď také střílet. Hráč však nemůže tuto vlastnost využívat neomezeně. Počet střel na obrazovce je zde omezen: pokud zde existují více než tři hráčovy kulky, pak mu loď nedovolí vystřelit. [1]



Obrázek 1.1: Obrázek původní hry [1]

Dále se v této hře nachází asteroidy. Obrázek 1.1 je ukazuje jako v prostoru létající mnohoúhelníky, které má hráč za úkol sestřelit. Když střela zasáhne asteroid, tak se vzájemně zničí. Jestliže však nebyl asteroid dostatečně malý, tak se místo zničení rozštěpí na několik menších. Pokud asteroid narazí do lodi, tak je zničena a hráč pokračuje s nižším počtem životů. Asteroidy se samovolně pohybují herní scénou, každý vlastním směrem, který je konstantní, dokud není asteroid rozštěpen. Každý z úlomků si poté nastaví vlastní směr. Každý sestřelený asteroid dává hráči body do skóre. Za každých 10 000 bodů získává hráč život navíc.

Ve hře se také objevují čas od času nepřátelé ve tvaru létajících talířů. Velké ufo létá prostorem a střílí náhodně okolo sebe. Malé ufo se snaží mířit

na hráčovu loď. Jejich pohyb je velmi jednoduchý. Létají po úsečce, která je náhodně dlouhá a jakmile dosáhnou jejího konce, tak se vypočítá nová úsečka, podle které budou dále postupovat. Po dosažení určitého skóre se objevují jen malé létající talíře.

Dále na obrázku 1.1 vidíme v levém horním rohu ukazatel skóre a počet životů, který je znázorněný jako několik malých vesmírných lodí vedle sebe.

V původní hře bylo nastaveno, že asteroidy nemohly narážet mezi sebou, pravděpodobně kvůli hardwarovým limitacím. Toto by se dalo považovat za optimalizaci, protože se herní model tímto zjednodušuje. Detekce kolizí se tedy počítá jen mezi nepřáteli, hráčem a kulkami. Každý z těchto objektů samozřejmě počítá i s tím, že by se mohl srazit s asteroidem.

Důležitým prvkem ve hře je takzvaný „wraparound“. Pokud některý z objektů v herní scéně vyjede z obrazovky, tak je přemístěn na opačný okraj obrazovky se stejným směrem a rychlostí jako předtím. Hra tedy za použití této techniky nepotřebuje více herních scén, vše se může generovat (zčásti) náhodně. Další výhodou jsou vlastnosti pohledu na scénu. Není zde nutnost počítat relativní pozici objektů vůči „kameře“, protože ta zůstává vždy na stejném místě. Vše se děje ve dvojrozměrném prostoru, kde se všechny body dají určit pomocí kartézských souřadnic.

Hra nemá různé velikosti scén, vše se odehrává na celé obrazovce. Není v ní ani žádná možnost přiblížení, transformace kromě rotace objektů (a to jenom co se týče vlastní lodi) se zde neobjevuje.

Graficky se hra jeví vcelku primitivně. Barevná paleta má rozsah pouze dvou barev, a to bílé a černé. Za to může tehdejší rozhodnutí vyvíjet hru pro vektorové displeje, i když původně chtěli vývojáři z Atari[2] použít rastrovou grafiku [1].

V originální hře nebyla možnost vyhrát hru. Cílem hráče bylo nahromadit si co největší možné skóre a přežít co nejdéle. Jediným možným způsobem jejího ukončení byla ztráta všech životů.

Přestože jde o velmi jednoduchou hru (ať už mechanicky či esteticky), jedná se o velký hit tehdejší doby. Hra oficiálně vyšla v roce 1979 [3]. Je připisována vedoucímu tohoto projektu Edwardu Loggovi [3]. Automatů s první veřejnou verzí této hry společnost Atari prodala přes 70 000 [4]. V dalších letech byly úspěšně vydalány novější verze [4].

1.3 Vlastní implementace

V mojí implementaci jsem se rozhodl některé prvky původní hry odstranit a některé přidat. Například graficky hra vypadá odlišně. Tvary jsem nechal vyplněné, takže hra nepůsobí tolik prázdně.

Životy byly z této implementace úplně odstraněny. Jelikož účelem této hry byla následná implementace neuronové sítě, tak nebyl důvod mít mechaniku

oživování. Místo ní může hráč restartovat hru kolikrát chce a jeho počet pokusů na hru není omezen.

V originální implementaci bylo omezeno střílení tím, že na obrazovce mohl být pouze určitý počet střel (v originální hře 4). Využil jsem však jiný způsob střelby, který se používá v moderních hrách. Loď má určenou hodnotu, jak často za vteřinu může střílet. Pomocí této hodnoty se dopočítává, jestli v daný snímek hry může střílet.

V původní hře stále existovaly části, které se daly vylepšit. Toto demonstruji například na režimu více hráčů na lokální síti. V původní hře se tento režim používal pouze na jednom automatu. Teoreticky můj klon hry dokáže zpracovávat velké počty (až stovky) hráčů na lokální síti.

Původní hra měla jenom jednu úroveň, kde se od začátku vyskytovala hráčova loď. Dále se zde v dávkách několika málo jednotek objevovaly asteroidy, pokud již žádný nebyl na obrazovce. Čas od času se zde také objevilo ufo. V mojí implementaci je vše uloženo v paměti počítače a nahráno najednou. Úrovně jsou vytvářeny v kódu programu. Přepínání scén funguje pouze po stisknutí patřičné klávesy.

Nepřátelé se v mé hře pohybují a vypadají jinak. Neobjevují se během scény, ale musejí být vloženi do určité úrovně před jejím začátkem, aby mohli existovat. V každém kroku scény hledají po obrazovce nejbližší objekt a letí za ním. Pokud jsou dostatečně blízko, tak se jej snaží sestřelit. Další rozdíl je ten, že jsem neměl důvod je graficky připodobnit původním ufům, takže vypadají podobně, jako hráčova loď, s tou výjimkou, že mají jinou barvu. Nepřátelé sestřelují asteroidy a tím kazí hráči skóre. Škodí tedy hráči pouze nepřímo.

V programu jsem měl naprogramované kolize mezi asteroidy, při nichž se měly od sebe odrazit. V pozdějších testech jsem toto zavrhl kvůli rychlosti algoritmu. Při normálním běhu hry (tj. 60 snímků za vteřinu) běželo vše hladce, ale evoluci to zbytečně zdržovalo. Proto se v pozdějších testech zde již nepočítají kolize. Lodě samozřejmě kolize s ostatními předměty stále počítají.

1.4 Popis programu

Pro tuto hru jsem si zvolil programovací jazyk Java kvůli mým zkušenostem s tímto jazykem a širokou zásobou prostředků, které tento jazyk podporuje a poskytuje.

V ukázce programu 1.1 vidíme rozhraní třídy `AbstractCollider`, které umožňuje veškeré kolize. Metoda `circleCollide` určuje, jakým způsobem bude collider detekovat kolize s kruhovým objektem. Podobně metoda `lineCollide` určuje kolize s úsečkou. Obecná metoda `collide` podle své třídy pouze deleguje detekci kolize na druhý collider a volá na něm metodu svého geometrického objektu. Pokud tedy kruh bude testovat kolizi s úsečkou, zavolá se na kruhu metoda `collide`. V ní bude návratová hodnota volání metody `circleCollide` na colliderú úsečky s parametrem vlastního colliderú. Metoda detekce kolize

```
public abstract class AbstractCollider {

    protected final GameObject gameObject;

    public AbstractCollider(GameObject gameObject) {
        this.gameObject = gameObject;
    }

    protected abstract boolean circleCollide(Ellipse2D
       circle);
    protected abstract boolean lineCollide(Line2D line);
    public abstract boolean collide(AbstractCollider
       other);
    public Point2D getPosition() {
        return gameObject.getPosition();
    }
    public abstract void draw(GraphicsContext context);
}
```

Úryvek programu 1.1: Rozhraní třídy `AbstractCollider`

na úsečce má v tuto chvíli všechny potřebné informace o druhém collideru a může vyhodnotit tuto detekci. Podobně fungují všechny ostatní collidery.

Hlavní třídou programu je `GameObject`, ze kterého dědí všechny další objekty, které se mohou vyskytnout ve scéně. Uchovává v sobě kartézské souřadnice daného objektu, rotaci a `AbstractCollider`. Veškerý výpočet běhu probíhá v metodě `update`. Každý objekt si také počítá, jestli se nenachází mimo obrazovku a jestli má použít „wraparound“.

Dále je velmi důležitá třída `AbstractEntity`, ze které dědí všechny objekty lodí. Uvnitř této třídy se nachází mimo jiné i `AbstractShipController`, který má na starosti veškeré ovládání lodi. Jsou zde metody, které umožňují její rychlé ovládání pomocí změny přiřazené Booleovské proměnné. Pokud bychom chtěli po lodi, aby se začala otáčet doleva, stačí zavolat metodu `setTurningLeft(true)`. Zbytek logiky otáčení již obstará loď při metodě `update()`, která se volá jednou, za každý snímek. V hráčově lodi zaregistruje klávesnicové vstupy na ovládání lodi. U umělých lodí se právě zde nasadí neuronová síť.

Další objekt, který je nutné zmínit, je objekt `Level`. Je to kontejner na objekty, které si hra bude postupně načítat. Do něj se načtou objekty přes konstruktor, který přijímá proměnlivý počet `GameObject`ů.

Celou hru ovládá objekt `Game`. V ní se načte patřičný `Level`, a poté probíhá smyčka, ve které se procházejí všechny načtené objekty a volá se na nich `update`. Tím se zaručuje, že se všechny objekty pohybují či vykonávají svou

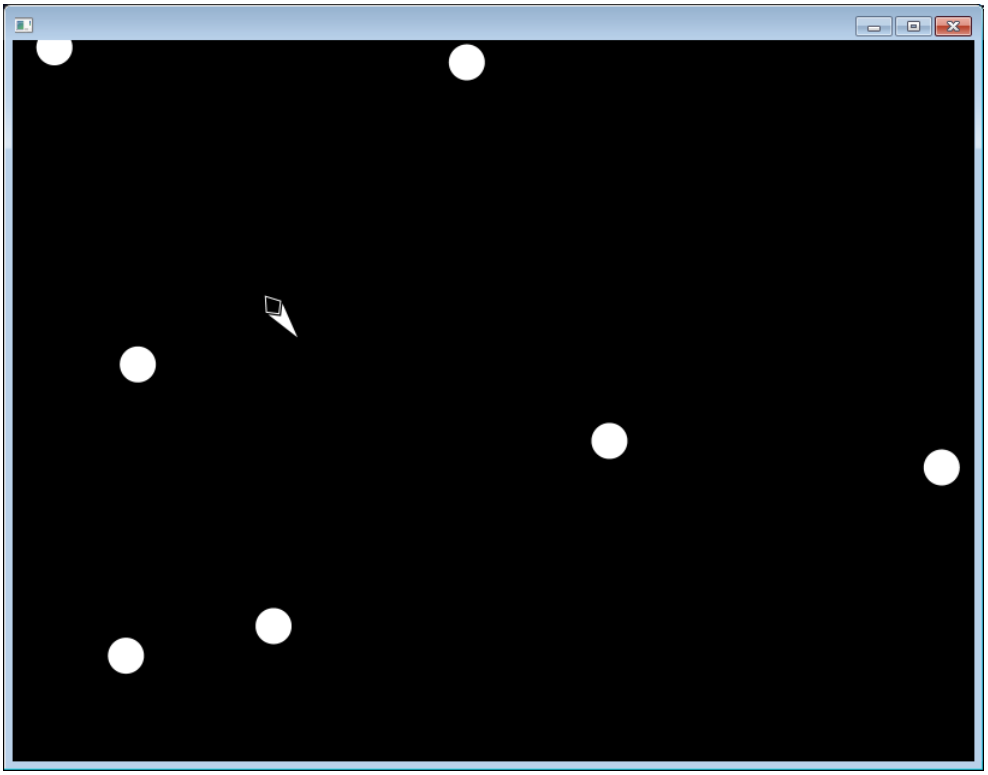
činnost.

Objekt `Game` obsahuje dvě hlavní vlákna, které dále řídí hru. První se stará o to, aby objekty byly na správných místech. Již bylo zmíněno, že všechny objekty mají implementovanou metodu `update()`, která zařizuje přesně toto chování. Druhé vlákno obhospodařuje vykreslování objektů na scéně. Každý objekt má metodu `draw(GraphicsContext context)`. Zde se používá třída JavaFX 8 `GraphicsContext`, která umožňuje na obrazovku kreslit jednoduché tvary [5]. Obě vlákna jsou nyní nastavena tak, že provedou svou činnost 60 krát za sekundu.

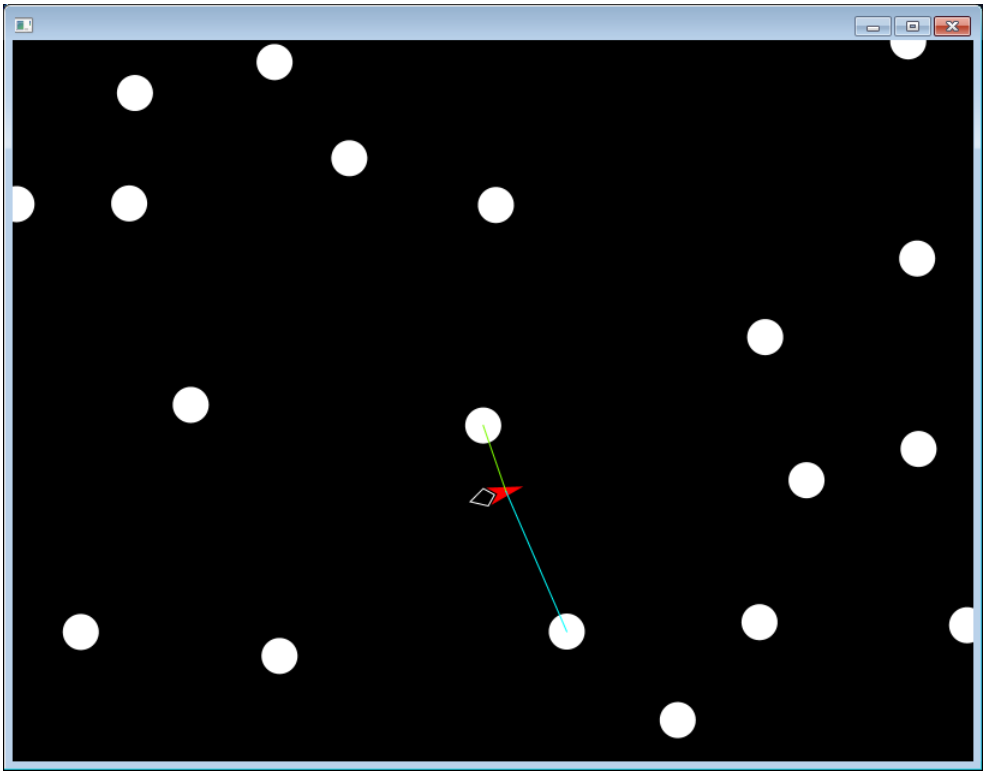
1.5 Uživatelské zobrazení

Zde jsou vidět změny oproti původní hře. V obrázku 1.2 je vidět hráčova loď (v bílém). Je patrné, že je její obsah barevně vyplněn, kdežto v původní hře byla jen souborem čar. V klonu hry se podobně změnily i asteroidy. Vzhledem k tomu, že práce nebyla zaměřena na co nejbližší kopii původní hry, jsem ponechal asteroidům tvar vyplněného kruhu. V uvedených obrázcích 1.2 a 1.3 se nacházejí pouze malé asteroidy, které se již nemohou rozpadnout na menší.

Na obrázku 1.3 je vidět kromě většího počtu asteroidů i loď umělé inteligence, která je oproti hráčově lodi rozlišená červenou barvou. Z ní vedou dvě čáry: jedna zelená a jedna modrá. Zelená označuje nejbližší objekt vůči lodi a modrá druhý nejbližší.



Obrázek 1.2: Obrázek z upravené hry



Obrázek 1.3: Obrázek scény s nepřítelem v upravené hře

Kapitola 2

Umělá inteligence ve hrách

Se záměnou hráče za umělou inteligenci již bylo úspěšně experimentováno [6] [7]. Existují i různé soutěže, kde se lidé snaží naprogramovat co možná nejlepší AI do různých her. Uvádím zde například TheAIGames [8] a Pogamut Cup[9]. Velmi úspěšným projektem byl také AlphaGo od společnosti DeepMind [10], kde se podařilo 4 krát z 5 zápasů porazit velmistra hry Go.

2.1 Obecná umělá inteligence

Existuje několik definic umělé inteligence (AI - z anglického Artificial Inteligence), všechny jsou velmi filozoficky zavádějící. Po AI buďto požadujeme, aby se chovala/přemýšlela lidsky, nebo aby se chovala/přemýšlela racionálně [11]. Pro umělou inteligenci je dále důležitý pojem **agent**. Neformálně je agentem cokoliv, co se dokáže nějakým způsobem chovat. Zde se budeme zabývat hlavně racionálně chovajícím se agentem, neboli agentem, který se snaží maximalizovat svoje zisky. V multiagentních systémech mohou agenti spolupracovat, zde se však omezíme pouze na jednoho agenta ve scéně.

2.2 Možnosti ovládání hry

Způsobů, jak nahradit lidského pilota vesmírné lodi umělým agentem, je několik. Mohli bychom například naprogramovat jednoduchá pravidla, kterými se stroj bude řídit. Tyto pravidla je pak jednoduché poskládat do stavového automatu a přiřadit přechody jednotlivým stavům chování. Tento způsob ovládání umělých bytostí v počítačových hrách byl používán poměrně hojně [12] [13]. Bohužel je velmi obtížné i u takto jednoduché hry zjistit, jaké ovládání stroje je optimální. Častokrát ani nelze lehce stroji popsat, jak se hra správně ovládá.

Je zde možné simulovat chování agenta zcela náhodně. Akcí je jen několik, implementačně tedy není s tímto způsobem ovládání lodi žádný problém. Její

```
public void update() {
    findNearestClosestObject();
    if (chasedObject != null) {
        aimAt(chasedObject);
        if (isCloseTo(chasedObject)) {
            setThrusting(false);
            entity.shoot();
        } else {
            setThrusting(true);
        }
    }
}
```

Úryvek programu 2.1: Chování počítačové lodi

zmatené pohyby na obrazovce však nejspíš neposunou vývoj umělé inteligence o mnoho dál. Agent nemá žádnou tendenci směřovat nějakým dlouhodobým cílem.

Tabulku pravidel jsem původně implementoval jako součást hry. Tyto pravidla se používaly pro základní umělou inteligenci nepřátel. V úryvku programu 2.1 je zobrazená původní implementace umělé inteligence. Nejprve si agent najde nejbližší objekt. Pokud takový existuje, tak se za ním otáčí, a pokud je blízko, tak po něm začne střílet. Jestliže je však příliš daleko od daného objektu, tak zapne trysky a tím se snaží dostat co nejblíže k vybranému cíli. Takovíto agenti se ve hře nedostávají k vysokým výsledkům a pokud ano, vyžadují zkušeného experta, který dokáže popsat toto perspektivní chování.

2.3 Stavový prostor

Pro reprezentaci mnoha modelů lze použít stavový prostor. V něm existují dvě hlavní části: stavy a akce. Obecně je stav množinou vlastností. Akcemi se z jednoho stavu dostáváme do jiného. Pokud budeme hledat sekvence akcí, které budou reprezentovat chování, je pro stavový prostor důležitý počáteční stav a množina koncových stavů. Stavový prostor si tedy lze představit jako orientovaný graf. Ten vytvoříme tak, že vrcholům přiřadíme stav a hranám akce [14].

Stav je v našem případě shromáždění pozic objektů a jejich směrů pohybu. Pro našeho agenta existuje ve stavu také číslo označující čas, kdy může opět vystřelit. Akce u nás je kombinace otáčení (vlevo, vpravo či žádné), použití trysek nebo střelba. Celkový počet akcí je tedy 12 – 3 možnosti otáčení, 2 možnosti používání trysek a 2 možnosti používání střelby. Bohužel pro tento případ vzrůstají nároky na paměť a výpočetní čas neúnosně rychle. Pro zmí-

něných 12 akcí narůstá problém exponenciálně. I kdybychom negenerovali při každé akci nový stav a nejprve se pokusili najít tento stav již vygenerovaný, tak se stejně bude prostor rozpínat příliš rychle. Navíc hledání stavu v těchto již existujících bude také časově náročné.

2.4 Neuronová síť

Již existují frameworky (například [15] a [16]), které se zabývají neuronovými sítěmi. Součástí této práce je však implementovat vlastní neuronovou síť, a tak se jimi dále nebudeme v této práci zabývat.

2.4.1 Původ

Inspirací pro umělou neuronovou síť (NN - neural network) je biologický neuron. Je to základní stavební a funkční jednotka neuronové sítě. Specializace buňky (neuronu) je příjem, šíření a zpracování informací. Skládá se z těla a výběžků rozlišitelných podle funkce (axon a dendrity). Velikost jejich těla kolísá mezi 6 a 100 μm [17]. Z těla neuronu vystupuje zpravidla několik dendritů, které se směrem ven z těla zužují a opakovaně větví. Z nich se šíří vzruch směrem do těla. Neuron má většinou jeden výběžek označovaný jako axon, vedoucí vzruchy směrem od neuronu. Neurony se mezi sebou propojují pomocí synapsí [17].

2.4.2 Umělá neuronová síť

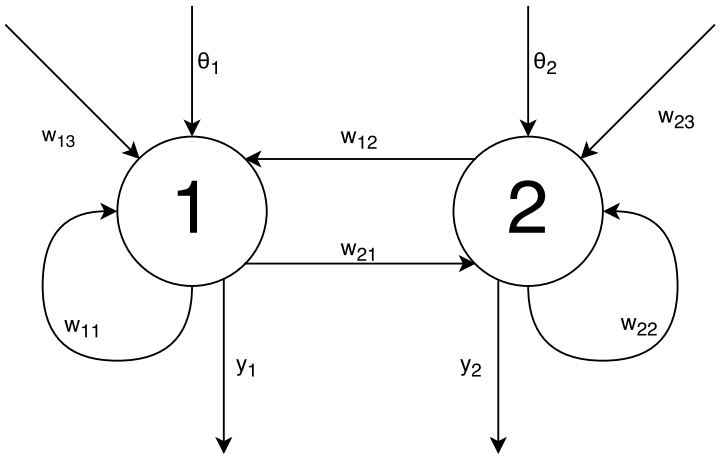
Podobně jako u lidské NN i umělá verze také přijímá, šíří a zpracovává informace. Umělý neuron má určené vstupy a výstupy. Za vstupy se berou čísla, která reprezentují kvantifikovatelnou informaci (například uplynulý počet vteřin). Každý vstup má určenou váhu, která značí, jak je daný vstup pro síť podstatný. Pro váhu w_n a hodnotu a_n neuronu n, kterých je $N \in \mathbb{N}$, provede neuron operaci:

$$f(\sum_{n=1}^{N} w_n a_n + \theta_n)$$

kde θ je práh aktivace neuronu. Funkce $f(x)$ je takzvaná aktivační funkce neuronu. Díky této funkci může neuronová síť reprezentovat nelineární funkci [11].

Neuronová síť se dá učit dvěma známými způsoby: zpětnou propagací chyb nebo evolučním algoritmem. Zpětná propagace funguje na principu pokus–omyl. Při špatných výstupech NN se váhy nejvíce účastněných neuronů korigují. Tento přístup se bude obtížně aplikovat u NN v počítačových hrách. Těžko totiž poznáme, jaká akce byla právě ta, která způsobila největší ztráty. Navíc se nejspíš bude jednat o soubor akcí.

U druhé metody nemusíme znát „špatnou akci“ vůbec. Stačí přiřadit ke každé neuronové síti číselné ohodnocení popisující, jak dobře si neuronová síť



$$\begin{pmatrix} w_{11} & w_{12} & \theta_1 & w_{13} \\ w_{22} & w_{21} & \theta_2 & w_{23} \end{pmatrix} \begin{pmatrix} y_{1,k-1} \\ y_{2,k-1} \\ 1 \\ x_{1,k} \end{pmatrix} = \begin{pmatrix} y_{1,k} \\ y_{2,k} \end{pmatrix}$$

Obrázek 2.1: Obrázek dvou-neuronové plně rekurentní sítě a váhové matice

vede. Funkce, která by toto vypočítávala, by se dala odvodit od počítání skóre. Váhy synapsí se tedy budou upravovat a ohodnocovat několikrát. To je velká výhoda této metody, nemusíme neuronové síti nijak popisovat, jak vypadá kvalitní řešení. Ručně určit či vypočítat hodnoty synapse tak, aby loď přežila co nejdéle a přitom sbírala body skóre, je velmi obtížné.

2.4.3 Plně rekurentní neuronová síť

Plně rekurentní neuronové sítě se vyznačují tím, že jeden z výstupů neuronu je použit i jako jeho vstup [11]. V kroku k dostává neuron data z kroku $k-1$. První krok inicializuje vstupy tohoto druhu na nulu. V tomto typu rekurentní sítě je také zaručeno, že každý neuron je propojen se všemi ostatními.

Plně rekurentní NN se dá jednoznačně popsat váhovou maticí. Na obrázku 2.1 je znázorněna plně rekurentní síť se dvěma neurony a způsob reprezentace této struktury pomocí matice. Pokud vynásobíme váhovou matici (vlevo) vektorem vstupů (vpravo), pak dostaneme přímo vektor následujících výstupů. Hodnoty y_k představují výstupy a x_k vstupy sítě v kroku k.

V určitém smyslu je plně rekurentní síť zobecněním dopředné sítě (ve které se nepovolují smyčky). Pokud bychom nastavili na nulu váhu synapse vedoucí zpět do zdrojového neuronu a synapse, které způsobují smyčky, tak bude výsledkem právě dopředná neuronová síť. Tento typ NN však obvykle neposkytuje výstupy na všech neuronech, a tak je i toto nutné ošetřit.

2.5 Evoluční algoritmus

Jde o techniku iterační optimalizace. Kandidátní řešení problému se postupně vylepšuje („šlechtí“), dokud nesplňuje námi určenou podmínku. Tou může být například počet iterací nebo celková kvalita řešení. Evoluční algoritmy pracují na principu přirozené evoluce. V algoritmu se využívá takzvané **populace**, což je souhrn **jedinců**. Každý jedinec si v sobě uchovává **genom**, zakódované řešení daného problému. Důležité je, aby se genomy mohly navzájem ovlivňovat. K tomu slouží genetické operátory – **křížení** a **mutace**. Křížení vybere z populace několik jedinců a vytvoří nové jedince získané kombinací genomů rodičů. Mutace může samovolně pozměnit genom jednoho jedince. Často uživatel v evoluci může určit, kolik jedinců z populace se bude křížit a jak velká část genomu bude mutovat. Toto zajišťují v algoritmu pravděpodobnosti křížení a mutace. Ohodnocení kvality jedince určuje funkce **fitness**. Pokud má jedinec hodnotu této funkce větší než jiný, znamená to, že je lepším řešením problému.

Tento druh algoritmů má široké využití. Častokrát dokáží přijít na poměrně dobré řešení i NP-úplných problémů [18]. Třída NP-úplných problémů je zjednodušeně řečeno množina problémů, pro které není znám algoritmus, který by zaručeně našel nejlepší řešení a zároveň nenašel toto řešení jiným způsobem, než je postupné zkoušení všech možností [18].

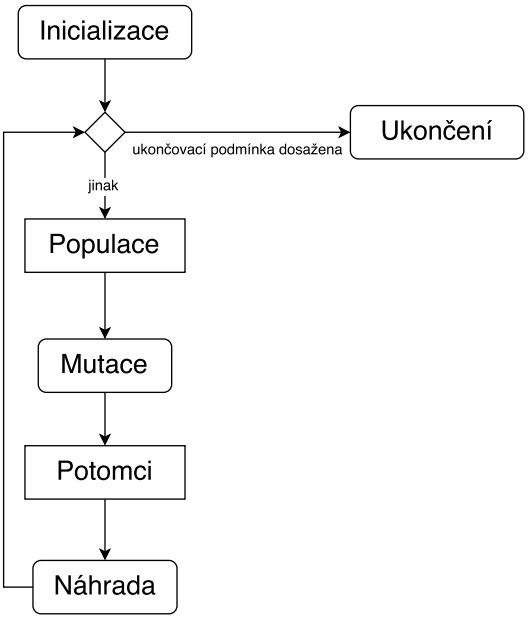
Je důležité zmínit, že po běhu evolučního algoritmu není garantován optimální výsledek. Častokrát není výstup ani uspokojivý. Funkce fitness často pracuje s mnoha hodnotami a její obor hodnot má mnoho lokálních optim. Jedinci mohou postupně sklouzávat do těchto oblastí a zůstat v nich. Existují však postupy, jak se jim vyhnout nebo jak se z nich dostat.

Nevýhodou může být dlouhá doba běhu. Zvažme následující případ: nastavíme populaci o velikosti 500 jedinců. Doba výpočtu simulace a ohodnocení každého jedince bude trvat 100 ms. Při 2000 generací se simulace ukončí přibližně po 27 hodinách při sekvenčním zpracování. Z toho vyplývá, že je důležité odladit všechny chyby na menších instancích běhu. Teprve poté začneme pouštět program s delší vyhodnocovací dobou.

2.6 Diferenciální evoluce

Jednou z podkategorie evolučních algoritmů je diferenciální evoluce (DE) vynalezena Rainerem Stornem a Kennethem Pricem [19]. Jako genom jedinců bývá zde použit vektor reálných čísel. DE kombinuje operátory křížení a mutace do jednoho. Schéma 2.2 zobrazuje její průběh. V inicializační fázi nastavíme populaci jedinců tak, že jim do genomu vkládáme náhodná čísla v rozsahu $[-\frac{1}{2}, \frac{1}{2}] \in \mathbb{R}$.

Ve vektoru n vypočteme nové hodnoty $i < W \in \mathbb{N}$, kde W je počet vah, pomocí následujících parametrů:



Obrázek 2.2: Schéma diferenciální evoluce

- tři různí rodiče z populace p_1, p_2, p_3,
- diferenční váha $F \in [0, 2] \subset \mathbb{R}$,
- pravděpodobnost mutace genu $M \in [0, 1] \subset \mathbb{R}$,
- hodnota $x < W$,
- náhodné číslo $r \in [0, 1] \subset \mathbb{R}$.

Hodnotu váhy i v novém řetězci n dostaneme následujícím způsobem:

$$n^i = \begin{cases} p_1^i + F \times (p_2^i - p_3^i) & \text{pokud } r < M \text{ nebo } i = x, \\ p_1^i & \text{jinak.} \end{cases}$$

Náhrada probíhá tak, že pokud je vyvinutý kandidát lepší (má větší hodnotu fitness) než p_1, tak tento jedinec převezme místo svého prvního rodiče. V opačném případě rodič v populaci zůstane.

Vyvstává zde problém zakódování neuronové sítě do genomu. Učení neuronových sítí pomocí DE již popsali Ronald J. Williams a David Zipser ve své práci [20]. Jestliže neuronová síť využívá matici, ve které jednotlivá čísla určují váhy neuronových synapsí, pak za genom prohlásíme vektor čísel, který dostaneme spojením řádků matice.

Ohodnocení fitness je spíše časově náročné, než obtížné. Musíme načíst scénu, vytvořit patřičný `Controller` a vložit jej do lodi, odsimulovat určený

počet kroků ve scéně a teprve poté budeme vědět, jak dobře si neuronová síť vede.

Kapitola 3

Vytváření neuronového ovladače lodi

Zde se zabývám konkrétními experimenty evolucí neuronových sítí. Ve všech testech se používaly pouze malé asteroidy. Nebylo tedy testováno, jak by se neuronová síť chovala při rozštěpení asteroidu na několik menších. Moje domněnka je taková, že by se stejně výsledné chování lodi nezměnilo, protože její úkol by zůstal stále stejný.

Postup učení neuronové sítě evolučním algoritmem je, jak již bylo zmíněno, velmi časově náročný. Častokrát se mi stalo, že jsem nechal na počítači spuštěnou evoluci, u které jsem poté zjistil, že vstupy do sítě nejsou správně nastaveny. Předzpracování dat je zde stejně (ne-li více) důležité jako implementace vlastní sítě.

Neuronová síť musí být nějak napojena na řídící prvky lodi. Jako aktivační funkce neuronů byla použita standardní logistická sigmoida [21]:

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

Tato funkce má obor hodnot v intervalu $[0, 1]$. Pokud je výstup větší než 0.5, pak je patřičná funkce ovládání aktivována, jinak nikoliv. Výjimkou je otáčení lodi, kde se používá aritmetický průměr dvou neuronů. Pokud je hodnota větší, než 0.6, pak loď rotuje doprava. Pokud je nižší, než 0.4, pak rotuje doleva. V ostatních případech se neotáčí. Tímto bylo dosaženo toho, že nemohlo být zároveň aktivováno otáčení doleva a doprava zároveň.

Fitness se počítá v některých experimentech jinak než v ostatních. Základ fitness tvoří počet kroků, které loď přežila ve scéně. V experimentech, kde se penalizuje srážka asteroidu s lodí, se za každou kolizi odečítá určitý počet bodů. Ten je nastavený na počátku simulace jedince na 200 a s každou kolizí se zvyšuje. V pokusech, kde se podporuje střelba se za každý sestřelený asteroid přičte 100 bodů a za každou netrefenou střelu se odečítá 50 bodů.

Počet generací ve zde uvedených grafech je vždy 1000. Byly provedeny experimenty ohledně vyšších generací, ale jedinci se vyvinuli s téměř stejným chováním. Toto potvrzuje nízký růst fitness ke konci těchto evolucí.

Mutační konstanta byla nastavena na 30 %. Tuto hodnotu bylo nutné odhadnout „rozumně“ vysokou. Při testech s nízkou pravděpodobností mutace se jedinci vyvíjeli příliš pomalu. S vysokou mírou mutace však nedosahovali dostatečně vysokých hodnot fitness.

Populace obsahovala v evolucích 300 jedinců. Vyšší počty umožňují větší a různorodější genetický základ na úkor zpomalení doby běhu programu.

Počet kroků hry byl nastaven na 2000, což odpovídá přibližně 30 vteřinám hry simulované původní rychlostí. Toto je dostatečně dlouhá doba na vystavení lodi nástrahám hry. Proto je nucena se asteroidům vyhýbat a tím si osvojuje žádané schopnosti.

3.1 Původní návrh

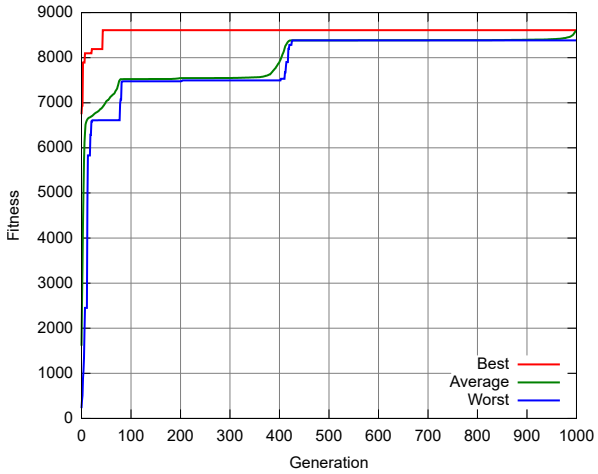
Původní návrh byl, že na vstupy neuronové sítě budu posílat vzdálenost a relativní úhel několika nejbližších objektů. Na obrázku 1.3 jsou vidět dvě čáry směřující od lodi k asteroidům. Tyto čáry znázorňují dva nejbližší objekty ve scéně. Původně jsem experimentoval se dvěma a později třemi objekty. Teoreticky se totiž loď dokáže objektům vyhýbat, pokud o nich bude vědět jejich směr, rychlost, velikost a pamatovat si, kde přibližně mohou být další okolo ní.

Toto chování jsem vyvíjel pro 6 neuronů, každý sudý dostal na vstupu úhel, liché dostaly vzdálenost. Takto vyvinutá loď se chovala velmi podivně. Nedokázala se vyhýbat asteroidům, pokud přicházely z více než dvou směrů nebo pokud se často měnily nejbližší objekty. Když okolo lodi prolétne velmi blízko asteroid, který nemá s lodí kolizní trajektorii, tak jí zastíní jeden vstup, přičemž v herních scénách, které mají mnoho asteroidů, může mít vzdálenější asteroid větší šanci ji zasáhnout. Ke zlepšení chování lodi nepřispělo ani to, že jí bylo povoleno střílet. Vyvinula si tedy takový způsob chování, že se nepřetržitě otáčela pouze jedním směrem a neustále střílela.

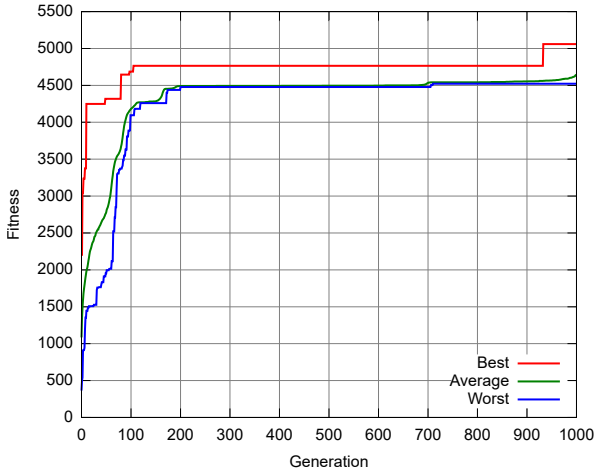
Dle grafu 3.1 vývoje populace se nejlepší jedinec vyvine během několika prvních desítek generací. Poté již evoluce nenalezne lepší řešení. Celkově populace postupně pomalu konverguje k nejlepšímu jedinci. Mezi 400 a 500 generací populace dosáhla lokálního optima a již se z něj nedostala. Fitness může dosahovat takto vysokých hodnot díky náhodnému střílení okolo sebe.

3.2 Vypnutí střelby

V dalším kroku jsem tedy chtěl zjistit, jestli se agent dokáže lépe vyhýbat objektům, když vypneme lodi možnost střelby. Toto jí však také nepomohlo. Výsledek evoluce je stejný jako ten předchozí s tou výjimkou, že lodi byl vypnut



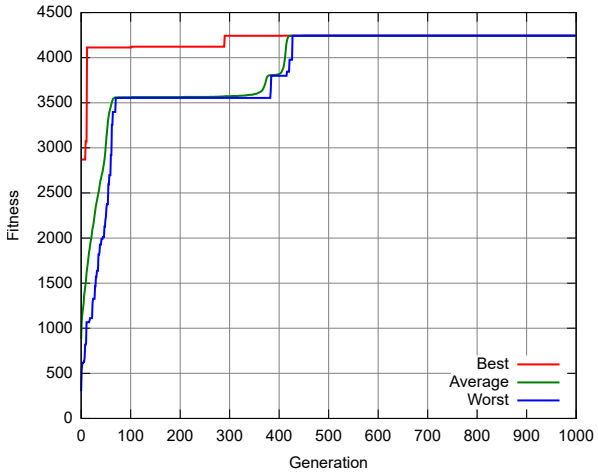
Obrázek 3.1: Vývoj populace ze sekce původního návrhu hry 3.1



Obrázek 3.2: Vývoj populace ze sekce vypnutí střelby 3.2

výstup, který byl mapován na střelbu. Loď se sice snažila asteroidům vyhýbat, ale díky omezenému počtu snímaných objektů nemohla vědět, že k přežití bude potřebovat komplikovanější manévr. Mezi různé možnosti vylepšení se řadí zvýšení počtu vstupů (a zároveň s tím počtu neuronů), změna vstupních informací neuronové sítě nebo zmenšený počet asteroidů ve scéně.

Z dat uvedených v obrázku 3.2 je viditelné výrazné zhoršení fitness, protože lodi bylo zakázáno střílet. V první stovce generací se vývoj dostal do lokálního optima a podařilo se mu z něj dostat pouze jednou na konci evoluce.



Obrázek 3.3: Vývoj populace ze sekce změny vstupního úhlu 3.3

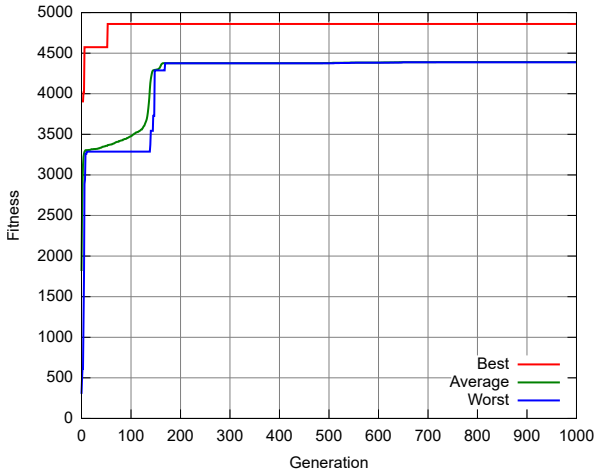
3.3 Změna vstupního úhlu

Dalším pokusem o nápravu chování jedince bylo transformovat informace tak, aby vstupní neurony nedostávaly čistý úhel, ale sinus a cosinus úhlu. Z funkce, která čistě předávala úhel, se tímto stala periodická funkce s malým rozsahem oboru hodnot, což neuronovým sítím vyhovuje. Loď s 6 neurony tedy mohla dostávat tyto informace pouze o dvou asteroidech. Ukázalo se však, že chování agenta bylo téměř identické s předchozím případem.

Oproti předchozím případům se výsledky grafu 3.3 zhoršily. Od 500 generace se vývoj téměř nepohnul. Celá populace efektivně degenerovala na jednoho reprezentanta.

3.4 Varianty scén

Zvažoval jsem možnost, že se neuronová síť možná naučí více informací než by pro ni bylo výhodné, a tím ztratí schopnost generalizace. Přestože každá evoluce načetla randomizovanou scénu, neuronová síť si mohla v průběhu učení zapamatovat až příliš a potom si jí stačilo jenom vybavovat akce, které jí přinesou největší fitness. Toto chování se označuje jako přeučování. Jedinec se naučí testovaná data natolik, že poté neumí vztahovat naučená pravidla na jiné scény. Přistoupil jsem k tomuto kroku s tím, že nechám jedince se učit na scénách, kde se fitness bude počítat ze součtu přežitého času ve všech scénách. Tímto se samozřejmě doba výpočtu zvýšila. Vždy jsem použil scénu, která má pokaždé stejné pozice asteroidů a lodi. Z této scény jsem vždy vygeneroval několik variant scény, kde varianta pouze mění počáteční směry asteroidů. Toto se jeví jako velmi malá změna a člověku se může zdát, že se ve scénách mnoho nezmění a že se loď stále může přeučovat. Ačkoliv je to možné, není



Obrázek 3.4: Vývoj populace ze sekce variant scén 3.4

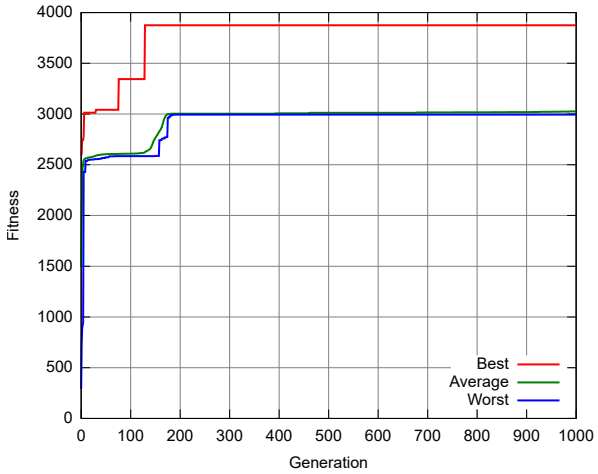
to velmi pravděpodobné. V testovací scéně je přibližně 20 asteroidů, každý má vlastní směr. I když mezi variantami bude mít 10 objektů stejný či velmi podobný směr, po nějakém čase se bude scéna velice lišit.

V údajích z obrázku 3.4 je vidět malé zlepšení. Pravděpodobně se jedná ale o odstranění chyb sinu a cosinu jakožto předzpracování vstupů. Varianty scén mohou mít však přínos v budoucích testech.

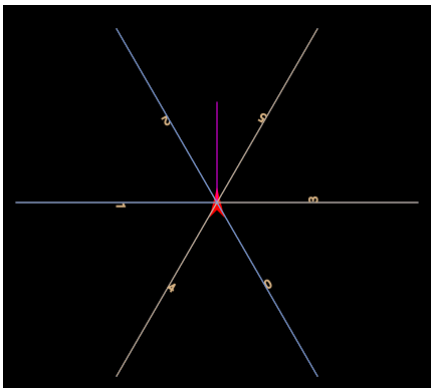
3.5 Změna periody střelby

Ani v tuto chvíli se stále nejevila neuronová síť jako inteligentní pilot lodi. Pokusil jsem se zapnout střelbu s tou podmínkou, že loď se nesmí chovat jen tak, že bude rotovat uprostřed a střílet s vysokou frekvencí. Tím by se nedosáhlo nijak užitečné strategie, takže jsem omezil periodu střelby. Po menších experimentech jsem nechal loď střílet jen jednou za 240 snímků hry. To znamená, že při 60 snímcích za vteřinu loď mohla vystřelit pouze jednou za 4 vteřiny. Ani s tímto chováním lodi však stále nepřicházel žádný úspěch. Loď se sice občasně trefila do nějakého z asteroidů, ale její schopnosti vyhýbat se asteroidům se nezlepšily ani při snížení počtu asteroidů jejich postupným sestřelováním.

V grafu 3.5 je opět vidět dlouhá perioda, kde se neuronová síť mnoho nenaučila. Zajímavý je zde rozestup nejlepšího, průměrného a nejhoršího jedince. Pravděpodobně sdílením jejich rysů nedosáhli lepších výsledků, a tak uvázli v lokálních optimech.



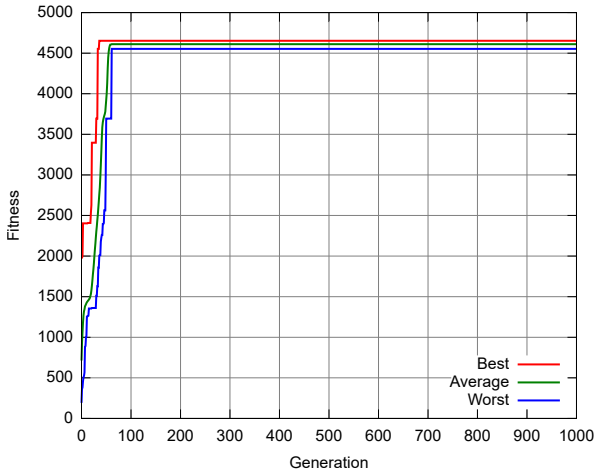
Obrázek 3.5: Vývoj populace ze sekce změny periody střelby 3.5



Obrázek 3.6: Obrázek šesti senzorů okolo lodi

3.6 Senzory

Pravděpodobně jsem se do této doby díval na problém ze špatného úhlu. Pokoušel jsem se najít jiný způsob předávání informací o okolních objektech do neuronové sítě. Některé moderní automobily mají senzory na zadním (někdy i předním) nárazníku, které snímají blízké objekty v jejich dosahu a pomáhají tak řidičům přesněji určit vzdálenost od překážek. Implementoval jsem tedy podobné senzory i u tohoto agenta. Použil jsem 6 senzorů rovnoměrně rozmístěné okolo lodi, každý se stejně velikou výsečí. 3.6 Každý snímač sleduje stejně velkou výseč směrem od lodi. Senzory jsou rozmístěny rovnoměrně okolo lodi, takže loď dostává vjemy z každé strany okolo sebe. Bohužel ani tímto způsobem se loď neposunula svým chováním nikam dál. Přesto se menší zlepšení vyskytlo ve střelbě, která byla o trochu přesnější. Toto si vysvětluji tím,



Obrázek 3.7: Vývoj populace ze sekce senzory 3.6

že u nejbližších objektů se často hodnoty vstupních neuronů přepínaly kvůli přepočítávání informací ohledně nejbližších asteroidů. U senzorů víme vždy, jestli je v blízkosti hráče objekt, a dokonce víme i přibližný úhel. Z tohoto neuronová síť pozná, jestli se vyplatí střílet.

Obrázek 3.7 nám ukazuje, že jedinci se senzory nemají tak velký rozptyl fitness. Populace sice není plná různých jedinců, ale vypadá, že je snazší se orientovat pomocí tohoto vnímacího modelu. Opět se zde však jeví nadbytečnost vysokého počtu generací.

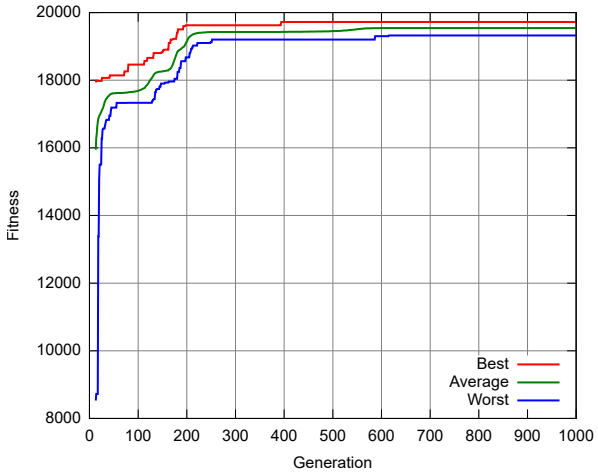
3.7 Nesmrtelnost

Další nápad, který jsem chtěl vyzkoušet, byl nastavit lodi nesmrtelnost a místo počítání fitness jen z kroků, které loď přežila, i penalizaci jedince za každou kolizi. Doteď mohla evoluce snižovat fitness spoustě jedincům, kteří mohli mít zajímavé chování, protože velmi rychle narazili do asteroidu. I toto se však projevilo jako nedostatečné, chování bylo stále velmi podobné předchozímu. V některých variantách herních scén bych řekl, že si dokonce loď vedla hůře. Zvýšil jsem zde i počet variant scén ze 3 na 10.

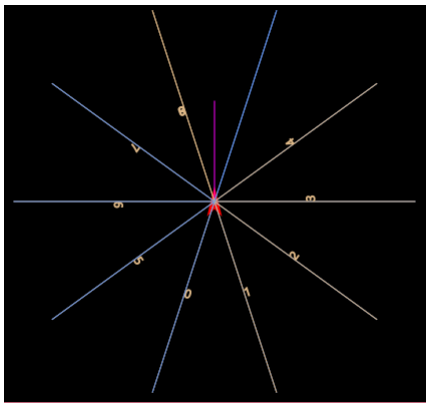
Na grafu 3.8 je znovu vidět strmý růst na začátku evoluce. Téměř se zde dosáhlo i maximální fitness (20 000). To ovšem nesvědčí o kvalitě chování neuronové sítě, které je stále neuspokojivé.

3.8 Zvýšení počtu senzorů

Jelikož je řešení snímání prostoru se senzory jednoduše škálovatelné, vyzkoušel jsem rozšířit jejich počet na 10. Velikosti výsečí, které senzory snímaly, vypa-



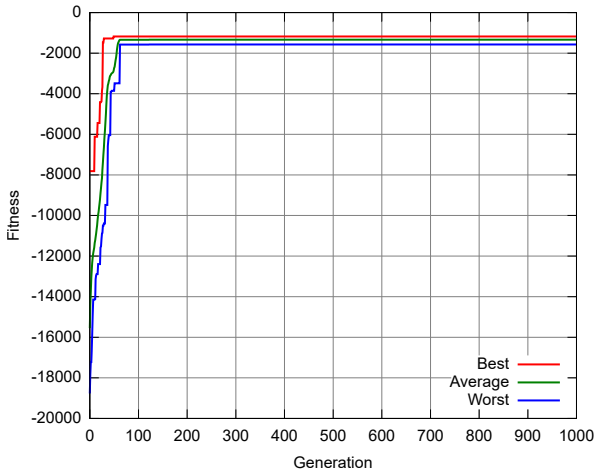
Obrázek 3.8: Vývoj populace ze sekce nesmrtelnosti 3.7



Obrázek 3.9: Obrázek deseti senzorů okolo lodi

daly dostatečně veliké na zachycení informací o blížících se asteroidech. (viz obrázek 3.9). Každá výseč byla blízko lodi velmi malá, takže se loď mohla (teoreticky) velmi dobře orientovat podle odvozených pozic. Výsledek ale doposud nebyl uspokojivý. Loď stále zůstávala uprostřed scény a snažila se přežít tím, že sestřelila všechny ji ohrožující asteroidy. Těch bylo však v testovací scéně mnoho a lodi bylo umožněno střílet pouze s malou kadencí, takže dlouho nepřežívala. Dále zde byla zvýšená penalizace kolize asteroidů.

Tato populace nedosáhla ani kladných hodnot fitness, jak je vidět na obrázku 3.10. Takto drastické snížení je důsledkem striktnějšího vyhodnocování srážek.



Obrázek 3.10: Vývoj populace ze sekce zvýšení počtu senzorů 3.8

3.9 Transformace výstupu senzorů

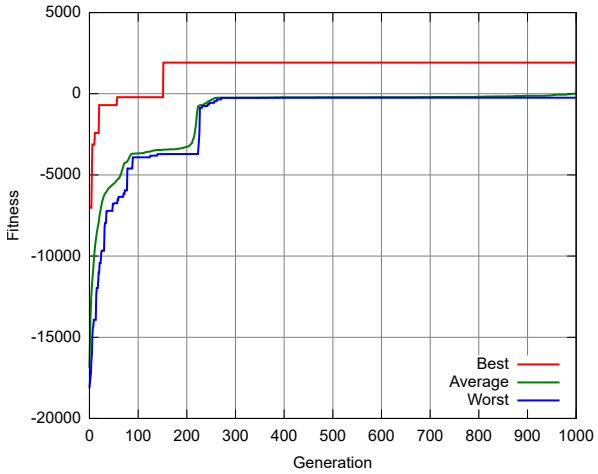
U úhlu a vzdálenosti nejbližších objektů před použitím senzorů jsem použil funkce sinus a cosinus na změnu rozsahu oboru hodnot. Podobně jsem zde transformoval vzdálenosti do rozsahu $[0, 1] \in \mathbb{R}$. Blízké objekty budou v senzorech zaznamenány hodnotou blízké 1, vzdálené hodnotou 0. Dosud byla vstupní hodnotou neuronů přímo eukleidovská vzdálenost objektu od lodi. To znamená, že loď viděla neomezeně daleko. Nyní jsem byl nucen omezit vjemy lodi, abych dokázal zaručit, že obor hodnot bude opravdu v rozsahu $[0, 1]$. Pokud je vzdálenost od objektu větší než maximální vzdálenost vnímaná senzorem, pak síť dostane na vstupu 0. Při bližší vzdálenosti se výstup senzoru počítá pomocí vzorce:

$$SENSOR(x) = \frac{-DIST(x)}{MAX\_DIST} + 1$$

3.10 Wraparound

Před testováním chování lodi s vlastnostmi, které jí přibyly transformací výstupu senzorů, jsem odstranil problém s tím, že loď neviděla za hranu obrazovky. S již zmíněným „wraparoundem“ se totiž nepočítalo při výpočtu úhlu a tudíž loď se blízko krajů scény ochudila o přibližně polovinu svých senzorů. Předpokládám, že neochota pohybu lodi pochází právě z tohoto „zastínění“ senzorů. Po nápravě tohoto problému se již při krátkém testování (10 generací) objevily známky použitelného chování. Loď se vyhýbala asteroidům a zároveň se na ně pokoušela střílet.

Při delších evolucích se však loď stále chovala jako předtím: zůstávala převážně na svém původním místě, při některých instancích se otáčela a střílela,



Obrázek 3.11: Vývoj populace ze sekce wraparound 3.10

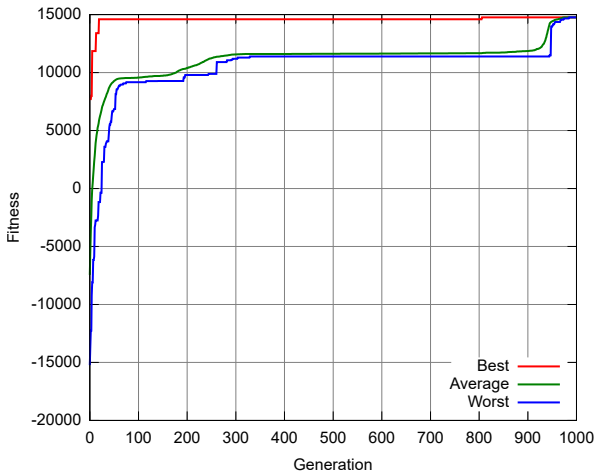
jak jen mohla. Zdá se, že lodě měly buď stále tendence se přeučovat (přestože byly herní scény zčásti náhodně generované), nebo bylo v učící scéně příliš mnoho asteroidů, čímž ze scény udělaly nevhodného kandidáta na učení. Dále mě také napadlo, že díky nesmrtelnosti si lodě nedělají velkou starost ohledně kolizí s ostatními asteroidy. Takže ani odstranění nesmrtelnosti nevedlo k lepším výsledkům.

V grafu 3.11 je vidět zlepšení oproti minulému experimentu. Lepší informace o prostředí umožňují neuronovým sítím vykonávat správné reakce.

3.11 Kompletní randomizace scén

Vyzkoušel jsem tedy generovat scény náhodně na začátku evoluce a ty pak používat. Stále se jednalo o to, že se prostředí generovala náhodně, ale nyní se randomizovaly i pozice asteroidů. Dále se také měnil počet asteroidů. V původní scéně jich bylo přibližně 20. Nyní se generovaly scény s asteroidy v rozmezí 5 až 10 asteroidů. Jednalo se tedy i o zjednodušení scény a testování, jestli je vůbec možné v prostoru s takovýmto počtem asteroidů dostatečně dlouho přežít.

Díky těmto úpravám se slibně změnilo chování lodi. Sice stále dokola rotovala, ale když se nějaký asteroid přiblížil k lodi, tak docházelo k úhybným manévrům. Jedna z vyvinutých neuronových sítí pojala tento manévr tak, že stačí vhodně rotovat okolo asteroidu a že není nutné vůbec aktivovat trysky (ve vzácných konfiguracích asteroidů ale trysky také zažehla). Toto se jevilo jako zneužívání chyby mnou implementované hry, kde se kolize mezi polygony počítaly jen mezi vrcholy mnohostěnu. Pokud by se počítaly pomocí bodů dotyku úseček, pak by toto chování nemohlo uspět. U jiné z evolucí se loď



Obrázek 3.12: Vývoj populace ze sekce kompletní randomizace scén 3.11

vyhýbala trochu lépe. Snažila se asteroidům stavit bokem a když byl příliš blízko, tak zapnula trysky, a vyhnula se z trajektorie asteroidu úplně. Obě tyto neuronové sítě měly střelbu zapnutou po celou dobu jejich života.

Moje domněnka je taková, že rotace lodí jim pomáhala více, než kdyby pouze staticky čekala na přiblížení asteroidu. Při rotaci střílely projektily pokaždé jiným směrem, a tak střelba pokrývala i asteroidy, které loď ignoruje. Druhým důvodem mohlo být získání přesnější pozice okolních objektů. Při změně senzoru, ve kterém se daný asteroid vyskytuje, se dá totiž vypočítat přesně relativní pozice asteroidu vzhledem k lodi.

Z obrázku 3.12 vidíme úspěch tohoto experimentu. Fitness jedinců se vyšplhala velmi vysoko oproti předchozím případům. Naučila se používat správně senzory a proto je pro ni jednodušší přežívat déle.

Kapitola 4

Zhodnocení výsledků

I když se zpočátku nejevily postupy práce jako perspektivní, podařilo se vyvinout lodi ovladač, který se vcelku spolehlivě dokáže vyhýbat ohrožujícím objektům. Vyvinutým ovladačům lodí se daří sestřelit většinu asteroidů. Některé evoluce se dokonce naučily používat střelbu jenom tehdy, když byl asteroid v blízkosti. Toto chování simuluje chování lidské. Hráč se snaží přežít v minovém poli asteroidů a zároveň jich sestřelit co nejvíce. Neuronová síť a hráč mají však jinou strategii střelby. Hráč může například mířit na delší vzdálenosti, kdežto vyvinutá NN má tendence střílet jen tehdy, když je asteroid blízko. Považuji práci za úspěšnou. Podařilo se napodobit část lidského uvažování – úhyb lodi před objekty s kolizní trajektorií.

Veškeré výpočty probíhaly na stroji s procesorem Intel i3-6100 3.7 GHz se dvěma jádry a čtyřmi vlákny. Paralelní výpočet nebyl použit, i když jej tento typ evoluce podporuje. Další evoluce (500 jedinců, 5000 pohybů ve scéně a 3000 generací) trvaly desítky hodin.

Při delších evolucích bylo vidět, že se neuronová síť již nemůže učit více. Například u evolucí s 5000 generacemi se většina evolucí zastavila v růstu fitness okolo 300 generace. Zdá se tedy, že nebylo nutné tolik výpočetních prostředků, což je také cenný poznatek práce.

4.1 Možné navazující práce

U této práce vidím možnosti, jak na ni navázat dalšími výzkumy. Změna rychlosti asteroidů či jiné drobnější změny ve fyzikální podstatě hry by mohly přinést zajímavé poznatky do světa inteligentních automobilů. Hra totiž do určitých mezí modeluje dopravní prostředky reálného světa. „Wraparound“ by mohl být odebrán a místo něj zavedeno tření, setrvačnost a jiné další fyzikální omezení. Hra by pak simulovala dopravu ještě více.

Pokud by navazující práce chtěla experimentovat s neuronovou sítí, která na vstupu dostává pouze obraz (tj. pixely) herní scény, pak můžeme zvolit konvoluční neuronové sítě. Dají se zde vyzkoušet schopnosti jiných modelů NN,

například dopředné neuronové sítě. Všem objektům by mohlo být přiřazeno gravitační pole, které by ovlivňovalo všechny blízké objekty, čímž bychom získali základ pro astrofyzikální modely.

Závěr

V práci jsem se zabýval použitím neuronových sítí v pozici člověka hrajícího počítačovou hru Asteroids. Součástí byla i implementace klonu zmíněné hry. Díky mé vlastní implementaci hry bylo jednoduché do ní vkládat různé druhy umělé inteligence. Odlišné druhy řízení lodi ve hře jsou podporovány a může je kdokoli přidat pomocí vlastního potomka třídy `AbstractUpdateController`. Součástí práce je také implementace algoritmu diferenciální evoluce a vlastní plně rekuretní neuronové sítě. Oba tyto moduly jsou vytvořeny univerzálně tak, aby bylo možné jejich použití v případných dalších projektech.

Realizace vlastní rekurentní neuronové sítě bylo úspěšné. Neuronové sítě fungují správně a dle očekávání. Diferenciální evoluce se ukázala jako vhodná metoda pro učení ovladače lodi, který je založený na plně rekurentních neuronových sítích, v počítačové hře Asteroids.

Vyvinuté ovladače lodí byly rovněž úspěšné. Dokáží se efektivně vyhnout asteroidům a střílet po nich. Chování těchto ovladačů je vesměs stejné: rotují do jedné strany a střílí. Pokud je ohrožuje nějaký objekt, tak se mu vyhnou kolmo na směr jeho trajektorie.

Na základě experimentů jsem zjistil, jak moc je evoluce neuronových sítí ovlivněna učícím prostředím. Správné transformace vstupních informací jsou zde velmi důležité, protože některé z nich neumožní neuronovým sítím používat správné akce. Nespojitost v úhlu nejbližších objektů či vysoké rozsahy oborů hodnot funkcí se jeví jako nevhodné vlastnosti vstupů neuronové sítě. Konkrétně pro tuto hru bylo výhodné nevěnovat se příliš informacím o nejbližších objektech, ale dívat se na vzdálenost objektů z různých stran lodi.

Literatura

[1] Asteroids Designer Ed Logg Honored With Pioneer Award. *Wired*, listopad 2011, [cit. 2017-04-28]. Dostupné z: https://www.wired.com/2011/11/ed-logg-pioneer-award/

[2] Atari - About Atari, SA. [online], [cit. 2017-05-14]. Dostupné z: https://www.atari.com/corporate/about-atari-sa

[3] All About Asteroids - The Atari Times. [online], [cit. 2017-05-12]. Dostupné z: http://www.ataritimes.com/index.php?ArticleIDX=174

[4] Wolf, M. J.: *The video game explosion: a history from PONG to Playstation and beyond*. ABC-CLIO, 2008.

[5] JavaFX API - Overview (JavaFX 8). [online], [cit. 2017-05-13]. Dostupné z: https://docs.oracle.com/javase/8/javafx/api/toc.htm

[6] Bureš, M.: Implementace počítačové hry využívající prvky umělé inteligence. 2012.

[7] Keßler, D.: *Parallel Agent Models for a Realtime Strategy Game*. Dizertační práce, University of Kassel, 2008.

[8] The AI Games. [online], [cit. 2017-05-11]. Dostupné z: http://theaigames.com/

[9] Pogamut Cup. [online], [cit. 2017-05-12]. Dostupné z: http://www.pogamutcup.com/

[10] Silver, D.; Huang, A.; Maddison, C. J.; aj.: Mastering the game of Go with deep neural networks and tree search. *Nature*, ročník 529, č. 7587, 2016: s. 484–489.

[11] Russel, S.; Norvig, P.: *Artificial Inteligence - A Modern Approach*. Prentice Hall, třetí vydání, ISBN 978-0-13-604259-4.

[12] Nareyek, A.: AI in computer games. *Queue*, ročník 1, č. 10, 2004: str. 58.

[13] Orkin, J.: Three states and a plan: the AI of FEAR. In *Game Developers Conference*, ročník 2006, 2006, str. 4.

[14] Valmari, A.: Stubborn sets for reduced state space generation. *Advances in Petri Nets 1990*, 1991: s. 491–515.

[15] GitHub - neuroph/neuroph: Java Neural Framework Neuroph. [online], [cit. 2017-05-11]. Dostupné z: `https://github.com/neuroph/neuroph`

[16] DeepLearning4J. [online], [cit. 2017-05-11]. Dostupné z: `https://deeplearning4j.org/`

[17] Druga, R.; Grim, M.; Dubový, P.: *Anatomie centrálního nervového systému*. Galén, první vydání, ISBN 978-80-246-1895-1.

[18] De Jong, K. A.; Spears, W. M.: Using genetic algorithms to solve NP-complete problems. In *ICGA*, 1989, s. 124–132.

[19] Storn, R.; Price, K.: Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, ročník 11, č. 4, 1997: s. 341–359.

[20] Williams, R. J.; Zipser, D.: A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, ročník 1, č. 2, 1989: s. 270–280.

[21] Karlik, B.; Olgac, A. V.: Performance analysis of various activation functions in generalized MLP architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, ročník 1, č. 4, 2011: s. 111–122.

Příloha **A**

Seznam použitých zkratek

AI Artificial inteligence - umělá inteligence

DE Diferenciální evoluce

NN Neural network - neuronová síť

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF