VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POSILOVANÉ UČENÍ PRO HRANÍ ROBOTICKÉHO FOTBALU

REINFORCEMENT LEARNING FOR ROBOTIC SOCCER PLAYING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ADAM BRYCHTA**
AUTHOR

VEDOUCÍ PRÁCE **doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Brychta Adam**
Program: Informační technologie
Název: **Posilované učení pro hraní robotického fotbalu**
Reinforcement Learning for Robotic Soccer Playing
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s metodami hraní her s využitím posilovaného učení.
2. Prostudujte podmínky a omezení prostředí definované v rámci soutěže RoboCup 3D Soccer Simulation League a Google Football Benchmark.
3. Implementujte herní strategii založenou na posilovaném učení a nezbytná rozhraní pro zařazení do soutěže.
4. Vyhodnoťte vytvořený systém srovnáním s ostatními účastníky.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle doporučení vedoucího

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

Vedoucí práce: **Smrž Pavel, doc. RNDr., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 29. dubna 2020

Abstrakt

Tato práce se zabývá vytvořením agenta pro hraní robotického fotbalu. V práci používám metodu hlubokého Q-učení, která využívá hlubokou neuronovou síť. Praktická část práce se zabývá implementací agenta pro posilované učení. Cílem agenta je vybrat nejlepší akci pro daný stav prostředí. Trénování agenta probíhá v různých scénářích situací na hřišti. Výsledek této práce ukazuje přístup k hraní fotbalu pomocí strojového učení.

Abstract

The aim of this thesis is to create a reinforcement learning agent that is able to play a soccer. I'm working with the deep Q-learning algorithm, which uses deep neural network. The practical part of this work is about implementing the agent for reinforcement learning. The goal of the agent is to choose the best action possible for a given situation. The agent is being trained in a variety of scenarios. The result of this thesis shows an approach to control soccer player using machine learning.

Klíčová slova

posilované učení, posilované učení pro hraní fotbalu, Q-učení, hluboké Q-učení, dvojité hluboké Q-učení

Keywords

reinforcement learning, reinforcement learning for soccer, Q-learning, deep Q-learning, double deep Q-learning

Citace

BRYCHTA, Adam. *Posilované učení pro hraní robotického fotbalu*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. RNDr. Pavel Smrž, Ph.D.

Posilované učení pro hraní robotického fotbalu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. RNDr. Pavla Smrže, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Adam Brychta
1. června 2020

Poděkování

Děkuji vedoucímu bakalářské práce dor. RNDr. Pavlu Smržovi, Ph. D. za odbornou pomoc.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá vytvořením agenta pro hraní fotbalu pomocí posilovaného učení. Agent vytvořený v této práci se snaží najít optimální strategii ve zvoleném scénáři situace na hřišti. Tuto úlohu plní pomocí algoritmu hlubokého Q-učení a dvojitého hlubokého Q-učení. Lze si jednoduše vytvořit vlastní scénář, který je potřeba vyřešit a upravovat ho dle potřeby. Výsledkem trénování agenta na zvoleném scénáři je natrénovaná neuronová síť, která poskytuje pro určitý stav prostředí možnou akci. Agenta lze používat v libovolném prostředí, pokud z tohoto prostředí lze získat stav prostředí a je možné v tomto prostředí ovládat entitu hráče. Lze také nastavit jakou množinu akcí má agent k dispozici.

Posilované učení je jedním ze tří základních typů strojového učení. Na rozdíl od učení s učitelem nepotřebuje posilované učení žádná tréninková data nebo příklady činnosti. V případě fotbalu nepotřebuje žádné již odehrané hry, kdy by bylo potřeba zapojit do učení člověka nebo počítačového hráče. Při posilovaném učení agent prozkoumává prostředí ve kterém se nachází a snaží se vyřešit zadaný problém. Zkouší vykonávat různé akce a sleduje stav prostředí, které je ovlivněno těmito akcemi. Za akce dostává odměnu nebo penalizaci, což je ohodnocení toho, jak prospěšná daná akce byla pro vyřešení zadaného problému. Cílem agenta je se dostat k nejvyšší možné odměně.

Pro natrénování agenta se musí odehrát velké množství her. To se skutečnými robotickými fotbalisty není možné, a proto se využívá simulátor, který poskytuje prostředí pro hraní zápasů bez rizika poškození skutečných robotických fotbalistů a poskytuje možnost odehrát velké množství zápasů v krátkém čase. V této práci je pro natrénování agenta použit simulátor a rovněž je použit pro srovnání natrénovaného agenta s ostatními fotbalovými týmy.

Existuje několik mezinárodních soutěží, které využívají robotické fotbalisty k hraní fotbalu. Tyto soutěže mnohdy používají reálné roboty na reálném hřišti, kteří jsou naprogramováni účastníky soutěže. Těchto soutěží se účastní týmy z různých světových univerzit, které mají v soutěži dlouholeté zkušenosti a jejich programování robotických hráčů fotbalu je na vysoké úrovni. Několik týmů se pokouší strategii svých fotbalistů vylepšit pomocí posilovaného učení.

V této práci je agent použit pro vytvoření optimální posloupnosti akcí při útoku na bránu. Úlohou agenta je vstřelit gól, a pokud se mu to podaří, získává nejvyšší odměnu. Byly navrženy tři funkce odměny, které odměňují agenta za úspěchy a neúspěchy v plnění této úlohy.

V první kapitole je popsána motivace této práce. Druhá kapitola probírá rozbor teorie strojového učení, posilovaného učení, umělých neuronových sítí a algoritmu pro posilované učení. Třetí kapitola se zabývá použitými nástroji, popisuje simulátory fotbalu a pravidla

fotbalu, kterými se řídí. Dále se zabývá návrhem řešení agenta pro hraní fotbalu s využitím posilovaného učení pro vylepšení útoku na bránu a popisuje postupy použité při řešení daného problému. Čtvrtá kapitola popisuje implementaci navrženého agenta, jeho umělé neuronové sítě a implementaci algoritmu posilovaného učení. Pátá kapitola popisuje experimenty prováděné s agentem, jeho trénování a diskutuje dosažené výsledky z experimentů. V závěru je vyhodnocen výstup této práce.

Kapitola 2

Rozbor řešené problematiky

V této kapitole budou vysvětleny a rozebrány pojmy a metody, které jsou v práci použity. Nejprve bude popsáno strojové učení a jaké je jeho základní rozdělení. Poté bude popsáno samotné posilované učení, jeho základní prvky a Markovův rozhodovací proces. Následně budou vysvětleny algoritmy Q-učení a hlubokého Q-učení. Poté bude rozebrána základní problematika umělých neuronových sítí a bude popsán jejich stavební prvek umělý neuron. Bude popsána aktivační funkce umělého neuronu, která je použita v této práci. U umělých neuronových sítí bude vysvětleno, co je hluboká neuronová síť a konvoluční neuronová síť.

2.1 Strojové učení

Strojové učení je nejpoužívanější oblast umělé inteligence [11]. Jde o získávání zkušeností, upravování nabytých zkušeností a o schopnost se přizpůsobovat řešenému problému. Ve strojovém učení se stroj učí a přizpůsobuje se zadanému problému. Právě pro tuto vlastnost se strojové učení používá v mnoha různých oborech a pro mnoho různých úloh. Používá se například pro autonomní řízení automobilů, rozpoznávání tváří v bezpečnostních systémech, převodu textu na řeč (a naopak) a mnoho dalších využití [9].

Strojové učení se dělí podle typu učení na tři kategorie [19]. První typ je učení s učitelem, druhý typ je posilované učení a třetí typ je učení bez učitele. Každá z těchto kategorií používá rozdílnou metodu učení a je vhodná pro jiný typ úloh. Rozdělení strojového učení do kategorií je na obrázku 2.1. Pro hraní her je vhodnou metodou posilované učení, protože není omezeno množstvím trénovacích dat [23].

Učení s učitelem

Učení s učitelem využívá pro učení omezené množství trénovacích dat, které jsou již rozřazeny do skupin podle vzorů (tříd) [20]. Tato označená data používá k předpovídání označení nově získaných dat. Trénovací data jsou příklady činnosti stroje [26]. Je to dvojice, která obsahuje vstup a očekávaný výstup. Očekávaný výstup poskytuje algoritmu učení s učitelem zpětnou vazbu. Algoritmus učení s učitelem se naučí generovat výstup podle vstupu z množiny trénovacích dat. Čím více je poskytnuto trénovacích dat, tím pravděpodobněji vygeneruje pro nová data očekávaný výstup.

Výstup algoritmu učení s učitelem lze ověřit pomocí ověřovacích dat [8]. Tímto testováním se zjistí jak je naučený model účinný a spolehlivý. Pro testování výkonnosti modelu je vhodné používat testovací data na kterých nebyl model učen.

Obrázek 2.1: Rozdělení strojového učení na tři kategorie.

Algoritmy učení s učitelem se dělí na algoritmy pro klasifikaci a regresi [6]. Algoritmy pro klasifikaci řadí data do určeného počtu skupin, zatímco algoritmy pro regresi predikují spojité hodnoty.

Nevýhodou učení s učitelem je potřeba mít dostatečný vzorek trénovacích dat [6]. Navíc při učení s učitelem může dojít k podučení nebo přeučení [11]. Při přeučení se po maximalizaci výkonnosti sytému začne výkonnost zhoršovat. U dat na které je model naučen je jeho výkonost maximální, ale pokud na vstup přijdou neznámá data, která se nenacházejí v trénovacích datech, je výkonnost modelu nízká. Došlo tedy k přeučení a model nedosahuje dostatečné generalizace [19]. Při podučení dochází naopak k příliš velké generalizaci a model zařazuje data do skupin kam nepatří.

Algoritmus učení bez učitele

Algoritmus učení bez učitele se snaží identifikovat ve vstupních datech sobě podobná data (vzor) [6]. Algoritmus nezná při učení očekávaná výstupní data, proto jde o učení bez učitele. Nemá tedy žádného externího pozorovatele, který by mu dal příklad očekávaného vstupu. Algoritmy učení bez učitele pracují na principu shlukování [8]. Dle vstupních vzorků se provádí klasifikace, najde se vzor a následně se data třídí do shluků. U roztříděných dat ve shluku nelze nijak zkontrolovat zda jsou zařazena do správného shluku. Příkladem učení bez učitele může být komprese dat a shlukování dat [19].

2.2 Posilované učení

Úkolem posilovaného učení je naučit agenta řešit zadaný komplexní problém [27]. V posilovaném učení se agent snaží naučit, které akce vedou k nejvyšším odměnám [23]. Tuto nejvyšší odměnu získá pokud se naučí vybírat optimální nebo téměř optimální akce. Snaží se tedy vybírat akce tak, aby maximalizoval konečnou kumulativní odměnu. Agent dopředu neví, jaké akce vedou k dosažení nejvyšší odměny a musí prozkoumat za jaké akce dostane nejvyšší odměnu. Agent prozkoumává své prostředí a metodou pokus a omyl zjišťuje nejlepší akce. Vždy po akci, kterou agent ovlivní prostředí, je agentovi poskytnuta zpětná vazba (odměna). Podle této zpětné vazby agent zjistí, jak vhodné jsou jím zvolené akce. Tato odměna může být i záporná, což je penalizace. Tímto způsobem může agent objevit nové strategie a postupy v hraní hry. Vztah mezi agentem a prostředím je na obrázku 2.2.

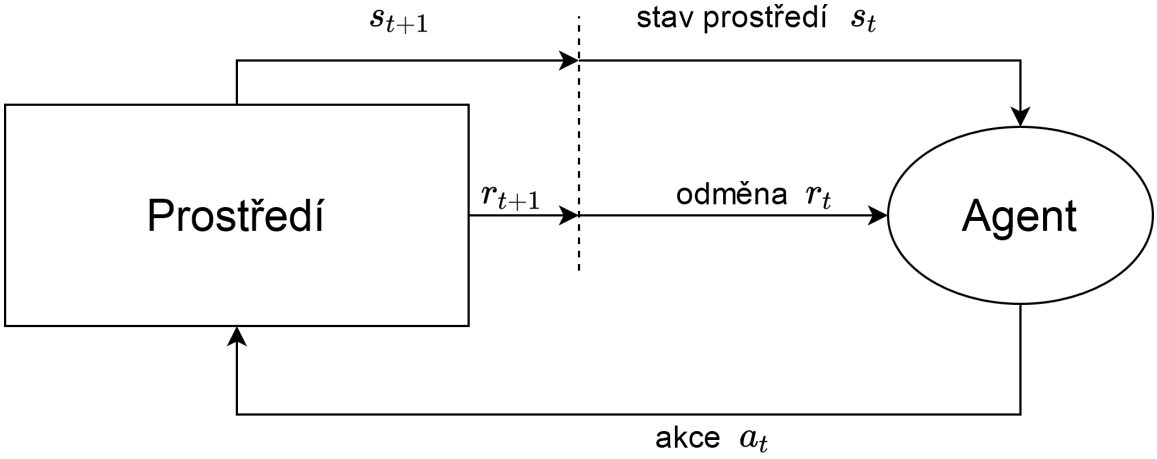
V posilovaném učení jsou čtyři základní elementy [27]:

Agent je učící se entita [14]. Agent vykonává akce, kterými ovlivňuje stav prostředí, sleduje stav prostředí a získává odměny.

Akce jsou konečná množina činností, které agent v určeném prostředí může vykonávat. Pomocí akcí agent mění stav prostředí.

Odměna je reprezentace vhodnosti akce. Agent dostává za své akce odměnu, která je reprezentována číslem. Při posilovaném učení se agent snaží svými akcemi maximalizovat obdrženou kumulativní odměnu.

Stav prostředí jsou informace o stavu řešeného problému. Tyto informace jsou předávány agentovi. Rozsah informací předávaných agentovi je určen prostředím.



Obrázek 2.2: Interakce agenta s prostředím. Akce a_t symbolizuje akci v čase t, odměna r_t symbolizuje odměnu v čase t a stav s_t symbolizuje stav prostředí v čase t. Agent provádí určenou akci v čase t a v reakci na akci, prostředí posílá agentovi odměnu a stav v čase $t + 1$.

Agent, který se nachází v prostředí musí umět vykonávat akce s jejichž pomocí ovlivňuje toto prostředí. Algoritmus posilovaného učení se učí pouze z interakcí s tímto prostředím. Agent se učí v epizodách, což je sekvence stavů, akcí a odměn od začátku agentovy činnosti

do konečného stavu jeho činnosti [23]. Je třeba, aby agent měl daný nějaký cíl, který musí splnit vykonáváním akcí a ovlivňováním prostředí. Jde tedy o metodu zaměřenou na splnění cíle [25].

2.2.1 Srovnání posilovaného učení a učení s učitelem

Hlavní rozdíl mezi posilovaným učením a učením s učitelem je ten, že učení s učitelem obsahuje externího pozorovatele [14]. Tento externí pozorovatel poskytne agentovi příklad činnosti, která řeší zadaný problém. Pokud je prostředí ve kterém se agent učí rozsáhlé, může být problém získat dostatečně reprezentativní vzorek všech příkladů akcí ze všech možných stavů. Tento problém posilované učení nemá, protože agent posilovaného učení prozkoumává výhodnost akcí metodou pokus omyl. Agent posilovaného učení tedy není omezen vzorkem všech příkladů akcí ze všech možných stavů.

2.2.2 Základní definice v posilovaném učení

V posilovaném učení jsou tři základní definice [19]. Jde o strategii, kumulativní odměnu a model.

Strategie definuje chování agenta v prostředí a jedná se o dvojici stavu prostředí přiřazenému k akci agenta: $\pi : S \rightarrow A$ [6]. Strategie se značí π, S je stav a A je akce. Strategie definuje akci, která bude zvolena v daném stavu. Hodnota strategie $V^{\pi}(s_t)$ je očekávaná kumulativní odměna, která bude obdržena pokud agent bude následovat strategii a bude začínat ve stavu s_t [6]. Agent hledá akci, která mu přinese nejvyšší odměnu v dlouhodobém měřítku. Pokud by vybíral pouze akce, které by mu přinesly nejvyšší okamžitou odměnu, nikdy by se nenaučil vyřešit problém s nejvyšší možnou odměnou.

Kumulativní odměna je suma odměn z počátečního stavu až do konce epizody. První odměna je z výchozího stavu a do této sumy se přičítají odměny za akce až do konce epizody. Suma odměn v daném čase určuje, jak je výhodný daný stav z dlouhodobého hlediska [23].

Model popisuje prostředí ve kterém se agent nachází. Model není pro posilované učení nutný, ale dokáže zlepšit jeho výsledky. Pro stav a akci dokáže model předpovědět jaký bude výstupní stav a odměna. Pomocí modelu lze plánovat agentovy akce a doporučovat agentovi následující akce. Model obsahuje konečnou množinu stavů a konečnou množinu akcí agenta [15].

2.3 Markovův rozhodovací proces

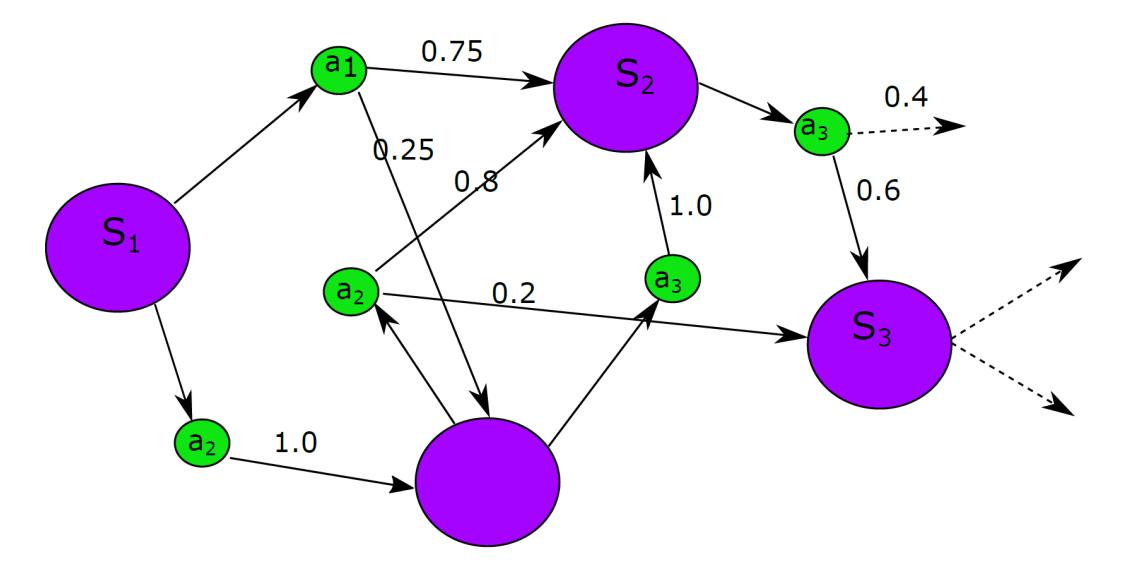
Markovův rozhodovací proces je matematický rámec pro řešení problému pomocí posilovaného učení [23]. V tomto procesu agent ve stavu s může vybrat akci a, která je dostupná v tomto stavu. Proces se poté přesune do nového stavu. Tento proces je zobrazen na obrázku 2.3

Při posilovaném učení dělá agent rozhodnutí, které jsou založené na aktuálním stavu prostředí. Pokud odměna a pravděpodobnost přechodu do dalšího stavu záleží pouze na aktuálním stavu a poslední akci a ne na žádném z předchozích stavů, má tento problém Markovskou vlastnost [29]. Pokud prostředí má tuto vlastnost, je popsáno jako Markovův

rozhodovací proces. V Markovském rozhodovacím procesu lze zvolit optimální akci pouze podle informací poskytnutých z aktuálního stavu [29].

Markovův rozhodovací proces je popsán jako pětice (S, A, P, R, γ).

1. S je konečná množina všech možných stavů do kterých se může agent dostat.
2. A je konečná množina všech možných akcí, které může agent vykonávat.
3. $P(s, a, s')$ je funkce přechodu. Tato funkce představuje pravděpodobnost přechodu do stavu s', při provedení akce a ve stavu s .
4. $R(s, a, s')$ je funkce odměny, kterou agent dostane při vykonání akce a ve stavu s a přechodu do stavu s'.
5. γ je diskontní faktor. Jeho hodnota je v intervalu [0,1].



Obrázek 2.3: Markovův rozhodovací proces. Stavy jsou značeny S, akce a a pravděpodobnosti přechodu jsou hodnoty u šipek. Stavy jsou reprezentovány fialovými kruhy, akce jsou zelené kruhy a hodnoty šipek jsou pravděpodobnosti přechodů. Obrázek převzat z [25]

Prostředí ve kterém se agent učí musí splňovat Markovskou vlastnost [18]. Odměna tedy musí záležet pouze na současném stavu. Agent vybírá akce v každém kroku, které ovlivní následující stav, ve kterém se bude agent nacházet podle funkce přechodu stavu.

Cílem agenta je najít optimální strategii π, aby byla maximalizována kumulativní odměna [7]:

$$\pi : S \rightarrow A \tag{2.1}$$

která maximalizuje budoucí očekávanou odměnu v čase t.

$$V_t = \sum_{k=1}^{\infty} \gamma^k r(s_t, s_{t-1}), \tag{2.2}$$

kde γ je diskontní faktor a měl by být v intervalu [0,1]. Tento diskontní faktor snižuje váhu následujících odměn. Pokud agent provede akci, která mu přinese vysokou okamžitou

odměnu, může tuto akci upřednostňovat před akcemi, které by mu přinesly vyšší odměnu z dlouhodobého hlediska. Protože je cílem agenta nasbírat co nejvyšší kumulativní odměnu, je třeba nastavit γ na hodnotu blízkou jedničce [14]. Pokud je hodnota γ blízká nule, agent bude upřednostňovat okamžité odměny a nebude se snažit získat maximální konečnou odměnu.

2.4 Metody posilovaného učení pro hraní her

Existuje mnoho typů her a pro každý typ hry může být vhodný jiný algoritmus posilovaného učení [23]. Pro jednodušší hry s omezeným stavovým prostorem lze využít Q-učení nebo Monte-Carlo. Takové hry jsou například Dáma a šachy. Složitější hry jako například počítačové strategické hry, akční hry a další, mají velmi komplexní prostředí a je třeba zvolit metody posilovaného učení, které si s dokáží poradit s velkým stavovým prostorem. Tyto metody používají hluboké neuronové sítě, které se naučí řešit složitý problém úpravou vah svých umělých neuronů. Do těchto algoritmů patří hluboké Q-učení, které lze použít i pro hraní akčních her (například hra *Doom*) [23].

2.4.1 Q-učení

Velmi hojně používaný algoritmus posilovaného učení je Q-učení [19]. V tomto algoritmu se sleduje, jak výhodná akce je v určitém stavu. Toto ohodnocení akce se nazývá Q-hodnota. Zajímá se tedy pouze o páry stavu a akce.

Algoritmus používá Q-tabulku, což je datová struktura sloužící k vypočítání maximální očekávané kumulativní odměny (Q-hodnoty) za akci v určitém stavu [19]. Tato tabulka uchovává Q-hodnoty pro všechny akce ve všech stavech. Podle Q-tabulky, dokáže agent volit akce, které ho dovedou k maximální kumulativní odměně.

V algoritmu Q-učení se používá funkce $Q(s, a)$, jejímž výstupem je Q-hodnota pro stav s a akci a. Optimalizace Q-hodnoty probíhá podle rovnice:

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma maxQ(s', a') - Q(s, a)), \tag{2.3}$$

kde s je stav, a je akce, γ je diskontní faktor a r je odměna. Hodnota $maxQ(s', a')$ je nejvyšší Q-hodnota v následujícím stavu, ve kterém se agent nachází po provedení akce a ve stavu s. Hodnota α je koeficient učení, který stanovuje jak moc se maximální Q-hodnota nadcházejícího stavu promítne do právě počítané Q-hodnoty. Koeficient učení nabývá hodnoty v intervalu (0, 1].

Algoritmus musí nejprve prozkoumávat akce a jejich Q-hodnoty [29]. Kdyby neprozkoumal dostatečně možné akce, mohl by zvolit strategii, která nevede k nejvyšší kumulativní odměně. Z tohoto důvodu během učení vybírá náhodné akce s pravděpodobností danou hodnotou ϵ. Vybráním náhodné akce může objevit novou a lepší strategii. Hodnota ϵ je v intervalu (0, 1] (pokud nabývá hodnoty 1, vybírá pouze náhodné akce). Během učení se hodnota ϵ snižuje.

Algoritmus nejprve inicializuje všechny Q-hodnoty u akcí na náhodné hodnoty [19]. Poté vybere akci s nejvyšší Q-hodnotou nebo náhodnou akci s pravděpodobností ϵ. Pokud jsou akce ohodnoceny stejnou Q-hodnotou, je z nich vybrána náhodná akce. Agent vykoná akci a ocitne se v dalším stavu. Q-hodnota provedené akce v předchozím stavu se upraví

podle vzorce 2.3. Po optimalizaci této Q-hodnoty algoritmus znovu vybírá akci a vše znovu opakuje až do ukončovacího stavu. Algoritmus Q-učení:

Algorithm 1: Algoritmus *Q-učení*

Parametry algoritmu: $epsilon > 0$;
Inicializace $Q(s, a)$, pro všechny stavy $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, na libovolné hodnoty;
foreach *epizoda* **do**
  Inicializace stavu s;
  foreach *krok v epizodě* **do**
    Vyber akci a ze stavu s, kdy je pravděpodobnost výběru náhodné akce *epsilon* nebo akci s nejvyšší hodnotou Q; Provedení akce a, pozorování odměny r a následujícího stavu s';
    $Q(s, a) = Q(s, a) + (r + \gamma maxQ(s', a') - Q(s, a)),$;
    $s = s'$;
  end
end

2.4.2 Hluboké Q-učení

Hluboké Q-učení je metoda posilovaného učení používající hlubokou neuronovou síť (umělá neuronová síť je popsána v 2.6) na jejímž vstupu je stav prostředí a na jejím výstupu je Q-hodnota pro každou možnou akci agenta. Hluboké Q-učení se používá pro řešení problémů v prostředích s vysokým počtem možných stavů [23]. Tento vysoký počet stavů nedovoluje použít tabulku s Q-hodnotami a nahrazuje ji hlubokou neuronovou sítí (nazývanou Q-síť). Tato Q-síť je optimalizována optimalizátorem, aby se snížil výstup ztrátové funkce. Tímto optimalizováním se síť učí.

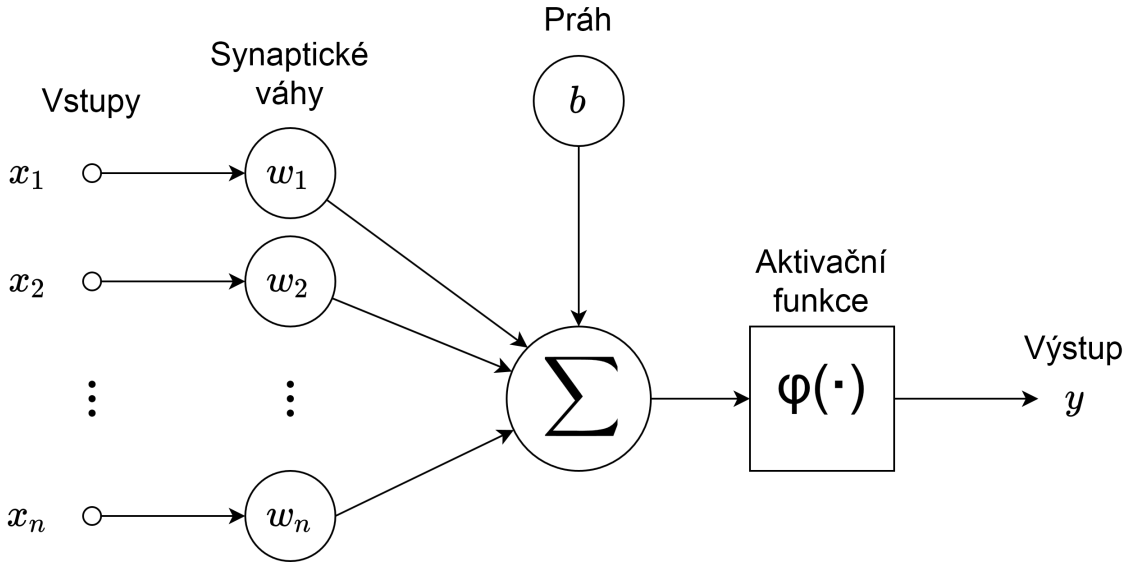
Existuje vylepšení algoritmu pomocí paměti zkušeností [23]. Q-síť je učena během kroků v epizodě s pomocí paměti zkušeností. Paměť zkušeností slouží pro uložení a přípravu trénovacích dat pro učení neuronové sítě. Každý záznam v paměti zkušeností obsahuje stav prostředí, vykonanou akci, odměnu a stav prostředí po vykonání akce. Při každém učení neuronové sítě se vybere z paměti zkušeností náhodný vzorek záznamů. Tento vzorek je nazýván dávka a má předem určený počet prvků. Dávka je použita pro vytvoření trénovacích dat pro učení neuronové sítě. Tato trénovací data jsou poté použita optimalizátorem pro snížení výstupu ztrátové funkce.

Metoda hlubokého Q-učení nemusí vždy vést k nalezení optimální strategie [23]. Tento problém lze vyřešit použitím dvojitého hlubokého Q-učení. Metoda dvojitého hlubokého Q-učení je modifikace hlubokého Q-učení, která řeší problém nalezení optimální strategie [19]. V této metodě posilovaného učení jsou použity dvě hluboké neuronové sítě. První síť slouží k vybrání akce pro druhou síť, která vypočítá Q-hodnotu pro tuto akci.

2.5 Umělý neuron

Umělý neuron je inspirován neurony v mozku [23]. Je to základní prvek umělých neuronových sítí. Každý umělý neuron má několik vstupů a výstup. Umělé neurony upravují svoje vstupní signály a předávají upravený signál dalším neuronům. Každý vstup umělého neuronu má svůj váhový koeficient, který určuje jak moc je výstup neuronu závislý na daném vstupu a jak je tento vstup důležitý [9]. Pokud je vážená suma vstupů ovlivněných jejich váhovými koeficienty větší než práh, je výstup umělého neuronu určen aktivační funkcí a

váženou sumou vstupů ovlivněných jejich váhovými koeficienty. Model umělého neuronu je na obrázku 2.4.



Obrázek 2.4: Model umělého neuronu. Každý vstup x_i je vynásoben svoji váhou w_i. Z těchto vstupů se vytvoří suma. Pokud je součet vetší než prahová hodnota, je na výstupu hodnota daná aktivační funkcí.

Výstup umělého neuronu popisuje rovnice:

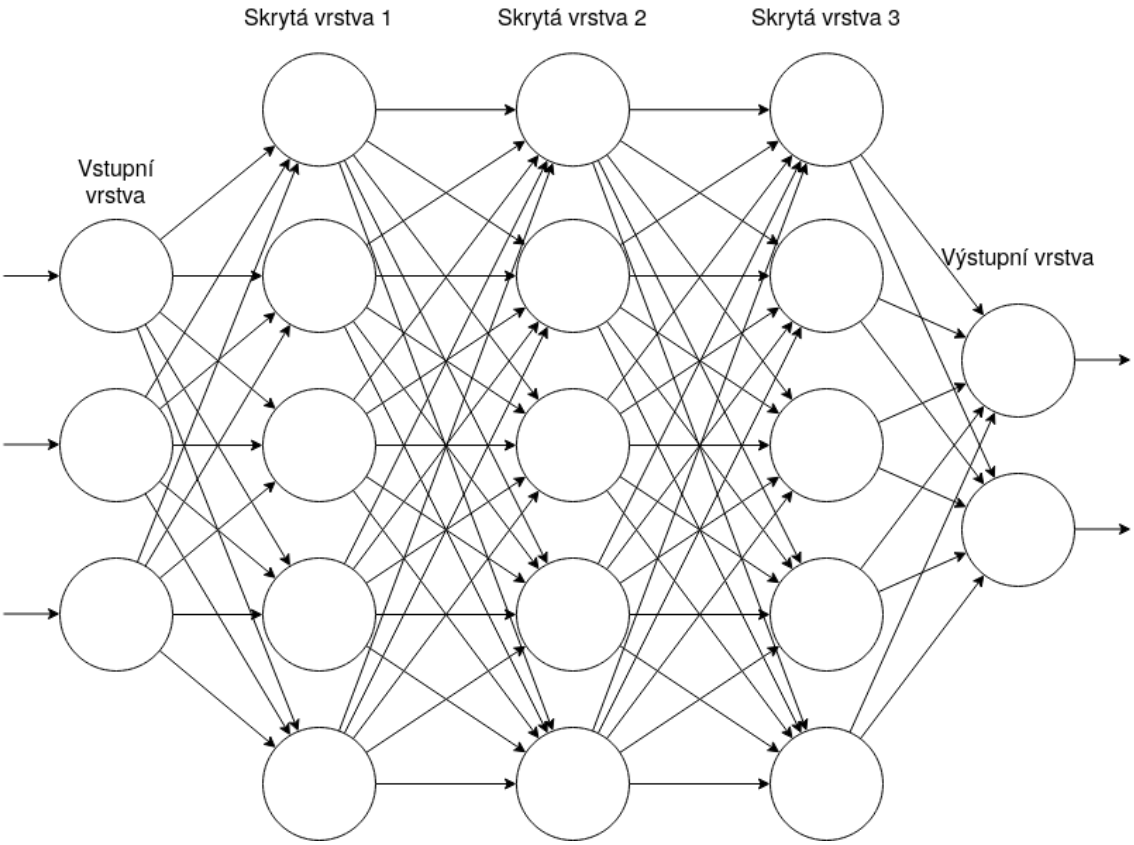
$$y = \phi(b + \sum_{i=0}^{n} w_i x_i), \tag{2.4}$$

kde výstup umělého neuronu y je určen aktivační funkcí ϕ. Práh neboli *bias* je b. Vstupy jsou značeny x a jejich váhové koeficienty jsou w.

2.6 Umělá neuronová síť

Umělá neuronová síť je model v oboru strojového učení [9]. Tento model je vzdáleně inspirován neuronovou sítí v mozcích savců [21]. Skládá se z umělých neuronů, které mají vzájemně propojené vstupy a výstupy a přenášejí mezi sebou signály. V umělé neuronové síti se upravují váhy vstupů umělých neuronů a tím se umělá neuronová síť učí a přizpůsobuje se zadanému problému [19]. Vrstvy se rozdělují na vstupní vrstvy, skryté vrstvy a výstupní vrstvy. Pokud v umělé neuronové síti mají neurony každé vrstvy propojeny výstupy s vstupy neuronů následující vrstvy, tak se tato neuronová síť nazývá dopředná neuronová síť [20]. Pokud je každý neuron v jedné vrstvě propojen s každým neuronem v následující vrstvě, jedná se o plně propojenou síť. Pokud síť má alespoň jednu skrytou vrstvu, nazývá se hluboká neuronová síť. Ukázka hluboké neuronové sítě je na obrázku 2.5.

Při učení sítě je třeba počítat chybu sítě [12]. Tato chyba sítě je počítána pomocí ztrátové funkce (anglicky *loss function*). Čím více výstup z umělé neuronové sítě podává nepřesnější výsledky, tím vyšší hodnoty dává ztrátová funkce. Trénování neuronové sítě tedy probíhá snižováním hodnoty ze ztrátové funkce. Ztrátová funkce střední kvadratické chyby (anglicky *Mean squared error*) patří mezi nejpoužívanější ztrátové funkce [28]. Výstup této funkce je



Obrázek 2.5: Model hluboké neuronově sítě. Síť má 3 externí vstupy, každý z nich vede do neuronu ve vstupní vrstvě. Následují tři skryté vrstvy, každá z nich má pět neuronů. Poslední je výstupní vrstva obsahující dva neurony. Síť má dva výstupy. Každý neuron má výstupy vedoucí do všech neuronů následující vrstvy, jde tedy o plně propojenou síť.

definován rovnicí:

$$MSE = \frac{1}{n}\sum_{t=1}^{n}(y_t - o_t)^2, \tag{2.5}$$

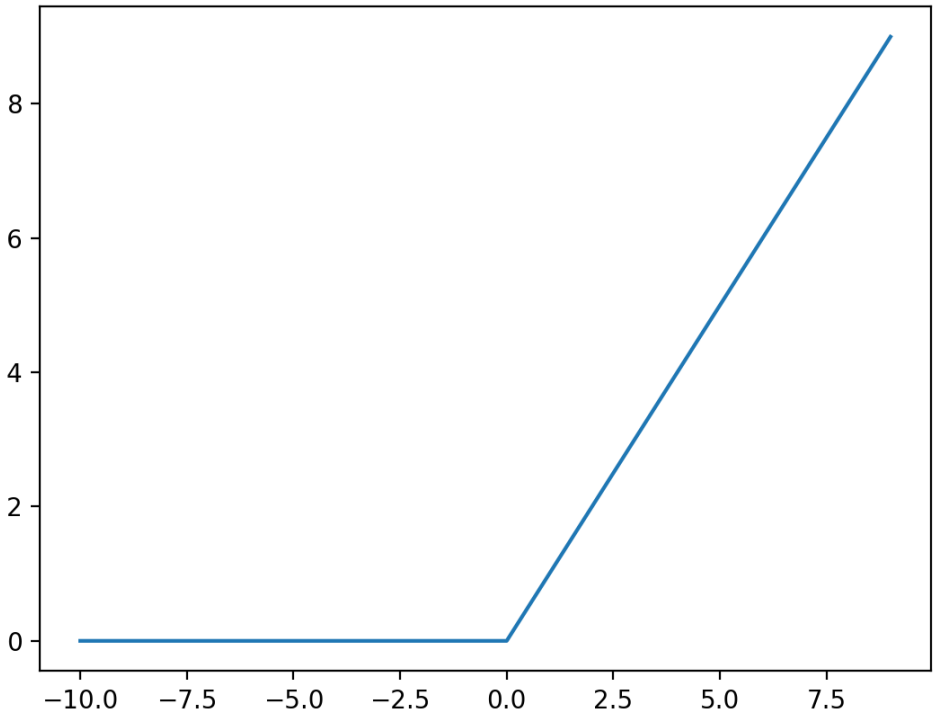
kde y_t je skutečná hodnota a o_t je požadovaná hodnota.

Aktivační funkce ReLU

Jedná se o aktivační funkci, která se používá v hlubokých neuronových sítích [23]. Výstup funkce je maximum mezi nulou a vstupní hodnotou. Pokud je vstupní hodnota záporná, výstup funkce je nula. Tato funkce je velice matematicky jednoduchá. Výstup funkce je na obrázku 2.6.

$$R(z) = \begin{cases} z & z > 0 \\ 0 & z <= 0 \end{cases}, \tag{2.6}$$

kde z je vstupní hodnota.

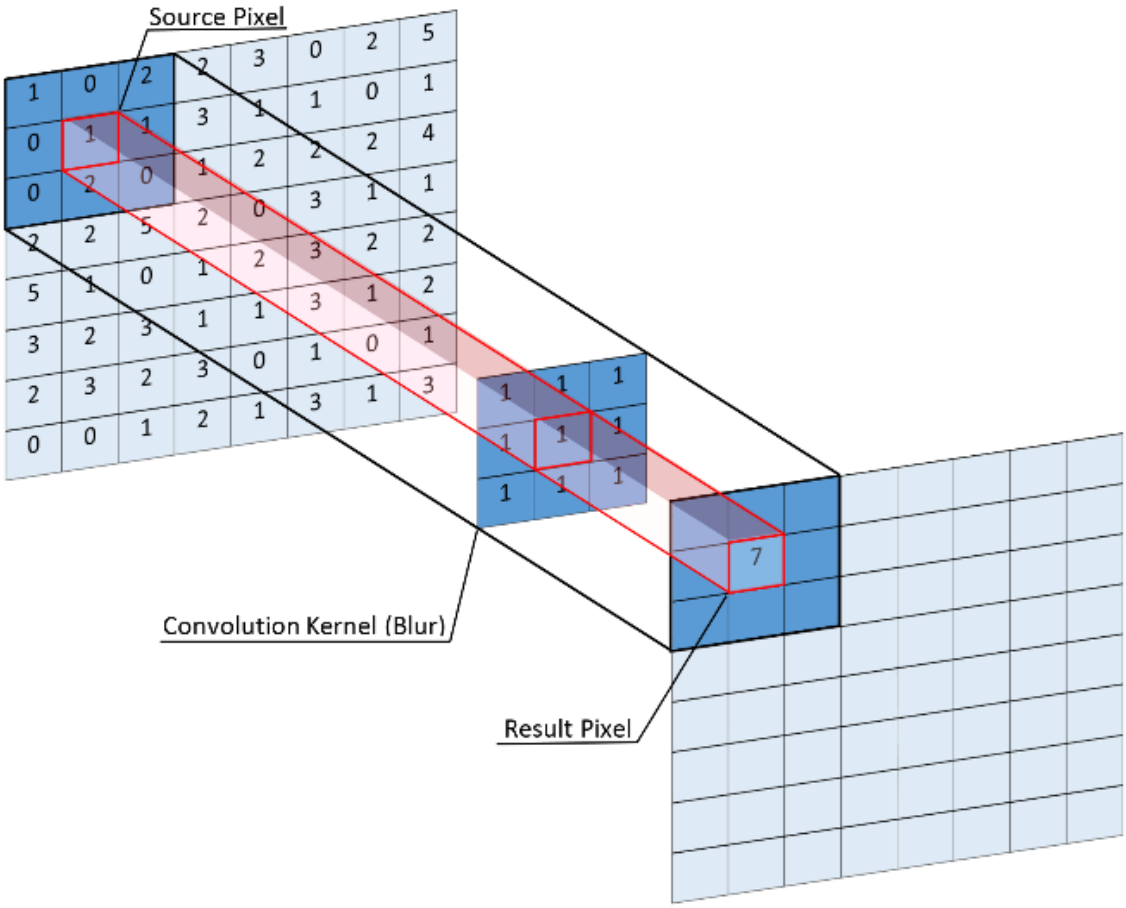


Obrázek 2.6: Průběh funkce ReLU. Pro záporný vstup vrací funkce nulový výstup. Pro kladný vstup funkce vrací $R(z) = max(0, z)$

2.6.1 Konvoluční neuronová síť

Konvoluční neuronové sítě se používají pro zpracování obrazových a zvukových dat [9]. Konvoluční síť obsahuje alespoň jednu konvoluční vrstvu. Konvoluční vrstva je základní stavební blok konvoluční sítě [19]. Tato vrstva provádí operaci konvoluce, což je lineární matematická operace.

Operace konvoluce vytváří ze několika vstupních hodnot jednu výstupní hodnotu. Pro vstupní hodnotu operace konvoluce extrahuje příznaky [9]. Tyto příznaky jsou zredukované informace ze vstupu. Při této operaci je na vstup použit filtr, který má určenou výšku, šířku a hloubku. Tímto filtrem se postupně prochází vstupní obrázek daným krokem a pixely obrázku jsou násobeny hodnotami ve filtru, které je překrývají. Krok (*stride*) je počet pixelů o který se filtr vždy posune. Poté jsou tyto získané hodnoty sečteny, je získána hodnota a dochází k redukci na menší vstup. Tento postup je zobrazen na obrázku 2.7.



Obrázek 2.7: Operace konvoluce. Hodnoty ze zdrojových pixelů (*source pixel*) jsou vynásobeny hodnotami z konvolučního jádra (*convolution kernel*), které je překrývají. Obrázek převzat z [1].

Kapitola 3

Návrh aplikace

V této kapitole jsou popsány nástroje použité při implementaci aplikace a návrh aplikace. Bude stručně popsána knihovna *NumPy*, která je použita pro práci s reprezentací stavu prostředí a knihovna *TensorFlow* pro práci s umělými neuronovými sítěmi. Je vysvětleno, jakým způsobem pracuje RoboCup Soccer Simulator a jak se k němu připojují klienti. Dále je vysvětleno jak klienti ovládají svého agenta a jaká pravidla fotbalu musí dodržovat. Bude popsán simulátor *Google Research Football* a budou vysvětleny jednotlivé komponenty simulátoru. Pro *Google Research Football* budou vysvětleny akce, které agent může vykonávat, jaké reprezentace prostředí agent může získávat a jaké scénáře může agent trénovat.

3.1 Použité nástroje

Agent je implementován v jazyku Python 3.6. Pro trénování a vývoj agenta je použito vlastní prostředí *Simple Football Environment* a simulátor *Google Research Football*. Pro práci s neuronovými sítěmi je použita knihovna *Keras* a *TensorFlow*. Pro práci se získanými informacemi z prostředí je použita knihovna *NumPy*.

3.1.1 Knihovna *TensorFlow*

TensorFlow je soubor knihoven, nástrojů a zdrojů sloužící jako platforma pro strojové učení. *TensorFlow* je používán výzkumníky v oblasti strojového učení k vývoji nových algoritmů z oblasti strojového učení [13]. Poskytuje aplikační rozhraní pro vývoj a učení modelů založených na strojovém učení. Aplikační rozhraní je dostatečně komplexní a lze díky němu řídit všechny stránky strojového učení [11]. K trénování a vytváření modelů založených na strojovém učení slouží aplikační rozhraní *tf.keras*, které je součástí *TensorFlow* a je to varianta aplikačního rozhraní *Keras* upravená přímo pro *TensorFlow*.

3.1.2 Keras

Keras je knihovna pro programovací jazyk Python. Tato knihovna poskytuje aplikační rozhraní pro tvorbu hlubokých neuronových sítí s vysokou mírou abstrakce [11]. Klade se důraz na jednoduchost a rychlost proto-typování modelů konvolučních a rekurentních sítí. Podporuje výpočty na mikroprocesoru i na grafické kartě. Knihovna *Keras* je součástí *TensorFlow*. Důležité vlastnosti knihovny *Keras* jsou uživatelská přívětivost, modularita a rozšiřitelnost. Díky modularitě lze přidávat vlastní modely nebo je různě kombinovat [13].

Základem knihovny *Keras* je objekt `Model`. Tento objekt spravuje vrstvy neuronové sítě. Nejjednodušší typ sítě je sekvenční model neuronové sítě. Do sekvenčního modelu jsou vrstvy neuronové sítě přidávány lineárně. Pro práci s neuronovými sítěmi poskytuje modul `NN`, který se používá pro definování komplexních neurálních sítí. Tento modul poskytuje bohaté aplikační prostředí pro aplikace používající umělé neuronové sítě a umožňuje vytvářet a spojovat vrstvy umělých neuronových sítí [13]. Další modul, který je poskytován je modul `Optim`, který definuje různé implementace optimalizačních algoritmů, které jsou používány pro vytváření neurálních sítí. V této práci je z tohoto modulu použit optimalizátor Adam.

3.1.3 Optimalizátor Adam

Optimalizátor Adam je efektivní metoda pro stochastickou optimalizaci. Jedná se o optimalizaci založenou na gradientu (gradient based). *Gradient descent* metody jsou nejoblíbenější algoritmy pro optimalizaci neurálních sítí [16]. Tyto metody se používají pro hraní her [22].

Optimalizátor je všestranný algoritmus, který se přizpůsobuje úlohám ve strojovém učení. Existují tři varianty optimalizátoru: Adam, Adamax a Nadam [24]. Název algoritmu Adam vyplývá z jeho anglického názvu *adaptive moment estimation*. Je používán pro optimalizaci neuronové sítě při hraní her a nahradil některé starší algoritmy (například RMSProp) [12].

3.1.4 Knihovna *NumPy*

NumPy je knihovna pro programovací jazyk Python. Tato knihovna je používána pro matematické výpočetní operace. Knihovna je využívána hlavně ve vědeckém sektoru a byla vytvořena v roce 2006 [11]. Knihovnu NumPy používá mnoho vědeckých projektů [13].

Základ knihovny je multidimenzionální objekt typu pole homogenních dat. Prvky tohoto objektu musí být tedy stejného datového typu. Toto multidimenzionální pole se nazývá `ndarray`. Tento objekt má pevnou velikost, která se určuje při inicializaci objektu a pokud je třeba změnit velikost objektu `ndarray`, musí být objekt znovu zcela vytvořen a originální objekt je zahozen. Objekt `ndarray` je vysoce optimalizován a operace nad ním jsou velmi efektivní.

3.2 Pravidla hry

Hráči jsou rozděleni do dvou týmů. Každý tým má 11 hráčů. Jeden hráč z týmu je brankář a ostatní jsou hráči v poli. Cílem hráčů je dostat míč do branky druhého týmu. Brankář jako jediný může míč chytit. Hraje se na dva poločasy a při každém poločase si týmy vymění strany hřiště. Oba poločasy začínají výkopem. První poločas provádí výkop hráč družstva, které je zvoleno například losem. Druhý poločas provádí výkop jejich soupeř.

Výkop je proveden jedním hráčem ze středu hřiště. Pokud jeden z týmů vstřelí branku, hráči se přesunou do svých výchozích pozic a výkop je proveden jejich soupeřícím týmem. Na dodržování pravidel dohlíží rozhodčí. Tento rozhodčí dává pokyn k výkopu a kontroluje zda nedošlo k autu nebo k ofsajdu. Aut je stav kdy se míč dostane mimo hrací plochu. Tehdy dojde k přerušení hry a novému výkopu. Rozhodčí kontroluje zda nedochází k ofsajdu, který nastává, když je hráč na polovině hrací plochy soupeře a je blíže k brankové čáře než míč a předposlední soupeřův obránce. Toto pravidlo se uplatňuje pouze, když se míče dotkne některý z jeho spoluhráčů.

3.3 *RoboCup* simulátor

RoboCup soccer simulator poskytuje prostředí pro hraní robotického fotbalu. Události v simulátoru probíhají v reálném čase. Tento simulátor se používá pro výzkum a mezinárodní soutěže [10]. *RoboCup soccer simulator* poskytuje realistické prostředí, které je co nejvíce podobné *RoboCup* soutěži se skutečnými roboty a simuluje senzory, jejich omezení a komplexní vnímání světa. Na obrázku 3.1 je zobrazeno hřiště v prostředí *RoboCup*.

Obrázek 3.1: Simulátor RoboCup 3D. Po hřišti se pohybují roboti Nao. Obrázek převzat z [5].

RoboCup Soccer Simulator obsahuje dvě základní komponenty [4]. První komponenta je *RoboCup Soccer Server*. Tento server umožňuje klientům se připojit k prostředí. Těmto klientům je poté umožněno spolu hrát zápas. Druhá komponenta je monitor. Monitor převádí dění v prostředí na grafické zobrazení. Umožňuje rozhodčímu sledovat zápas a vyhodnocovat co se děje na hrací ploše. Umožňuje rozhodčímu ovlivňovat dění hry a interagovat s herním prostředím. Mezi tyto interakce patří pokyn k výkopu, přerušení hry nebo její ukončení. V průběhu hry je tedy připojen jeden monitor, který funguje jako rozhodčí a dohlíží na dodržování pravidel.

Robot v *RoboCup Soccer Server* má tři typy senzorů [4]. První typ jsou vizuální senzory. Tyto vizuální senzory poskytují hráčovi vizuální informace o části hřiště, kam je právě hráč otočen. Vizuální informace jsou odeslány serverem ke klientovi každých 150 milisekund [4]. Vizuální informace z vizuálních senzorů jsou použity pro zjištění druhu objektu, vzdálenost, směr a rychlost objektu vzhledem k hráči. Druhý typ senzoru jsou zvukové senzory. Zvukové senzory se používají pro příkaz `hear`, který vrací zvukové informace z okolí robota. Zvukový projev je možné použít pomocí příkazu `say`. Tímto způsobem spolu mohou hráči komunikovat. Třetí typ senzorů jsou senzory tlaku. Přesnou pozici hráče nelze od serveru jednoduše získat. Jeho pozice se určuje podle senzorů, ale informace ze senzorů jsou pouze z pohledu hráče. To komplikuje určení polohy. Pro zjednodušení určení polohy jsou používány praporky na krajích hrací plochy.

Server komunikuje s klienty v časových cyklech [4]. Každý cyklus má 100 milisekund. Mezi cykly server naslouchá požadavkům od klientů. Na konci každého cyklu server aktu-

alizuje stav simulace. Každý klient ovládá vždy pouze jednoho hráče. Pokud klient vyšle více požadavků během jednoho cyklu, je serverem zpracován pouze jeho poslední přijatý požadavek. Zbytek požadavků od klienta je zahozen. Pokud server nedostane od klienta žádný požadavek, klientovi možnost poslat serveru požadavek propadá.

Klient se k RoboCup Soccer Server připojuje přes UDP/IP sokety [4]. Proto je možné klienta vyvíjet v jakémkoliv programovacím jazyce, který podporuje sokety. Pomocí soketů klient posílá serveru svoje požadavky, aby jím ovládaný hráč provedl požadovanou akci. Server dostane od klienta požadavek, podle něj ovládá hráče a aktualizuje prostředí. Po určitém intervalu jsou klientům poslány informace ze senzorů a stav prostředí. Klienti spolu nesmí komunikovat přímo přes sokety. Musí to udělat přes server a své senzory. K tomuto účelu slouží protokol `say` a `hear`.

3.4 Google Research Football

Google Research Football je simulátor fotbalu vytvořený firmou Google. Simuluje fotbalový zápas, kde lze ovládat hráče. Klient může ovládat jednoho nebo více hráčů svého týmu. Prostředí modeluje trojrozměrný fotbalový zápas a poskytuje fyzikální model [17]. Simulátor je naprogramován v C++ a jeho zdrojový kód je dostupný na Github [1] [17]. Simulátor se skládá z *Football engine* a samotného prostředí *Google Research Football* [2]. *Football engine* vytváří komplexní simulaci fotbalového zápasu. Toto prostředí lze spouštět s grafickým prostředím, kde lze vidět na obrazovce aktuální stav hry nebo bez grafického prostředí.

Simulátor poskytuje přizpůsobitelné prostředí a hráč může hrát různé scénáře [17]. Je například možné upravit počet hráčů, pozměnit ukončovací podmínky hry nebo upravovat pozici hráčů a míče. Z tohoto simulátoru lze získat informace jako pozici míče, pozici hráče, pozice ostatních hráčů, stav hry a obrazovou reprezentaci prostředí. Simulátor poskytuje scénáře, kde je je hráč vystaven častým situacím, které nastávají v průběhu fotbalového zápasu [2].

Při ovládání jednoho hráče se tým dělí na neaktivní hráče a aktivního hráče [17]. Jako aktivní hráč je považován hráč s míčem nebo nejblíže k míči ze všech spoluhráčů, kteří jsou považováni za neaktivní hráče. Tito neaktivní hráči jsou kontrolováni počítačem. Chovají se podle svojí nastavené role a pronásledují míč, čekají v obraném pásmu nebo postupují s aktivním hráčem.

Základ prostředí tvoří *Football Engine*, což je simulátor fotbalu naprogramovaný v jazyku C++ [3]. Obsahuje základní pravidla fotbalu jako ofsajdy a auty. Pro měření délky hry se používají snímky [17]. Hra podle standardních pravidel trvá 3000 těchto snímků.

Množina akcí v Google Research Football

V simulátoru Google Research Football může hráč fotbalu vykonávat celkem 19 akcí [17]. Jakoukoliv akci může vykonat kdykoliv, ale není zaručeno, že akce bude mít nějaký účinek. Pokud například aktivuje sprint dvakrát za sebou, druhá aktivace sprintu nebude mít žádný účinek.

Množina akcí, které může hráč vykonávat:

Top je akce pohybu směrem nahoru (sever).

Bottom je akce pohybu směrem dolů (jih).

[1] https://github.com/google-research/football

Left je akce pohybu doleva (západ).

Right je akce pohybu směrem doprava (východ).

Top-Left je akce pohybu směrem vlevo nahoru (severozápad).

Top-Right je akce pohybu směrem vpravo nahoru (severovýchod).

Bottom-Left je akce pohybu směrem vlevo nahoru (severozápad).

Bottom-Right je akce pohybu směrem vlevo nahoru (severozápad).

Short Pass je akce, která vykoná přihrávku spoluhráči na krátkou vzdálenost.

High Pass je akce, která vykoná vysokou přihrávku spoluhráči. Touto akcí lze přihrávat spoluhráči přes protihráče.

Long Pass je akce, která vykoná přihrávku spoluhráči na dlouhou vzdálenost.

Shot je akce střely na bránu soupeře.

Sliding je akce sklouznutí po trávě.

Dribble je akce driblování.

Stop-Dribble je akce zastavení driblování.

Sprint je akce aktivující sprint.

Stop-Moving je zastavení pohybu hráče.

Stop-Sprint je akce přerušení sprintování.

Do-Nothing nevykoná žádnou akci.

3.5 Vlastní prostředí simulovaného fotbalu

Pro účel vývoje a trénování agenta bylo navrženo a implementováno prostředí *Simple Football Environment*. Toto prostředí poskytuje objekty, základní pravidla a akce simulovaného fotbalu.

Prostředí *Simple Football Environment* obsahuje tyto typy objektů:

- Agent (aktivní hráč) je jeden z hráčů týmu, který je ovládán agentem. Aktivní hráč je vždy ten člen týmu, který drží nebo je nejblíže míči. Tento hráč je ovládán agentem s využitím posilovaného učení.
- Hráči levého týmu. Z tohoto týmu je vybírán aktivní hráč, který je ovládán agentem.
- Hráči pravého týmu, kteří se snaží získat míč a znemožnit, tak levému týmu vstřelit gól.
- Aktivní protihráč se snaží získat míč nebo ho sebrat agentovi. Jedná se o hráče pravého týmu, který je nejblíže míči. Agent nesmí dopustit, aby mu tento hráč znemožnil přístup k míči.

Prostředí objekty udržuje v datovém typu pole. Při vytváření scénářů situací na hřišti je vytvořeno prostředí a poté jsou do něj umístěny tyto objekty. Pro každý objekt musí být specifikován řádek a sloupec, kam má být v hřišti umístěn. U hráčů je třeba specifikovat navíc jejich tým.

Každý objekt v prostředí *Simple Football Environment* obsahuje:

- Řádek počáteční pozice, který značí umístění objektu v prostředí na určeném řádku na začátku hry.
- Sloupec počáteční pozice, který značí umístění objektu v prostředí na určeném sloupci na začátku hry.
- Současný řádek pozice v prostředí.
- Současný sloupec pozice v prostředí.
- Velikost jednoho bodu v pixelech.
- Barva, kterou objekt nabývá při vykreslování.
- Velikost objektu v pixelech.
- Povolení, zda se má objekt vykreslovat.
- Objekt z knihovny `TkInter` použitý pro vykreslení objektu.

Velikost v pixelech, barva a objekt z knihovny `TkInter` jsou použity pouze v případě když je povoleno vykreslení prostředí na obrazovku. Vykreslení zpomaluje proces učení, proto je vhodné povolit vykreslení pouze pro demonstrační účel.

Akce v prostředí *Simple Football Environment*

Agent může vykonávat stejné aktivity jako v prostředích *Google Football Environment* a RoboCup. V každém prostředí mají akce jinou číselnou hodnotu a výstup z agenta se musí namapovat na správné hodnoty akcí. Mapování číselných hodnot akcí má na starosti rozhraní mezi agentem a prostředím.

Agent může v prostředí *Simple Football Environment* vykonávat tyto akce:

- Akce zahálení. Při vykonání této akce agent nijak neovlivní prostředí.
- Čtyři akce pohybu do čtyřech základních světových stran.
- Čtyři akce pohybu do vedlejších světových stran.
- Přihrávka nejbližšímu spoluhráči.
- Přihrávka nejbližšímu spoluhráči v daném směru.
- Akce střely na bránu. Při této akci musí být hráč v pozici, ve které má šanci vstřelit branku, jinak ztrácí míč.

Těmito akcemi agent ovlivňuje prostředí. Může se tedy pohybovat, sebrat míč, přihrát sebraný míč spoluhráči a vystřelit na branku soupeře. Pokud agent nevlastní míč a vykoná akci pracující s míčem (přihrávka nebo střela na bránu), neovlivní prostředí a může mu být udělena penalizace. Pokud agent vykoná akci střely v dostřelu a ve správném úhlu k brance protihráče, je vstřelen gól a prostředí ukončuje jednu epizodu (hru) a vrací odměnu s hodnotou 1.

Informace poskytované prostředím

Prostředí poskytuje informace o poloze a stavu objektů na hřišti. Tyto informace jsou převedeny do snímku prostředí (popsán v sekci 3.7.1) a tento snímek je poskytnut agentovi. Prostředí poskytuje agentovi i další užitečné informace. Informuje agenta o rozměrech herní plochy a tuto informaci využívá agentova funkce odměny. Dále poskytuje informaci, který tým drží míč. To je důležité pro funkci odměny.

V tomto prostředí je možné sledovat kolize mezi objekty. Například mezi hráči, hráčem a míčem, hráčem a koncem hřiště. Díky tomu, lze odměňovat agenta za různé vykonané úkony (například za získání míče, dosáhnutí nějaké pozice a odměna za sebrání míče) a i agenta penalizovat (například za přenechání míče protihráči nebo za pohyb proti konci hřiště).

3.6 Návrh agenta

Implementovaný agent interaguje s prostředím, naviguje se v něm a pozoruje jeho stav. Průběh učení agenta je shrnut do třech kroků [23]:

1. Agent získá vstup z prostředí, tak že pozoruje aktuální stav prostředí a bere si snímek prostředí jako vstup.
2. Agent se na základě vstupu (reprezentace prostředí) rozhodne jakou vykoná akci.
3. Agent provede akci a od prostředí dostane jeho stav po vykonání této akce. Podle tohoto stavu je agentovi udělena odměna z funkce odměny. Odměna reprezentuje jak byla právě provedená akce dobrá či špatná. Tímto se agent učí jaké akce jsou s kladnou odměnou a jaké akce jsou s penalizací nebo nižší odměnou. Agent se tedy učí jaké akce mu přinesou nejvyšší kumulativní odměnu, takže jeho učení je založeno na pozorování odměny.

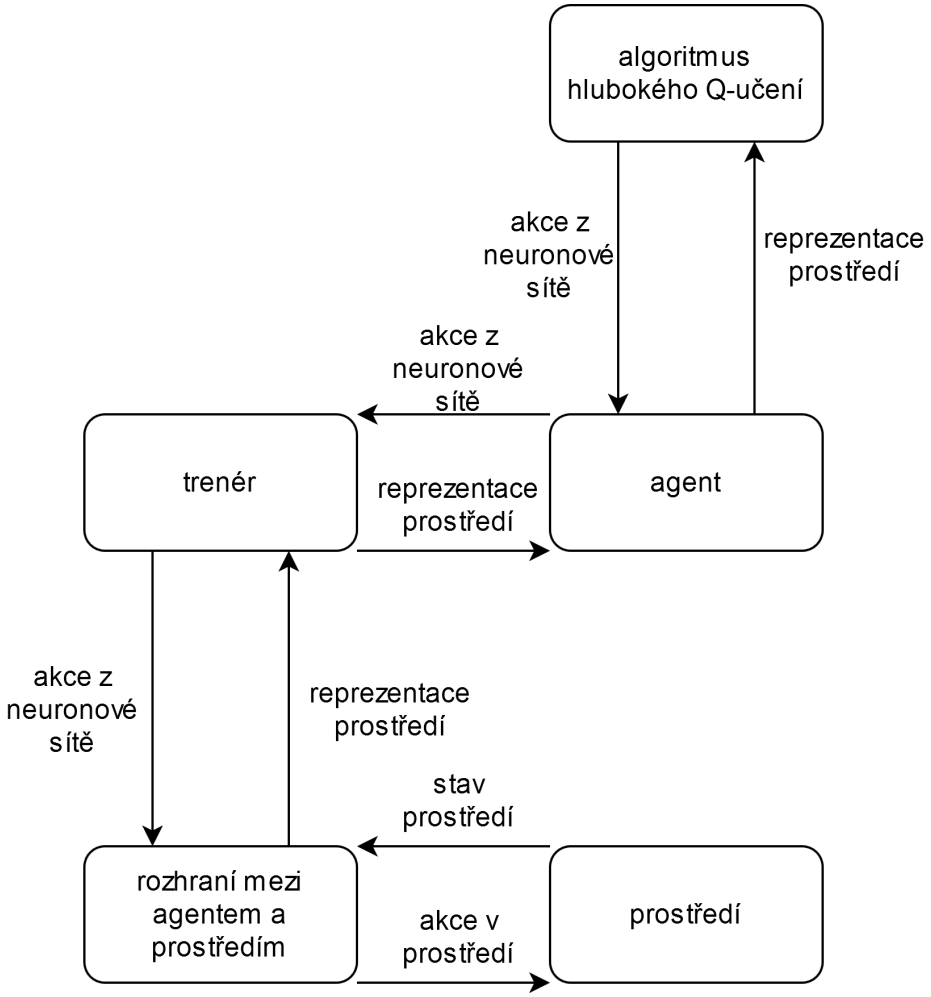
Agent se učí v prostředí, které mu musí poskytovat potřebné funkce a informace. Pro každý použitý simulátor je nutné implementovat rozhraní mezi tímto prostředím a agentem. Agent ovlivňuje stav prostředí pomocí funkce `step`, které předává číslo akce. Rozhraní musí implementovat tyto funkce:

1. Funkce `step`, která provede jeden časový krok v prostředí. Agent předá této funkci číslo akce, kterou chce vykonat. Krok je určitý časový úsek v prostředí, kdy může agent vykonat nějakou akci. Návratová hodnota této funkce je stav prostředí po vykonání akce.
2. Funkce `reset` uvede prostředí do výchozího stavu.
3. Funkce `close` ukončí prostředí.

Cílem je implementovat rozhraní pro zvolená prostředí, spojení mezi těmito rozhraními a agentem, samotného agenta a algoritmus posilovaného učení. Tyto funkční bloky a informace, které si mezi sebou předávají jsou na obrázku 3.2.

3.6.1 Rozhraní mezi agentem a prostředím

Každé prostředí může poskytovat informace o svém stavu v jiném formátu. Z tohoto důvodu je potřeba vytvořit pro každé použité prostředí rozhraní, které poskytuje agentovi snímek prostředí ve formátu, který agent očekává.



Obrázek 3.2: Funkční bloky aplikace. Mezi jednotlivými bloky aplikace je předáván stav prostředí a akce. Tyto informace jsou v jednotlivých blocích převáděný na formáty, které jsou potřeba v jejich sousedních blocích.

Dalším důležitým úkolem rozhraní je převést index zvolené akce z agentovi umělé neuronové sítě do hodnoty akcí v použitém prostředí. Agentova umělá neuronová síť má počet výstupů shodný s počtem akcí, které může agent vykonávat. Z hodnot výstupů této neuronové sítě je vybrán index výstupu s nejvyšším ohodnocením. Tento index je zvolená akce, která má být provedena. Index však nekoresponduje s číselnou hodnotou akce v prostředí a musí se převést. Po převodu je získána jedna nebo více číselných hodnot reprezentujících akce ve zvoleném prostředí.

3.6.2 Trenér

Trenér je spojení mezi rozhraním a agentem. Stará se o provedení zadaného počtu epizod a kroků v epizodě. Každý krok v epizodě předá agentovi snímek prostředí a získá od agenta index výstupu z neuronové sítě s nejvyšším ohodnocením. Tento index reprezentující číslo akce je poslán do rozhraní, kde je přeložen na jednu nebo více akcí v prostředí. Po provedení

těchto akcí je získána z rozhraní odměna, snímek prostředí a stav hry (zda probíhá nebo ne). Tyto informace jsou předány agentovi, který je využije pro učení své neuronové sítě.

3.7 Průběh učení agenta

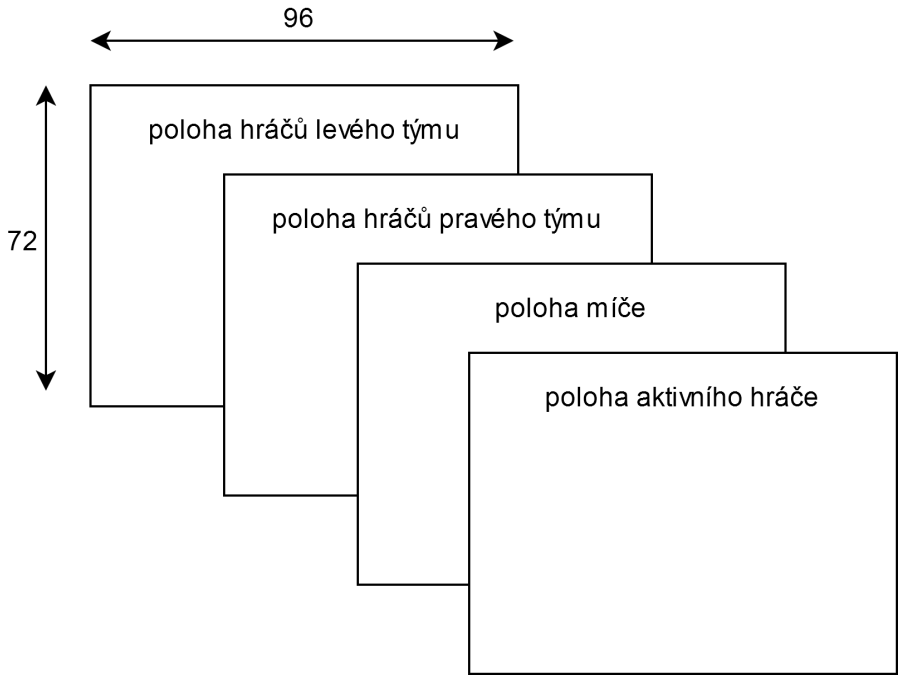
Po spuštění aplikace je vytvořeno rozhraní mezi agentem a prostředím, ve kterém je agent trénován. Je třeba definovat, jaké akce má agent k dispozici a podle toho převádět akce agenta do akcí v prostředí. Číselné hodnoty z agentovi neuronové sítě neodpovídají hodnotám akcí v prostředí a musí být přeloženy na hodnoty odpovídající akcím v prostředí, které agent chce vykonat. Dále je vytvořeno samotné prostředí s objekty (hráči a míč). U každého vytvářeného objektu je určena poloha.

Agent vytvoří neuronovou síť pro ohodnocování akcí. Lze načíst i dříve natrénovanou síť a použít ji. Algoritmus hlubokého Q-učení probíhá v epizodách. Každá epizoda začíná uvedením prostředí do výchozího stavu, což je stav kdy prostředí není ovlivněno žádnou akcí agenta. Hráči a míč jsou na začátku epizody umístěni na výchozí pozice v hřišti. Epizoda končí při splnění ukončovacích podmínek epizody. Ukončovací podmínka je splnění zadaného úkolu (získání míče nebo vstřelení gólu) nebo dosažení zadaného maximálního počtu kroků epizody.

Během každého kroku epizody agent získává reprezentaci prostředí (popsáno v 3.7.1), což je jeden snímek prostředí. Algoritmus hlubokého Q-učení vybere s pravděpodobností danou ε náhodnou akci nebo akci z výstupu neuronové sítě. Pokud vybere akci z výstupu neuronové sítě, nejdříve zpracuje reprezentaci prostředí konvoluční neuronovou sítí a takto zpracovaný snímek prostředí je vstupem do Q-sítě, jejíž výstup jsou ohodnocené akce (každá akce je ohodnocena Q-hodnotou). Z takto ohodnocených akcí je vybrána akce s nejvyšším ohodnocením (Q-hodnotou). Vybraná akce je poslána do prostředí (funkcí `step`), kde se provede a agent získává nový stav prostředí, který nastal po provedení akce a odměnu za provedení akce. Následně si agent do paměti zkušeností uloží stav prostředí před provedením akce, stav prostředí po provedení akce, samotnou akci a odměnu, kterou za tuto akci získal. Pokud nastává konec epizody nebo uplynul zadaný počet kroků, agent provede učení neuronové sítě (Q-sítě) pomocí trénovacích dat. Tato trénovací data jsou získána z dávky, která je náhodně vybrána ze vzorků z paměti zkušeností. Krok epizody agenta končí a pokud epizoda není ukončena, tak agent v následujícím kroku epizody získává stav prostředí po provedení akce v tomto kroku epizody.

3.7.1 Reprezentace stavu prostředí

Agent volí akci podle snímku stavu prostředí a pro reprezentaci tohoto stavu byla navržena reprezentace stavu znázorněná na obrázku 3.3. Snímek prostředí má nejčastěji rozměr $72 \times 96 \times 4$, ale může být nastaven i jiný rozměr. Hodnota pixelu je 0 nebo 255. Hodnota 0 značí nepřítomnost čehokoliv. Hodnota 255 značí přítomnost objektu (například hráče nebo míče). Každá vrstva uchovává pozici jiného typu objektů. První vrstva uchovává polohu hráčů levého týmu. Druhá vrstva uchovává stejným způsobem pozice hráčů pravého týmu. V třetí vrstvě je na souřadnicích míče hodnota 255. Čtvrtá vrstva uchovává souřadnice aktivního hráče, který je ovládán agentem. Tato reprezentace prostředí je zpracována agentem.



Obrázek 3.3: Reprezentace prostředí zpracovávaná agentem. Rozměry této reprezentace jsou $72 \times 96 \times 4$.

3.8 Funkce odměny

Funkce odměny poskytuje agentovi odměnu za vykonanou akci v určitém stavu. Tato funkce je velmi důležitá pro správné fungování algoritmu hlubokého Q-učení [23]. Agentovi poskytuje za žádoucí akce pozitivní odměnu (například přiblížení se k brance) a za nežádoucí akce (například ztráta míče) poskytuje agentovi penalizaci. Tímto funkce motivuje agenta, aby vykonával akce, které ho přibližují k splnění zadaného úkolu. V této práci jsou implementovány tři funkce odměny.

- Funkce odměny A je základní funkce, která odměňuje agenta hodnotou 1, pokud agent nalezne cíl (míč nebo střed branky). Agent je penalizován hodnotou -0.5, pokud se snaží o pohyb mimo hřiště, což ho motivuje nevykonávat akce vedoucí k pohybu mimo hřiště. Pokud ztratí míč nebo je mu míč sebrán protihráčem, je agentovi udělena penalizace -1. Pokud uplyne maximální počet kroků epizody, agent nedosáhl cíle, tak je prostředí uvedeno do výchozího stavu a není mu udělena žádná odměna ani penalizace. Funkce penalizuje agenta za pohyb, což ho motivuje vykonat co nejméně kroků. Za každou akci pohybu do základních směrů je agent penalizován hodnotou -0.1, při diagonálním pohybu je penalizován hodnotou -0.141.
- Funkce odměny B rozšiřuje funkci odměny A použitím dílčích odměn za přibližování se k cíli. Odměny jsou stejné jako ve funkci A, až na odměňování agenta za snížení vzdálenosti od cíle. Agent využívá toho, že zná pozici cíle a pomocí odměn je veden směrem k tomuto cíli. Za každou akci, která přiblíží agenta ke splnění cíle, dostává agent odměnu. Pokud se agent vzdaluje od cíle, získává penalizaci. Pokud agent sníží vzdálenost od cíle, je mu poskytnuta odměna s hodnotou 0.08, což agenta motivuje postupovat směrem k cíli.

- Funkce odměny C rozšiřuje funkci odměny B. Místo jednotné hodnoty odměny za přiblížení míče k cíli, násobí odměnu vzdáleností, o kterou se míč k cíli přiblíží. Funkce motivuje agenta vykonávat akce, které přiblíží míč co nejvíce k bráně soupeře. Pokud si při přihrávce může vybrat mezi více spoluhráči, získává nejvyšší odměnu, pokud přihraje spoluhráči, který je nejblíže brance.

Kapitola 4

Implementace aplikace

V této kapitole bude popsána implementace agenta, který je schopen se naučit pomocí posilovaného učení vést útok na branku soupeře. Bude popsána implementace neuronové sítě, která na vstupu bere reprezentaci prostředí a na výstupu poskytuje agentovi ohodnocení akcí, které agent může vykonávat. Poté bude popsána implementace algoritmu hlubokého Q-učení a jak se agent učí ze spuštěného prostředí.

Agent je implementován v programovacím jazyku Python verze 3.6 a lze tedy přímo pracovat s prostředím *Google Research Football*. Pro práci s neuronovými sítěmi byla zvolena knihovna *Keras* a *TensorFlow*.

4.1 Struktura aplikace

Strukturu aplikace tvoří třídy pro trénování agenta, třídy prostředí *Simple Football Environment*, rozhraní pro *Google Research Football*, třídy implementovaného agenta a algoritmus posilovaného učení. Pro trénování agenta slouží třídy `Coach`, `Worker` a `TrainingHistory`. Tyto třídy manipulují se vstupem do prostředí (akce od agenta) a s výstupem z prostředí (snímek stavu prostředí). Agent je implementován ve třídě `Agent`. Algoritmus hlubokého Q-učení je implementován ve třídách `QNet`, `DQN` a `DDQN`. Rozhraní mezi agentem a prostředím *Google Research Football* je implementováno ve třídě `GFootballEnv`. Prostředí pro rychlé trénování agenta je implementováno ve třídě `SimpleFootballEnv`. Funkce odměny je implementována ve třídě `RewardFunction`.

Po spuštění aplikace je vytvořeno prostředí, ve kterém se bude agent trénovat. Následně je vytvořen samotný agent (instance třídy `Agent`) s parametry pro algoritmus posilovaného učení, tvarem reprezentace prostředí, množinou akcí, které může vykonávat v prostředí a zvolenou funkcí odměny. Poté je vytvořena instance třídy `Coach`, která se stará o trénování agenta po určený počet epizod a kroků. Pro prostředí a instanci třídy `Agent` vytvoří instance třídy `Coach` jednu instanci třídy `Worker`. Následně probíhá po zadaný počet epizod trénování agenta. Každou epizodu je na instanci třídy `Worker` volána funkce `start`. V této funkci probíhá trénování agenta po zadaný počet kroků.

4.1.1 Třída `Coach`

Instance této třídy má na starosti trénování agenta v prostředí po určený počet epizod a po určený počet kroků. Při vytvoření instance se do atribut ukládá vytvořené prostředí ve kterém bude agent trénovat. Třída `Coach` uchovává atributy, které udávají maximální počet epizod, po které se agent může učit a maximální počet kroků, které může agent vykonat

v každé epizodě. Instance této třídy uchovává ve svém atributu vždy pouze jednoho agenta. Pro každé prostředí je vytvořena jedna instance třídy `Worker`, která se stará o výstup z prostředí a agentův vstup do prostředí (akci). Po určeném počtu epizod učení je otestována účinnost agentova modelu. Pokud dosahuje nejlepšího dosud naměřeného výsledku, je model neuronové sítě uložen.

4.1.2 Třída `Worker`

Instance třídy `Worker` pracuje s jednou instancí prostředí a jednou instancí agenta. Má metodu `start`, kde probíhá učení agenta po jednu epizodu a po určený maximální počet kroků. Učení agenta probíhá v krocích. Každý krok je jeden průběh cyklu, kde se volá funkce `step` u prostředí a funkce `train` u agenta. Třída předává každý krok prostředí akci, která se má vykonat. Poté z prostředí obdrží extrahovaný stav po této akci. Stav je předán agentovi ve funkci `train`. Každý krok učení se zaznamenává do historie učení. Zaznamená se akce vybraná agentem a odměna za tuto akci. Odměna je získána z funkce odměny, která poskytuje agentovi odměny podle vykonané akce a stavu prostředí.

Pokud agent splní úkol nebo vyčerpá maximální počet kroků, je epizoda ukončena a instance třídy `Worker` vrací instanci třídy `Coach` historii průběhu učení agenta během epizody. Podle informací získaných z průběhu učení jsou generovány obrázky grafů.

4.1.3 Třída Agent

Instance této třídy slouží jako rozhraní mezi informacemi získanými z prostředí a algoritmem posilovaného učení. Uchovává v atributu instanci třídy `DQN`, což je třída implementující algoritmus posilovaného učení. Pokud je použito dvojité Q-učení, je místo instance třídy `DQN` použita instance třídy `DDQN`. Této instanci předává stav prostředí a získává akci určenou k vykonání v dalším kroku učení.

V atributech tato třída uchovává poslední akci získanou z algoritmu posilovaného učení, instanci třídy `RewardFunction`, stav prostředí před vykonáním akce a instanci třídy s algoritmem posilovaného učení. Instance třídy `RewardFunction` slouží k volání funkce, která poskytuje algoritmu odměnu za akci v určitém stavu.

Po vykonání získané akce je instanci třídy předán stav, ve kterém je prostředí po vykonání této akce. Tento stav je předán funkci odměny, která vypočítá odměnu za akci, která vyvolala tento stav prostředí a booleovskou hodnotu zda má být epizoda ukončena (například v případě, že agent splní úkol nebo mu je znemožněno splnit úkol protihráčem). Po získání odměny, třída `DQN` zaznamená stav prostředí po provedení akce, tuto akci a odměnu za akci. Poté probíhá učení sítě ve funkci `learn`. V případě, že je konec epizody, je snížena hodnota `epsilon` (popsáno v 4.1.4).

4.1.4 Třída DQN

V této třídě je implementován algoritmus hlubokého Q-učení. V jejích atributech jsou uloženy hyperparametry pro algoritmus hlubokého Q-učení a informace pro vytvoření umělých neuronových sítí. Atributy třídy potřebné pro vytvoření Q-sítě:

- Vstupní tvar, což jsou dimenze reprezentace prostředí (například $72 \times 96 \times 4$).
- Velikost akčního prostoru, což je počet akcí, ze kterých má neuronová síť vybírat.
- Velikost dávky z paměti zkušeností pro učení Q-sítě.

Atributy dále obsahují instanci třídy `QNet` a instanci třídy `ReplayBuffer`, která slouží jako paměť zkušeností pro metodu hlubokého Q-učení. Při každém trénování sítě pomocí funkce `learn` je z paměti zkušeností zvolen vzorek zkušeností. Z tohoto vzorku jsou vytvořena trénovací data a Q-síť je podle nich optimalizována optimalizátorem Adam (popsán v 3.1.3).

4.1.5 Třída `ReplayBuffer` a paměť zkušeností

V této třídě je implementována paměť zkušeností pro algoritmus hlubokého Q-učení. Zkušenost je záznam, který obsahuje stav prostředí, akci zvolenou pro tento stav prostředí, odměnu získanou za tuto akci a nový stav prostředí, který nastává po provedení akce. Agent sbírá zkušenosti během kroků epizody. Za každý krok epizody získá jeden záznam zkušenosti.

Učení hluboké neuronové sítě (Q-sítě) probíhá na konci kroku epizody, ale pouze pokud nastává konec epizody nebo po uplynutí zadaného počtu kroků. Pro učení Q-sítě jsou vytvořena trénovací data z dávky zkušeností. Dávka zkušeností je vzorek n záznamů zkušeností, který je vybrán náhodně ze všech zkušeností. Při trénování se tedy agentova Q-sít učí z již proběhlých zkušeností v minulosti. Protože je dávka z paměti zkušeností vybrána náhodně, nejsou záznamy v dávce propojeny a nedochází k učení sítě na velmi podobných stavech jdoucích po sobě [23].

4.1.6 Třída `QNet`

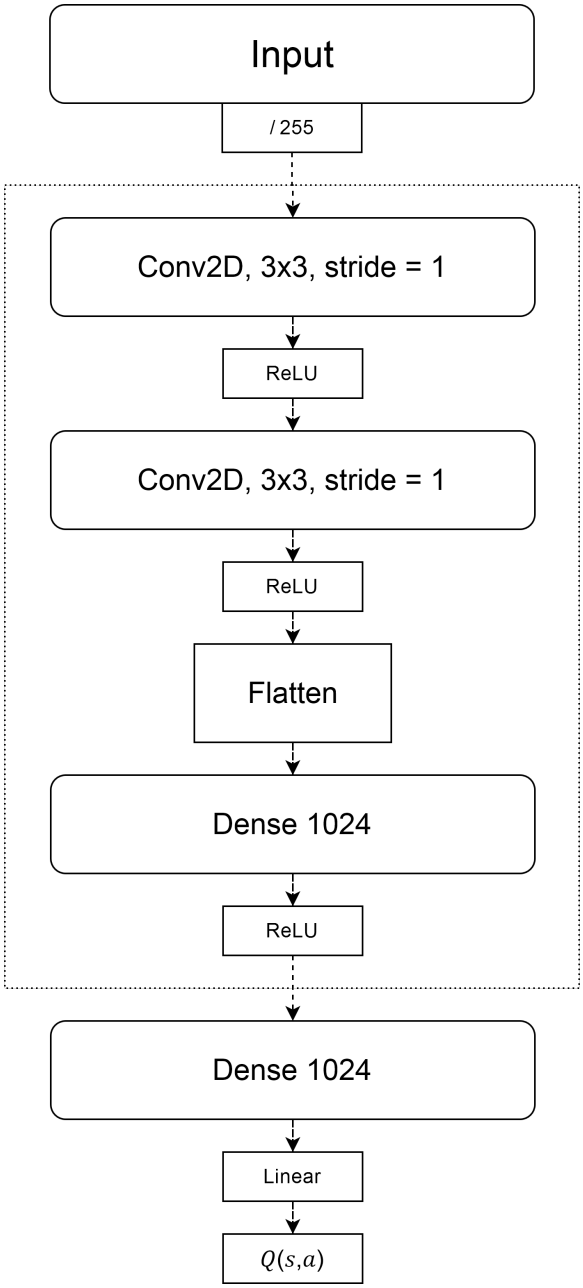
V této třídě je vytvářen model hluboké neuronové sítě s konvolučními a plně propojenými vrstvami. Konvoluční vrstvy zpracují reprezentaci prostředí a výsledek předají do plně propojených vrstev. Tyto plně propojené vrstvy se v algoritmu hlubokého Q-učení nazývají Q-síť. Schéma sítě je na obrázku 4.1

Při vytváření Q-sítě je třeba určit, kolik bude mít vstupních a výstupních hodnot. Počet vstupních hodnot je volen podle výstupu konvolučních vrstev (1024 výstupních hodnot z plně propojené vrstvy s 1024 neurony). Počet výstupních hodnot je počet akcí, které může agent vykonávat v prostředí. Výstupem této sítě je ohodnocení (Q-hodnota) jednotlivých akcí, které agent může vykonávat.

Pro zpracování reprezentace prostředí jsou určeny konvoluční vrstvy sítě. Tato síť byla navržena, tak aby dokázala zpracovat reprezentaci prostředí popsanou v sekci 3.7.1. Agent získá reprezentaci prostředí a zpracuje ji pomocí konvoluční neuronové sítě.

Konvoluční vrstva očekává na vstupu jeden snímek prostředí (například o velikosti $72 \times 96 \times 4$). Rozměr obrázku je popsán jako výška×šířka×počet kanálů. Před zpracováním snímku prostředí jsou pixely obrázku normalizovány. Normalizace pixelů probíhá vydělením hodnoty pixelů číslem 255. První konvoluční vrstva má 64 filtrů a velikost konvolučního jádra je 3×3. Tato vrstva má krok 1, který vyjadřuje posun konvolučního jádra. Druhá konvoluční vrstva má 32 filtrů, velikost konvolučního jádra je 3×3 a velikost kroku je 1.

Po třech konvolučních vrstvách následuje vrstva `flatten`. Na výstupu konvolučních vrstev jsou vícerozměrná data a vrstva `flatten` je sloučí do jednoho dlouhého vektoru vstupních dat pro následující plně propojenou vrstvu. Následuje plně propojená vrstva, která má 1024 neuronů a její výstup je 1024 hodnot. Výstup z této předposlední vrstvy je vstup pro Q-síť. Výstup z Q-sítě jsou ohodnocené akce (jejich počet je určen nastavením Q-sítě), které agent může vykonávat.



Obrázek 4.1: Implementovaná hluboká neuronová síť pro zpracování reprezentace prostředí a získání Q-hodnoty pro dvojici stavu a akce. Ohraničení s vynechávanou čarou označuje konvoluční část sítě. Vstupní reprezentace ve formátu $72 \times 96 \times 4$ pixelů je normalizována vydělením každého pixelu hodnotou 255. Následující dvě konvoluční vrstvy zpracují tuto reprezentaci a operace `flatten` upraví dimenze výstupu. Následující plně propojená vrstva upraví výstup konvoluční části sítě. Z následující plně propojené vrstvy jsou získány Q-hodnoty pro akce v daném stavu.

Všechny vrstvy této sítě používají aktivační funkci ReLU (rovnice 2.6). Pro úpravu vah neuronů této neuronové sítě je použit optimalizační algoritmus `Adam` (popsán v sekci 3.1.3). Tento algoritmus používá ztrátovou funkci MSE (rovnice 2.5).

4.1.7 Ostatní třídy

Pro testování aplikace a prostředí byla vytvořena třída `KBAgent`. V této třídě je implementován agent ovládaný klávesnicí. Slouží k testování aplikace (hlavně tříd `Coach` a `Worker`) a funkcí poskytujících odměny a penalizace. Jsou v něm implementovány stejné funkce jako v agentovi posilovaného učení s rozdílem, že akci nezískává z neuronové sítě, ale z klávesnice.

Třída `EpisodeHistory` slouží k zaznamenání průběhu učení agenta během jedné epizody. Zaznamenávají se akce, odměny, počet provedených kroků a výsledky ztrátové funkce. Instance této třídy je vytvořena vždy na začátku epizody a po jejím konci je předána instanci třídy `Coach`, kde je uložena do instance třídy `TrainingHistory`. Instance třídy `TrainingHistory` si ukládá záznamy učení do datového typu pole a ukládá je na pevný disk. Po ukončení učení agenta, jsou tyto informace použity k vytvoření obrázků grafů, které zobrazují průběh učení.

4.1.8 Propojení agenta a prostředí

Vytvořeného agenta lze propojit s jakýmkoliv prostředím, ale je třeba vytvořit rozhraní mezi agentem ovládajícím hráče a tímto prostředím. Agent očekává, že prostředí bude implementovat funkce `reset`, `step` a `close`.

Pro prostředí `Google Research Football` bylo vytvořeno rozhraní, které umožňuje agentovi používat akce pohybu, přihrávky a akci střely na bránu. Toto rozhraní je implementováno ve třídě `GFootballEnv` a obsahuje potřebná rozhraní pro hraní fotbalu v prostředí *Google Research Football*. Stará se o přeložení akcí z agenta do akcí v prostředí a o získávání reprezentace prostředí. Každá akce agenta je převedena na jednu nebo více akcí v prostředí a následně je vykonán potřebný počet kroků v prostředí. Toto rozhraní převádí stav prostředí *Google Research Football* na formát popsaný v sekci 3.7.1. Tento převod provádí instance třídy `SMMWrapper`.

Kapitola 5

Experimenty

V této kapitole jsou popsány experimenty, které probíhají na navrženém a implementovaném agentovi. Nejdříve jsou zjištěny optimální hyperparametry pro algoritmus hlubokého Q-učení, které budou vzápětí použity pro řešení situací v simulovaném fotbalu. Pro ověření funkčnosti agenta byly navrženy scénáře různých situací, které nastávají při hraní fotbalu. Agent při jejich řešení musí správně využívat dostupné akce a získat co nejvyšší kumulativní odměnu. Agent ovládá vždy agenta, který drží míč. Pokud míč není držen žádným hráčem, je získán pomocí strategie založené na pravidlech (kromě experimentu 1). Po získání míče je hráč s míčem označen jako aktivní hráč a je ovládán agentem.

Parametry pro algoritmus hlubokého Q-učení jsou:

Dostupné akce: Pohyb hráče do čtyř základních světových stran a pohyb do čtyř vedlejších světových stran. Přihrávka do zvoleného směru a střela na bránu.

Odměna: Agent získává odměnu s hodnotou 1 za splnění úlohy (získání míče nebo vstřelení branky). Při použití funkce odměny B agent získává odměnu 0.08 za přiblížení míče k cíli. Při použití funkce odměny C agent získává odměnu 0.01 za přiblížení míče k cíli, která je vynásobena uraženou vzdáleností.

Penalizace: Agent, který se pokusí o pohyb mimo hřiště získává penalizaci -0.5 bodu. Při použití akce, která způsobí ztrátu míče získává -1 bod. Agent je za akci pohybu na základní světovou stranu penalizován hodnotou -0.1 bodu. Za akci pohybu na rozšířenou světovou stranu je penalizován hodnotou -0.141 bodu, protože urazí vetší vzdálenost, než při pohybu na základní světovou stranu. Za akci přihrávky dostává penalizaci -0.141. Pokud je použita funkce odměny B, tak agent obdrží penalizaci -0.09 za zvýšení vzdálenosti míče od cíle. Při použití funkce odměny C, agent obdrží penalizaci -0.011 za zvýšení vzdálenosti míče od cíle. Tato penalizace se násobí vzdáleností, která byla uražena.

Maximální počet episod je 1000 pro experiment 1. V dalších experimentech je počet epizod navýšen na 2000.

Maximální počet kroků, které agent může vykonat je 64.

Gamma má hodnotu 0.95. Tato hodnota byla vybrána podle doporučení v [17].

Epsilon má počáteční hodnotu 1.0. V první epizodě agent vybírá zcela náhodné akce. Postupně se tato hodnota snižuje a agent začne vybírat akce podle Q-hodnoty.

Faktor snižování hodnoty epsilon je určen počtem epizod, tak aby dosáhl minimální hodnoty v posledních epizodách.

Minimální hodnota epsilon je 0.1. Agent i na konci trénování prozkoumává nové akce s pravděpodobností 10%.

Faktor učení je 0.00025. Tato hodnota byla vybrána podle doporučení v [17].

Velikost dávky je 128 záznamů. Tato hodnota byla zvolena podle maximálního počtu kroků. Při učení agent používá záznamy minimálně ze dvou epizod.

Kapacita paměti zkušeností je 6400 záznamů. Tato kapacita byla zvolena tak, aby bylo možno uložit všechny záznamy kroků po 1000 epizod.

Ukončovací podmínky epizody: K ukončení dojde při splnění úlohy nebo vyčerpání maximálního povoleného počtu kroků.

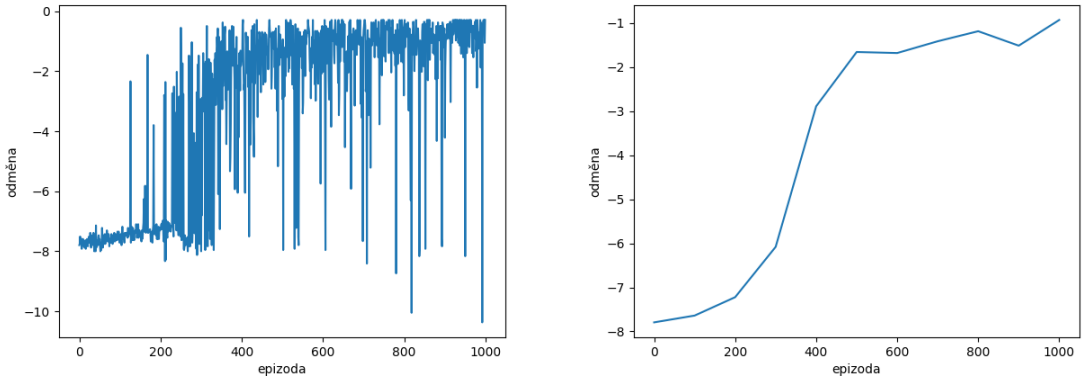
Ukončovací podmínky trénování agenta: K ukončení dojde při provedení maximálního počtu epizod. Pokud je natrénovaný model schopen vykonat úlohu s dosud nejvyšší naměřenou odměnou, je uložen zvlášť. Tento model je poté možno používat pro hraní fotbalu.

5.1 Experiment 1 - sbírání míče

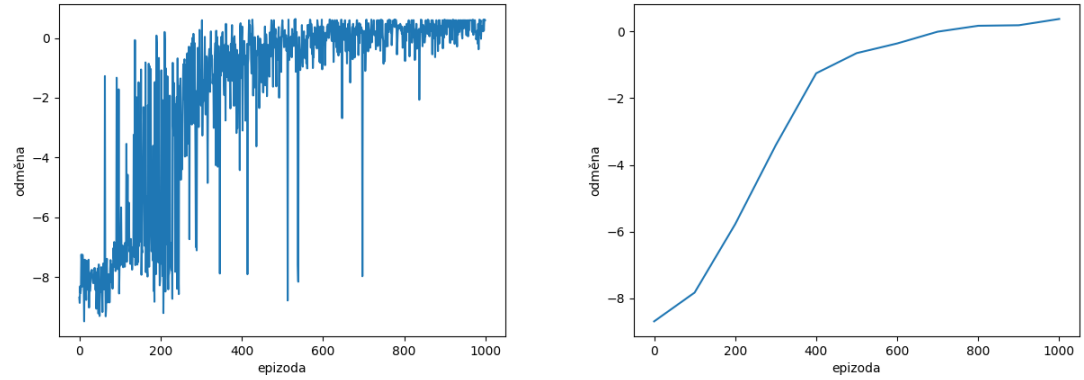
V tomto experimentu je na hrací plochu umístěn jeden hráč levého týmu a míč. Úkolem agenta je najít a získat míč. Úlohu musí splnit za co nejmenší počet kroků a s největší kumulativní odměnou. Agent má k dispozici pouze množinu akcí pro pohyb. Pokud agent dorazí na pozici míče, sebere ho a epizoda končí s odměnou za splnění úlohy. Každou desátou epizodu je model testován, a pokud dosáhne dosud nejlepšího výsledku, uloží se. Experiment je rozdělen na části, kdy v každé části je použita jiná funkce odměny (popsány v sekci 3.8). Experiment je proveden pouze v prostředí *Simple Football Environment*, protože hráč v prostředí *Google Research Football* nemůže být ovládán agentem, pokud začne hru bez míče (míč je nejdříve sebrán automaticky).

V první části experimentu agent používá funkci odměny A, která nepoužívá informaci o pozici míče. Agent hledá míč a není jakkoliv pomocí odměn naváděn k jeho pozici. Agent získává odměnu pouze za sebrání míče. Průběh trénování agenta je na obrázku 5.1.

V druhé části experimentu je použita funkce odměny B. Agent dostává malé odměny za snížení vzdálenosti vzdálenosti mezi hráčem a míčem. Navádět agenta k míči tímto způsobem lze díky tomu, že v reprezentaci prostředí je známa poloha míče. Agent je odměnami motivován, aby zmenšoval vzdálenost od míče a penalizacemi je motivován, aby ji naopak nezvětšoval. Průběh trénování agenta je na obrázku 5.2.



Obrázek 5.1: Obrázky grafů průběhu trénování agenta při úloze nalezení míče s použitím funkce odměny A. Agent dostává odměnu pouze při splnění úlohy. Vlevo je obrázek grafu kumulativních odměn jednotlivých epizod, vpravo je obrázek grafu průměrů kumulativních odměn za 100 epizod. Doba trénování nejlepšího modelu je 1960 sekund. Nejlepšího výsledku model dosahoval v 240 epizodě. Největší kumulativní odměna byla -0.205. Celková doba trénování je 5073 sekund. Průměrná odměna za celou dobu trénování je -3.221 bodů.



Obrázek 5.2: Obrázky grafů průběhu trénování agenta při úloze nalezení míče s použitím funkce odměny B. Agent získává malou odměnu, pokud se přiblíží k míči. Vlevo je obrázek grafu kumulativních odměn jednotlivých epizod, vpravo je obrázek grafu průměrů kumulativních odměn za 100 epizod. Doba trénování nejlepšího modelu je 1640 sekund. Nejlepšího výsledku model dosahoval v 170 epizodě. Největší kumulativní odměna byla 0.597. Celková doba trénování je 5378 sekund. Průměrná odměna za celou dobu trénování je -1.858 bodů.

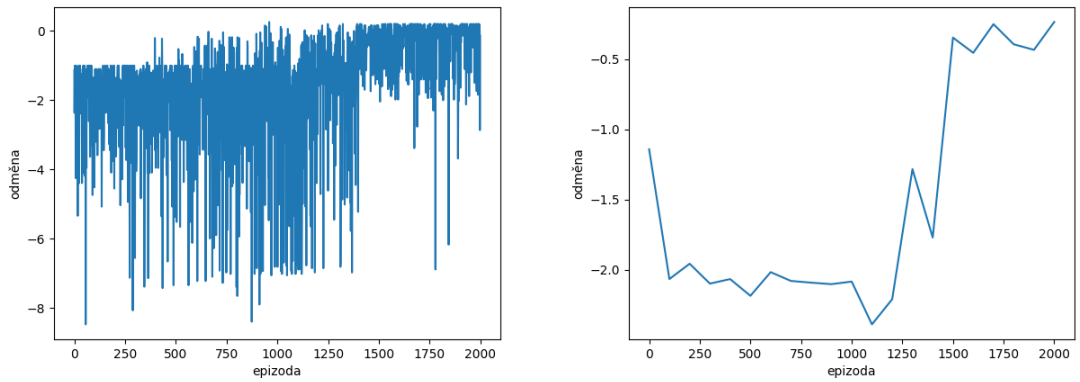
Vyhodnocení experimentu 1

Experiment potvrdil, že implementovaný agent dokáže najít pozici míče a sebrat ho. Při použití dílčích odměn za snižování vzdálenosti k míči, agent mnohem dříve a mnohem častěji začne nacházet míč. Agent, který používá funkci odměny B se naučí získat míč o 320 sekund dříve, než agent bez dílčích odměn. Pokud jsou agentovi poskytovány dílčí odměny, algoritmus Q-učení mnohem rychleji nalezne nejlepší akce k vyřešení problému.

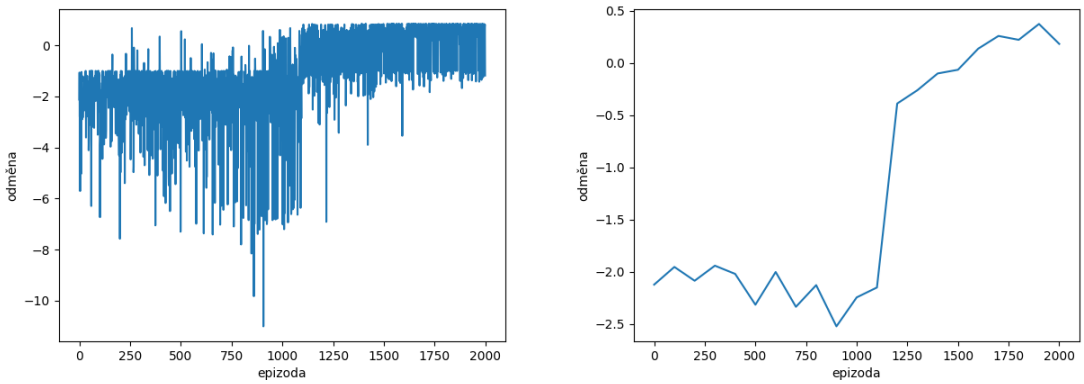
5.2 Experiment 2 - útok na bránu

Cílem tohoto experimentu je ověřit, zda je agent schopen se naučit útočit na bránu soupeře. V tomto experimentu útočí jeden hráč na bránu. Hráč začíná uprostřed hřiště s míčem a jeho úkolem je vstřelit míč do branky. Hráč ovládaný agentem se musí dostat na dostřel k brance (do střelecké pozice) s míčem, a poté použít akci střely na bránu. Střelecká pozice je kruhová výseč před bránou, ve které je velká šance na zásah středu branky.

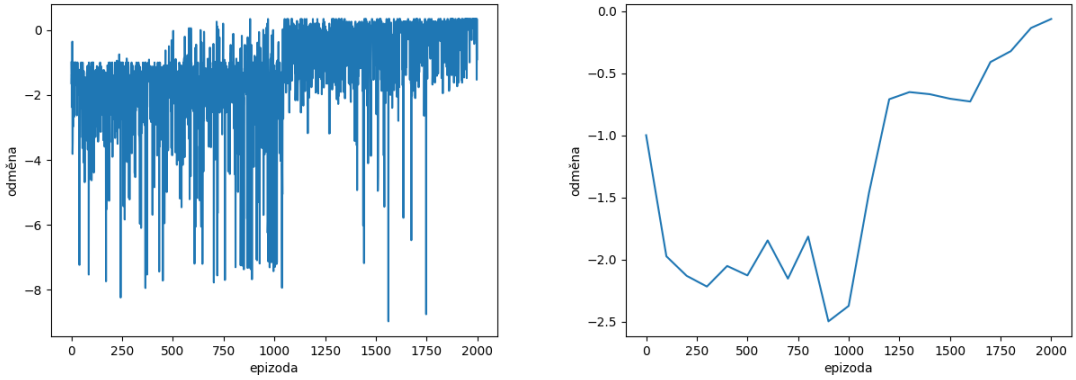
V experimentu byly použity tři funkce odměny. Dílčí odměny získává agent za pozici míče vzhledem k bráně soupeře. Při použití funkce odměny A je průběh trénování na obrázku 5.3, při použití funkce B je na obrázku 5.4 a průběh s funkcí C je na obrázku 5.5.



Obrázek 5.3: Obrázky grafů průběhu trénování agenta ve scénáři útoku na bránu. Pro odměňování agenta je použita funkce odměny A. Vlevo je obrázek grafu kumulativních odměn jednotlivých epizod, vpravo je obrázek grafu průměrů kumulativních odměn za 100 epizod. Doba trénování nejlepšího modelu je 4803 sekund. Nejlepšího výsledku model dosahoval v 1250 epizodě. Největší kumulativní odměna byla 0.2. Celková doba trénování je 7304 sekund. Průměrná odměna za celou dobu trénování je -1.5255.



Obrázek 5.4: Obrázky grafů průběhu trénování agenta ve scénáři útoku na bránu. Pro odměňování agenta je použita funkce odměny B. Vlevo je obrázek grafu kumulativních odměn jednotlivých epizod, vpravo je obrázek grafu průměrů kumulativních odměn za 100 epizod. Doba trénování nejlepšího modelu je 4596 sekund. Nejlepšího výsledku model dosahoval v 1100 epizodě. Největší kumulativní odměna byla 0.84. Celková doba trénování je 7677 sekund. Průměrná odměna za celou dobu trénování je -1.1672.



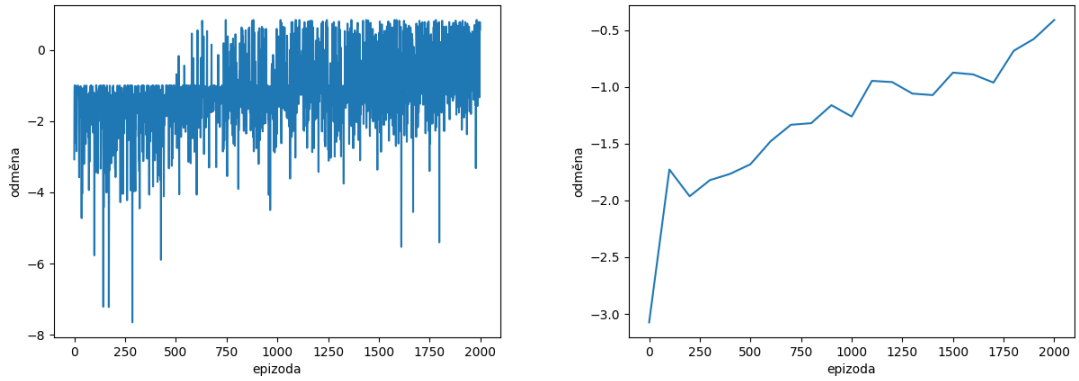
Obrázek 5.5: Obrázky grafů průběhu trénování agenta ve scénáři útoku na bránu. Pro odměňování agenta je použita funkce odměny C. Vlevo je obrázek grafu kumulativních odměn jednotlivých epizod, vpravo je obrázek grafu průměrů kumulativních odměn za 100 epizod. Doba trénování nejlepšího modelu je 2987 sekund. Nejlepšího výsledku model dosahoval v 717 epizodě. Největší kumulativní odměna byla 0.257. Celková doba trénování je 4753 sekund. Průměrná odměna za celou dobu trénování je -1.3532.

Vyhodnocení experimentu 2

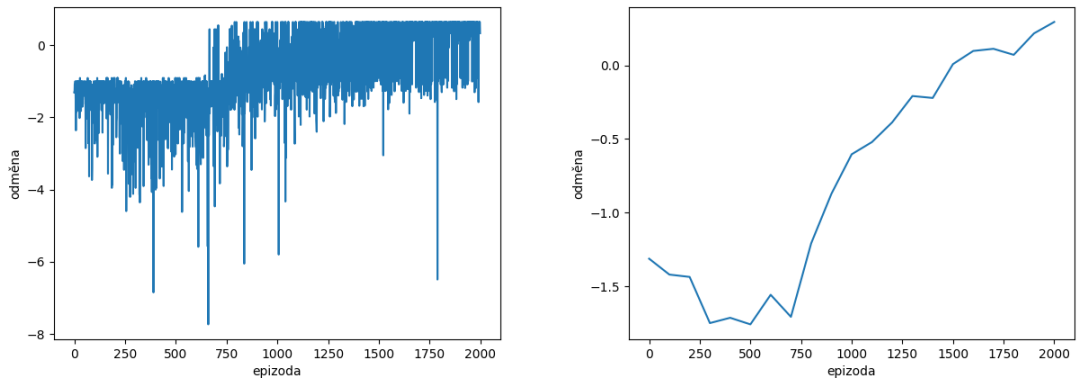
Experiment slouží k ověření, zda agent dokáže najít střeleckou pozici před bránou soupeře a vstřelit gól. Agent dokáže splnit úlohu při použití jakékoliv z funkcí odměny, ale funkce odměny B a C dosahují nejlepších výsledků. Agent používající funkci odměny B se naučil splnit scénář o 207 sekund dříve, než s použitím funkce A.

5.3 Experiment 3 - přihrávání

Cílem tohoto experimentu je zjistit, zda agent založený na hlubokém Q-učení dokáže pomocí přihrávek optimalizovat útok na bránu. V tomto scénáři jsou v hřišti 2 hráči levého týmu. Hráč, který je nejvzdálenější od brány má míč a je ovládán agentem. Cílem scénáře je vstřelit branku. Hráči si tedy budou muset přihrávat, tak aby se míč dostal k hráči, který je nejblíže k střelecké pozici. Funkce odměny A není použita, protože v předchozích experimentech dosahovala nejhorších výsledků. Průběh trénování agenta s funkcí B je na obrázku 5.6 a s využitím funkce C 5.7.



Obrázek 5.6: Obrázek grafu průběhu trénování agenta na úloze útoku na bránu s přihrává-ním. Agent používá funkci odměny B. Doba trénování nejlepšího modelu je 3548 sekund. Nejlepšího výsledku model dosahoval v 747 epizodě. Největší kumulativní odměna byla 0.839. Celková doba trénování je 7272 sekund. Průměrná odměna za celou dobu trénování je -1.1972 bodů.



Obrázek 5.7: Obrázek grafu průběhu trénování agenta na úloze útoku na bránu s přihrávání. Agent používá funkci odměny C. Doba trénování nejlepšího modelu je 2161 sekund. Nejlepšího výsledku model dosahoval v 730 epizodě. Největší kumulativní odměna byla 0.639. Celková doba trénování je 6301 sekund. Průměrná odměna za celou dobu trénování je -0.728 bodu.

Vyhodnocení experimentu 3

V předchozím experimentu musel hráč dojít na střeleckou pozici a vystavovat se riziku ztráty míče. Během akce pohybu se může k hráči přiblížit protihráč a sebrat mu míč. V tomto experimentu se agent naučí, jak minimalizovat pohyb s míčem pomocí přihrávek spoluhráčům. Počet akcí pohybu minimalizuje přihrávkou spoluhráči, který je blíže k bráně soupeře a místo několika rizikových akcí pohybu, vstřelí gól jeho spoluhráč. Nejlepšího výsledku dosahovala funkce odměny C. Natrénování modelu při použití funkce C trvalo o 1387 sekund méně, než u funkce B. To je dáno tím, že při použití funkce C, agent získává za přihrávku spoluhráči velkou odměnu a dává přednost přihrávce před pohybem (oproti funkci B).

5.4 Vyhodnocení použitých funkcí odměny

V sekci 3.8 byly popsány tři funkce odměny. Byly použity v provedených experimentech a byl zkoumán jejich vliv na trénování agenta.

Vyhodnocení funkce odměny *A*

Funkce odměny *A* poskytuje agentovi odměnu za splnění úlohy a penalizace za akce pohybu. Díky těmto penalizacím se agent snaží najít nejoptimálnější cestu ke splnění cíle a získat nejvyšší odměnu. Z tohoto důvodu je tato funkce vhodná pro hledání nejkratší cesty k cíli. Nevýhodou této funkce je, že nepoužívá žádné dílčí odměny a nevyužívá skutečnost, že agent zná polohu cíle. Agent tedy musí prozkoumávat celý prostor kolem sebe, aby objevil stav, ve kterém dostane odměnu. Tato funkce odměny dosahovala v experimentech nejhorších výsledků.

Vyhodnocení funkce odměny *B*

Druhá funkce odměny bere v potaz znalost pozice cíle (cíl může být míč nebo branka soupeře). Během experimentů jsem zjistil, že tato funkce odměny vede agenta směrem k cíli pomocí malých dílčích odměn. Použití průběžných dílčích odměn urychluje trénování agenta. Agent s touto funkcí dokázal splnit všechny scénáře rychleji než s funkcí A.

Vyhodnocení funkce odměny *C*

Funkce odměny C vylepšuje funkci B a bere v potaz vzdálenost, kterou urazí míč směrem k cíli. Funkce poskytuje agentovi odměnu za přiblížení míče k cíli, která je vynásobená vzdáleností, kterou urazí. Agent preferuje akce, které dostanou míč co nejblíže k bráně. Místo akcí pohybu směrem k cíli, preferuje přihrávky spoluhráčům, kteří jsou blíže brance. V experimentu s přihráváním dosahuje tato funkce nejlepších výsledků, protože agent upřednostňuje akci přihrávky, která dostane míč nejblíže k brance.

5.5 Srovnání s ostatními týmy

V tomto experimentu je srovnávána strategie útoku na bránu natrénovaného agenta a týmem, který se účastní soutěže *Google Research Football*. Žádný z týmů nevyužívá strojového učení. Pro srovnání s ostatním týmy byly vytvořeny tři scénáře útoku na bránu. V těchto scénářích musí hráči vhodně nahrávat, pohybovat se a střílet na bránu, tak aby splnili scénář v nejnižším možném čase. Agent používá funkci odměny C, protože v předchozích experimentech dosahovala nejlepších výsledků.

Ve scénářích mají hráči levého týmu za úkol vstřelit branku v co nejnižším čase a s co nejmenším počtem vykonaných akcí. Popis scénářů:

Scénář 1: Na hřišti je jediný hráč s míčem. Hráčova počáteční pozice je společně s míčem uprostřed hřiště. Hráč musí nalézt branku protivníka a vstřelit gól z optimální vzdálenosti.

Scénář 2: Na hřišti jsou dva hráči. V tomto scénáři je pro co nejrychlejší gól nutné přihrát spoluhráči.

Scénář 3: Na hřišti jsou čtyři hráči. Pro nejlepší výsledek je potřeba vhodně přihrávat.

Tabulka 5.1: Tabulka vyhodnocení týmů

	scénář 1	scénář 2	scénář 3
SunBear	51	69	71
SharpEye1	44	73	68
Raveman	45	43	51
můj agent	49	39	46

Vyhodnocení srovnání

Počet časových kroků, které potřebovali ke splnění úkolu jednotlivé týmy je v tabulce 5.1. Jeden časový krok v *Google Research Football* je časový úsek, kdy může hráč vykonat jednu akci.

V prvním scénáři dosahují všechny týmy podobných výsledků. Liší se pouze ve vzdálenosti, při které střílí na bránu. V druhém scénáři dosahuje můj agent a tým *Raveman* podobných výsledků. Další dva týmy nepoužívají akci přihrávky, dokud je neohrozí protihráč. Jejich hráči tedy musí doběhnout až k bráně a poté použít akci střely na bránu. Nedokáží tedy splnit tento scénář v minimálním možném čase a s minimálním počtem akcí. Takové chování může dát soupeři šanci k přesunutí se na výhodnější pozice. Můj agent a *Raveman* nepotřebují k aktivaci přihrávání protihráče a přihrají spoluhráči, který je ve výhodnější pozici. Tým *Raveman* používá pro tuto vlastnost výpočet výhodnosti přihrávky, jehož hodnota značí, jak výhodné je přihrát danému hráči. Můj agent se tuto vlastnost naučí pomocí posilovaného učení, protože díky přihrávce nemusí vykonat několik akcí a získává vyšší kumulativní odměnu.

Ve třetím scénáři musí hráč s míčem nahrát správnému spoluhráči. Můj agent se naučí nahrát správnému spoluhráči, protože za přihrávku získá nejvyšší odměnu. Tým *Raveman* přihrává vždy stejnému spoluhráči a nezjistí, že přihrávka jinému vede k rychlejšímu gólu. Z tohoto důvodu má horší výsledek, než agent s posilovaným učením.

Pomocí posilovaného učení lze získat posloupnost akcí, která vede ke splnění úkolu s nejmenším počtem akcí. Podle získaných posloupností lze optimalizovat chování hráče založeného na pravidlech. Dva ze srovnávaných týmů nepřihrávají spoluhráči, dokud k tomu nejsou donuceni protihráčem a lze je vylepšit.

Kapitola 6

Závěr

V této práci jsem nejprve popsal strojového učení, poté posilované učení a algoritmus, který byl pro posilované učení vybrán. Dále byla probrána neuronová síť, umělý neuron a konvoluční neuronová síť. Byla popsána dvě prostředí pro simulované hraní fotbalu a pravidla fotbalu, kterými se hráči řídí.

Implementoval jsem agenta pro hraní fotbalu, který je založen na hlubokém Q-učení. V algoritmu hlubokého Q-učení jsem použil vylepšení pomocí paměti zkušeností a dvojitého Q-učení. Pro agentovo vnímání stavu prostředí jsem vytvořil konvoluční neuronovou síť, která je schopna zpracovat snímek prostředí. Vytvořil jsem několik scénářů situací na hřišti, které ověřují schopnost agenta vyřešit danou situaci.

Pro testování a trénování agenta jsem si navrhl svoje vlastní prostředí pro hraní fotbalu. Výstupní snímek tohoto prostředí je stejný jako v *Google Research Football*. Implementovaného agenta lze použít v libovolném prostředí, ale je nutné vytvořit rozhraní mezi agentem a tímto prostředím. V této práci jsou vytvořena taková rozhraní dvě, a to pro *Google Research Football* a moje vlastní prostředí.

Pro srovnání jsem použil záznamy hraní scénářů, které jsem obdržel od několika tvůrců týmů do *Google Research Football*. Tyto scénáře jsem vytvořil, aby jsem prozkoumal jakým způsobem jednotlivé týmy útočí na branku. Většina srovnávaných týmů nevykonávala nejvhodnější akce při útoku na bránu soupeře. Je zde tedy prostor pro zlepšení hraní fotbalu pomocí posilovaného učení nebo použít získané posloupnosti akcí pro vylepšení hráčů založených na pravidlech.

Implementovaná aplikace umožňuje uživateli spustit vlastní scénář situace na hřišti, pro který chce najít nejoptimálnější posloupnost akcí. Takto lze odhalit nedostatky týmů se strategií založenou na pravidlech a vylepšit je pomocí získaných výsledků.

Výstupem práce jsou dvě rozhraní mezi použitým prostředím a agentem. Další výstup práce je implementovaný agent využívající algoritmus hlubokého Q-učení. Tento agent obsahuje hlubokou neuronovou síť, jejímž vstupem je reprezentace prostředí a výstupem je ohodnocení akcí. Posledním výstupem této práce jsou funkce odměny, které lze použít pro hraní fotbalu. Pomocí experimentů byla jako nejlepší vybrána funkce odměny, která odměňuje agenta za přibližování míče k cíli.

Celá aplikace je navrhnuta tak, aby ji bylo možné jednoduše rozšířit. Lze jednoduše přidat více algoritmů posilovaného učení a přidat rozhraní, aby bylo možné používat aplikaci i v dalších prostředích. Další vývoj práce by bylo navrhnutí a implementování rozhraní pro prostředí RoboCup, které by umožňovalo ovládat robota Nao.

Literatura

[1] *Convolution and Normalized Cross Correlation on Kepler Architecture* [online]. 2020 [cit. 2020-4-29]. Dostupné z: https://sipl.eelabs.technion.ac.il/projects/convolution-and-normalized-cross-correlation-on-kepler-architecture-2/.

[2] *Google Research Football Environment* [online]. 2020 [cit. 2020-3-27]. Dostupné z: https://hub.packtpub.com/google-research-football-environment-a-reinforcement-learning-environment-for-ai-agents-to-master-football/.

[3] *Introducing Google Research Football: A Novel Reinforcement Learning Environment* [online]. 2020 [cit. 2020-3-27]. Dostupné z: https://ai.googleblog.com/2019/06/introducing-google-research-football.html.

[4] *The RoboCup Soccer Simulator Users Manual* [online]. 2020 [cit. 2020-3-14]. Dostupné z: https://rcsoccersim.github.io/manual/.

[5] *Simspark wiki* [online]. 2020 [cit. 2020-4-25]. Dostupné z: http://simspark.sourceforge.net/wiki/.

[6] Alpaydin, E. *Introduction to Machine Learning*. 3. vyd. The MIT Press, 2014. ISBN 0-262-02818-2.

[7] Atienza, R. *Advanced Deep Learning with Keras*. 1. vyd. Packt Publishing, 2018. ISBN 1-78862-941-8.

[8] Barber, D. *Bayesian Reasoning and Machine Learning*. 1. vyd. Cambridge University Press, 2012. ISBN 0-521-51814-8.

[9] Bishop, C. *Pattern Recognition and Machine Learning*. 1. vyd. Springer-Verlag New York, 2006. ISBN 0-387-31073-8.

[10] Chalup, S. *RoboCup 2019 : Robot World Cup XXIII*. 1. vyd. Cham, Switzerland: Springer, 2019. ISBN 3-030-35698-1.

[11] Chollet, F. *Deep Learning with Python*. 1. vyd. Manning Publications, 2017. ISBN 1-61729-443-8.

[12] De, S., Mukherjee, A. a Ullah, E. *Convergence guarantees for RMSProp and ADAM in non-convex optimization and an empirical comparison to Nesterov acceleration* [online]. 2018 [cit. 2020-3-23]. Dostupné z: https://arxiv.org/abs/1807.06766.

[13] Gulli, A. *Deep learning with TensorFlow 2 and Keras : regression, ConvNets, GANs, RNNs, NLP, and more with TensorFlow 2 and the Keras API.* Birmingham: Packt Publishing Ltd, 2019. ISBN 1-83882-341-7.

[14] Hutter, M. *Universal Artificial Intelligence.* 1. vyd. Springer-Verlag Berlin Heidelberg, 2005. ISBN 3-642-06052-6.

[15] Kaelbling, L. P., Littman, M. L. a Moore, A. W. Reinforcement Learning: A Survey. [Journal of Artificial Intelligence Research, Vol 4, (1996), 237-285]. 1996, [cit. 2020-3-28]. Dostupné z: https://arxiv.org/abs/cs/9605103.

[16] Kingma, D. P. a Ba, J. *Adam: A Method for Stochastic Optimization* [online]. 2014 [cit. 2020-3-23]. Dostupné z: https://arxiv.org/abs/1412.6980.

[17] Kurach, K., Raichuk, A., Stańczyk, P., Zając, M., Bachem, O. et al. *Google Research Football: A Novel Reinforcement Learning Environment* [online]. 2019 [cit. 2020-3-28]. Dostupné z: https://arxiv.org/abs/1907.11180.

[18] Li, Y. *Deep Reinforcement Learning* [online]. 2018 [cit. 2020-3-28]. Dostupné z: https://arxiv.org/abs/1810.06339.

[19] Lonza, A. *Reinforcement Learning Algorithms with Python.* 1. vyd. Packt Publishing Ltd., 2019. ISBN 1-78913-111-6.

[20] Marsland, S. *Machine Learning: An Algorithmic Perspective.* 2. vyd. Chapman and Hall/CRC, 2014. ISBN 1-4665-8328-2.

[21] Mehlig, B. *Artificial Neural Networks* [online]. 2019 [cit. 2020-3-23]. Dostupné z: https://arxiv.org/abs/2001.00629.

[22] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I. et al. *Playing Atari with Deep Reinforcement Learning* [online]. 2013 [cit. 2020-3-23]. Dostupné z: https://arxiv.org/abs/1312.5602.

[23] Ravichandiran, S. *Hands-On Reinforcement Learning with Python: Master reinforcement and deep reinforcement learning using OpenAI Gym and TensorFlow.* 1. vyd. Packt Publishing, 2018. ISBN 1-78883-652-9.

[24] Ruder, S. *An overview of gradient descent optimization algorithms* [online]. 2016 [cit. 2020-3-23]. Dostupné z: https://arxiv.org/abs/1609.04747.

[25] Sarkar, A. *A Brandom-ian view of Reinforcement Learning towards strong-AI* [online]. 2018 [cit. 2020-3-23]. Dostupné z: https://arxiv.org/abs/1803.02912.

[26] Smola, A. a Vishwanathan, S. *Introduction to Machine Learning.* 1. vyd. Cambridge University Press, 2008. ISBN 0-521-82583-0.

[27] Sutton, R. S. a Barto, A. G. *Reinforcement Learning: An Introduction.* 2. vyd. The MIT Press, 2018. ISBN 0-262-19398-6.

[28] Yang, I.-S. *A Loss-Function for Causal Machine-Learning* [online]. 2020 [cit. 2020-3-23]. Dostupné z: https://arxiv.org/abs/2001.00629.

[29] Zai, A. a Brown, B. *Deep Reinforcement Learning in Action.* 1. vyd. Manning, 2020. ISBN 1-61729-543-4.

Příloha A

Obsah přiloženého paměťového média

doc/ Bakalářská práce v PDF a plakát.

doc/src/ Zdrojové soubory bakalářské práce a plakátu.

doc/README.txt Návod pro použití aplikace.

src/ Zrojové soubory agenta, rozhraní a prostředí.