FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Lokální koordinace a plánování pro robotický fotbal
Student:	Tomáš Valenta
Vedoucí:	doc. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je navrhnout abstraktní metody řízení hráčů v robotickém fotbale. Předpokládáme, že řešitel bude pracovat se zjednodušením, kdy je hrací plocha reprezentována diskrétně. Čas ale bude modelován spojitě, aby bylo možné pracovat s různou rychlostí hráčů a různou rychlostí vystřeleného míče. Hráči budou řízeni lokálně jako individuální agenti. Úkoly uchazeče jsou následující:

1. Prostuduje relevantní techniky návrhu řízení agentů s možností aplikace v robotickém fotbalu.
2. Navrhne vlastní nebo modifikuje existující techniku lokálního řízení hráčů pro zvolenou abstraktní variantu robotického fotbalu.
3. Návrh implementuje formou softwarového prototypu a experimentálně ověří úspěšnost automaticky řízených hráčů v syntetických scénářích.

–

[1] Devin Schwab, Yifeng Zhu, Manuela M. Veloso: Zero Shot Transfer Learning for Robot Soccer. AAMAS 2018: 2070-2072

[2] Juan Pablo Mendoza, Reid G. Simmons, Manuela M. Veloso: Online Learning of Robot Soccer Free Kick Plans Using a Bandit Approach. ICAPS 2016: 504-508

[3] Marika Ivanová, Pavel Surynek, Katsutoshi Hirayama: Area Protection in Adversarial Path-finding Scenarios with Multiple Mobile Agents on Graphs - A Theoretical and Experimental Study of Strategies for Defense Coordination. ICAART (1) 2018: 184-191

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 29. října 2020 v Praze.

Bakalářská práce

LOKÁLNÍ KOORDINACE A PLÁNOVÁNÍ PRO ROBOTICKÝ FOTBAL

Tomáš Valenta

Fakulta informačních technologií ČVUT v Praze
Katedra aplikované matematiky
Vedoucí: doc. RNDr. Pavel Surynek, Ph.D.
13. května 2021

Odkaz na tuto práci: Tomáš Valenta. *Lokální koordinace a plánování pro robotický fotbal.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především doc. RNDr. Pavlu Surynkovi, Ph.D. za vedení této práce, trpělivost a hlavně za jeho cenné rady. Dále bych chtěl poděkovat své přítelkyni Kláře a rodině za podporu při psaní této práce. Nakonec mé díky patří všem osobám, díky kterým se podařilo sesbírat důležitá data.

Prohlášení

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

V Praze dne 12. května 2021 .....

Abstrakt

Předmětem této práce je prozkoumání relevantních technik návrhu lokálního řízení agentů s možností aplikace v robotickém fotbale. Jako vnitřní mechanismus agentů jsou použity rozhodovací stromy a k jejich konstrukci je využit algoritmus ID3. Dále je vytvořen softwarový prototyp a z něho jsou sesbíraná relevantní data ze simulací jednoduchých situací. Nakonec jsou vytvoření agenti testováni v různých simulacích. Výsledky jsou poté srovnány s jinými přístupy.

Klíčová slova robotický fotbal, multiagentní systém, lokální řízení, rozhodovací stromy, supervizované učení, C#, Unity3D

Abstract

The goal of this thesis is examine relevant techniques of design local agent coordination with the possibility of application in robotic football. Decision trees are used for inner mechanism of agents and for their construction we are using the ID3 algorithm. Furthermore, software prototype is created and relevant data are gathered from simulations of simple situations. Finally, the created agents are tested in various simulations. Their results are compared with other approaches.

Keywords robotic football, multiagent system, local coordination, decision trees, supervised learning, C#, Unity3D

Seznam zkratek

AI	Umělá inteligence
BFS	Breadth-first search
CART	Classification And Regression Trees
DFS	Depth-first search
FIFA	Fédération Internationale de Football Association
FIFO	First-in, First-out
FIRA	Federation of InternationalRobot-soccer Association
ID3	Iterative Dichotomiser 3
MAE	Mean Absolute Error
MarLÖ	Multi-Agent Reinforcement Learning in MalmÖ
MAS	Multiagentní systém
MSE	Mean Squared Error
PEAS	Performance, Environment, Actuators, Sensors

Kapitola 1

Úvod

Robotický fotbal je jedno z velmi probíraných praktických témat na poli umělé inteligence (dále jen AI). Neoficiálně se řadí se mezi sporty a organizuje ho FIRA (Federation of International Robot-soccer Association, česky Mezinárodní federace roboticko-fotbalové asociace). Od roku 1997 se po celém světě organizuje akce jménem RoboCup, kde týmy soutěží v různých kategoriích (simulace, reální roboti). Poprvé se zúčastnilo přes 40 týmů a akci sledovalo přes 5 000 diváků. Minulý rok se kvůli pandemii COVID-19 RoboCup přesunul do online prostředí a přítomni byli jen domácí účastníci z Japonska.

Práce se zabývá návrhem lokálního řízení agentů s aplikací v robotickém fotbale. Existuje celá řada metod, jak k tomuto problému přistoupit. Multiagentní systémy a jejich fungování jsou známy již několik let, ale jejich využívání v praxi je stále větší. Vnitřní mechanismus agentů je realizován pomocí rozhodovacích stromů.

Tématem této práce je vytvoření a učení agentů, kteří budou schopni účinkovat v simulaci robotického fotbalu. Vnitřní mechanismus agenta – rozhodovací strom – je zkonstruován z dat získaných odehráním jednoduchých situací reálnými hráči. Tito agenti budou podléhat experimentům proti různým agentům (týmům agentů) s jiným vnitřním mechanismem. Data budou sbírána od dvou skupin lidí. První skupina bude složena z hráčů fotbalu a druhá nikoliv.

Motivací k výběru tématu byl převážně vlastní zájem o hlubší prozkoumání tématu robotického fotbalu a umělé inteligence obecně. K motivaci přispěl i první ryze online RoboCup pořádaný v Japonsku, kde se řeší pouze řízení jednotlivých hráčů a odpadá řešení problémů s hardwarem.

V první části bude čtenář seznámen se základními teoretickými poznatky o využívaných technikách řízení agentů, rozhodovacími stromy a rešerší existujících řešení. Také se zde dozví základní pravidla fotbalu, resp. robotického fotbalu. V další části se práce zabývá konkrétní implemetací simulace a seznámením s úpravou (relaxací) pravidel. V poslední části se práce zaobírá získáním dat, vybráním vhodných příznaků a následně experimenty s naučenými agenty.

1.1 Cíle práce

Cílem práce je navrhnout vlastní nebo modifikovat existující techniku lokálního řízení agentů. Tento návrh poté implementovat formou softwarového prototypu a experimentálně ověřit fungování tohoto návrhu.

Cílem teoretické části práce je seznámit se s inteligentními agenty a jejich zapojení do multiagentního systému. Dalším cílem je prozkoumat algoritmy konstrukce a učení rozhodovacích stromů. Posledním cílem této části je provést rešerši již existujících řešení nebo řešení podobných tomuto problému.

Cílem praktické části práce je implementovat simulaci robotického fotbalu, která bude mít interaktivní a autonomní část. Z interaktivní části bude možnost získat data z právě probíhané simulace a ze získaných dat vytvořit příznaky vhodné pro konstrukci rozhodovacích stromů. Tato konstrukce bude probíhat v autonomní části, kde bude zároveň možnost pozorovat simulace agentů při hře proti sobě. Posledním cílem je provádět experimenty nad získanými daty s vybranými metodami a algoritmy. Cílem této práce není vytvoření realistické simulace robotického fotbalu ani konstrukce fyzických agentů – robotů.

Kapitola 2

Teoretická východiska práce

V této kapitole se budeme zabývat teoretickými východisky práce a vymezením používaných pojmů. Také si ukážeme existující řešení problémů podobných našemu. Popíšeme základní problematiku multiagetních systémů i agenty samotné. Ukážeme možné přístupy a metody k řízení agentů. Dále se budeme zabývat definicí rozhodovacích stromů, různých algoritmů k jejich učení a konstrukci. Posléze ukážeme existující řešení zabývající se robotickým fotbalem či řízením multiagentního systému pomocí rozhodovacích stromů.

Problém koordinace v robotickém fotbalu se dá převést na problém Multiagentního systému v nepřátelském prostředí.

2.1 Multiagentní systémy

Multiagentní systémy jsou zkoumány už od roku 1980 a většího zájmu se dostalo v polovině 90. let minulého století.[1] Za tímto zájmem a popularizací tématu stojí rozvoj internetu, kdy se předpokládalo, že agenti jsou vhodné softwarové paradigma při využití v obrovských otevřených distribuovaných systémech.

Brown a Shoham za multiagentní systém považují „*systémy obsahující více autonomních entit (agentů) s různými zájmy, informacemi nebo obojím.*“ [2, překlad vlastní] Autoři doplňují, že se nejedná o přesnou definici multiagentního systému, kvůli spoustě nekonzistentních odpovědí na tuto otázku. Podle Wooldridge jsou MAS „*systémy sestavené z vícero interaktivních a počítacích elementů, známých jako agenti.*“ [1, překlad vlastní]

2.1.1 Agent

Definic samostatného agenta existuje spousta, ale všechny se upínají stejným směrem. Russell a Norvig se zaměřují na situování do určitého prostředí.

▶ **Definice 2.1.** *„Agent je cokoliv, co dokáže vnímat prostředí přes svoje senzory a na základě těchto informací vykonávat akce v tomto prostředí pomocí dostupných prostředků.“* [3, překlad

vlastní]

Wooldridge přidává definici agenta další rozměr, a to vztah mezi agentem a jeho designérem.

▶ **Definice 2.2.** *„Agent je počítačový systém, který je situován do nějakého prostředí a je schopný autonomních akcí v tomto prostředí za účelem dosažení úkolů a cílů, které jsou určeny jeho tvůrcem.“* [1, překlad vlastní]

Autoři pak doplňují, že agent je schopen pouze ovlivňovat prostředí a nedokáže ho plně řídit. Musí být pak připraven na možnost selhání. K tomu ještě reálné prostředí není vždy deterministické, s čímž se také musí počítat. Russel a Norvig rozlišují různé vlastnosti prostředí[3]:

Přístupné vs. nepřístupné – Přístupné prostředí je takové, o kterém dokáže agent aktuálně, kompletně a přesně zjistit informace. Prostředí reálného světa jsou spíše nepřístupná.

Deterministické vs. nedeterministické – V deterministickém prostředí mají všechny akce jasně daný a garantovaný efekt. Narozdíl od nedeterministického, kde máme jakousi nejistotu o stavu, který bude výsledkem akce.

Statické vs. dynamické – Statické prostředí není měněno ničím jiným než pomocí akcí agentů. V praxi se ovšem více setkáváme s dynamickým prostředím, které je ovlivňováno jinými procesy (fyzické jevy, poruchy atd.).

Diskrétní vs. spojité – V diskrétním prostředí je pevný a konečný počet akcí a vjemů (např. pohyb na mřížce rozměrů: $n * n \mid n \in \mathbb{N} \wedge n$ je konečné).

Známé vs. neznámé – Známé prostředí je (jak z pohledu agenta, tak z pohledu designéra agenta) takové, kde jsou známé určité zákonitosti. V neznámém prostředí se musí agent *zorientovat* a přijít na fungování tohoto prostředí. Příkladem neznámého prostředí by mohlo být řízení auta v cizí zemi bez navigace. Toto prostředí je přístupné (vidíme silnici, značky, ostatní auta), ale musíme přijít na jeho fungování (na jaké straně se jezdí, kam vede silnice).

Každý agent by měl splňovat následující vlastnosti a měl by být:

- Autonomní – Agent k dosažení vlastního cílu nepotřebuje zásah nebo vedení od uživatele. Nemáme nad ním **žádnou přímou kontrolu**.
- Reaktivní – Agent interaguje s okolím a reaguje na jeho případné změny.
- Proaktivní – Agent se pokouší dosáhnout svých cílů. Není řízen událostmi, ale přebírá iniciativu.
- Společenský – Agent interaguje s ostatními agenty skrze spolupráci (pracování spolu jako tým, za dosažení stejného cíle, např. v našem příkladě dát co nejvíce gólů a zároveň jich co nejméně dostat), koordinaci (řízení vzájemných závislostí mezi akcemi vícero agentů) a vyjednávání (schopnost dosáhnout dohody ve věcech společného zájmu).

„*Práce umělé inteligence je designovat* ***program agenta****, který implementuje agentní funkci – mapování vjemů na akce.*“[3, překlad vlastní] Tento program funguje na nějakém fyzickém zařízení se senzory a aktuátory, což nazýváme architekturou a platí [3]:

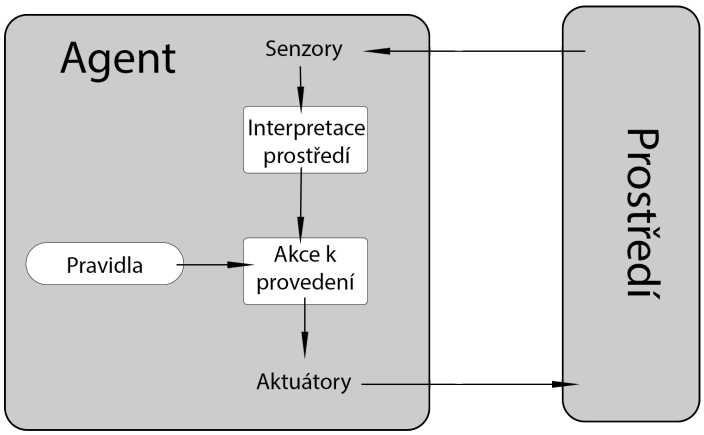
$$agent = architektura + program$$

Programy agenta mají stejnou strukturu (vstup ze senzorů, výstup z aktuátorů) a podle Russela a Norviga rozlišujeme 4 základní druhy programů – agentů:

Jednoduchý reflexní agent

Nejjednodušší druh agenta. Na základě současných vjemů volí akci, přitom ignoruje veškerou historii vjemů. Na obrázku 2.1 a v kódu 1 můžeme vidět abstraktní strukturu programu. Nejdříve musíme získat nynější stav z vjemu ze senzorů (např. vypočítat pozici na základě obrazu z kamery). Následně musíme najít pravidlo, které odpovídá nynějšímu stavu a nakonec vrátit akci nalezeného pravidla. Seznam pravidel je předem znám a je čistě koncepční. „*Implementace může být triviální stejně jako například soubor logických hradel implementující boolovský obvod.*“[3, překlad vlastní] Tento typ agentů je velmi jednoduchý, ale jeho inteligence je velmi omezená. „*Jednoduchý reflexní agent bude fungovat, když na základě aktuálních vjemů může být správně rozhodnuto – to je jen tehdy, když je prostředí kompletně přístupné.*“[3, překlad vlastní]

Jeden z větších problémů nastává při zacyklení v programu (agent čeká na učitý vstup, který nemusí nastat). Řešením může být částečná randomizace výstupu, která může být více prospěšná v multiagentních systémech, než v situaci s jedním agentem.[3]



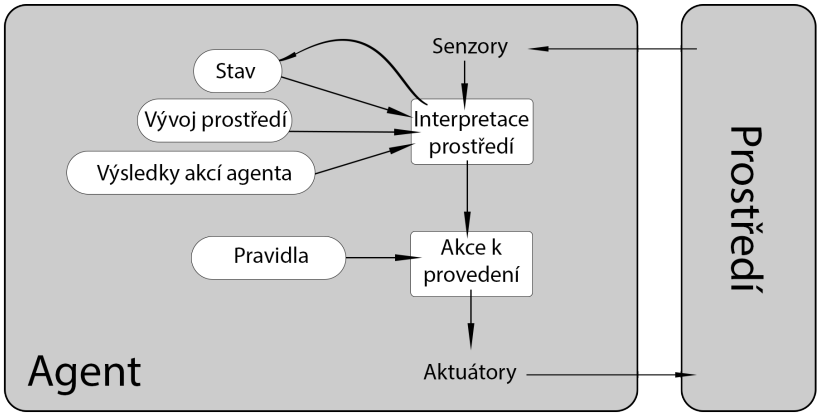
Obrázek 2.1 Architektura reflexního agenta [3]

Algoritmus 1: Jednoduchý reflexní agent

Vstup: vjem ze senzorů, pravidla
Výstup: akce
stav = intepretaceVstupu(vjem)
pravidlo = najdiPravidlo(stav,pravidla)
akce = pravidlo.akce
return akce

Modelově založený reflexní agent

„*Nejefektivnější cesta, jak zvládnout částečnou přístupnost, je pamatovat si část prostředí, která je pro agenta nepřístupná.*“ [3, překlad vlastní] To znamená, že agent si musí udržovat vnitřní stav na základě historie vstupů. Dále se agent chová stejně jako reflexní. K aktualizaci vnitřního stavu potřebujeme dvě informace. První je informace o chování prostředí nezávisle na agentovi a druhá je, jak agent svými akcemi může toto prostředí změnit. Tyto informace o prostředí nazýváme modelem, proto mluvíme o **Modelově založeném reflexním agentovi**.



Obrázek 2.2 Architektura Modelově založeného agenta [3]

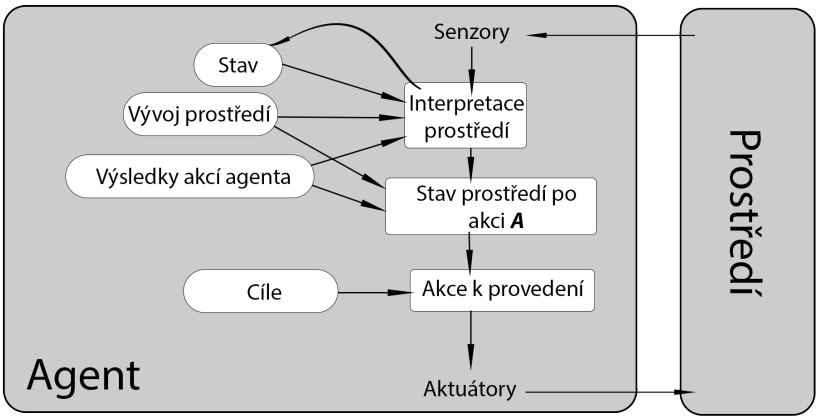
Algoritmus 2: Modelově založený reflexní agent

Vstup: vjem ze senzorů, pravidla, model, předchozí akce
Výstup: akce
stav = aktualizaceStavu(vjem,pravidla,model,předchozí akce)
pravidlo = najdiPravidlo(stav,pravidla)
akce = pravidlo.akce
return akce

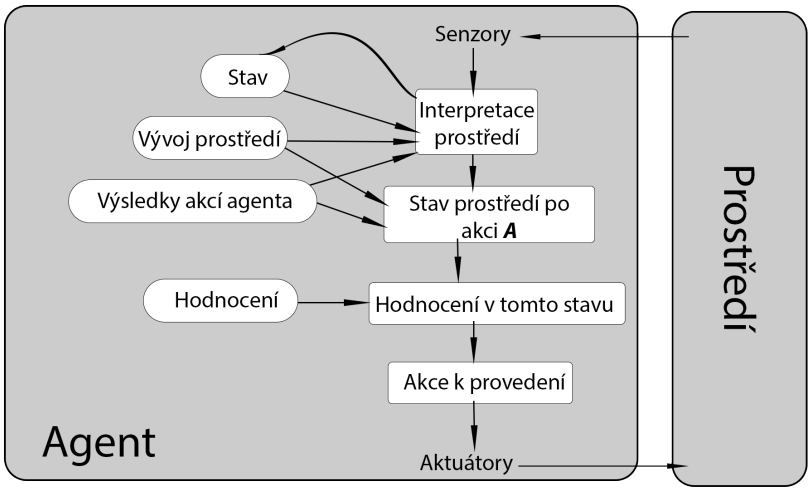
Agent na základě cíle

„*Vědět informace o nynějším stavu prostředí není občas dost ke správnému rozhodnutí.*“ [3,

překlad vlastní] Občas je důležité znát informace, které popisují cíle agenta. Program agenta pak kombinuje tyto informace s modelem (stejný jako v modelově založeném agentovi) a na základě této kombinace vybírá akci. Narozdíl od reflexního agenta je lépe flexibilní a není problém definovat nové cíle (v reflexním agentovi musíme přepisovat celá pravidla, pokud chceme nový cíl).



Obrázek 2.3 Agent na základě cíle [3]



Obrázek 2.4 Prospěšně založený agent [3]

Prospěšně založený agent

„*Cíle samotné nejsou dostatečné pro kvalitní chování v mnoha prostředí*“ [3, překlad vlastní]. Občas existují rychlejší, levnější a kvalitnější způsoby, jak cíle splnit. Cíle jsou buď splněny nebo nesplněny, a proto potřebujeme funkci, která nám bude měřit kvalitu stavu po provedení nějaké

akce. Agent si pak vybere tu akci, která byla vyhodnocena touto funkcí jako nejprospěšnější. Tito agenti si lépe poradí s částečně přístupným a nedeterministickým prostředím.

Vnitřní mechanismus agenta může být implementován mnoha způsoby. „*Každý mechanismus má stejnou kostru: na vstupu berou data ze senzorů a na výstupu vracejí akce do pohonů.*“ [3, překlad vlastní] Můžeme například využít konečné automaty, kde musíme vstupy převést na abecedu a koncové stavy budou reprezentovat akce. Mezi další metody můžeme zařadit plné konvoluční sítě a reprezentace akcí pomocí obrázků [4]. V našem případě bude vnitřní mechanismus reprezentován rozhodovacím stromem.

2.2 Druhy učení

Jedna z důležitých činností na poli umělé inteligence je učení, kdy se zlepšuje vlastní výkon k dosažení lepších budoucích výsledků. Učení se pohybuje od triviálních až po komplexní metody. Podle Russella a Norviga se učení dělí na tři, resp. čtyři kategorie: [3]

- Nesupervizované učení – Učení bez jakékoliv odezvy od „učitele“ – autor musí v datech sám najít nějaké závislosti.
- Posilované učení – Učení pomocí odměn nebo trestů.
- Supervizované učení – Učení pomocí ukázkových vstupů neznámé funkce f a jim odpovídajících výstupů. Z těchto dvojic pak hledáme funkci aproximující f.
- Semi-supervizované učení – Kompromis mezi supervizovaným a nesupervizovaným učením. V praxi nemáme vždy pravdivé výstupy k ukázkovým vstupům. Zohledňuje systematické chyby, kde jejich odhalení je problém nesupervizovaného učení.

2.3 Rozhodovací stromy

▶ **Definice 2.3.** *„Rozhodovací strom reprezentuje funkci, která jako vstup očekává vektor příznaků a vrací jednu hodnotu – rozhodnutí“ [3, překlad vlastní]*

Vstupy i výstupy této funkce mohou být:

- Diskrétní – nabývá spočetně mnoha hodnot. Například: $\{0, 1\}$[1], $\{x \in \mathbb{N} \mid x < 10\}$, počet obyvatel.
- Spojité – nabývá nespočetně mnoha hodnot. Například: $\langle 0, 1)$, $\mathbb{R}$, výška nebo váha člověka.

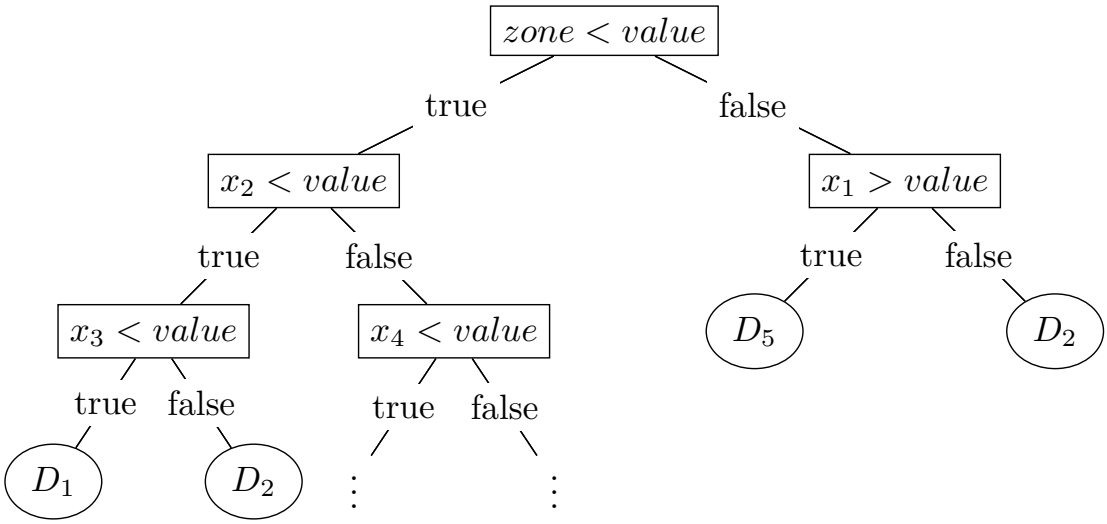
Celý rozhodovací strom bychom mohli graficky znázornit pomocí stromové struktury (viz. obrázek 2.5), nejčastěji binárního stromu. V běžném životě můžeme rozhodovací stromy přirovnat k různým manuálům. Skládá se z:

- Kořene – Uzel bez žádného předka. Vždy bude vnitřním vrcholem.

[1] Při binárním výstupu můžeme mluvit o Boolovské klasifikaci.

- Vnitřních vrcholů – Uzly testující příznaky podle předem daných podmínek. Jejich následníci mohou být vnitřní vrcholy nebo listy.
- Listů – Uzly obsahující výstup (rozhodnutí). Nemají žádné následníky.

Jako stromy známé z teorie grafů mají stejně definovanou hloubku, která se používá jako jeden z hyperparametrů při jeho konstrukci.



Obrázek 2.5 Schéma rozhodovacího stromu

Při diskrétním, resp. spojitém výstupu můžeme mluvit o klasifikaci, resp. regresi.

2.3.1 Algoritmy pro konstrukci stromů

Konstrukce rozhodovacích stromů je jedním z příkladů supervizovaného učení. K tomu potřebujeme trénovací data, která se skládají z matice příznaků a vektoru vysvětlované proměnné.

2.3.1.1 Hyperparametry

Hyperparametry modelu jsou takové parametry, které se používají při konstrukci stromu. Určují používané metriky a hodnoty podmínek větvení, resp. vytváření listů. Hledání nejoptimálnějších hodnot se nazývá **ladění hyperparametrů** a jedná se o systematické zkoušení různých kombinací hodnot a vyhodnocování výsledků modelu. Mezi základní hyperparametry patří [5]:

Kritérium – Funkce, která měří kvalitu rozdělení příznaků. Nejčastější jsou Gini index nebo Entropie. Více v 2.3.1.2. Tato funkce se pak použije při výpočtu informačního zisku.

Maximální hloubka – Maximální hloubka rozhodovacího stromu. V každém úplném binárním stromě s hloubkou h je přesně 2^{h-1}. Kdybychom měli h příznaků [2], tak počet všech možnách kombinací je také 2^{h-1}, čímž by strom „degradoval“ na slovník. Díky tomuto hyperparametru můžeme této „degradaci“ předejít.

[2] Opět pro jednoduchost uvažujeme binární klasifikaci.

Váha příznaků – Slouží k vážení důležitosti příznaku.

Minimální počet výsledků v listu – Pokud při konstrukci stromu není dostatek příznaků, vytvoří se list.

2.3.1.2 Používané metriky

Pro algoritmy pro konstrukci stromů potřebujeme nějakým způsobem měřit, jak určitý příznak rozdělí vektor vysvětlované proměnné. Hledáme tedy funkci, která je nezáporná, maximální pro stejné počty výsledků a nulová pro vektor se stejnou proměnnou [6]. Můžeme použít odhad **Entropie** na základě dat [6]:

$$Entropie(\mathcal{D}) = H(\mathcal{D}) = -\sum_{i=0}^{k-1} p_i \log(p_i),$$

kde $\mathcal{D}$ je množina dat, p_i je poměr počtu i v $\mathcal{D}$ a platí $\sum_{i=0}^{k-1} p_i = 1$.

Jelikož chceme vybrat příznak, který nejvíce sníží neuspořádanost, využijeme **informační zisk** [6]:

$$InformacniZisk(\mathcal{D}, X_i) = IG(\mathcal{D}) = H(\mathcal{D}) - \sum_{j=0}^{k-1} t_j H(\mathcal{D}_j),$$

kde $\mathcal{D}_j$ je podmnožina $\mathcal{D}$ pro které $X_i = j$, t_j je podíl počtu prvků v $\mathcal{D}_j$ a $\mathcal{D}$.

Další metrikou je tzv. **Gini index**, která udává míru toho, že nově přidaný prvek bude špatně klasifikován. Přesný výpočet je [6]:

$$GiniIndex(\mathcal{D}) = GI(\mathcal{D}) = \sum_{i=0}^{k-1} p_i(1 - p_i)$$

Gini index můžeme využit při výpočtu informačního zisku, kdy jen ve vzorci nahradíme $(\mathcal{D})$ za $GI(\mathcal{D})$.

Jako poslední uvádíme dvě podobné metriky a to **MSE = Mean Squared Error** a **MAE = Mean Absolute Error**. Obě metriky měří, jak moc se liší hodnoty od střední hodnoty. Podobně jako u ostatních metrik pracujeme vlastně s odhadem těchto dvou veličin a jejich výpočty jsou [6]:

$$MSE(\mathbf{Y}) = \frac{1}{N} \sum_{i=0}^{N-1} (Y_i - \overline{Y})^2$$

$$MAE(\mathbf{Y}) = \frac{1}{N} \sum_{i=0}^{N-1} |Y_i - \overline{Y}|$$

2.3.1.3 ID3

„ID3 (Iterative Dichotomiser 3) byl vyvinut v roce 1986 Rossem Quinlanem. Algoritmus vytvoří

rozhodovací strom pomocí hladového přístupu.“[7, překlad vlastní] Na vstupu se očekávají kategorická (diskrétní) data.

Nevýhody:

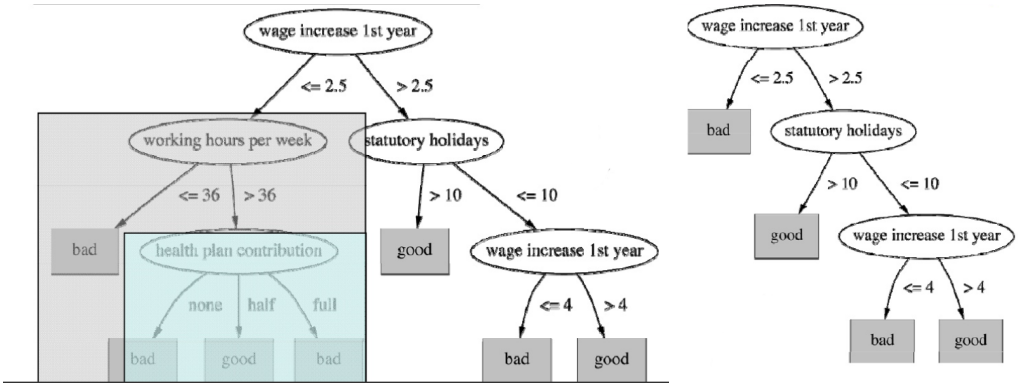
- Přeučení při malých datasetech.
- Neporadí si se spojitými daty a chybějícími hodnotami.

Algoritmus 3: Iterative Dichotomiser 3

Vstup: Matice příznaků, vektor vysvětlované proměnné
Výstup: Rozhodovací strom
if *žádná data* **then**
  return fail list
else if ∀ *příklady mají stejný výsledek* **then**
  return list, který vrací výsledek klasifikace
else if *je splněna podmínka hyperparametru* **then**
  return list, který vrací výsledek klasifikace
else
  A ← příznak nejlépe rozdělující vektor výsledků;
  r, l = split(A,X,y);
  levý_syn ← ID3(rX,rY);
  pravý_syn ← ID3(lX,lY);
end

2.3.1.4 C4.5, C5.0

Jedná se o vylepšeného nástupce ID3, který už dokáže pracovat i se spojitými daty i chybějícími hodnotami. Od svého předchůdce se liší technikou **prořezávání**, při které po konstrukci stromu prochází a odstraňuje zbytečné větve. Tato technika nám pomůže proti přeučení stromu. Existují dva druhy prořezávání, a to pre-prořezávání (při zjištění nespolehlivé informace přestaneme dále rozvíjet současnou větev) a post-prořezávání (nejdříve se zkonstruuje strom a zbytečné části se odstraní). [8]



Obrázek 2.6 Ukázka rozhodovacích stromů před a po prořezávání [8]

Algoritmus C4.5 nakonec ještě dokáže převést celý rozhodovací strom na pravidla. [8]

C5.0 je Quinlanova poslední verze algoritmu, která využívá méně paměti, generuje menší pravidla, ale je i přesnější než C4.5[7].

2.3.1.5 CART

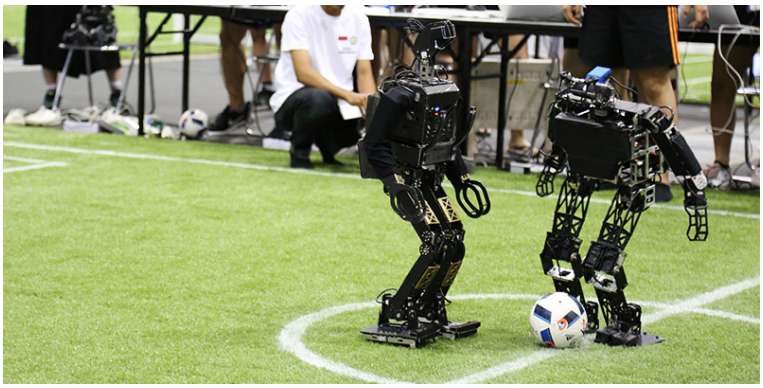
CART nebo-li Classification And Regression Trees je velice podobný algoritmu C4.5. Narozdíl od něj ale umí pracovat se spojitou vysvětlovanou proměnou (tj. regrese). „*CART konstruuje binární stromy pomocí prahových hodnot, které přinášejí největší zisk informací v každém uzlu.*“ [7, překlad vlastní]

2.3.2 Robotický fotbal

Fotbal (anglicky football) patří do kolektivních míčových sportů ve kterém proti sobě hrají dvě družstva. Zároveň se také řadí mezi nejpopulárnější sporty. Ve každém ze 2 družstev hraje 11 hráčů (10 v poli a 1 v bráně) a jejich cílem je vstřelit více gólů než soupeř. Gólem se rozumí přechod míče přes brankovou čáru celým jeho objemem. Hraje se na obdélníkovém hřišti, nejčastěji na travnatém povrchu. Hřiště je ohraničeno tzv. *outovou čárou*, za kterou se míč nesmí dostat (následuje vhazování). Hráči ke hře využívají zejména nohy, ale mohou používat veškeré tělo, kromě rukou. To však neplatí u brankáře, který může v blízkosti své branky hrát i rukama.

V robotickém fotbalu jsou pak lidští hráči nahrazeni roboty, kteří jsou adekvátně konstruováni. Neslouží jen k zábavě[3], ale spíše pro výzkum kooperativních robotů a multiagentních systémů v dynamickém a nepřátelském prostředí. Organizace RoboCup v rámci svých každoročních akcí RoboCupSoccer pořádá ligy nebo turnaje v několika kategoriích [9]:

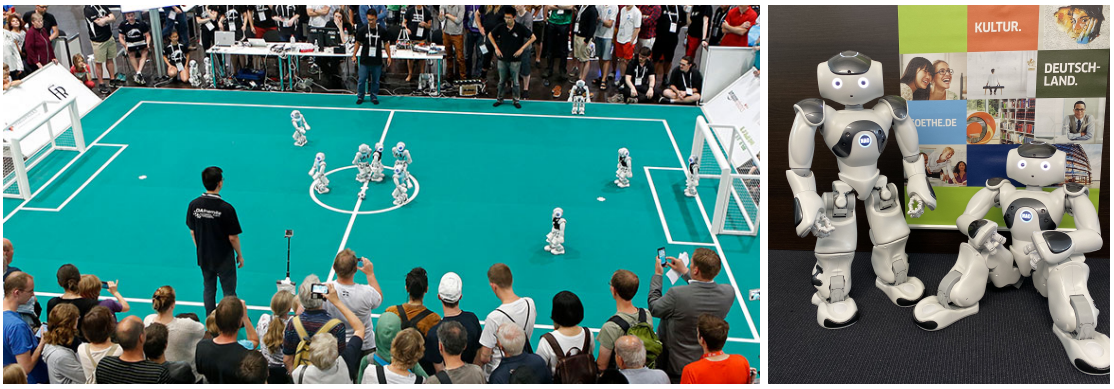
Humanoid – Tato kategorie se dál dělí na další tři podkategorie, a to na základě výšky robotů. Roboti zde chodí na vlastních nohách a svým vzhledem připomínají člověka. Zatím sice nedokáží udržet rovnováhu stejně jako člověk, ale jejich kopání, týmové strategie a rozpoznání ostatních robotů a hřiště se každý rok zlepšují. „*Očekává se, že liga humanoidů podstatně přispěje k vývoji důležitých technologií pro budoucí vývoj humanoidů.*“ [9]



Obrázek 2.7 Humanoid liga v roce 2019 [9]

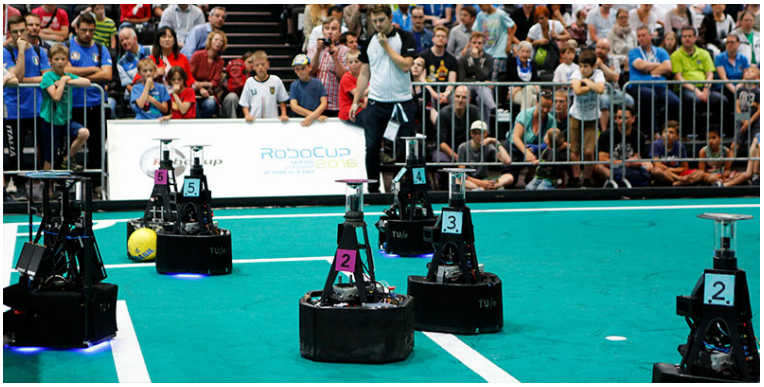
[3] Narozdíl od ryze lidského fotbalu.

Standart platform – V této kategorii všichni účastníci soutěží se stejnými modely robotů, konkrétně používají NAO komunikačního robota (viz. 2.8). Všichni tedy mají stejné hardwarové podmínky. „*Robot NAO má omezené pohybové schopnosti, protože nebyl designovaný pro robortický fotbal. Roboti spolu komunikují a každý pak hraje pomocí svých autonomních rozhodnutí.*“ [9]



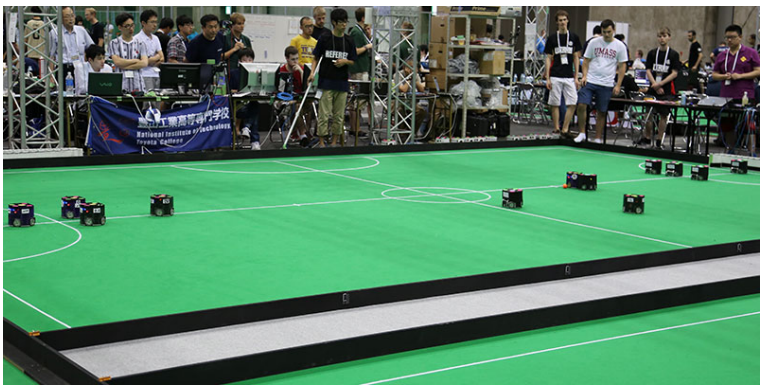
Obrázek 2.8 Standart platform liga v roce 2019 [9] a NAO komunikační robot [10]

Middle Size Teams – Dva týmy po pěti autonomních robotech, kteří jsou vysocí do 80 cm, soutěží v robotickém fotbalu na hřišti ze zeleného koberce o maximální rozměrech $14 * 22$ m. „*Middle Size Teams ligové hry používají stejně velký balon velikosti 5, jako se používá v lidském fotbalu a jeho barva je specifikována vždy před soutěží.*“ [9]



Obrázek 2.9 Middle Size Teams liga v roce 2019 [9]

Small Size Teams – Hry se odehrávají mezi dvěma týmy po osmi robotech. Průměr robota nesmí být větší než 18 cm a musí být nižší než 15 cm. Hraje se na menším hřišti než v kategorii Middle Size Teams, a to konrétně na koberci velikosti $12 * 9$ m. Jako míč se zde používá oranžový golfový míček. Kamery nejsou umístěny na robotech, ale nad celým hřištěm a informace z nich jsou posílány všem robotům. Nejatraktivnější věcí jsou zde rychlé pohyby a kooperace mezi agenty. „*Tato kategorie je doporučena pro fanoušky lidského fotbalu.*“ [9]



Obrázek 2.10 Small Teams liga v roce 2019 [9]

Simulation – Zápasy nejsou hrané mezi skutečnými roboty, ale v simulaci na serveru **SoccerServer** [9]. Hraje zde jedenáct hráčů používající virtuální senzory, které každému hráči dávají informace o prostředí (situace okolo hráče, pozice míče). Díky absenci hardwaru a problémů s ním spojených zde můžeme pozorovat sofistikované strategie.

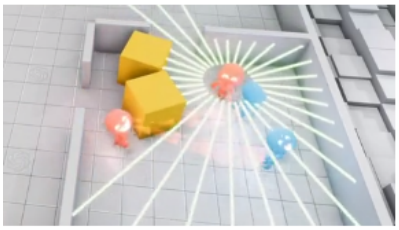
Ostatní – RoboCup nepořádá jen soutěže v robotickém fotbalu, ale i v jiných odvětvích spojených s robotickou. Jedním z nich je soutěž v zachraňování obětí pomocí robotů nebo v simulaci. Nakonec se také soutěží ve využití robotů v průmyslu nebo v logistice. [9]

2.4 Rešerše existujících řešení

Multiagentní systémy

Existuje spousta aplikací multiagetních systémů, ale jen malá část z nich využívá rozhodovací stromy pro řízení agentů. Velká část aplikací totiž využívá neuronové sítě.

OpenAI - Emergent Tool – Tento projekt pozoruje učení jednotlivých agentů ve známé hře „na schovávanou“. Jedná se o problém multiagentní koordinace v prostředí s nepřítelem s lokálním řízením agentů. Nacházejí se zde dva týmy (*Hledaní* a *Hledači*), kde každý má jiný cíl a jinou odměnovou funkci. *Hledaní* mají za úkol se schovat před *Hledači*. Před začátkem kola mohou využít prostředí nebo pohybovat s určitými předměty. *Hledači* jsou zpočátku na určitou dobu nehybní a poté se mohou pohybovat po místnosti a pohybovat s neuzamknutými předměty. V šesti fázích simulace se jednotlivé týmy vždy naučí porazit nepřátele díky tomu, že se naučí nové taktiky. K dostatečnému naučení agentů bylo potřeba provést řádově 10^{10} simulací. [11]



Obrázek 2.11 OpenAI - Emergent Tool - senzory agentů [11]

Project Malmo a MarLÖ – Projekt Malmo je platforma postavena nad oblíbenou hrou Minecraft a je designována k základnímu výzkumu AI. „*Vize projektu Malmo je umožnění AI technologiím spolupráci s lidmi*“. [12, překlad vlastní] Nad tímto frameworkem byl vytvořen projekt MarLÖ (Multi-Agent Reinforcement Learning in MalmÖ). Ve hře specifikuje různé úkoly a využívá MAS a posilované učení k jejich řešení. Jedná se například o:

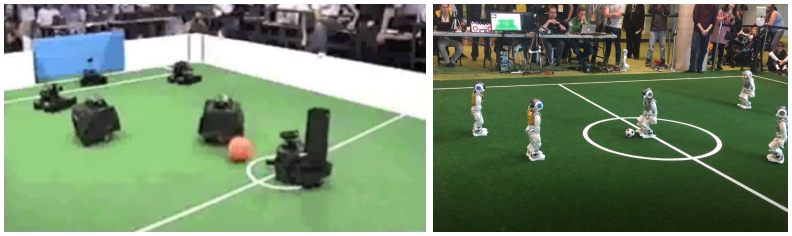
- Hledání pokladu – Jeden agent hledá nebo nese poklad a ostatní brání tým před nepřáteli
- Soutěž ve stavění – Agenti spolupracují při stavění určitého objektu a rychlejší agenti dostávají větší odměny
- Nahánění zvířat – Agenti musí spolupracovat k úspěšnému chycení zvířete (Tento úkol byl využit v soutěži *Malmo Collaborative AI Challenge*)

Tento projekt je Open Source. [13]

Robotický fotbal

Existující řešení, zabývající se robotickým fotbalem, můžeme rozdělit na dvě katogorie:

- Simulace a teoretické – Na základě znalostí informatiky, aplikované matematiky a umělé inteligence jsou vytvořeny programy, které pouze simulují hru. Neřeší se zde fyzická stránka robota.
- Praktické – Získané zkušenosti z teoretických poznatků a simulací se uplatňují při sestavování a návrhu skutečných robotických hráčů. Můžeme se zde setkat s ovlivňováním výsledků díky fyzikálním vlastnostem prostředí a různými nedeterministickými nedokonalostmi, které se těžko simulují.



Obrázek 2.12 Porovnání RoboCupu z roků 1998 [14] a 2019 [15]

RoboCup – „*Myšlenka robotického fotbalu byla poprvé zmíněna profesorem Alanem Mackworthem (Univerzita Britské Kolumbie, Kanada) v práci jménem On Seeing Robots a publikována v roce 1993 knize Computer Vision: System, Theory, and Applications*“. [16, překlad vlastní] Za méně než 5 let v roce 1997 byl uspořádán první RoboCup turnaj a konference. „*Zúčastnilo se přes 40 týmů a přišlo přes 5 000 diváků*“.[16, překlad vlastní] Od té doby jsou pořádány pravidelné akce zabývající se robotickým fotbalem z praktického hlediska [4]. V průběhu let čím dál více roboti vypadají „lidštěji“. Tento vývoj můžeme vidět na obrázku 2.12. Společným cílem poté je v roce 2050 sestavení plně autonomního týmu robotů a podle oficiálních FIFA pravidel porazit posledního výherce Mistrovství světa ve fotbale. [2, překlad vlastní]

[4] V roce 2020 se kvůli pandemii COVID-19 uskutečnily pouze online soutěže v simulacích.

Kapitola 3

Vlastní návrh

V této kapitole se budeme zabývat vlastní implementací softwarového prototypu v jazyce C#. Ukážeme zde popis prostředí a konkrétní návrh agentů. Také v této kapitole popíšeme relaxovaná pravidla robotického fotbalu a ukážeme rozdíly mezi tímto prototypem a reálným světem.

3.1 Prostředí

Celé prostředí je realizováno pomocí 2D mřížky. Agenti se po mřížce mohou pohybovat do osmi směrů (tj. i diagonálně). Diagonální pohyb je samozřejmě pomalejší kvůli zachování základních fyzikálních vlastností světa. Vzhledem k Russelovi a Norvigovi[3] můžeme toto prostředí klasifikovat jako:

Přístupné – Agent dokáže kompletně a jednotně zjistit informace o prostředí. Ví, na jak velkém hřišti se hraje, kde je brána protihráče a ví, kam nesmí. Tyto informace agent nezjistí sice hned, ale trvá mu to minimum času (pouze stačí získat data ze senzorů).

Deterministické – Narozdíl od reálného fotbalu je toto prostředí deterministické. Nepůsobí na něj žádné vnější fyzikální jevy a všechny akce zde mají jasný a garantovaný efekt.

Statické – Pouze akce agenta dokáží změnit prostředí. Bez agentů je prostředí neměnné.

Diskrétní/spojité – Na první pohled se zdá toto prostředí diskrétní. Na mřížce $m*n$ s k akcemi existuje konečný počet akcí. Neurčitost v klasifikaci prostředí nám přidává spojitý čas. Pokud nepovažujeme stejné akce, akorát provedené v jiném čase, za totožné pak je prostředí spojité. Pokud uvažujeme opak, prostředí je diskrétní.

Známé – Z pohledu agenta i designéra je prostředí známé. Známe rozměry hřiště (aspoň poměr), počet hráčů i počet protihráčů.

Protředí je rozměrově podobné reálnému fotbalovému hřišti a je rozděleno do 7 zón: Brána, pokutové území, obranné pásmo, střední pásmo, útočné pásmo, soupeřovo pokutové území a

soupeřeva brána. Kromě středního pásma jsou ostatní k sobě zrcadlově symetrické a každý tým je má uspořádané naopak. Celé hřiště je pak ohraničeno outovou čárou, na které se ještě může hrát. Kdyby měl míč překročit outovou čáru, zastaví se. Vhazování po odehrání mimo hřiště tedy není v tomto prototypu implementováno.

3.1.1 Míč

Míč jako takový sám o sobě nemůže provádět žádné akce. Pouze agenti za pomocí svých akcí mohou míč ovládat. Míč může agent vzít a pohybovat se s ním, nebo ho může odkopnout na určitou pozici s určitou silou. Během střely míč ztrácí na rychlosti a může se i zastavit před cílem.

K určení trajektorie míče jsme použili Bresenham Line-Drawing algoritmus[17]. Algoritmus se sice používá pro vykreslování přímek v pixelovém rastru, ale diskretizace trajektorie míče je podobný problém, jelikož se v našem prototypu míč pohybuje jenom po přímce.

„*Pokud omezíme rutinu kreslení čar tak, aby při vykreslování vždy zvyšovala x, je zřejmé, že po vykreslení bodu na* $(x, y$ *má rutina výrazně omezený rozsah možností, kam může umístit další bod: buď na* $(x + 1, y)$ *nebo* $(x + 1, y + 1)$.“[17] Nejdřív si definujme chybu ϵ, která říká, o kolik se reálná y souřadnice liší od vykreslené. Chyba ϵ se tedy pohybuje v intervalu $-0.5 < \epsilon < 0.5$. Při pohybu z x do $x + 1$ zvyšujeme reálnou hodnotu y o sklon přímky m. Vykreslíme[1] pixel na souřadnicích $(x + 1, y)$ pokud rozdíl mezi novou hodnotou a y je menší než 0.5.

$$y + \epsilon + m < y + 0.5$$

Jinak vykreslíme $(x+1, y+1)$. Poté budeme muset přepočítat ϵ. Tento postup využívá desetinná čísla, ale po jednoduchém vynásobení podmínky výše 2 a změnou x můžeme využívat jen celá čísla.

$$\epsilon + m < 0.5$$

$$\epsilon + \Delta y/\Delta x < 0.5$$

$$2 * \epsilon * \Delta x + 2 * \Delta y < \Delta y$$

Následující pseudokód 4 ukazuje vykreslování přímky v prvním oktantu a pouze za pomocí celých čísel.

▶ **Věta 3.1.** *Bresenhamův algoritmus je konečný a jeho časová složitost je* $O(n)$ *vzhledem k délce* n *výsledné přímky* p.

Důkaz. Algoritmus na vstupu získá dva body a iteruje přes souřadnice x, resp. y, od jednoho bodu k druhému s konstantní velikostí kroku = 1. Jelikož jsou oba vstupní body umístěné v nějakém prostoru, rozdíl jejich souřadnic x, resp. y, musí být konečný. Z toho vyplývá časová složitost úměrná délce výsledné přímky. □

Pro vykreslení přímky v jiných oktantech stačí patřičně prohodit x a y souřadnice nebo podmínku vynásobit změnou y.

[1] Ve skutečnosti pixely nevykreslujeme, ale ukládáme do pole, které vytvoří trajektorii míče.

Algoritmus 4: Bresenhamův algoritmus [17]

Vstup: $(x_1, y_1)(x_2, y_2)$
Výstup: Přímka p

```
1  dx = x2 − x1
2  dy = y2 − y1
3  y = y1
4  eps = 0
5  p = []
6  forall x ← x1; x ≤ x2 do
7      p.append((x,y))
8      eps += dy
9      if eps ≥ dx then
10         y++
11         eps − = dx
12     end
13 end
14 return p
```

3.2 Agent

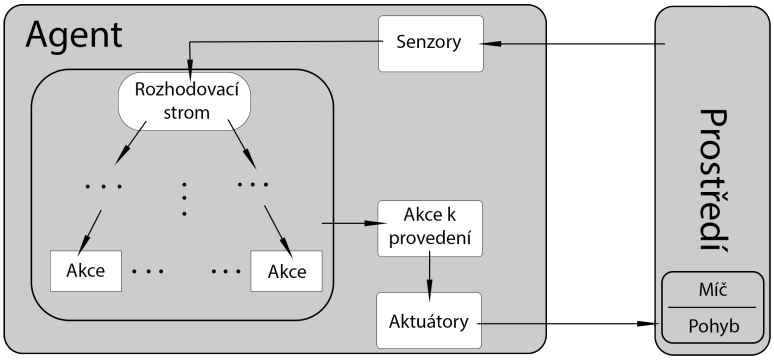
„*Při návrhu agenta musíme jako první specifikovat problém, co nejvíce to jde.*“[3, překlad vlastní] Problém můžeme popsat pomocí PEAS (Performance, Environment, Actuators, Sensors, česky Výkon, Prostředí, Aktuátory, Senzory)[3].

Tabulka 3.1 PEAS agenta

Typ Agenta	Měření výkonu	Prostředí	Aktuátory	Senzory
Robotický fotbalista	Vstřelené góly Min inkasovaných gólů	Fotbalové hřiště (mřížka) Míč Spoluhráči Protihráči	Nohy k pohybu Nohy k odkopnutí míče	Kamera

Přístupů k návrhu agentů existuje celá řada. K našim potřebám poslouží nejlépe reflexní agent, který reaguje na aktuální podněty a neuvažuje jejich historii (což se v reálném fotbalu nehodí – hráči se učí z chyb). Specifický návrh je vidět na obrázku 3.1. Diagram popisuje princip vnitřního fungování agenta: Nejdříve agent svými senzory získá informace o prostředí. Tyto informace jsou pak předány jako vstup rozhodovacímu stromu, který byl předem zkonstruován z trénovacích dat. Strom je vlastně funkce $f(x)$, která vrací rozhodnutí – v našem případě akci. Tato akce se předá ke zpracování aktuátorům agenta a provede se. Akce může[2] ovlivnit prostředí, a to jen svoji aktuální pozici nebo pozici míče.

[2]Když agent provede akci kopnutí míče, ale míč nemá, tak neovlivní prostředí.



Obrázek 3.1 Diagram vnitřního návrhu agenta

3.2.1 Akce

Agent může pomocí svých aktuátorů vykonávat akce a tím ovlivňovat prostředí. Všechny jeho akce se dají rozdělit na dvě kategorie:

Pohyb

Základní akce agenta. Umožňuje mu pohybovat se do osmi směrů. Může se pohybovat na určitou pozici nebo do určité zóny. K nalezení cesty je využíván algoritmus A* upravený pro hledání cesty v měnícím se prostředí (ostatní agenti).

Práce s míčem

Agent může míč vystřelit určitou silou na jemu určenou pozici, jemu určené zóny nebo některému z jeho spoluhráčů. Také může míč vzít a pohybovat se s ním.

Senzory

Tento agent má jediný senzor, a to kameru, ze které vypočítává veškeré informace. Dokáže z nich získat polohu spoluhráčů, protihráčů i míče. Senzor byl vybrán kvůli korespondenci s reálným fotbalem, kde hráči získávají nejvíce informací zrakem. V realitě sice hráči vnímají i pomocí sluchu a hmatu, ale pro tuto simulaci tyto senzory neuvažujeme.

3.3 Simulace spojitého času

V diskrétním modelu času robot potřebuje jednu jednotku času k pohybu na sousední pole.[2] Jelikož chceme pracovat s různými rychlostmi hráčů a míče, musíme využít model spojitého času. K jeho simulaci využijeme modifikovanou datovou strukturu **prioritní fronta** (dále jen fronta).

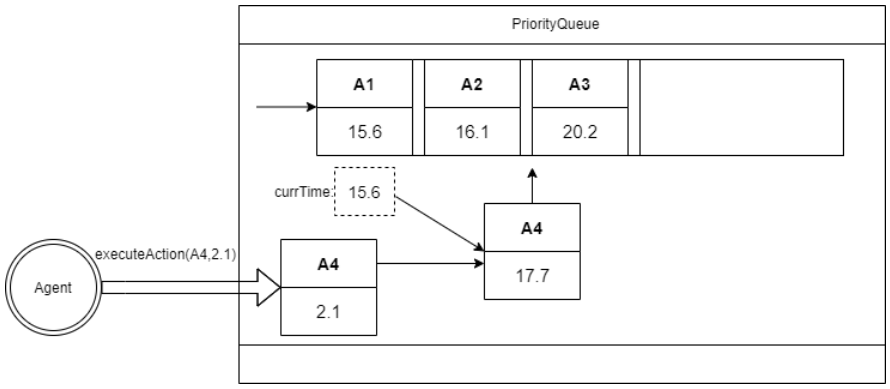
Výpis kódu 3.1 Abstraktní třída vrcholu v rozhodovacím stromu

```
public class CAction {
  Action action;
  float priority;
}
```

▶ **Definice 3.2.** *Fronta je abstraktní datová struktura, která ukládá prvky. Prvky z ní vystupují v pořadí, ve kterém přišly (typ FIFO = First-In, First-Out, česky První dovnitř, první ven). Z prioritní fronty vystupují nejdříve prvky s nejvyšší prioritou získané na základě prioritní funkce.* [3, překlad vlastní]

Do fronty budeme vkládat struktury reprezentující akce (kód 3.1):

Fronta, narozdíl od klasické definice, bude zpracovávat nejdříve prvky s nejnižší prioritou, která reprezentuje čas vykonání akce. Agenti při pokusu provedení nějaké akce, musí „vznést požadavek“ této frontě (pomocí struktury *CAction*, kde *priority* je doba trvání akce). Ta si musí udržovat hodnotu právě zpracovávaného času a tu přičte k prioritě příchozí struktury. Následně tuto strukturu vhodně umístí do své paměti (celý proces lze vidět na obrázku 3.2).



Obrázek 3.2 Proces vkládání nové akce do fronty

Fronta tedy postupně zpracovává požadavky a při jejich vyjmutí provede definovanou akci. Agent tedy provede akci a zároveň vnese nový požadavek frontě. Máme tedy zajištěno pracování s různými délkami trvání jednotlivých akcí a tím získáváme simulaci spojitého času.

Problém může nastat, když se ve frontě budou nacházet dvě akce se stejnou prioritou. V některých scénařích by mohlo zásadně záležet na pořadí provedení těchto kolizních akcí, ale v našem případě provedeme dřív akci, která do fronty přišla jako první.

3.4 Implementace rozhodovacích stromů

Rozhodovací strom se skládá z kořene, vnitřních vrcholů a listů. Pro všechny tyto uzly si vytvoříme abstraktní třídu, ze které budou dědit. Společné atributy tedy jsou (viz. 3.2) hloubka uzlu (jak daleko od kořene se nachází) a rozlišovací flag, jestli se jedná o vnitřní uzel.

Výpis kódu 3.2 Abstraktní třída vrcholu v rozhodovacím stromu

```
public abstract class DecisionTreeNode {
    protected bool isInner;
    public int depth;
}
```

Vnitřní vrchol (viz. 3.3) si musí udržovat informace o svých synech, které jsou typu *DecisionTreeNode*, takže se může jednat o vnitřní vrchol nebo list. Dále pak musí mít specifikovanou hodnotu, která se bude porovnávat se získanými daty od agenta při průchodu stromem (3.5). Vnitřní vrcholy musí mít definované oba syny, jinak můžeme při průchodu narazit na chybu (každý průchod stromem musí vracet akci, ale když není definovaný jeden syn, může nastat situace, kdy nemáme možnost vrátit akci).

Výpis kódu 3.3 Implementace vniřního vrcholu

```
public class InnerNode : DecisionTreeNode{
    public DecisionTreeNode trueSon;
    public DecisionTreeNode falseSon;
    public float value;

    public bool Check(float toCheck) => toCheck > value;
}
```

Listy narozdíl od vnitřních vrchoů už nemohou mít žádné potomky. Obsahují pouze určitou akci agenta (struktura CAction), kterou při průchodu stromem (zde průchod končí, viz. 3.5) předá frontě simulující spojitý čas. Listy mají ještě pomocnou metodu k získání jména akce, která ale není pro funkčnost důležitá, používá se jen při vizualizaci rozhodovacího stromu.

Výpis kódu 3.4 Implementace listu

```
public class LeafNode : DecisionTreeNode {
    private CAction _cAction;

    public void Execute() => _cAction.Commit();
}
```

Při spuštění programu nebo po vykonání nějaké akce se opět prochází rozhodovací strom agenta. Nejdříve musíme získat data z jeho senzorů. Poté začneme procházet strom od kořene. Pokud se jedná o vnitřní vrchol, porovnáme určitý příznak ze získaných dat s hodnotou vrcholu. Porovnání probíhá pouze pomocí operátoru $>$. Při použití inverzního operátoru se pouze prohodí synové všech vnitřních uzlů, takže není potřeba implementovat použití libovolného operátoru.

Výpis kódu 3.5 Funkce pro průchod rozhodovacím stromem

```
public void Proceed(){
    RequestData(); //Get data from agent
    DecisionTreeNode node = _root;
    while (true) {
```

```
        if (!node.IsInner()) {
            LeafNode n = (LeafNode) node;
            n.Execute(); // Execute LeafNode action => Add to TimeQueue
            break;
        }
        else{
            InnerNode n = (InnerNode) node;
            // Compare data and Node value
            // Choose correct son (trueSon/falseSon)
            if (n.Check(_data[n.indexToCheck]))
                node = n.trueSon;
            else
                node = n.falseSon;

        }
    }
}
```

3.5 Hledání cest

Agenti ke svému pohybu do určitého místa potřebují nějaký algoritmus k nalezení cesty. Existuje řada těchto algoritmů např.: BFS, DFS, Greedy Search, A*, Dijkstra, atd. Ne všechny algoritmy dokážou najít nejkratší cestu (DFS, Greedy Search) nebo potřebují více dostupných prostředků než je skutečně potřeba. Pro náš případ jsme zvolili A*, a to hlavně kvůli našemu prostředí, což je $m * n$ mřížka. Princip fungování A* můžeme nejlépe popsat na ukázaní rozdílů mezi ním a Greedy Searchem. Nejprve ale musíme definovat pojem heuristika a zmínit některé vlastnosti, protože se v obou algoritmech využívá. Také uvažujme, že cestu hledáme v prostoru $\mathcal{X}$ a prvky $x \in \mathcal{X}$ jsou nějaké pozice, mezi kterými můžeme měřit vzdálenost funkcí $d(x_1, x_2)$.

▶ **Definice 3.3** (Heuristika)**.** *„Heuristika $h(x)$ je (v kontextu hledání nejkratších cest) odhadovaná cena (délka) nejlevnější (nejkratší) cesty z určené pozice do cíle.“* [3, překlad vlastní]

Pro popis vlastností si definujeme **optimální heuristiku** $h^*(x)$, která vrací skutečnou cenu (vzdálenost) od pozice k cíli. Tato heuristika sice existuje, ale vypočítat jde pouze hrubou silou, tudíž se to výpočetně nevyplatí. Lepší heuristiku než $h^*(x)$ nelze sestavit.

▶ **Definice 3.4** (Přípustnost heuristiky)**.** *„Přípustná heuristika je taková, která nikdy nenadhodnotí cenu (vzdálenost) do cíle.“*[3, překlad vlastní] Formálně:

$$\forall x \in \mathcal{X} : h(x) \leq h^*(x)$$

Přípustné heuristiky jsou někdy taky označovány jako optimistické.

▶ **Definice 3.5** (Konzistence (Monotónnost) heuristiky)**.** *„Heuristika je konzistetní, jestliže pro každou pozici x a každého jeho souseda x' je odhadovaná cena (vzdálenost) do cíle z x není větší*

než reálná cena (vzdálenost) z x do x' plus odhadovaná cena (vzdálenost) do cíle z x'."[3, překlad vlastní] Formálně:

$$\forall x, x' \in \mathcal{X} : h(x) - d(x, x') \leq h(x')$$

▶ **Věta 3.6.** *Algoritmus A* je konečný a s monotónní heuristikou je jeho výstupem optimální cesta.*

Důkaz. Algoritmus iteruje přes množinu *open*, dokud není prázdná. Při každé iteraci odstraníme jeden prvek x a do *open* vložíme jeho sousedy, kteří nejsou v množině *closed*. Nakonec x vložíme do množiny *closed* $\implies$ žádný prvek nebude v množině *open* víckrát, než jednou a jelikož je počet prvků konečný, algoritmus se zastaví. Důkaz optimality viz [3, s. 95] □

Algoritmus 5: A*

```
   Vstup: start, target
   Výstup: Nejkratší cesta
 1 open ← new HashSet()
 2 closed ← new HashSet()
 3 open.Insert(start)
 4 while !open.Empty() do
 5     x ← open.ExtractMin()
 6     if x == target then
 7         return ReconstructPath(x)
 8     end
 9     forall y ∈ x.Neighbors() ∧ y ∉ closed do
10         temp_d ← GetDistance(x,y) + x.gCost
11         if y ∉ open ∨ temp_d < y.gCost then
12             y.prev = x
13             y.gCost = temp_d
14             y.fCost = y.gCost + GetDistance(start,y)
15             if y ∉ open then
16                 open.Insert(y)
17         end
18     end
19     closed.Insert(x)
20 end
```

Algoritmus musíme modifikovat kvůli hledání cesty v prostředí s více agenty [18]. V podmínce v 5 na 11. řádce stačí přídat ke konjunkci kontrolu obsazenosti pozice. To znamená upravit podmínku následovně: **if**((y ∉ open ∨ temp_d < y.gCost) ∧ y.Empty())

3.5.1 Rychlostní optimalizace

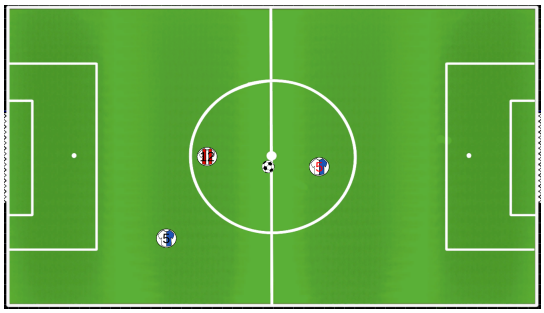
Funkce pro hledání cesty se musí volat před každým provedeným pohybem. Nejvíce času ($O(n)$) se pak spotřebuje na hledání minima na řádku 5. Musíme tedy vybrat novou a vhodnou strukturu pro množninu *open*. Takovouto strukturou je minimální binární halda. Časová složitost řádku 5 se tedy změní na $O(\log n)$. Touto optimalizací však zaplatíme zpomalením vkládáním do *open* na řádku 16, z $O(1)$ na $O(\log n)$.

Kapitola 4

Získání dat a experimenty

V této kapitole se budeme zabývat získáváním potřebných dat (testováním) k supervizovanému učení, konkrétně pro konstrukci rozhodovacích stromů. Popíšeme zde rozdělení účastníků, postup získání dat a jejich strukturu. Nakonec, na základě těchto dat, budeme provádět experimenty se vzniklými agenty.

Vzhledem ke specifičnosti tématu a netriviálnosti získání dat jsme nezvolili hromadné a anonymní testování. Softwarový prototyp nemá webovou verzi a tím se testování ztížilo. Testování tedy bylo uskutečněno kontaktně nebo online s dozorem s menším počtem lidí.



Obrázek 4.1 Ukázka situaci ze SW prototypu využitého k získání dat

4.1 Popis testované skupiny

Celého testování se zúčastnilo 12 lidí. Vzhledem k tématu práce se nabízelo prozkoumat rozdíl mezi daty aktivních hráčů fotbalu (dále jen fotbalisté) a lidí nehrajících fotbal (dále jen nefotbalisté). Skupinu tak tvoří 6 fotbalistů a 6 nefotbalistů. Testování napříč skupinami se nijak nelišilo, bylo úplně totožné.

4.2 Získání dat

K získání dat nám posloužily předem nadefinované a jednoduché situace. Na každou roli agenta připadají 2-4 situace. Uživatel ovládá jednoho agenta podle instrukcí, které jsou dopředu známy. Při každém vstupu od uživatele se do souboru uložila data ze senzoru ovládaného agenta. Každá akce se ale jednoduše nedá převést na číslo. Nejdříve musíme podle typu vstupu analyzovat danou akci. To znamená že např. při zmáčknutí šipek víme, že se jedná o pohyb, ale musíme ho blíže specifikovat (může se jednat o pohyb k míči, nebo pohyb do určité zóny hřiště). To samé platí o vstupu z myši, kdy se jedná o akci s míčem.

Tabulka 4.1 Struktura dat

Jméno příznaku	Datový typ	Ukázka dat
role	enum - $\{0, 1, 2, 3\}$	1
hasBall	bool - $\{0, 1\}$	0
agentX	int	15
agentY	int	17
closestEnemyDistance	float	10.63
closestEnemyX	int	10
closestEnemyY	int	3
enemyHasBall	bool - $\{0, 1\}$	0
enemyDistanceToBall	float	3
zone	enum - $\{0 \dots 7\}$	5
enemyDistanceToClosestPlayer	float	10.63
enemyDistanceToSecondPlayer	float	5
enemyWithBallDistance	float	$2.15 * 10^9$

V určitých situacích není možné určité příznaky zjistit nebo dopočítat. Například když protihráč nemá míč, nelze zjistit jeho vzdálenost s míčem. Nebo v určitých situacích nejsou spoluhráči k dispozici, tudíž opět nemůžeme zjistit jejich vzdálenosti. Aby nedocházelo k problémům s chybějícími daty, neznámým příznakům přiřadíme vždy maximální hodnotu 32 bitového celého čísla $= 2.15 * 10^9$

4.3 Experimenty

V experimentální části práce budeme porovnávat týmy sestavené z různých typů agentů a to z:

Syntetických agentů – Agenti, kteří nejsou řízeni rozhodovacím stromem, ale jejich chování je pevně naprogramované v kódu.

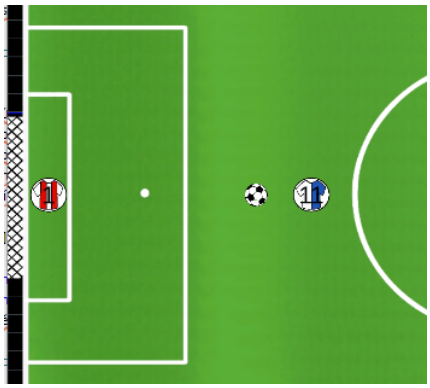
Natrénovaných agentů – Agenti jsou řízeni rozhodovacím stromem, který je automaticky zkonstruovaný ze získaných dat pomocí algoritmu ID3. Tuto kategorii ještě můžeme rozdělit na další tři, kde budeme strom konstruovat na základě dat pouze od fotbalistů, nefotbalistů a z obou skupin dohromady.

Ručně vytvořených agentů – Agenti jsou sice řízeni rozhodovacím stromem, ale ručně vytvořeným.

V experimentech budeme nejdříve testovat jednoduchou situaci, a to střelu na brankáře. Poté otestujeme hru s celým týmem. Při testování střelby budeme sledovat závislost maximální hloubky rozhodovacího stromu na úspěšnost a při testování hry s celým týmem budeme testovat agenty s pouze jednou, předem určenou, maximální hloubkou stromu.

Střely na bránu

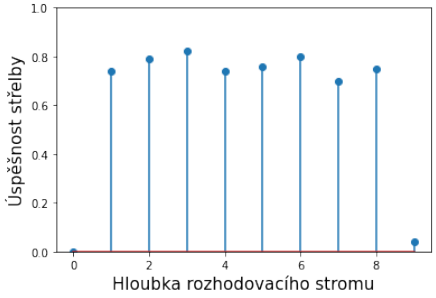
První experimentem je testování úspěšnosti střelby. Tuto situaci (viz. Obrázek 4.2) můžeme přirovnat k reálnému fotbalu a to konkrétně k pokutovým kopům (penaltám).



Obrázek 4.2 První experiment: Střely na bránu

Golman vs. útočník

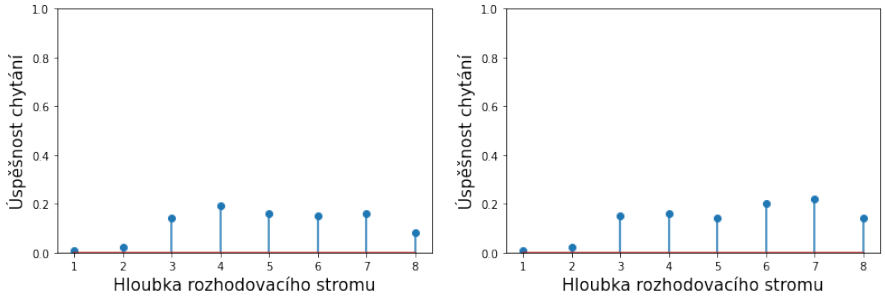
Nejdříve otestujeme přesnost agenta s zkonstruovaným stromem ze všech dat. Přesnost budeme testovat s různými maximálními hloubkami rozhodovacího stromu. Jak vidíme z grafu 4.3, maximální hloubka stromu nemá na střelbu vliv. Přesnost je nízká jen v krajních hodnotách. Při hloubce stromu rovné 9 je přesnost 4%, kdy se jedná o náhodný šum, protože se agent pokoušel nahrát spoluhráči a občas se podařilo dát gól.



Obrázek 4.3 Přesnost hráče v závislosti na hloubce rozhodovacího stromu

Útočník vs. golman

Dále budeme testovat gólmana. Situace bude velmi podobná s předchozí, jen budeme sledovat úspěšnost chytání míče agenta s automaticky vytvořeným rozhodovacím stromem. Na grafech 4.4 vidíme úspěšnost gólmana, kdy na levém grafu je znázorněna úspěšnost proti syntetickému hráči a v pravém grafu vidíme úspěšnost proti natrénovanému hráči z dat.



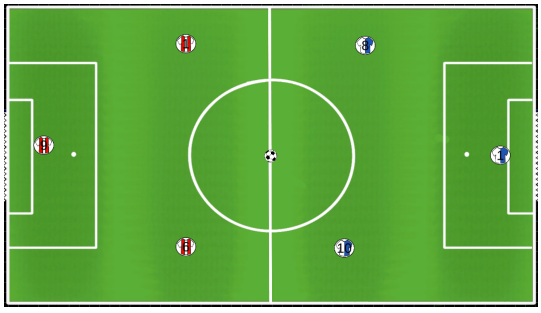
Obrázek 4.4 Úspěšnost gólmana v závislosti na hloubce rozhod. stromu

Tyto úspěšnosti se zásadně neliší a při tvorbě stromu gólmana má maximální hloubka větší vliv na výsledky než u střelce.

Výsledky můžeme porovnat s reálnou statistikou pokutových kopů. Ze 100 000 kopů skončilo v bráně 75%, 17.5% střel bylo chyceno gólmanem, 3.5% trefilo brankovou konstrukci a 4% střel vůbec netrefilo bránu [19]. Reálné statisktiky se velmi podobají přesnostem a úspěšnostem našich agentů.

Klasická hra

V tomto testu budeme měřit úspěšnost agentů podle vstřelených/inkasovaných gólů. Od reálného zápasu fotbalu se bude lišit jiným počtem hráčů (minimálně tři, maximálně pět), kvůli *zaseknutí* simulace při větším počtu agentů.



Obrázek 4.5 Druhý experiment: Klasická hra se třemi hráči

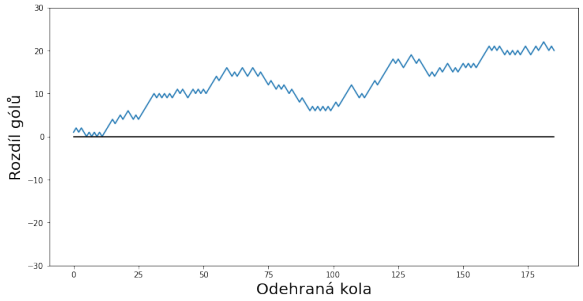
Dále se budou lišit pozice všech hráčů po vstřeleném góly. V lidském fotbalu rozehrává tým, který inkasoval gól. V této simulaci se míč po gólu umístí do středu hřiště a agenti jsou rozmístěni na své určené pozice (tj. nikdo nerozehrává). Díky této změně, se celá jedna simulace, např. s

maximálním skóre sto, tváří jako sto simulací s maximálním skóre jedna. Tím pádem dosažené skóre v průběhu simulace nemá vliv na další průběh.

Syntetičtí vs. Natrénovaní agenti

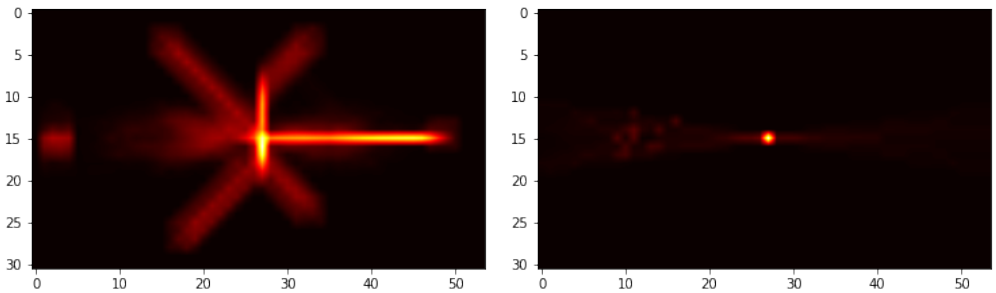
Syntetický agent má specifikované chování přímo ve zdrojovém kódu. Nevyužívá žádné heuristiky, ani žádné metody umělé inteligence. Útočník se pouze pohybuje směrem k míči a při kontaktu s ním se snaží vystřelit na bránu. Brankář kopíruje pozici míče pouze v souřadnicích y a při kontaktu se snaží míč chytit nebo odkopnout.

Rozhodovací stromy natrénovaných agentů budeme tvořit nejdříve z dat pouze od fotbalistů, poté od nefotbalistů a nakonec od všech dohromady.



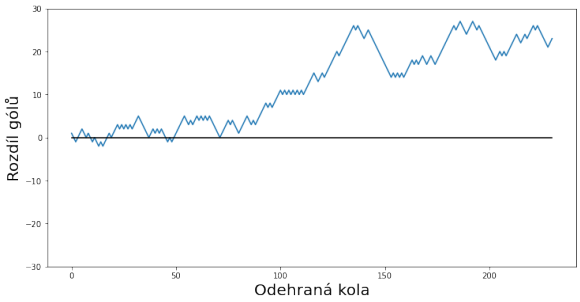
Obrázek 4.6 Vývoj skóre: data fotbalisté

Z grafu 4.6 můžeme vyčíst přehlednou výhru natrénovaných agentů. V polovině odehraných kol můžeme pozorovat mírný výkyv ve prospěch syntetickách agentů, který ale nějak významně neohrozil pozitivní trend vývoje skóre. Na grafech 4.7 můžeme vidět *heatmapu* znázorňující pohyb agentů a míče. Jako zajímavost můžeme vyzdvihnout chování pravého gólmana, který se místo čekání v bráně připojil k útočící dvojici.



Obrázek 4.7 Pohyb agentů a míče: data fotbalisté

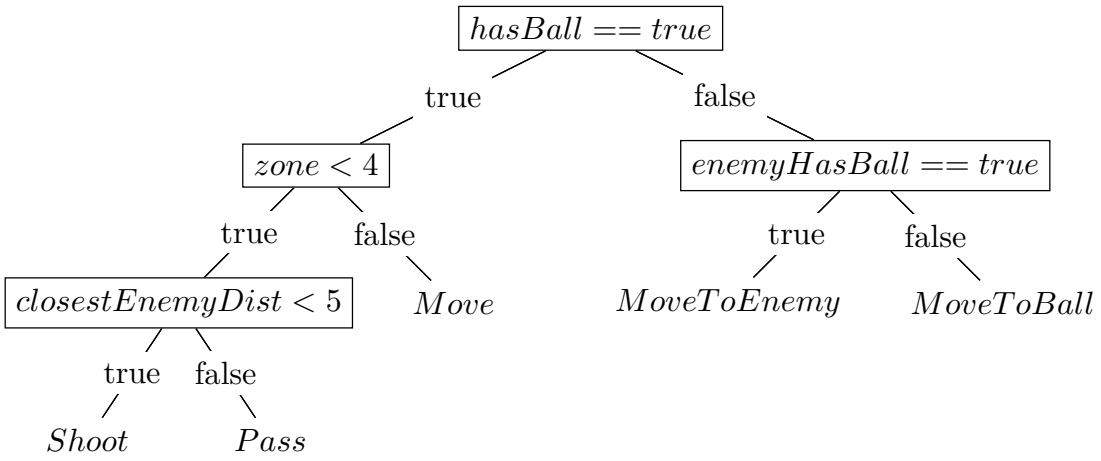
Pokud použijeme data od nefotbalistů, dočkáme se velmi podobných výsledků i velmi podobných *heatmap* znázorňující pohyb. Na grafu 4.8 vidíme průběh simulace, který skončil podobně jako s daty fotbalistů.



Obrázek 4.8 Vývoj skóre: data nefotbalisté

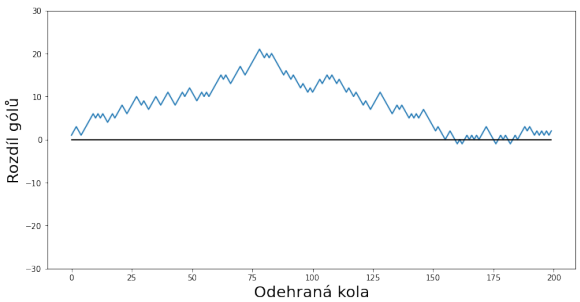
Ručně vytvoření vs. Natrénovaní agenti

Ručně vytvoření agenti jsou agenti se stejným vnitřním mechanismem jako natrénovaní, ale strom jsme vytvořili ručně. Strom jsme vytvářeli jednoduchý, jen podle pouhé intuice a po krátkém zamyšlení.



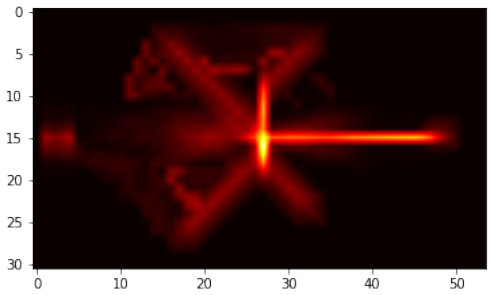
Obrázek 4.9 Schéma ručně vytvořeného rozhodovacího stromu

Na grafu 4.10 můžeme zezačátku vidět pozitivní trend pro natrénované agenty. I přes to simulace skončila s velmi těsným skórem. Můžeme zde vypozorovat, že i jednoduchý rozhodovací strom se vyrovná automaticky vytvořenému.



Obrázek 4.10 Vývoj skóre

Chování natrénovaných agentů na obrázků 4.11 vypadá téměř totožně, jak v předchozím experimentu na obrázku 4.7. Zato chování ručně vytvořených agentů vypadá zajímavěji než všichni předchozí. Vidíme lepší využití prostoru i jiné pohyby než pouze směřující do středu hřiště.



Obrázek 4.11 Pohyb agentů

4.4 Výsledky

Agenti s rozhodovacími stromy, které byly zkonstruovány algoritmem ID3 a pomocí získaných dat, byli využiti v několika experimentech. Nejdříve jsme testovali úspěšnost střelby v situacích podobným pokutovým kopům. Dále jsme ve stejné situaci testovali gólmana a jeho úspěšnost chytání. Získané úspěšnosti jsou velice podobné statistikám z reálného fotbalu získané z [19].

Dalším experimentem byla normální hra tří proti třem proti syntetickým agentům a proti ručně vytvořeným. Syntetické agenty se podařilo porazit o značný rozdíl, ale s ručně vytvořenými se už jednalo o vyrovnanou hru. Za velké zklamání považujeme celkový pohled na hru a pohyb automaticky vytvořených agentů. Na vizualizacích 4.7 a 4.11 můžeme vidět relativně monotónní pohyby, které jen směřují do středu hřiště.

V mnoha simulacích docházelo k „patovým situacím“, kdy si pouze hráči navzájem do nekonečna brali od sebe míč. Kvůli tomutu jsme do prototypu museli přidat možnost ručního i automatického restartování kola. Tyto „patové situace“ skóre nijak neovlivňovaly.

Data ke konstrukci rozhodovacích stromů jsme získavali od dvou skupin: fotbalistů a nefotbalistů. Po experimentech se neukázal žádný rozdíl mezi použití dat od jednotlivých skupin nebo všech najednou.

Kapitola 5

Závěr

Robotický fotbal je jedno z probíraných praktických témat na poli umělé inteligence. Cílem společnosti je do roku 2050 postavit robotické družstvo, které porazí lidské hráče – budoucího vítěze Mistrovství světa ve fotbale 2050.

Úkolem této práce bylo navrhnout vlastní nebo modifikovat existující techniku lokálního řízení hráčů robotického fotbalu. Tento návrh poté implemenotvat formou softwarového prototypu a provádět experimenty s automaticky řízenými hráči v různých scénářích. Na základě výzkumu byly zvoleny rozhodovací stromy.

Ve vytvořeném softwarovém prototypu bylo možné hrát ve vytvořených situacích a tím tak sbírat data o hře. Také se zde, na základě těchto dat, dají konstruovat rozhodovací stromy pro vnitřní mechanismus agentů a následně tyto agenty pozorovat v simulaci.

Vytvoření agenti podléhali různým experimentům. Při testování úspěšnosti střelby a chytání golmana jsme se přiblížili reálným statistikám z lidského fotbalu. Při testování celého týmu v normální hře jsme agenty postavili proti naprogramovaným a ručně vytvořeným agentům. Syntetické agenty se podařilo porazit s velkým náskokem, ale s ručně vytvořenými skončíla simulace vyrovnaně. Experimenty neprokázaly rozdíly mezi agenty zkontruovaných z dat od různých skupin respondentů. Za největší zklamání experimentl považujeme až monotónní pohyb agentů.

V budoucnosti by bylo možné aplikaci rozšířit o další přístupy k lokálnímu řízení agentů v multiagentních systémech nebo se na tento problém podívat úplně z jiného pohledu. Také by se dalo přemýšlet o využití již existujících řešeních v jazyce Python a to konkrétně v balíčku scikit-learn.

Příloha A

Použitý Engine

Unity3D - Unity je multiplatformní herní engine vyvinutý společností Unity Technologies. Byl použit pro vývoj her pro PC, konzole, mobily a web. První verze podporovala pouze OS X a byla představena na celosvětové konferenci Applu v roce 2005. Od té doby byl rozvinut o více než patnáct dalších platforem. Pro osobní a nekomerční použití je zdarma.

Bibliografie

1. WOOLDRIDGE, Michael J. *An introduction to multiagent systems*. 2. vyd. Chichester: Wiley, 2009. ISBN 978-0-470-51946-2.
2. SHOHAM, Yoav, LEYTON-BROWN, Kevin. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge: Cambridge University Press, 2009. ISBN 978-0-521-89943-7.
3. RUSSELL, Stuart J., NORVIG, Peter. *Artificial Intelligence: A Modern Approac*. 3. vyd. Upper Saddle River: Prentice Hall, 2010. ISBN 0-13-604259-7.
4. SCHWAB, Devin, ZHU, Yifeng, VELOSO, Manuela. Zero shot transfer learning for robot soccer. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems* [online]. 2018, s. 2070–2072 [cit. 2021-05-08]. Dostupné z: `http://ifaamas.org/Proceedings/aamas2018/pdfs/p2070.pdf`.
5. PEDREGOSA, Fabien et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* [online]. 2011, roč. 12, s. 2825–2830 [cit. 2021-04-25]. Dostupné z: `https://jmlr.csail.mit.edu/papers/volume12/pedregosa11a/pedregosa11a.pdf`.
6. KLOUDA, Karel, VAŠATA, Daniel. *FIT ČVUT: Předmět Vytěžování znalostí z dat - Rozhodovací stromy* [online]. 2021 [cit. 2021-05-10]. Dostupné z: `https://courses.fit.cvut.cz/BI-VZD/@B201/lectures/files/BI-VZD-01-cs-handout.pdf`.
7. ÖZER, Mehmet Ali. scikit-learn: Decision Trees [online]. 2021 [cit. 2021-04-25]. Dostupné z: `https://github.com/scikit-learn/scikit-learn/blob/main/doc/modules/tree.rst`.
8. PUIG, Albert Orriols i. *Introduction to Machine Learning: C4.5* [online]. 2009 [cit. 2021-05-10]. Dostupné z: `https://www.slideshare.net/aorriols/lecture6-c45`.
9. *RoboCup Sydney 2019 official website* [online]. 2019 [cit. 2021-05-08]. Dostupné z: `https://2019.robocup.org`.
10. *Czech Institute of Informatics, Robotics and Cybernetics official website* [online]. 2020 [cit. 2021-05-08]. Dostupné z: `https://www.ciirc.cvut.cz/robots-in-residence-robota-nao-v-goethe-institutu-programuji-vedkyne-z-ciirc-cvut/`.

11. BAKER, Bowen et al. Emergent Tool Use From Multi-Agent Autocurricula. *CoRR* [online]. 2019, roč. abs/1909.07528 [cit. 2021-04-25]. Dostupné z: http://arxiv.org/abs/1909.07528.

12. MICROSOFT RESEARCH. Project Malmo. *Microsoft Research official website* [online]. 2021 [cit. 2021-04-25]. Dostupné z: https://www.microsoft.com/en-us/research/project/project-malmo/.

13. MOHANTY, Sharada. *Project Marlo: Multi Agent Reinforcement Learning using MalmÖ* [online]. 2018 [cit. 2021-04-25]. Dostupné z: https://github.com/crowdAI/marLo.

14. ROBOCUP. *RoboCup 1997-2011 HD video* [online]. Youtube [video], 2018 [cit. 2021-04-25]. Dostupné z: https://www.youtube.com/watch?v=4QtBSDSC2pk&t=98s&ab_channel=RoboCup.

15. AUSSIEDIARIES. *Robots playing Soccer for Robocup 2019 — Sydney, Australia* [online]. Youtube [video], 2019 [cit. 2021-04-25]. Dostupné z: https://www.youtube.com/watch?v=Bam9WzQbtfM&ab_channel=AussieDiaries.

16. *RoboCup Federation official website* [online]. 2016 [cit. 2021-04-25]. Dostupné z: https://www.robocup.org/.

17. FLANAGAN, Colin. The Bresenham Line-Drawing Algorithm [online]. 1996 [cit. 2021-04-25]. Dostupné z: The%20Bresenham%20Line-Drawing%20Algorithm.

18. IVANOVÁ, Marika, SURYNEK, Pavel. Area Protection in Adversarial Path-Finding Scenarios with Multiple Mobile Agents on Graphs: a theoretical and experimental study of target-allocation strategies for defense coordination. *arXiv preprint arXiv:1708.07285* [online]. 2017 [cit. 2021-05-02]. Dostupné z: https://arxiv.org/abs/1708.07285.

19. *SQaF: Sports Quotes and Facts website* [online]. 2021 [cit. 2021-05-10]. Dostupné z: https://sqaf.club/what-percentage-of-penalties-are-scored-stats/.

Obsah přiloženého média

- `readme.MD` ..... stručný popis obsahu média
- `exe`
  - `DataPrototype` ..... adresář se spustitelnou formou implementace
  - `SimulationPrototype` ..... adresář se spustitelnou formou implementace
- `src`
  - `dataProcces` ..... zdrojové kódy práce s daty
  - `robotic-football` ..... projekt a zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `images` ..... adresář s obrázky
- `vid` ..... adresář s ukázkou běhu programu