UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

ROBOTICKÝ SYSTÉM PRE HRANIE STOLOVEJ HRY

2015 Peter Pukančík

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

ROBOTICKÝ SYSTÉM PRE HRANIE STOLOVEJ HRY

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Mgr. Pavel Petrovič, PhD.

Bratislava 2015 Bc. Peter Pukančík

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Peter Pukančík
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov: Robotický systém pre hranie stolovej hry

Cieľ: Cieľom je navrhnúť, zostrojiť, implementovať a odladiť robotický systém, ktorý s ľudským spoluhráčom alebo samostatne dokáže hrať stolové hry využitím vhodných algoritmov napr. z oblasti umelej inteligencie a strojového učenia. Súčasťou robota bude systém počítačového videnia, robotické rameno, softvér s používateľským rozhraním a systém autonómneho rozhodovania pri hraní hry.

Literatúra:
1. E. B. Corrochano: Geometric Computing for Perception Action Systems, Springer 2001.
2. C. M. Bishop: Pattern Recognition and Machine Learning, Springer, 2006.

Vedúci: Mgr. Pavel Petrovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. PhDr. Ján Rybár, PhD.

Dátum zadania: 15.10.2012

Dátum schválenia: 29.11.2012

doc. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestne prehlasujem, že som diplomovú prácu vypracoval samostatne, pod vedením vedúceho diplomovej práce, a za použitia uvedených zdrojov a literatúry.

Bratislava 4.5.2015 ….....

Peter Pukančík

Poďakovanie

Veľká vďaka patrí vedúcemu diplomovej práce Mgr. Pavlovi Petrovičovi, PhD., za množstvo hodín strávených konzultáciami, za usmernenia, pomoc a odborné vedenie pri tvorbe tejto diplomovej práce.

ABSTRAKT

Pukančík, Peter. Robotický systém pre hranie stolovej hry. Diplomová práca. Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky a informatiky. Katedra aplikovanej informatiky. Školiteľ: Mgr. Pavel Petrovič, PhD.

V práci sa venujeme návrhu a implementácii komplexného herného systému pre rôzne stolové hry. Najskôr predstavíme úvod do problematiky, popíšeme si teoretické informácie, prístupy a myšlienky, ktoré nám pomôžu pochopiť jednotlivé funkcionality komponentov výsledného systému: inverzná kinematika pre robotické rameno, kalibrácia kamery pre počítačové videnie, reprezentácia ľubovoľnej stolovej hry a zadefinovanie jej pravidiel v jazyku logického programovania, a použitie algoritmov umelej inteligencie pri procese hrania danej stolovej hry. Práca prejde všetkými fázami tvorby softvéru, čiže špecifikácia požiadaviek, návrh, implementácia a testovanie systému.

Kľúčové slová:
herný systém, stolové hry, robotické rameno, inverzná kinematika, logické programovanie, ASP, umelá inteligencia, algoritmus minimax, alfa-beta orezávanie

ABSTRACT

Pukančík, Peter. Robotic system for playing a board game. Master's Thesis. Comenius University in Bratislava. Faculty of mathematics, physics and informatics. Department of applied informatics. Diploma Thesis supervisor: Mgr. Pavel Petrovič, PhD.

This thesis deals with designing and implementation of nontrivial gaming system for playing various board games. At first, we present introduction to thesis theme and describe theoretical ideas and approaches that will help us understand functionality of each component from final system: inverse kinematics for robotic arm, camera calibration for computer vision, representation of arbitrary board game and its game rules in the language of logic programming, and application of algorithms of artificial intelligence in the game process. The whole software development process of our system is described in this thesis that means Requirements, Design, Implementation and Testing.

Key words:
gaming system, board games, robotic arm, inverse kinematics, logical programming, ASP, artificial intelligence, minimax algorithm, alpha-beta pruning

OBSAH

ÚVOD

Predkladaná práca sa venuje prieniku atraktívnych tém zahŕňajúcich robotiku, umelú inteligenciu a hry. Samotná robotika je už dospelý odbor, ktorý vo svojom primárnom zameraní dovoľuje efektívne vyrábať napríklad v automobilovo priemysle, či posielať robotické sondy na vzdialené planéty. Veľký prienik potrebných technológii umožňuje vznikať aj projektom mimo priemyslu, ktoré priblížia tento odbor a jeho produkty bežným ľuďom, či už je to dostupná programovateľná stavebnica, alebo možnosť naprogramovať si robotické rameno v zábavnom parku a posadiť sa do kresla na jeho konci.

Obdobne sa umelá inteligencia, svojho času viac prítomná vo filme a vedecko-fantastickej literatúre, v posledných dvoch desaťročiach dostáva z laboratórií do praxe. Vďaka zvyšovaniu výpočtového výkonu sa s výsledkom náročných prístupov, ako sú neurónové siete či evolučné algoritmy, stretáva aj bežný človek. Pričom našťastie nepôjde o filmového bojového robota, ale napríklad o personalizovanú ponuku produktov v online obchode vzhľadom na predošlé nákupy.

Herný priemysel sa od počiatkov z doby prvých domácich počítačov snaží zaujať konzumentov aplikáciou noviniek v oblasti informačných technológii. V poslednej dobe je, vďaka dostupnosti výkonu aj v prenosných zariadeniach, takou aj nasadenie komplexnejších metód umelej inteligencie na simuláciu ľudských súperov. Trendom je aj oživenie virtuálnych svetov pomocou integrácie fyzických predmetov či už snímaním na vstupe alebo rôznymi výstupnými zariadeniami ako aktuátormi.

Samotná idea spojiť princípy robotiky, umelej inteligencie a herných mechanizmov dokopy nie je nová, stačí si len spomenúť Kempelenovho šach hrajúceho „umelého Turka“ z konca 18 storočia. Ale pri súčasných technológia nielenže nemusí ísť o podfuk, ale je možné spraviť ekonomicky efektívne reprezentácie na akademickej pôde.

Môj stret s robotikou začal v rámci bakalárskej práce na úrovní teórie, hardvéru a obslužných programov a zanechal vo mne záujem pokračovať aj v diplomovej práci. Výber aplikácie robotického ramena do herných systémov mi dovolil rozšíriť a aplikovať vedomosti z oblasti umelej inteligencie, logického programovania a herných teórii.

Cieľom práce je vytvoriť systém pre použitie fyzického aktuátora, robotického ramena, ako výstupu herných algoritmov založených na aplikácii algoritmov umelej inteligencie. Systém má vedieť ovládať robotické rameno s funkcionalitou inverznej kinematiky, spracovať a reprezentovať hraciu plochu s hracími prvkami, mať možnosť zadefinovať a simulovať herný proces pre viacero stolových hier a simulovať hráča na základe algoritmov.

Výsledkami sú štatistiky pre porovnanie efektívnosti výpočtu, efektívnosti výhier či remíz, správnosť a úplnosť akcií medzi jednotlivými stavmi hracieho prostredia. Tiež aj simulácia pomocou robotického ramena a celkový systém použiteľný vďaka opraviteľnosti ako didaktickú pomôcku na demonštrovanie rôznych modelov.

Samotná práca je rozdelená na 2 časti a štruktúrovaná do 5 kapitol. Prvá časť a zároveň kapitola slúži ako prehľadová časť, ktorá obsahuje popis východísk k riešeniu zadaného cieľa práce. Druhá časť sa delí na kapitoly podľa štandardného postupu pri procese vývoja softvérového diela či projektu. V kapitole 2 prednesiem ciele práce a z nich vychádzajúce funkčné a nefunkčné požiadavky na systém. Kapitola 3 obsahuje analýzu požiadaviek a návrh systému. V kapitole 4 bude popísaná konkrétna realizácia a implementácia systému. V kapitole 5 bude zhodnotený výsledný systém, jeho funkčnosť a výsledky testov na konkrétnych testovacích hrách.

1 VÝCHODISKÁ

1.1 Ciele práce

Cieľom tejto práce je práve vytvoriť hernú platformu pre rôzne stolové hry, pričom sa má samotný hrací proces možnosť odohrávať na reálnej hracej ploche za použitia robotického ramena. Hráčom môže byť človek, alebo racionálny agent, autonómne rozhodujúci za pomoci algoritmov umelej inteligencie. S úmyslom interaktívnosti s reálnym svetom má obsahovať robotické rameno a počítačové videnie.

Cieľom nie je vytvoriť špecifické algoritmy pre efektívne hranie konkrétnych hier. Z toho dôvodu sa nebudeme zameriavať na efektívnosť výhier či výpočtu pre konkrétne algoritmy pre konkrétne hry, ale ich správnosť výpočtu a simulácie pre ľubovoľne zadefinovanú stolovú hru. Porovnávať môžeme efektívnosť medzi jednotlivými algoritmami použitými pre danú hru.

Pri procese návrhu treba komplexne analyzovať zadané špecifikácie systému a určiť funkčné a nefunkčné požiadavky na výsledný systém. Následne sa vypracovaný návrh pokúsiť implementovať, tak aby spĺňal hlavné zadefinované požiadavky. Robotické rameno má obsahovať funkcionalitu inverznej kinematiky. Počítačové videnie musí vedieť získať a spracovať informácie z prostredia o stave hry. Komponent autonómneho rozhodovania musí vedieť pomocou aspoň jedného algoritmu umelej inteligencie racionálne určovať akcie na dosiahnutie výhry pri procese hrania ľubovoľnej stolovej hry. Riadiaca časť musí vedieť komunikovať s jednotlivými komponentami, ale aj s ľudským hráčom.

Následne na príkladoch demonštrovať funkčnosť celého systému, jednotlivých komponentov a racionálnosť rozhodnutí umelej inteligencie.

1.2 Umelá inteligencia a hry

Umelá inteligencia je vedná oblasť so širokým záberom pôsobnosti. Zaoberá sa skúmaním, navrhnutím, zostrojením a pochopením inteligentných objektov, od ktorých očakávame, že sa na základe implementovaných metód umelej inteligencie budú správať racionálne. Hlavné ciele štúdia umelej inteligencie zahŕňajú reprezentáciu a odvodzovanie znalostí, plánovanie akcií, učenie sa, schopnosť vnímať a ovplyvňovať prostredie.

Podoblasti, ktorým sa umelá inteligencia venuje je veľa a pre každú z nich sa jej pôsobnosť špecificky upravuje.

Základy umelej inteligencie, ešte v rámci iných odborov, z ktorých umelá inteligencia vychádza, sa objavujú už v antike, hlavne v oblasti filozofie a formálnej logiky. Prvé algoritmy UI v hrách sa datujú od 50tych rokov minulého storočia, pre rôzne hry vrátane *dámy* či *šachu*. Avšak rozvoj umelej inteligencie prišiel až vďaka expertným systémom v 80tych rokoch minulého storočia, ktoré sa snažili simulovať ľudského experta pomocou reprezentácii znalostí a odvodzovacích pravidiel s cieľom odvodiť nové znalostí. Následne prišiel rozvoj aj vďaka zvyšujúcej sa výpočtovej schopnosti počítačov, kedy mohli prísť na radu aj zložité výpočtové koncepty akými sú neurónové siete či evolučné algoritmy. Za jeden z novodobých míľnikov umelej inteligencie v hrách sa považuje rok 1997 a počítač *Deep Blue,* kedy ako prvý šachový program porazil vtedajšieho šachového majstra sveta.

Venujme sa teraz konkrétnej časti, umelej inteligencie pre problémy zadefinované ako hry. Hry sú vhodnou oblasťou skúmania pre umelú inteligenciu, nakoľko ponúkajú dobre zadefinovaný problém, dobre vyjadriteľnú mieru úspešnosti a presnú reprezentáciu akéhokoľvek stavu a akcií medzi stavmi. [3]

V súčasnosti neexistuje jedna optimálna metóda umelej inteligencie, ktorou by sa dali riešiť všetky problémy, dokonca ani v rámci samotných špecifických pododborov. Ku každému problému treba pristupovať osobitne, vyskúšať rôzne prístupy a spomedzi tých čo sa dajú aplikovať vzájomným porovnávaním vybrať tie najefektívnejšie vzhľadom na vymedzené kritéria. Vďaka tomu sa ponúka priestor pre zaujímavú snahu hľadať, skúšať, testovať a porovnávať nové prístupy a algoritmy. [2]

V súčasnosti poznáme dva hlavné prístupy k riešenie problémov pomocou umelej inteligencie

- Symbolický – Ide o klasický prístup, kde sa uplatňuje symbolická reprezentácia znalostí a ich sekvenčné spracovanie. Tento prístup sa používa, ak si vieme problém zadefinovať v uzavretej doméne, pre jasne definované symboly a pravidlá. Medzi základné metódy patria algoritmy prehľadávania stromových štruktúr, expertné systémy, logické vyvodzovanie a iné.

  Tradičné prístupy majú však svoje obmedzenia, nakoľko veci z reálneho sveta sa často krát nedajú vyjadriť len symbolickým spôsobom, pravda-nepravda (true-

false). Navyše domény a znalosti v reálnom svete bývajú zväčša nekonečné, s čím má tradičný prístup taktiež problém s reprezentáciou.

- Subsymbolický – Ide o moderný prístup, ktorý sa ponúka pri problémoch, kde má klasický symbolický prístup ťažkosti. Dáva možnosť imitovať myslenie, ktoré je bližšie ľudskému, nakoľko vie simulovať schopnosť „pocitovo“ uhádnuť alebo pravdepodobnostnými metódami odhadnúť riešenie, čo je proces ktorý nevieme symbolicky zadefinovať a reprezentovať. Vďaka tomu sa tento prístup vie vysporiadať aj s problémami, pre ktoré nevieme celkovo popísať a zadefinovať jednoznačný postup. Tieto algoritmy sa na základe opakovaných pokusov snažia aproximovať kvalitu akcií bez toho, aby nutne rozumeli prečo ich považujú za dobré. Prevažujú tu prístupy za použitia neurónových sietí a evolučných algoritmov. [2]

V nasledujúcich kapitolách si zadefinujeme základné pojmy a algoritmy na riešenie hier.

1.2.1 Herný systém, základné pojmy

Existuje rôzne definície pre rôzne typy hier, v závislosti od ich komplexnosti či prostredia. Zamerajme sa teraz na konkrétnu podskupinu hier, stolové resp. doskové hry.

Z formálneho hľadiska môžeme hru zadefinovať ako problém, ktorý má určený počiatočný stav, jednoznačnú reprezentáciu akéhokoľvek stavu a prechodov medzi stavmi, jednoznačne identifikovateľné konečné stavy a dobre vyjadriteľnú mieru úspešnosti správania sa hráčov.

Stav hry resp. pozícia

Aktuálny stav prostredia vzhľadom na jeho vlastnosti a hracie objekty.

Počiatočný stav hry

Inicializačné nastavenie agentov a stav v ktorom sa nachádza prostredie pri začiatku každej hry.

Konečný stav hry

Stav v ktorom je hra ukončená a vieme pre neho určiť výsledok hry.

Akcia, prechod

Množina možných akcií, ktoré môže agent v danom stave hry vykonať a zmeniť tak stav hry na nasledujúci stav hry.

Stavový priestor

Množina všetkých stavov v akých sa môže nachádzať prostredie, vychádzajúc z počiatočného stavu a aplikáciou ľubovoľnej postupnosti možných legálnych akcii v každom stave takejto postupnosti.

Stromová štruktúra stavového priestoru

Orientovaný graf, kde uzly grafu reprezentujú jednotlivé stavy hry a hrany grafu reprezentujú prechodové akcie medzi stavmi.

Prehľadávanie stromovej štruktúry [3]

- Informované – máme dodatočnú informáciu o kvalite stavu na základe ktorého vieme určovať postupnosť rozvoja uzlov s cieľom vybrať najlepšie akcie v danom stave na základe tejto informácie, čím sa zvyšuje efektívnosť algoritmu.
- Neinformované – nemáme žiadnu dodatočnú informáciu o probléme alebo kvalite stavu v akom sa nachádzame, uzly rozvíjame postupne ako prichádzajú postupne na radu.

Úžitková funkcia (Utility/evaluating function)

Ide o presné vyjadrenie hodnoty konečného stavu hry stavu hry z pohľadu agenta

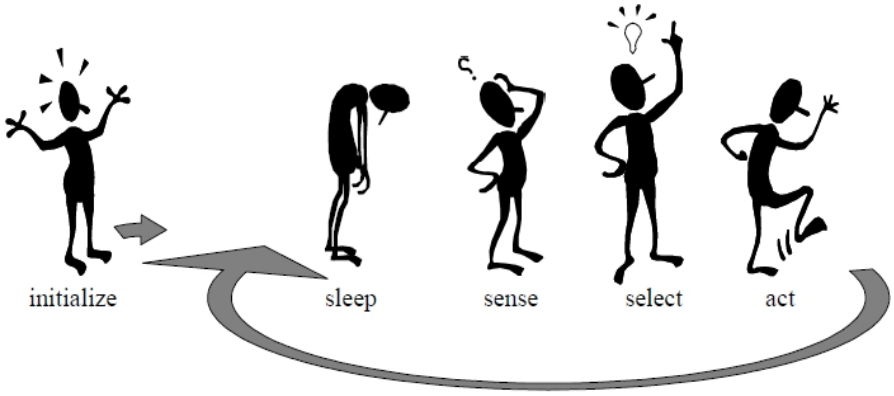
Heuristická funkcia

Heuristická funkcia vyjadruje spokojnosť agenta s daným stavom, ide o aproximáciu kvality stavu na základe konkrétnych parametrov ktoré berie do úvahy, nie je to vyjadrenie presného stavu prostredia ako pri úžitkovej funkcii, ale jeho odhad agentom. Agent môže mat aj viac heuristických funkcii a vyberať si spomedzi nich najlepšiu pre daný problém.

1.2.2 Agent a jeho P.E.A.S

Agent

Za agenta sa považuje entita, ktorá neustále vníma okolité prostredie za pomoci svojich senzorov a vykonáva akcie pomocou svojich výkonných prostriedkov. [5] Agent volí akcie na základe svojho systému rozhodovania sa, pričom sa výber jeho akcií riadi určitým zadaným cieľom. Životný cyklus agenta je znázornený na obrázku 1.1.



Obr. 1.1 Životný cyklus agenta [2]

Typy agentov

- Jednoduchý reaktívny
  Agent reaguje na podnety cez striktne zadefinované pravidlá Ak-Tak (if-then).
- Tvoriaci si model (model-based)
  Jednoduchý reaktívny agent, ktorý si naviac ukladá získané informácie na základe postupnosti stavov sveta a tvorí si z nich vnútorný model prostredia.
- Zameraný na cieľ (goal-based)
  Agent tvoriaci si model, ktorý vie aký má cieľ a akcie si vyberá podľa toho aby sa k nemu dostal.
- Zameraný na úžitok (utility-based)
  Agent tvoriaci si model, ktorý vie aký má cieľ a akcie si vyberá podľa toho za akú cenu sa dostane do cieľa, pričom sa cenu snaží minimalizovať.

Agent sa skladá zo štruktúry (fyzickej architektúry) a implementovanej agentovej funkcie.

Agentova funkcia

Funkcia, ktorá mapuje získané vnemy agenta na vybranú akciu $[f: V^* \rightarrow A]$.

Racionálny agent

Pre každú možnú postupnosť vnemov vyberá racionálny agent takú akciu, od ktorej očakáva maximalizáciu miery výkonu (performance measure). Agent používa svoj zadefinovaný systém rozhodovania sa, agentovu funkciu, na základe získaných vnemov, faktov a odvodených znalostí. Racionálny agent neznamená nutne optimálny, môže byť, ale nemusí.

P.E.A.S.

Pre agenta je nutné špecifikovať prostredie, v ktorom sa bude nachádzať a pre ktoré bude určený. Štruktúra agenta a agentova funkcia sa navrhuje a prispôsobuje podľa typu prostredia.

Miera výkonu (Performance Measure)

Objektívne (účelové) kritérium hodnotiace úspech agentovho správania sa (jeho stratégie), s ohľadom na kombináciu rôznych parametrov akými sú napríklad účinnosť, rýchlosť, počet vyriešených úloh, získané skóre, a podobne.

Výkonné prostriedky (Actuators)

Agent pomocou výkonných prostriedkov vykonáva akcie, ktorými môže ovplyvňovať iné objekty alebo časti prostredia. Napríklad robotická ruka, motory, a podobne.

Senzory (Sensors)

Agent spracováva výstupy zo senzorov a transformuje ich na vnemy, t.j. získané znalosti o prostredí. Napríklad kamera, senzor teploty, a podobne.

Prostredie (Environment)

Charakteristika prostredia v ktorom bude agent pôsobiť ovplyvňuje zvolenú štruktúru agenta a jeho agentovu funkciu.

- Plne pozorovateľné alebo Čiastočne pozorovateľné

  Prostredie je plne pozorovateľné, ak majú senzory agenta v každom momente možnosť vnímať kompletné a úplné informácie o stave prostredia.

- Deterministické alebo Stochaistické (Nedeterministické)

  Prostredie je deterministické, ak je nasledujúci stav prostredia kompletne určený aktuálnym stavom a akciou, ktorú vykoná agent. Zmena prostredia teda nie je ovplyvnená náhodnými udalosťami, alebo inými zdrojmi ako akcie agenta.

- Epizodické alebo Sekvenčné (Neepizodické)

  Prostredie je epizodické, ak je správanie sa agenta je rozdelené na atomické epizódy, pri ktorých agent získa znalosti vnemom a vykoná práve jednu akciu, pre ktorú sa agent rozhodne len na základe znalostí získaných počas tejto jednej epizódy.

- Statické alebo Dynamické alebo Semidynamické

  Prostredie je statické, ak sa stav prostredia nemení, pokým agent nevykoná akciu. Prostredie je semidynamické vtedy, ak sa v čase samotné prostredie nemení, ale mení sa miera výkonu (performance measure) agenta.

- Diskrétne alebo Spojité

  Prostredie je diskrétne, ak obsahuje konečný počet stavov a agent má konečný počet možných akcií.

- Jednoagentové alebo Multiagentové

  Prostredie je multiagentové, ak sa v ňom nachádzajú viacerí agenti, či už navzájom spolupracujúci alebo ako oponenti.

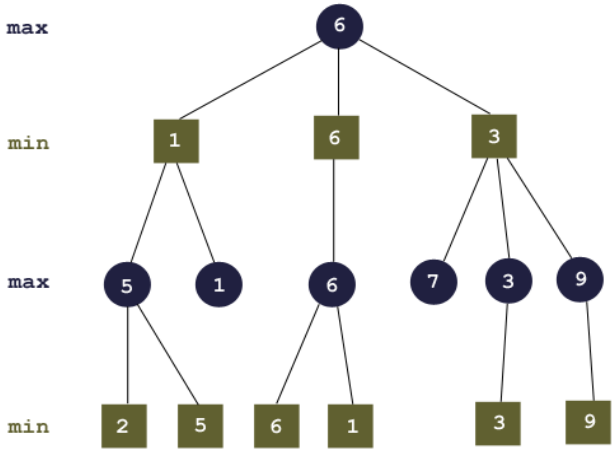
1.2.3 Minimax

Ide o tradičný symbolický prístup umelej inteligencie k riešeniu zadaného problému. Algoritmus Minimax je vo svojej podstate prehľadávanie stromovej štruktúry do hĺbky, pričom si spätne propagujeme získané hodnoty z listov postupne späť do koreňa stromu. Ide o metódu, ktorá vytvorí všetky možné postupnosti stavov, aké môžu nastať v hre. Aj preto sa nemodifikovaný algoritmus Minimax dá chápať ako metóda hrubej sily.

Hranie hry za použitia algoritmu Minimax môžeme zadefinovať ako prechádzanie postupnosti stavov hry (uzol stromu), pričom v každom stave môže hráč, ktorý je v ňom na ťahu vykonať konečný počet akcií (hrany stromu z daného uzla), ktoré zmenou aktuálneho stavu hry vedia vytvoriť možné nasledujúce stavy hry (nasledovníci uzla). Vzhľadom na konečný počet akcií pre každý uzol vieme vytvoriť úplný strom hry, pričom koreň stromu predstavuje aktuálny stav hry a listy stromu predstavujú konečné stavy hry. Listy stromu

sú ohodnotené podľa toho, či hráč, označme ho *Max*, pre ktorého je herný strom vygenerovaný vyhral, prehral alebo remizoval.

V takomto úplnom strome hry vieme potom určiť postupnosť akcií, ktorá by *Max* hráča doviedla k najlepšiemu možnému výsledku, pri predpoklade, že druhý hráč, označme ho *Min*, pre seba vyberá najoptimálnejšie akcie, vzhľadom na to aké by si na jeho mieste vybral *Max* hráč.



Obr. 1.2 Priebeh algoritmu Minimax [12]

Na obrázku 1.2 môžeme vidieť priebeh algoritmu Minimax. Hráči *Max* a *Min* sa postupne striedajú na ťahu. *Max* hráč je hráč, ktorý je na ťahu v stave hry pre koreň stromu a snaží sa maximalizovať svoj možný zisk výberom akcie, ktorá vedie do uzla s maximálnou hodnotou spomedzi svojich nasledovníkov. *Min* hráč je hráč, ktorý sa snaží minimalizovať svoju stratu výberom akcie, ktorá vedie do uzla s minimálnou hodnotou spomedzi svojich nasledovníkov.

Algoritmus Minimax:

1. v aktuálnom stave vygeneruj všetky možné nasledujúce stavy,
2. postupne prechádzaj zľava všetky možné nasledujúce stavy,
3. ak je nový stav konečný, vyhodnoť jeho hodnotu úžitkovou funkciou *Max* hráča, podľa toho či vyhral, prehral alebo remizoval,
4. ak nový stav nie je konečný, pokračuj z neho 1. krokom algoritmu,
5. ak už nie sú žiadne ďalšie nasledujúce stavy, priraď tomuto stavu hodnotu maxima spomedzi všetkých možných nasledujúcich stavov ak je na ťahu *Max* hráč, alebo minimum ak je na ťahu *Min* hráč.

Princíp algoritmu Minimax má svoje výhody, ale aj zásadné nevýhody.

Výhoda algoritmu

- Spôsob prehľadávania tak, že sa vytvorí úplný strom, znamená, že ak sa v strome nachádza optimálne riešenie problému, algoritmus ho určite nájde a vyberie. Avšak, ohodnotenia riešení problému sú založené na princípe, že *Min* hráč vyberá optimálne riešenia z pohľadu stratégie *Max* hráča, ako by sa on rozhodol na jeho mieste. Čiže ide o optimálne riešenie len z pohľadu stratégie *Max* hráča, nie objektívne optimum.

Nevýhody algoritmu

- Prvou nevýhodou je možnosť vytvorenia opakujúceho sa cyklu stavov v hre, čo znamená nekonečnú hĺbku pre danú vetvu stromu. V takomto prípade by sa algoritmus nikdy neskončil.
- Druhou nevýhodou je veľkosť úplného stromu. Priestorová zložitosť úplného stromu je $O(b^d)$, pričom b je faktor vetvenia (maximálny počet akcií v každom stave) a d je maximálna hĺbka stromu. Pri hrách s veľkým počtom stavov sa stáva prehľadávanie na úplnom strome nereálne z hľadiska výpočtovej zložitosti súčasných počítačov. Napríklad pri dáme sa uvádza veľkosť úplného stromu 10^{31}alebo pri šachu 10^{123}. [2]

Vzhľadom k týmto nevýhodám pristúpime k dvom spôsobom úpravy algoritmu Minimax

- orezávanie v určenej hĺbke stromu,
- orezávanie neperspektívnych vetiev stromu – *alfa beta* orezávanie.

1.2.3.1 Minimax s orezaním v hĺbke

Algoritmus prehľadávania stromu si upravíme tak, že pri dosiahnutí určenej hĺbky stromu sa orežú všetky nasledujúce uzly v danej vetve. Takto vzniknuté listy stromu sa ohodnotia určenou heuristickou evaluačnou funkciou.

Výhody algoritmu

- Odstráni sa možnosť nekonečnej vetvy stromu.
- Pri zvolení správnej hĺbky sa výrazne urýchli výpočet riešenia.

Nevýhody algoritmu

- Stratí sa vlastnosť optimálnosti algoritmu, čo znamená, že aj keď existuje optimálne riešenie, už nie je zaručené že ho algoritmus nájde, t.j. pokiaľ sa nachádza v strome hlbšie ako je určená hĺbka orezania tak o ňom algoritmus nevie.
- Úspešnosť riešenia je výrazne ovplyvnená kvalitou heuristickej evaluačnej funkcie a zvolenej hĺbky orezania.

1.2.3.2 Minimax s alfa-beta orezávaním

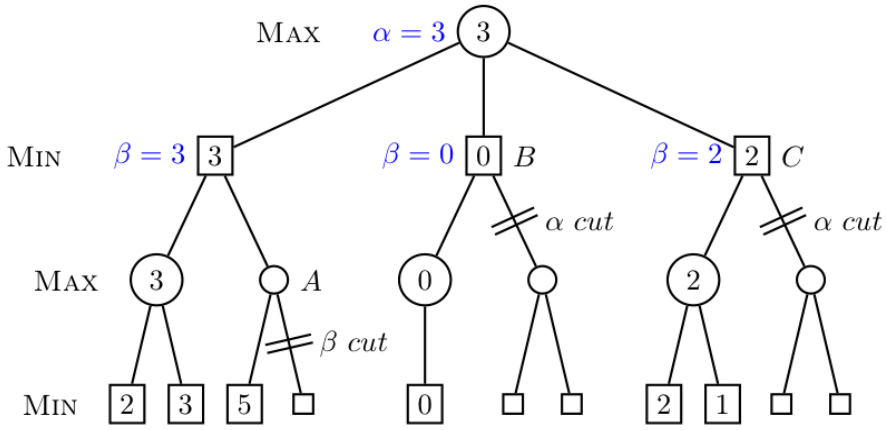
Algoritmus *alfa-beta* orezávanie je štandardná modifikácia algoritmu Minimax. Jeho aplikovaním sa snažíme vysporiadať s príliš veľkou priestorovou zložitosťou pri generovaní celého stromu stavového priestoru. Pointou je orezanie vetiev stromu, čiže možných postupností akcií, o ktorých už vieme, že nemôžu byť lepšie ako iná predošlá postupnosť akcií. Takýmto spôsobom zaručíme, že dosiahneme rovnaké riešenie ako pri štandardnom algoritme Minimax, avšak bez nutnosti vygenerovať a prehľadať úplný strom.

Algoritmus prehľadávania stromu si upravíme tak, že pridáme 2 parametre, *alfa* a *beta*, ktoré nám budú pri prechode stromom priebežne informovať hráčov v daných uzloch, aké už majú zaručené hodnoty z predchádzajúcich postupností akcií.

- ***Alfa*** – Parameter, ktorý v danom uzle určuje priebežnú zaručenú najlepšiu získanú hodnotu *Max* hráča v niektorej z predošlých postupností akcií, ktorá prechádza daným uzlom. *Max* hráč sa snaží parameter *alfa* maximalizovať.
- ***Beta*** – Parameter, ktorý v danom uzle určuje priebežnú zaručenú najlepšiu získanú hodnotu *Min* hráča v niektorej z predošlých postupností akcií, ktorá prechádza daným uzlom. *Min* hráč sa snaží parameter *beta* minimalizovať.
- ***Alfa*** orezanie – Nastáva v prípade, že očakávaný hodnota *beta* pre zisk *Min* hráča v danom uzle, je aspoň tak dobrá, alebo lepšia, ako očakávaný hodnota pre zisk *Max* hráča, z nejakej predchádzajúcej vetvy postupností akcií. Toto je z pohľadu *Max* hráča nevýhodné, preto ak je splnená podmienka $beta \leq alfa$, nastáva *Alfa* orezanie. Ide o prípad, že *Max* hráča zastaví príliš dobrý uzol pre *Min* hráča.
- ***Beta*** orezanie – nastáva v prípade, že očakávaný hodnota *alfa* pre zisk *Max* hráča v danom uzle, je aspoň tak dobrá, alebo lepšia, ako očakávaný hodnota pre zisk *Min* hráča, z nejakej predchádzajúcej vetvy postupností akcií. Ďalší rozvoj je z pohľadu

Min hráča zbytočný, preto ak je splnená podmienka *beta* ≤ *alfa*, nastáva *Beta* orezanie. Ide o prípad, že *Min* hráč zastaví príliš dobrý uzol pre *Max* hráča.

Na obrázku 1.3 môžeme vidieť oba typy orezanie. V uzle *A* si *Max* hráč nastaví parameter *alfa* na 5, pričom jeho parameter *beta* má hodnotu 3, nakoľko to je hodnota ktorú má už *Min* uzol o jednu úroveň vyššie zaručenú z predchádzajúcej vetvy. Keďže uzol *A* prináleží *Max* hráčovi, ten hodnotu *alfa* môže zmeniť ak len na ešte vyššiu. To pre hráča *Min* v uzle nad uzlom *A* znamená, že táto vetva je pre neho už teraz horšia ako predchádzajúca a už sa môže z jeho pohľadu len zhoršiť. Z toho dôvodu je zbytočné ďalej rozvíjať uzol *A* a algoritmus pristúpi k *beta* orezaniu. Pre *alfa* orezanie to platí analogicky.



Obr. 1.3 Priebeh algoritmu Minimax s *alfa-beta* orezávaním [12]

Zmeny v algoritme Minimax

1. Na začiatku algoritmu sa nastavia hodnoty pre parametre *alfa* ako záporné nekonečno a *beta* ako kladné nekonečno.
2. Pri postupnom prechádzaní nasledovníkov uzla sa do nich prenášajú aktuálne hodnoty *alfa* a *beta* z daného uzla.
3. Pri spätnej propagácii hodnoty z nasledovníkov uzla sa pre uzol *Max* hráča nastaví parameter *alfa* ako maximum z vrátenej hodnoty a aktuálnej hodnoty *alfa*, a pre uzol *Min* hráča nastaví parameter *beta* ako minimum z vrátenej hodnoty a aktuálnej hodnoty *beta.*
4. Po nastavení nových hodnôt parametrov sa skontroluje podmienka, *beta* ≤ *alfa*. Ak je splnená, rozvoj ďalších nasledovníkov sa zastaví.

Výhody algoritmu

- Vďaka orezaniu nepotrebných vetiev stromu sa zlepší časová aj priestorová zložitosť algoritmu. Miera zefektívnenie prehľadávania závisí od toho, v akom poradí sa budú prehľadávať jednotlivé uzly nasledovníkov, t.j. čím skôr sa prezrú stavy po vykonaní výhodnejších akcií, tým skôr sa začnú orezávať nepotrebné uzly.
- V najlepšom prípade sa zníži priestorová zložitosť na $O(\sqrt{b^d})$. [2] V najhoršom prípade sa môže stať, že pri nevhodnej postupnosti akcií sa neoreže žiadna vetva a vygeneruje sa znova úplný strom.

1.3 Reprezentácia znalostí

Reprezentácia znalostí je symbolickým prístupom v umelej inteligencii. Určuje formu a spôsob symbolického a abstraktného zápisu informácií z reálneho sveta do jazyka, ktorý bude lepšie pochopiteľný pre počítač. Reprezentácia znalostí je potrebná na vyjadrenie a porozumenie informácií, ale taktiež aj na ich spracovanie, získavanie a odvodzovanie nových znalostí.

V súčasnosti poznáme viacero formalizmov na reprezentáciu znalostí, napríklad sémantické siete, rámce, predikátovú logika, produkčné systémy a iné. Každý má svoju vlastnú syntax za zápis informácií a taktiež vlastné produkčné a odvodzovacie metódy na spracovanie a tvorbu nových znalostí.

Jeden z najpoužívanejších formalizmov v reprezentácií znalostí je predikátová logika. Z jej špecifickej verzie predikátovej logiky prvého rádu vychádza formalizmus logického programovania, ktoré si v nasledujúcej kapitole zadefinujeme.

1.3.1 Logické programovanie

Logické programovanie je špecifickou formou deklaratívneho programovania. To znamená, že na rozdiel od procedurálneho jazyka, neurčujeme postupnosť inštrukcií, ale pre zadefinovanú doménu problému špecifikujeme čo má program docieliť, nie ako.

Formalizmus a syntax logické programovania vychádza z formálnej logiky, najližšie má k predikátovej logike prvého rádu.

Abeceda jazyku (syntax) logického programovanie pozostáva z

- premenné A, B, C, ...
- funkčné symboly f, g, h, ...
- predikátové symboly p, q, r, ...
- spojky $\bot$,$\top$, $\leftarrow$, $\neg$, ~, $\vee$, $\wedge$
- pomocné symboly (,), {,}, ”, ', ...

Zadefinujme si hlavné pojmy: [6]

Arita

Arita funkcie alebo predikátu udáva počet parametrov funkcie alebo predikátu.

Term

Za term sa považuje konštanta (nulárna funkcia), premenná alebo funkcia, ktorá obsahuje len iné termy ako parametre. Term je **uzemnený (ground)** ak neobsahuje premenné.

Atóm

Za atóm sa považuje predikát, ktorý obsahuje len termy ako parametre. Atóm je **uzemnený (ground)** ak obsahuje iba uzemnené termy.

Formula

Za formulu sa považuje atóm, alebo negácia formuly $\neg F$, disjunkcia formúl $F \vee G$, konjunkcia formúl $F \wedge G$, alebo pravidlo $F \leftarrow G$.

Literál

Za objektívny literál za považuje atóm A, alebo explicitne negovaný atóm ┐A. Za subjektívny (defaultový) literál za považuje defaultne negovaný objektivny literal ~A alebo ~┐A.

Klauza

Klauza je disjunkcia literálov $(A_1 \vee \neg A_2 \vee \sim\neg A_3 \vee \sim\neg A_4 \vee \ldots \vee A_n)$.

Konjunktívna normálna forma (CNF, Conjunctive Normal Form)

Konjunktívna normálna forma je formula v tvare konjunkcie kláuz $(A_1 \vee A_2) \wedge (A_3 \vee A_4) \wedge \ldots \wedge (A_5 \vee \ldots \vee A_n)$. Ľubovoľnú formulu vieme previesť do

CNF tvaru pomocou distribučných a DeMorganových pravidiel a pravidla dvojitej negácie. V logickom programovaný využívajú CNF formu formúl odvodzovacie pravidlá, avšak zväčša ju zapisujeme v tvare pravidla programu kvôli lepšej zrozumiteľnosti pre človeka.

CWA

Predpoklad uzavretého sveta (CWA Close World Assumption) je formalizácia, ktorá umožňuje použitie tzv. **defaultovej negácie**, označenie ~ , resp. *not*. Táto formalizácia používa odvodzovacie pravidlo „negácia ako neúspech“ (Negation as failure), ktoré hovorí, že ak neviem dokázať pravdivosť atómu, môžem predpokladať, že je nepravdivý, až do momentu, kým nedokážeme opak. Defaultová negácia sa týmto líši od klasickej explicitnej negácie.

Pravidlo

Pravidlo je klauza, ktorá je zapísaná v ľudsky čitateľnom tvare. hlava$(R) \leftarrow$ telo(R) Pravidlo sa skladá z literálov v hlave pravidla a literálov v tele pravidla. $L_1, L_2, \ldots, L_n \leftarrow L_{(n+1)}, L_{(n+2)}, \ldots, L_m$ kde $L_1, \ldots, L_m$ sú literály.

Na základe logického pravidla $(A \Rightarrow B) \Leftrightarrow (\neg A \vee B)$ a distribučných a DeMorganových pravidiel vieme upraviť formulu v tvare klauzy na pravidlo logického programovania.

$(A_1 \wedge A_2 \wedge \ldots \wedge A_n \Rightarrow B) \Leftrightarrow (\neg(A_1 \wedge A_2 \wedge \ldots \wedge A_n) \vee B) \Leftrightarrow (\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \vee B)$

Poznáme dva špeciálne typy pravidiel, obmedzenia a fakty.

- **Fakt** je pravidlo, ktoré má prázdne telo pravidla. Sémanticky význam faktu sa dá vysvetliť ako pravidlo, ktorého telo je vždy splnené, čím je automaticky splnená aj hlava pravidla.

  hlava$(R) \leftarrow \top$ hlava$(R) \leftarrow$ True

- **Obmedzenie (constraint)** je pravidlo, ktoré má prázdnu hlavu pravidla. Pravidlo obmedzenia má tú vlastnosť, že pokiaľ je splnené telo pravidla, do modelu odvodí konštantu False, čím spôsobí zahodenie modelu ako nesplniteľného.

  $\perp \leftarrow$ telo(R) False $\leftarrow$ telo(R)

Logický program

Logický program je množina pravidiel.

- **Pozitívny (definitný)** logický program neobsahuje defaultovú negáciu.
- **Normálny** logický program môže obsahovať defaultovú negáciu iba v tele pravidiel.
- **Rozšírený** logický program môže obsahovať defaultovú negáciu aj v hlave pravidiel.
- **Disjunktívny** logický program môže obsahovať disjunkciu literálov v hlave pravidiel.

Herbrandovské univerzum

Množina všetkých uzemnených termov. Môže byť nekonečná.

Herbrandovské báza

Množina všetkých uzemnených atómov. Môže byť nekonečná.

Herbrandovská interpretácia

Interpretácia *I* prisúdi pravdivostnú hodnotu atómom. Herbrandovská interpretácia je podmnožinou uzemnených atómov z Herbrandovskej bázy, pričom tieto atómy z Herbrandovskej interpretácie bude chápať ako **splnené**. V prípade defaultovej negácie atómu ~A, chápeme tento literál ako splnený, pokiaľ sa atóm A nenachádza v Herbrandovskej interpretácií.

Herbrandovský model

Herbrandovský model logického programu je taká Herbrandovská interpretácia, pre ktorú platí, že vzhľadom na všetky uzemnené atómy z Herbrandovskej bázy, je každé pravidlo programu v tejto interpretácii splnené.

V logickom programovaní sa používajú rôzne metódy k odvodzovaniu nových znalostí. Dve najznámejšie sú:

- Forward chaining – extenzívnym aplikovaním odvodzovacích pravidiel na známe fakty a priebežne odvodené znalosti sa snažíme odvodiť všetky uzemnené znalosti patriace do modelu programu.

- Backward chaining – odvodzovacie pravidlá aplikujeme spätne na hypotézu žiadanej znalosti a za pomoci substitúcií a rekurzie sa snažíme potvrdiť splniteľnosť hypotézy.

Iný prístup k odvodeniu nových znalostí je cez riešenie zástupného SAT problému, zisťovania či existuje interpretácia pre danú formulu v CNF forme: DPLL algoritmus. (1.3.3)

1.3.2 ASP

ASP (Answer Set Programming) je deklaratívna paradigma, ktorej základnou myšlienkou je použitie algoritmov na hľadanie možnej interpretácie pre splnenie formuly reprezentujúcej logický program, ktorú použijeme na hľadanie riešenia pôvodného problému. Problém je zakódovaný ako logický program, tak aby jeho riešenia boli reprezentovateľné stabilným modelom. Na hľadanie modelu sa použije ASP solver, ktorý je najčastejšie založený na DPLL algoritme, čím sa zaručí absencia zacyklenia.

1.3.3 DPLL algoritmus

DPLL (Davis-Putnam-Logemann-Loveland) je úplny backtracking algoritmus pre riešenie SAT problému pre danú formulu v CNF forme. Základná metóda backtracking algoritmov, dosadzovanie hodnôt literálom a zisťovanie pravdivosti formuly, je vylepšená o nasledovné heuristiky:

- Jednotková propagácia (Unit propagation) - ak v danej klauzule ostane jediný literál s nepriradenou hodnotou, tak tá musí byť dosadená tak, aby bol pravdivý.
- Odstránenie čistých literálov (Pure literal elimination) - ak sa literál vyskytuje vo všetkých klauzách iba v jednom pravdivostnom tvare, tak jeho hodnota musí byť dosadená tak aby bol pravdivý.

Ak sa podarí postupne eliminovať všetky klauzy a zostane prázdna formula, problém je splniteľný.

1.4 Počítačové videnie

Počítačové videnie je technologická disciplína zaoberajúca sa zachytením vizuálnej informácie z priestoru reálneho sveta, následným spracovaním zachyteného obrazu a analýzou žiadaných konkrétnych informácií obsiahnutých v spracovanom výstupe počítačového videnia.

Štruktúra a funkcionalita modulu počítačového videnia závisí od konkrétneho zamýšľaného využitia. V procese analýzy špecifikácií pre plánovaný systém, či agenta, treba zhodnotiť aké množstvo a typ informácií o prostredí potrebujeme, a na základe toho si navrhnúť štruktúru komponentu počítačového videnia.

Pokiaľ si vystačíme s 2D informáciu o prostredí, použijeme jednoduchšiu formu, jedno kamerové zariadenia. Ak vyžadujeme 3D informáciu, musíme použiť stereo videnie, kde následne pretransformujeme a zlúčime získané údaje z viacerých kamerových zariadení do jednej priestorovej informácie.

Ako druhú vec si musíme analyzovať aké informácie potrebujeme zo zachytených údajov získavať. Funkcionalita počítačového videnia sa delí podľa časovej fázy jej aplikácie:

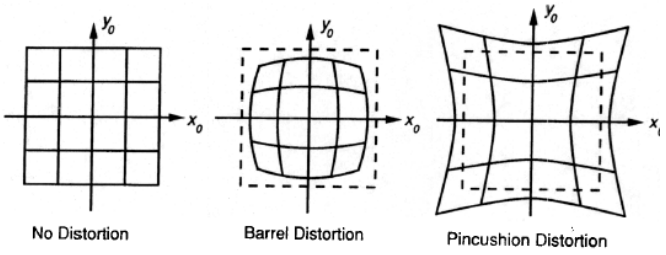
- predspracovanie obrazu – techniky používané zväčša na úpravu obrazových defektov, ktoré môžu byť špecifické pre každé zariadenie a treba ich odstrániť pred ďalšou fázou spracovania,
- spracovanie obrazu – všeobecné techniky na získavanie informácií z obrázku,
- po spracovanie obrazu – úprava obrázku do jeho konečnej formy pred zobrazením užívateľovi systému.

Z pohľadu herného systému, alebo agenta v danom systéme, ide o funkčný komponent slúžiaci na zachytenie informácie z reálneho sveta o stave prostredia, potrebnej pri procese interpretácie aktuálneho herného stavu do vnútornej reprezentácie stavu v počítačovom systéme.

1.4.1 Kalibrácia kamery

Bežné kamery spôsobujú skreslenie obrazu, ktoré je zapríčinené akým spôsobom sa premietnu lúče pri prechode cez šošovku kamery. Najbežnejším skreslením je tzv. radiálne skreslenie (radial distortion), čo spôsobuje skreslenie informácií, ktoré je najviditeľnejšie pri okrajoch obrázku (Obr. 1.4). Iný typ skreslenie, tzv. tangenciálne skreslenie (tangential

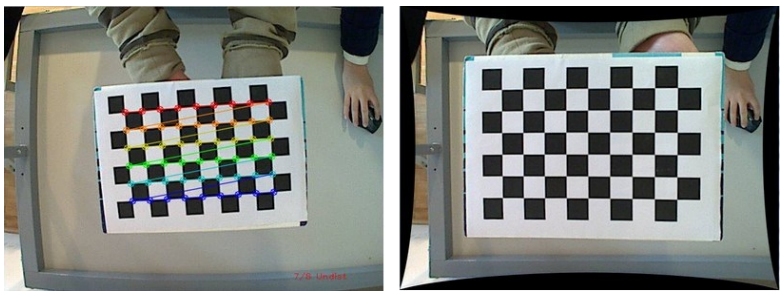
distortion), nastáva pri premietnutí roviny z reálneho sveta do 2D priestoru obrázku, pričom táto rovina nie je kolmá na os šošovky kamery.



Obr. 1.4 Skreslenie obrazu (radial distortion) [16]

Kalibrácia kamery je technika predspracovania obrazu, ktorá za pomocí transformácií vstupného obrazu dokáže tieto defekty odstrániť. Pri procese kalibrácie kamery sa opakovane zachytí referenčný objekt, ktorého rozmery a tvar sú nám známe (často sa používa tvar šachovnice), v rôznych pozíciách a rotáciách, čím sa pre danú kameru vyrátajú vnútorné a vonkajšie parametre, a transformačná matica kamery. [15] Tieto hodnoty stačí vyrátať pre konkrétnu kameru s konkrétnym rozlíšením len raz a môžeme ich opakovane používať.

Aplikáciou transformačnej matice získame nakalibrovaný obraz, vďaka čomu odstránime skreslenie obrazu a zjednotíme pomer medzi vzdialenosťami v reálnom svete a na obrázku.



Obr. 1.5 Štandardné kolekcie pre C++ a ich časová efektívnosť [15]

1.5 Robotika

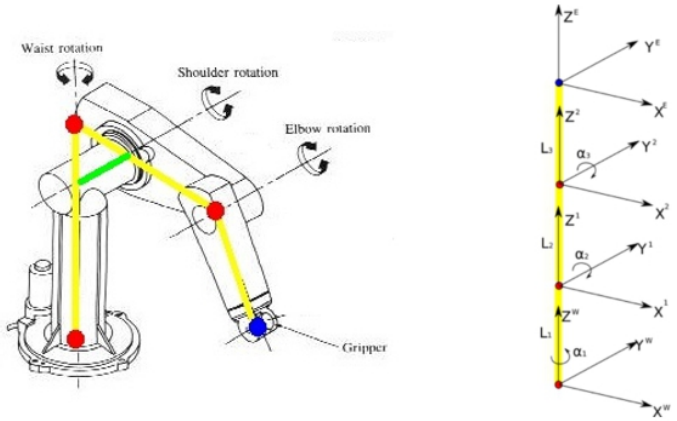
Robotika je technická disciplína zaoberajúca sa dizajnom, štrukturálnou konštrukciou, funkcionalitou, produkciou a spôsobom využitia robotov. Robotika sa venuje aj riadiacim a výkonným systémom určeným pre manipuláciu, ovládanie a komunikácia s robotom.

Pod pojmom robot si môžeme predstaviť agenta v reálnom prostredí, ktorý sa skladá zväčša z elektromechanických komponentov. Tvar a konštrukcia robota sa špecificky a osobitne prispôsobuje jeho účelu, zamýšľanej funkcionalite a do ktorého sa zamýšľa nasadiť.

Robot však nemusí byť len autonómny typ agenta, ale môže ísť jednoducho len o mechanický nástroj, ktorý vykonáva úlohy zadefinované iným agentom či systémom. V takom prípade sa jedná o robota s priamym riadením a môže vykonávať napríklad funkciu výkonného prostriedku (actuator) v nejakom robotickom systéme, ktorého je súčasťou. (1.2.2)

1.5.1 Robotické rameno

Robotické rameno je stacionárny typ robota, ktorý je mechanickou analógiou k ľudskej hornej končatine. Môže ísť o autonómneho agenta, vykonávajúceho samostatne určitý účel, alebo môže ísť o mechanické zariadenie na vykonávanie akcií s priamym riadeným od iného agenta alebo inej časti väčšieho systému.



Obr. 1.6 Robotické rameno s označenými DOF a znázornenie súradnicových systémov im prináležiacim [9]

Z pohľadu štrukturálnej koncepcie pozostáva robotické rameno zo 4 častí:

- Pevné nepohyblivé časti, ktoré majú svoju určenú nemennú dĺžku a slúžia ako prepojenia medzi kĺbmi ramena.
- Kĺby sú miesta, ktoré sú dynamickou časťou štruktúry ramena a môže v nich prichádzať k zmene aktuálnej podoby ramena. Charakter zmeny závisí od druhu kĺbu
  - posuvný kĺb – zmena má charakter presunu pevnej časti ramena prináležiacej danému kĺbu.

  - otočný kĺb – zmena má charakter rotácie pevnej časti ramena prináležiacej danému kĺbu.
- Manipulátor je súčasť ramena nachádzajúca sa na jeho konci a slúži na vykonávanie akcií pre ktoré je dané rameno určené, napr. použitie nástroja, uchytenie objektu a pod.
- Základňa je súčasť ramena nachádzajúca sa na jeho začiatku. Ide o časť, ktorou sa dané rameno pevne uchytí či do nejakej nehybnej štruktúry, napr. do steny budovy alebo konštrukciu nejakého mobilného robota. Súradnicový systém základne slúži ako referenčný súradnicový systém pre celé rameno.

Kĺby sú v robotickom ramene chápané ako tzv. stupne voľnosti (**DOF**, degrees of freedom). V týchto miestach dochádza k zmene aktuálneho tvaru ramena a to v neobmedzenej miere alebo obmedzenej, napr. maximálny možný rozsah otočného kĺbu. Zmena tvaru ramena závisí od konkrétneho typu kĺbu, posuvný alebo otočný, a taktiež podľa štruktúry kĺbu, čiže vzhľadom na akú os súradnicovej sústavy sa referencuje zmena. V 3 dimenzionálnom priestore teda poznáme 2 typy transformácií pevného telesa, rotáciu a posun, a obe sa ešte ďalej špecifikujú podľa toho, voči ktorej osi súradnicovej sústavy. [4] Z toho nám vyplýva 6 možných transformácií, stupňov voľnosti, v priestore 3D. Každý kĺb reprezentuje práve jeden stupeň voľnosti, viaceré stupne voľnosti sa môžu kombinovať dokopy ak je potrebný zložitejší transformačný aparát. Z dôvodu zníženia náročnosti výpočtov doprednej či inverznej kinematiky sa však snažíme udržať štruktúru ramena čo najjednoduchšiu a počet DOF čo najmenší. (1.5.4)

Každý kĺb spolu so svojou príslušnou pevnou časťou tvorí samostatnú logickú časť ramena a má pridelenú svoju vlastnú súradnicovú sústavu, ktorého počiatočný bod je tvorený pozíciou daného kĺbu (Obr. 1.6). Tieto jednotlivé súradnicové sústavy na seba nadväzujú tak, ako na seba nadväzujú cez pevné časti ramena jednotlivé kĺby. Na základe takéhoto zreťazenia sa pohyb jednotlivých častí ramena chápe ako postupnosť transformácií medzi jednotlivými súradnicovými sústavami. Túto postupnosť reprezentujeme ako nekomutatívne násobenie transformačných matíc.

V robotike sa pri procese návrhu rozloženia jednotlivých súradnicových sústav a ich vzájomnou polohou riadime štandardom Denavit-Hartenberg. (1.5.3) Držať sa štandardu samozrejme nie je nutné, nakoľko dané kinematické zreťazenie môžeme reprezentovať rôznymi spôsobmi, avšak jeho analýza môže byť príliš komplexná a vtedy nám štandard môže výrazne zjednodušiť celý proces. [8]

1.5.2 Súradnicové sústavy a transformácie

Jednotlivé transformácie medzi súradnicovými sústavami reprezentujeme transformačnými maticami. Pridaním ďalšieho rozmeru pre popis bodu v 3D priestore dostaneme homogénnu reprezentáciu súradníc $P = (x, y, z, 1)^T$ a transformačné matice rozmeru 4x4. Výhodou je, že v takomto tvare môžeme vyjadriť skladanie operácií, transformácií, ako nekomutatívne násobenie matíc, použiteľné nielen pre rotácie ale aj pre posun. [4]

Na jednoznačnú reprezentáciu objektov, akými sú aj pohyblivé časti robota, je v 3D priestore potrebné zaznamenať 3 pozičné a 3 rotačné veličiny. To znamená 6 transformačných matíc s ktorými je potrebné pracovať. [4]

1.5.3 Štandard Denavit-Hartenberg

Pri prideľovaní súradnicových sústav pre kinematické zreťazenie stupňov voľnosti robotického ramena je možné použiť viacero spôsobov, pričom štandardom je konvencia Denavit-Hartenberg. Tá používa na popis transformácií nadväzujúcich súradnicových sústav rovnomennú sadu parametrov zredukovanú na 4 parametre vzťahujúce sa na 2 osi, z a x.

Tieto 4 parametre popisujú vzťah medzi osami súradnicových sústav ***i-1*** a ***i***, ktorý na neho nadväzuje v poradí. Pričom os z_i je určená osou kĺbu a os x_i, je normálou k osiam z_{i-1} a z_i. Ich vzťah je nasledujúci: [8]

- a_i – určuje vzdialenosť od osi z_{i-1} k osi z_i vzhľadom na os x_{i-1}
- α_i – určuje uhol *(alfa)* medzi osou z_{i-1} a osou z_i vzhľadom na os x_{i-1}
- d_i – určuje vzdialenosť od osi x_{i-1} k osi x_i vzhľadom na os z_i
- θ_i – určuje uhol *(*theta) medzi osou x_{i-1} a osou x_i vzhľadom na os z_i

Transformácia medzi 2 súradnicovými sústavami sa dá zapísať jednou transformačnou maticou A_i získanou postupným vynásobením 4 transformačných matíc, ktoré reprezentujú jednotlivé kroky transformácie medzi sústavami. [8] Pomocou matice ${}^{i-1}A_i$ pre nadväzujúce sústavy ***i-1*** a ***i***, je možné jej násobením zľava vyjadriť súradnice bodu v súradnicovej sústave ***i*** v sústave ***i-1***.

1.5.4 Kinematika robotického ramena

Problematika kinematiky robotického ramena spadá do dvoch základných problémov:

- **dopredná kinematika** sa zaoberá nájdením súradníc pozície a natočenia manipulátora pri danej štruktúre a známom nastavení stupňov voľnosti ramena,
- **inverzná kinematika** má za cieľ nájsť aspoň jednu vhodnú konfigurácií variabilných parametrov robotického ramena na dosiahnutie žiadanej zadanej pozície manipulátora v priestore.

Pri doprednej kinematike vieme vďaka aplikácií konvencie Denavit-Hartenberg vypočítať celkovú transformáciu ${}^{0}T_{n}$, ktorá je výsledkom zreťazenia transformácií ${}^{0}A_{1} \; x \; \ldots \; x \; {}^{n-1}A_{n}$ a umožňuje nám vyjadriť pozíciu manipulátora v súradnicovej sústave základne.

Inverzná kinematika je vzhľadom na väčšiu variabilitu oveľa komplexnejší problém, ktorý môže mať množinu riešení s rôznymi úrovňami efektívnosti v praxi. Používa sa viacero typov prístupov k hľadaniu riešenia:

- Analytické prístupy – tieto metódy majú význam v prípade jednoduchej štruktúry robotického ramena. Problém si segmentuje na jednotlivé podproblémy až kým nebudú mať takú zložitosť aby sa dali jednotlivé náväznosti častí robotického ramena vyjadriť v rovniciach a z nich výsledok exaktne vypočítať.
- Iteratívne prístupy – zahŕňajú viacero podskupín podľa techník výpočtov pri jednotlivých krokoch iterácií. Základným numerickým prístupom je výpočet odchýlky pomocou parciálnych derivácií tzv. Jacobian technika IK a jej variácie. [7] Každý z iteratívnych prístupov má iné výhody a nevýhody, rozdiely sa týkajú presnosti aproximácie či výpočtového zaťaženia a z toho vyplývajúcej vhodnosti do jednotlivých aplikácii v praxi. [7]

1.6 Použité technológie

V tejto kapitole si stručne popíšeme technológie, ktoré budeme používať v tejto práci. Nakoľko ide o rozsiahle témy, zameriame sa na len na špecifické časti, ktoré sa týkajú našej práce.

1.6.1 DLV solver

DLV solver je deduktívny databázový systém, ktorý ponúka ASP formalizmus reprezentácie znalostí založený na disjuktívnom logickom programovaní. [11] (1.3.1)

Pri spustení programu sa ako vstupné parametre určí jeden alebo viac súborov obsahujúcich pravidlá logického programu. Odpoveď programu je množina literálov (answer set), ktorá tvorí Herbrandovský model pre daný logický program. Ak je ako odpoveď prázdny model, buď sa v programe neodvodili žiadne literály, alebo bol model vyhodnotený ako nekonzistentný.

Syntax jazyka DLV: [11]

Literál

nulárny predikát: *predicate,* unárny predikát: *predicate*(N),
n-árny predikát: *predicate*(A, …, N), explicitne negovaný literál: **-***literal*(X),
defaultovo negovaný literál: not *literal*(N),
ľubovoľné parametre v predikáte: *predicate*(A,_,_,B)

Pravidlo

hlava **:-** *telo***.** Telo pravidla je ukončené bodkou. V tele aj hlave pravidla môžu byť literály, ktoré sú oddelené čiarkami, čo reprezentuje konjunkciu.

Fakt

predicate(A,B)**.** Pravidlo bez tela je fakt.

Obmedzenie (constraint)

:- *literal_1*, *literal_2*, …, *literal_N***.** Pravidlo bez hlavy je obmedzenie.

Špeciálne vstavané predikáty

Funkčné predikáty **+, *, -,** / syntax napr. +(A,B,C) znamená A+B=C, pričom premenné A a B sa musia vyskytnúť v tele pravidla v inom predikáte, alebo môžu byť ľubovoľné čísla ak sú viazané predikátom #*int*(A), #*int*(B). Čísla sú nezáporné celé čísla, zhora ohraničené predikátom #*maxint*=N

Pri premenných treba dodržať **pravidlo bezpečnosti**: Ak sa premenná nachádza v hlave pravidla, alebo v tele pravidla pre defaultne negovaný literál, musí sa táto

premenná nachádzať v tele pravidla v aspoň jednom atóme, alebo explicitne negovanom literály, alebo ako posledná premenná vo predikáte vstavaných funkcií.

1.6.2 OpenCV

OpenCV (Open Source Computer Vision) je open source knižnica zameraná primárne na funkcionalitu počítačového videnia a spracovanie obrazu. [14]

1.6.3 C++

C++ je procedurálny, objektovo orientovaný (OOP) programovací jazyk. Na rozdiel od deklaratívneho programovania (1.3.1), kde neriešime ako sa program dopracuje k výsledku, ale len čo je cieľom, pri procedurálnom programovaní práveže naopak definujeme čo chceme vykonávať a cieľ nie je explicitne určený, ale implicitne pomocou postupnosti vykonávaných krokov.

Programovací jazyk C++ obsahuje štandardnú knižnicu, čo je množina tried a funkcií, ktoré sú štandardizované a sú natívnou súčasťou jazyka. Súčasťou štandardnej knižnice sú aj štruktúry na zoskupovanie rovnakého typu dát tzv. nádoby (containers), no poznáme ich ekvivalenty aj v iných programovacích jazykoch, zväčša pod názvom kolekcie (collections). Kolekcie sa líšia v spôsobe reprezentácie uložených dát, ich ukladania do pamäte, referenciami na susedné dáta v kolekcii, ako aj funkcionalitou nad danými dátami. Preto použitým vhodnej kolekcie, s ohľadom na zamýšľané použitie daných dát, vieme výrazne ušetriť časovú náročnosť pri vykonávaní operácií nad danými dátami.

Na obrázku 1.7 si môžeme porovnať efektívnosť daných kolekcií, pričom C znamená konštantný čas operácie a N počet členov kolekcie.

Container	Stores	Overhead	[ ]	Iterators	Insert	Erase	Find	Sort
list	T	8	n/a	Bidirect'l	C	C	N	N log N
deque	T	12	C	Random	C at begin or end; else N/2	C at begin or end; else N	N	N log N
vector	T	0	C	Random	C at end; else N	C at end; else N	N	N log N
set	T, Key	12	n/a	Bidirect'l	log N	log N	log N	C
multiset	T, Key	12	n/a	Bidirect'l	log N	d log (N+d)	log N	C
map	Pair, Key	16	log N	Bidirect'l	log N	log N	log N	C
multimap	Pair, Key	16	n/a	Bidirect'l	log N	d log (N+d)	log N	C
stack	T	n/a	n/a	n/a	C	C	n/a	n/a
queue	T	n/a	n/a	n/a	C	C	n/a	n/a
priority_ queue	T	n/a	n/a	n/a	log N	log N	n/a	n/a
slist	T	4	n/a	Forward	C	C	N	N log N

Obr. 1.7 Štandardné kolekcie pre C++ a ich časová efektívnosť [17]

1.7 Stolové hry

Stolová hra nemá nejakú jednoznačnú definíciu. Pod týmto pojmom teda môžeme chápať akúkoľvek hru, ktorej priestorové požiadavky sa zmestia na priestor stola, zväčša obsahuje hraciu plochu rozdelenú na hracie políčka a rôzne hracie objekty, ktoré na základe definovaných herných pravidiel môžu meniť svoju pozíciu alebo typ.
V tejto kapitole si zadefinujeme niekoľko stolových hier a ich pravidiel.

1.7.1 Žabky

Jedná sa o jednoduchú hru pre jedného hráča. Obsahuje 2 typy objektov, žabiek, a zväčša sa hrá na 7 hracích políčkach. Na začiatku hry sú objekty uložené tak, že jeden typ objektov je za sebou zoradených v rade, druhý typ objektov je zoradených za sebou na druhej strane a medzi nimi je práve jedno políčko prázdne. Cieľom je presunúť objekty z jednej strany na druhú, tak aby si dané typy objektov vymenili svoje pozície.

Možné akcie sú:

- objekt začínajúci zľava sa môže presunúť o jedno políčko vpravo ak je toto políčko voľné, objekt začínajúci sprava to isté len naopak,
- objekt začínajúci zľava môže preskočiť objekt druhého typu, pokiaľ sa tento nachádza o jedno políčko vpravo a nasledujúce políčko od neho napravo je voľné, objekt začínajúci sprava to isté len naopak.

V prípade, že hráč nemôže vykonať žiadnu akciu a hra nie je vo finálnom stave, hráč prehráva.

 O

Obr. 1.8 Začiatočná pozícia objektov v hre Žabky

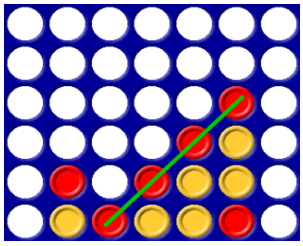
1.7.2 Connect4

Jedná sa o hru pre dvoch hráčov, každý hráč má jeden typ objektu, ktorý mu prináleží. Hracia plocha je tvaru obdĺžnika a jej veľkosť nie je určená, ale aby mala hra zmysel mala by mať minimálny rozmer 4x4. Na začiatku hry je hracia plocha prázdna.

Hráči sa na ťahu striedajú a každý ťah musí hráč vykonať akciu. V prípade, že je na hracej ploche už nie je prázdne políčko kam by sa dal položiť objekt, hra sa končí remízou. Hráč vyhráva v prípade, že sa mu podarí položiť objekty na plochu tak, že má v rade za sebou 4 svoje objekty, pričom môžu byť vertikálne, horizontálne alebo diagonálne.

Možné akcie sú:

- hráč môže položiť svoj objekt na prázdne políčko, pokiaľ sa jedná o políčko na najspodnejšom riadku hracej plochy, alebo sa na políčku o jeden riadok nižšie od tohto políčka nachádza ľubovoľný iný objekt.



Obr. 1.9 Ukážka konečného stavu v hre Connect4

1.7.3 Alquerque

Hra *Alquerque* je zjednodušenou verziou hry *Dáma*. Zadefinujeme si rozšírenú verziu pravidiel tejto hry. Hra je určená pre dvoch hráčov, každý z nich má jeden typ objektu, ktorý mu prináleží. Hráči sa na ťahu striedajú a každý ťah musí hráč vykonať akciu. V prípade, že hráč nemôže pohnúť žiadnym zo svojich objektov, prehráva. Hracia plocha je štvorcového tvaru zväčša rozmeru 5x5, alebo iné nepárne číslo. Na začiatku si hráči rozložia svoje objekty tak, aby ich mal každý na svojej strane a v strede zostalo jedno políčko prázdne. Hráč vyhráva ak sa mu podarí eliminovať z hracej plochy všetky objekty súpera, alebo ho dostať do stavu, že nemôže vykonať žiadnu akciu.

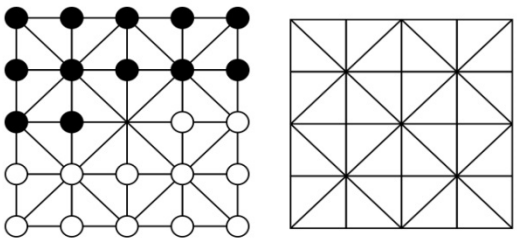
Možné akcie pre hráča začínajúci zospodu sú:

- Hráč môže svoje objekty pohnúť podľa mriežky hracej plochy na susediace voľné políčka smerom hore alebo na strany, nie smerom dole.
- Hráč môže preskočiť objekt svojho súpera, ak sa tento nachádza podľa mriežky hracej plochy na susediacom políčku a jedno políčko ďalej v tomto smere je prázdne. Pri tomto type akcie sa môže objekt pohnúť aj smerom nadol. Preskočený

objekt súpera je eliminovaný z hry. Hráč po tomto ťahu pokračuje na ťahu. Ak už hráč nemôže v takomto prípade ďalej vykonať žiadnu akciu, pokračuje v hre druhý hráč.

- Ak môže hráč eliminovať objekt súpera, musí vykonať tento typ akcie, nie obyčajný posun objektu.

Pre hráča začínajúceho zvrchu platia rovnaké pravidlá, len opačne.



Obr. 1.10 Ukážka začiatočného stavu v hre Alquerque a hracia mriežka

1.8 s3Games

Existujúci robotický multiagentový systém pre hranie stolových hier. Na zadefinovanie hry používa vlastný interpretovaný jazyk. Obsahuje rôzne algoritmy a prístupy umelej inteligencie, napr. neinformované prehľadávania, minimax, Monte Carlo a pod. Hrací proces sa dá simulovať, alebo hrať aj za pomoci robotického ramena na hracej ploche v reálnom svete. [18]

2 ŠPECIFIKÁCIA PRÁCE

Zadanie projektu

„Cieľom práce je navrhnúť, zostrojiť, implementovať a odladiť robotický systém, ktorý s ľudským spoluhráčom alebo samostatne dokáže hrať stolové hry využitím vhodných algoritmov napr. z oblasti umelej inteligencie a strojového učenia. Súčasťou robota bude systém počítačového videnia, robotické rameno, softvér s používateľským rozhraním a systém autonómneho rozhodovania pri hraní hry.“

Účel projektu

Projekt má za úlohu vyvinúť platformu:

- na hranie stolových hier, ktoré sa dajú pridávať podľa priloženej špecifikácie hier,
- kde ľudský hráč môže mať za oponenta autonómny systém rozhodovania, používajúci algoritmy umelej inteligencie,
- kde umelá inteligencia je doplnená fyzicky reálnym hraním za pomoci robotického ramena, čo zvyšuje zaujímavosť pre používateľa,
- na testovanie efektívnosti vyvinutých heuristík pre dané hry,
- na trénovanie učiacich sa algoritmov (strojové učenie, neurónové siete, …) a porovnanie ich výpočtovej a časovej efektívnosti voči iným algoritmom umelej inteligencie,
- na doplnenie funkcionality jednotlivých komponentov do iných, existujúcich projektov. (1.8)

Výsledný systém projektu by mal implementovať nasledujúce funkčné a nefunkčné požiadavky.

Špecifikácia funkčných požiadaviek

Systém má mať nasledujúcu funkcionalitu:

- vedieť komunikovať s jednotlivými súčasťami, získavať z nich informácie, slúžiť ako prostredník medzi jednotlivými súčasťami,
- spustiť ľubovoľnú zo zadefinovaných hier,
- jednoznačne identifikovať aktuálny stav prostredia a z toho stav hry,

- na základe zmeny stavu dať jednoznačné pokyny pre robotické rameno na vykonanie zmeny stavu objektov na hracej ploche,
- systém sa musí vedieť vysporiadať s nelegitímnymi akciami ľudského hráča, t.j. zachytiť vykonanie takejto akcie, upozorniť na situáciu a nedovoliť pokračovať v nelegitímnom stave hry,
- za vstupné dáta sa bude považovať konkrétna hra a jeden alebo viacerí hráči, s určenými vlastnosťami podľa špecifikácie pre hráčov,
- za výstupné dáta sa bude považovať súhrn informácií o priebehu a výsledku ukončenej hry.

Objekty sa po akcii ľudského hráča môžu nachádzať na ľubovoľnej pozícii hracej plochy alebo mimo nej, preto treba zabezpečiť schopnosť systému prispôsobiť sa novému stavu:

- pomocou systému počítačového videnia získať vizuálnu informáciu o stave prostredia v reálnom svete, vedieť vyhodnotiť jeho legitímnosť z pohľadu špecifikácie hry a vedieť tak zachytiť dynamicky sa meniaci stav na hracej ploche,
- robotické rameno musí byť schopné pracovať s ľubovoľnými pozíciami objektov na hracej ploche aj mimo nej v dosahu ramena, t.j. nie len s preddefinovanými pozíciami.

Systém autonómneho rozhodovania sa má mať nasledujúcu funkcionalitu:

- zistiť možné akcie hráča v danom stave hry,
- na základe definovaného algoritmu umelej inteligencie pre daného hráča vyhodnotiť možné akcie a vybrať najperspektívnejšiu akciu z pohľadu zadaných kritérií,
- po zvolení výslednej akcie vedieť túto informáciu odkomunikovať s príslušnou časťou systému za účelom zmeny stavu hry,

Špecifikácia nefunkčných požiadaviek:

- navrhnúť prostredie, fyzickú architektúru systému a umiestnenie jeho komponentov (systém počítačového videnia, robotické rameno, hracia plocha) v priestore, vzhľadom na parametre vybraných zariadení,
- systém má umožniť hrať stolové hry, kde sú akcie vykonávané jedným alebo viacerými hráčmi,
- hráčom môže byť ľudský jedinec alebo systém autonómneho rozhodovania sa za pomoci vhodných algoritmov umelej inteligencie,

- systém má obsahovať prostriedok na získanie vizuálnej informácie z prostredia – počítačové videnie,
- systém má obsahovať priestor s hracími objektami – hraciu plochu,
- systém má obsahovať prostriedok na manipuláciu s objektami na hracej ploche – robotické rameno,
- rozšíriteľnosť – systém by mal byť delení na samostatné modulárne komponenty, ľahšia rozšíriteľnosť v prípade zmeny prostredia.

Obmedzenia systému a časové priority:

- je možné zúžiť okruh potencionálnych hier podľa určitých obmedzujúcich podmienok,
- informácie o priebehu, stave a výsledku hry nemusia byť vizualizované, postačujúca bude aj textová informácia,
- v prvej fáze projektu postačujú ľubovoľné algoritmy pre umelú inteligenciu v autonómnom rozhodovacom systéme. S učiacimi algoritmami sa ráta až pre nasledujúce prípadné fázy projektu.

Atribúty kvality

Systém sa bude považovať za úspešný, ak:

- funkcionalita systému počítačového videnia bude vedieť získať, spracovať a poskytnúť všetky informácie potrebné pre hranie hry,
- funkcionalita robotického ramena zvládne s dostatočnou presnosťou pracovať s hracími objektami,
- autonómny rozhodovací systém bude vedieť rozumne reagovať na akcie súpera, t.j. reakcia nebude náhodná a správnosť akcie závisí od kvality heuristiky či kvality natrénovanej stratégie,
- reakcia autonómneho rozhodovacieho systému pri hre (nie trénovaní) s ľudským oponentom musí byť v rámci prípustného časového limitu, t.j. musí byť vykonaná dostatočne rýchlo, aby hra zostala pre ľudského hráča zážívná.

Kvalita samotného autonómneho rozhodovacieho systému sa určí na základe vyhodnotení výsledkov hier voči rôznym typom oponentov.

3 NÁVRH PRÁCE

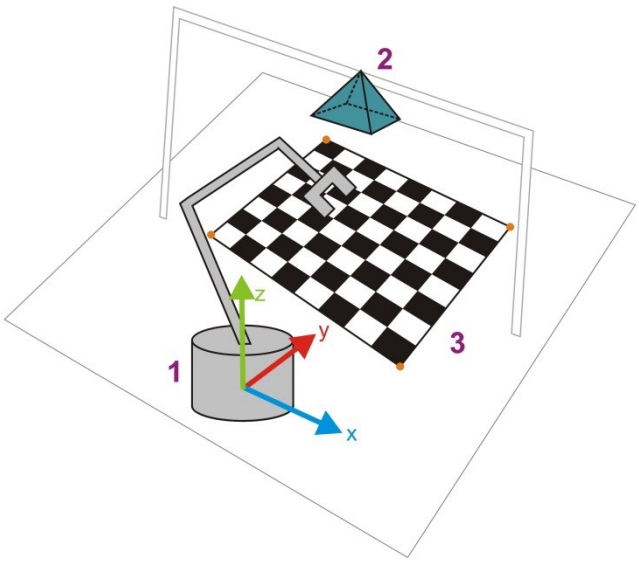
V nasledujúcich kapitolách si najskôr analyzujeme zadefinovanú špecifikáciu práce a následne sa pozrieme na navrhovanú každú časť systému osobitne.

3.1 Analýza špecifikácie práce

Pozrime sa na požiadavky z celkového obrazu a jednotlivé požiadavky na konkrétne časti systému si zhodnotíme v ďalšej podkapitole.

Charakteristika systému z pohľadu P.E.A.S. (1.2.2)

- ***Výkonné prostriedky (Actuators)*** – Robotické rameno ako jeden spoločný výkonný prostriedok pre všetkých počítačových agentov. Končatiny pre ľudského agenta.
- ***Senzory (Sensors)*** – Kamera pre ako jeden spoločný senzor pre všetkých počítačových agentov. Oči a hmat pre ľudského agenta.
- ***Miera výkonu (Performance Measure)*** – Počet výhier, dĺžka trvania hry, množstvo výpočtového výkonu, zložitosti pre počítačového hráča a iné.
- ***Prostredie (Environment)***
  - Plne pozorovateľné
    Kamera vníma a získava kompletnú informáciu o stave prostredia.
  - Deterministické
    Nasledujúci stav hry bude jednoznačne určený dvojicou stav – akcia.
  - Sekvenčné (Neepizodické)
    Agenti môžu využívať znalosti o predchádzajúcich stavoch hry.
  - Statické
    Legálne zmeny stavu hry sa udejú len po akcii agenta, ktorý je na ťahu.
  - Diskrétne
    Prostredie má konečný počet hracích pozícií, objektov a akcií agentov.
  - Multiagentové
    Systém je určený pre jedného alebo dvoch agentov.

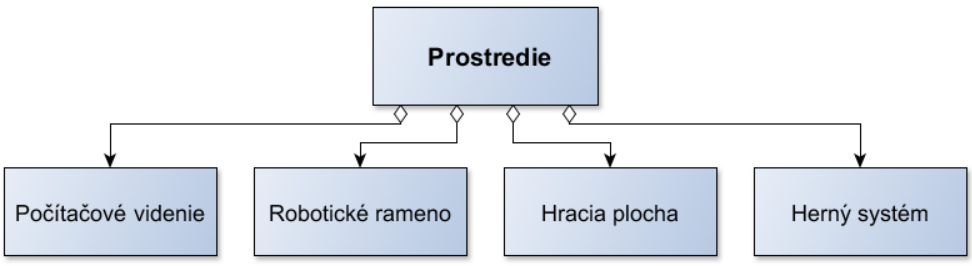


Obr. 3.1 Grafická ilustrácia navrhovaného systému

Ako môžeme vidieť na grafickej ilustrácii návrhu systému na obrázku 3.1, ako referenčný bod priestoru v reálnom svete pre celý systém si zvolíme stred základne robotického ramena.

3.2 Dekompozícia systému

Na základe nefunkčných a funkčných požiadaviek sa celý systém prirodzene delí na 4 samostatné moduly, ktoré sa od seba oddeľujú nielen z funkčného, ale aj z fyzického hľadiska, znázornené v diagrame na obrázku 3.2.



Obr. 3.2 Diagram dekompozície systému na jednotlivé komponenty

Kamera

Jedným zo zjednodušení systému bude úroveň vizuálneho vnemu prostredia. Budeme predpokladať, že si vystačíme s dvojrozmernou informáciou o stave prostredia. Postačíme si teda s jednou kamerou, ktorú umiestnime kolmo nad hraciu

plochu, tak ako je znázornené na grafickej ilustrácii návrhu systému na obrázku 3.1. Stratíme tak pojem o hĺbke priestoru na základe vizuálneho vnemu, avšak toto obmedzenie je zanedbateľné pri stanovení fixnej výšky umiestnenia kamery v priestore.

Nakoľko plánujeme aplikáciu v reálnom svete, nemôžeme sa nutne spoliehať na podmienku úplnej kolmosti kamery voči rovine hracej plochy určenej osami *x* a *y* súradnicovej sústavy systému (Obr. 3.1), ani na nulovú rotáciu obrazu hracej plochy voči osi *z*. Kým s prvou záležitosťou sa vyrovnáme kalibráciou kamery (1.4.1), druhú záležitosť budeme riešiť funkcionalitou v komponente hracej plochy.

Robotické rameno

Máme vybratú konštrukciu robotického ramena *ALB5* od firmy *Lynxmotion*, pričom ako rozhranie na ovládanie ramena využijeme existujúcu riadiacu dosku s firmwarom SSC-32 od rovnakej firmy. [19]

Štruktúru robotického ramena treba podrobne analyzovať a uchopiť jej reprezentáciu v priestore cez štandard Denavit-Hartenberg (1.5.2), ktorého aplikovaním získame priamo funkcionalitu doprednej kinematiky. (1.5.4) Riešenie úlohy inverznej kinematiky nie je tak priamočiare, ale nakoľko je štruktúra vybraného robotického ramena dostatočne nekomplikovaná, dá sa pristúpiť k hľadaniu analytického riešenia. Nakoľko ale z povahy problému inverznej kinematiky vyplýva existencia viacerých možných konfigurácií ramena na dosiahnutie zadanej pozície v priestore, bude sa nutné vysporiadať s touto nejednoznačnosťou.

Hracia plocha

Ako hraciu plochu si určíme tvar šachovnice. Ide o základný tvar hracej plochy pre stolové hry, pričom sa na nej dajú ľahko reprezentovať prechodové relácie medzi jednotlivými pozíciami hracej plochy. Šachovnicový tvar slúži len ako podklad, abstraktný tvar plochy, pričom sémantický význam hracej plochy ako možných pozícií hracích objektov a prechodov medzi týmito pozíciami sa určí svojsky pre každú konkrétnu hru pri jej špecifikovaní.

Tak ako sme už spomenuli pri komponente kamery, funkcionalita komponentu hracej plochy sa musí vedieť vysporiadať aj s vizuálnym vnemom šachovnice,

ktorý môže byť rotovaný voči pomyselnej osi *z* obrázku, teda strany šachovnice by neboli rovnobežné s osami *x* a *y* obrázku.

Taktiež musí implementovať transformačný aparát na prevod bodov z dvojrozmerného priestoru obrázku do trojrozmerného priestoru reálneho sveta.

Herné jadro

Komponent herného jadra bude mať na starosti reprezentáciu znalostí pre danú hru, reprezentáciu a simuláciu celého herného procesu, a zabezpečovať komunikáciu medzi agentami a prostredím, vrátane spracovania uskutočnených akcií.

Komponent musí vedieť komunikovať s externým modulom (programom) určeným na tvorbu možných akcií. Ako vstup do externého modelu musí vedieť v špecifikovanej syntaxi poskytnúť informáciu o aktuálnom stave prostredia. Ako výstup z externého modelu musí vedieť spracovať informáciu o možných akciách v aktuálnom stave. Tieto informácie budú v generickej syntaxi akcií, ktorých sémantický význam komponent nerieši. Po validácii parametrov akcie voči aktuálnemu stavu hry ich bude považovať za legálne akcie.

Externý modul

Externý modul v sebe skryje celú tvorbu možných akcií pre daný stav konkrétnej hry. Sémantika za zmyslom danej akcie sa implicitne ukrýva v zadefinovanom programe logického programovania. Komponent herného jadra už spracúva len generickú akciu, v špecifikovanej syntaxi, o ktorej zmysle nemá žiadnu informáciu.
Z hľadiska existujúcej optimalizovanej implementácie ASP paradigmy logického programovania pre riešenie problémov zadefinovaných ako hry, sme si vybrali tento prístup. Na hľadanie modelov, ASP odpovedí, pre zadefinovaný problém hry sme si vybrali program DLV solver. (1.6.1)

Obmedzenia požiadavky ľubovoľnosti hier a iné **zjednodušenia** vychádzajúce z nášho návrhu systému sú

- veľkosť hracieho priestoru je zhora ohraničená veľkosťou reálnej šachovnice ako abstraktnej hracej plochy,
- na jedno hracie políčko pripadá maximálne jeden hrací objekt,

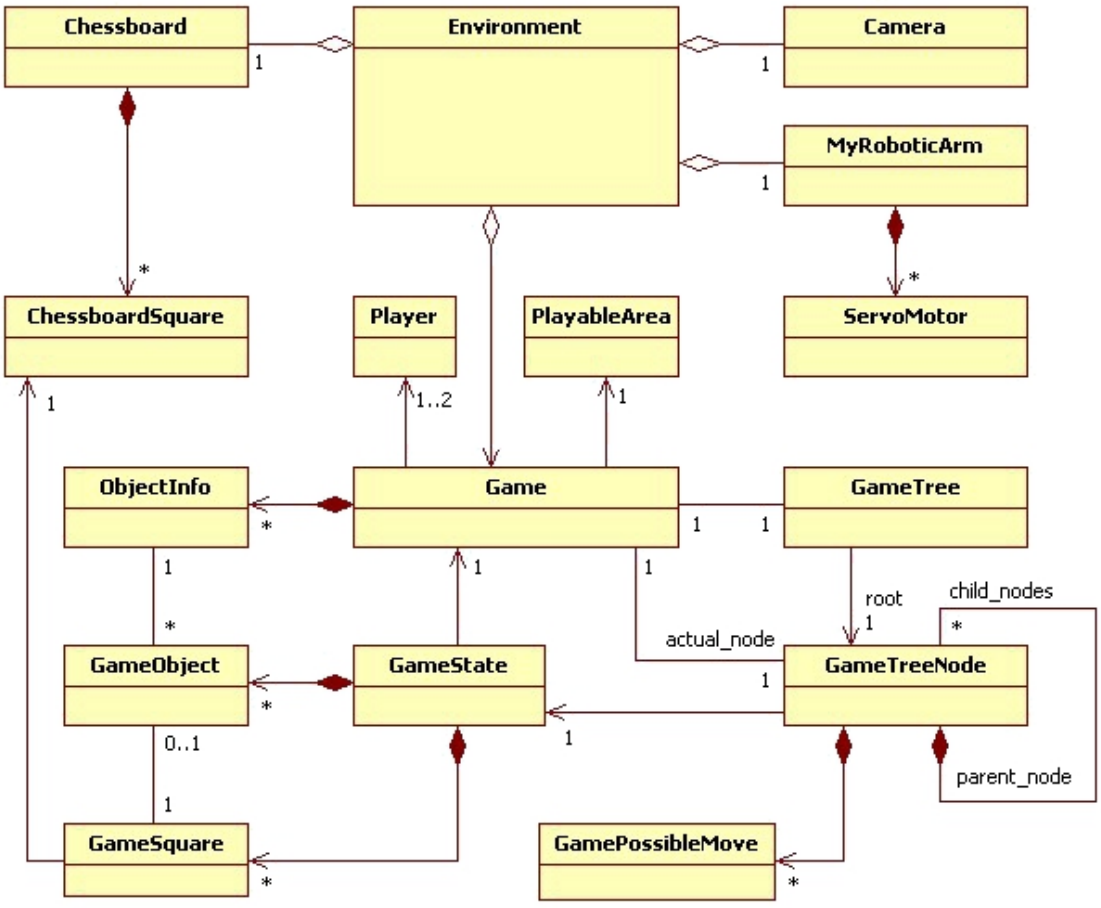
- hracie objekty nerozpoznávame subjektívne, ale len na základe ich typu, čím sa kombinácia (index hracej pozície, typ objektu na danej pozícii) stáva unikátnym identifikátorom,
- tvar hracích objektov musí byť jednoznačne rozpoznateľný pri navrhnutom zjednodušenom vizuálnom vneme prostredia, a ich veľkosť uchopiteľná navrhnutým manipulátorom robotickej ruky.

4 IMPLEMENTÁCIA PRÁCE

V tejto kapitole si popíšeme podstatné časti implementácie návrhu pre jednotlivé komponenty systému.

4.1 Implementácia systému

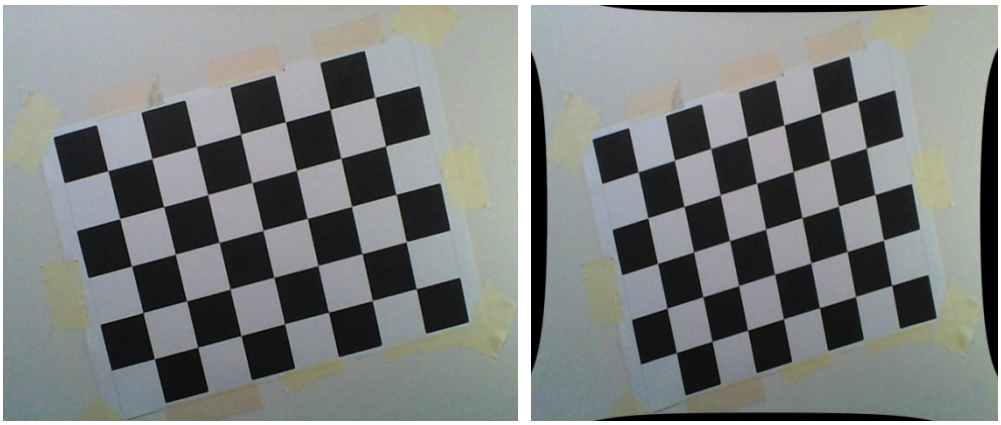
Systém sa nám podarilo implementovať podľa zadaných cieľov z väčšej časti návrhu. Na obrázku 4.1 je znázornený triedny diagram tých tried, ktoré sú podstatné z pohľadu štruktúry systému a jeho rozdeleniu na 4 hlavné komponenty.



Obr. 4.1 Triedny diagram vybraných tried systému

Realizácia komponentu Kamera (class Camera)

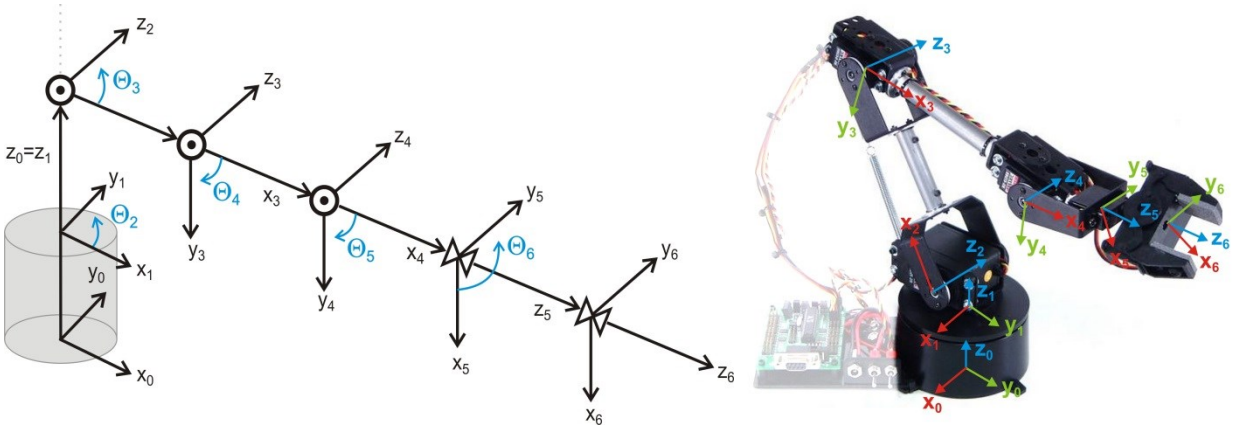
Pre komponent Kamera sa nám podarilo implementovať funkcionalitu kalibrácie kamery, pričom sa automaticky spustí v prípade, že systém nenájde uložené kalibračné hodnoty pre danú konkrétnu kameru.



Obr. 4.2 Ukážka kalibrácie kamery

Realizácia komponentu Robotické rameno (class MyRoboticArm)

Pre komponent Robotické rameno sa nám podarilo implementovať funkcionalitu doprednej a inverznej kinematiky. Na obrázku 4.3 sú graficky znázornené súradnicové systémy robotického ramena priradené na základe štandardu Denavit-Hartenberg (1.5.2). Konkrétne parametre na základe ktorých si vytvoríme transformačné matice [8] medzi jednotlivými súradnicovými sústavami sa dajú nájsť v tabuľke 4.1.

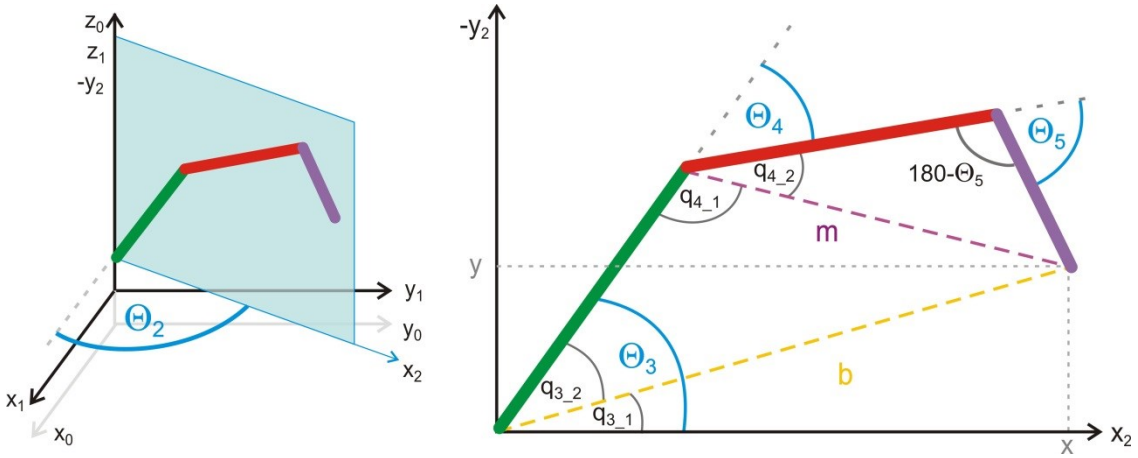


Obr. 4.3 Súradnicové systémy pre stupne voľnosti nášho robotického ramena

	a_i	α_i	d_i	Θ_i
6	58	0	0	Θ_6
5	56	90	0	$90 + \Theta_5$
4	182	0	0	Θ_4
3	147	0	0	$- \Theta_3$
2	0	-90	19	Θ_2
1	0	0	48	0

Tab. 4.1 Parametre robotického ramena podľa štandardu Denavit-Hartenberg (1.5.3)

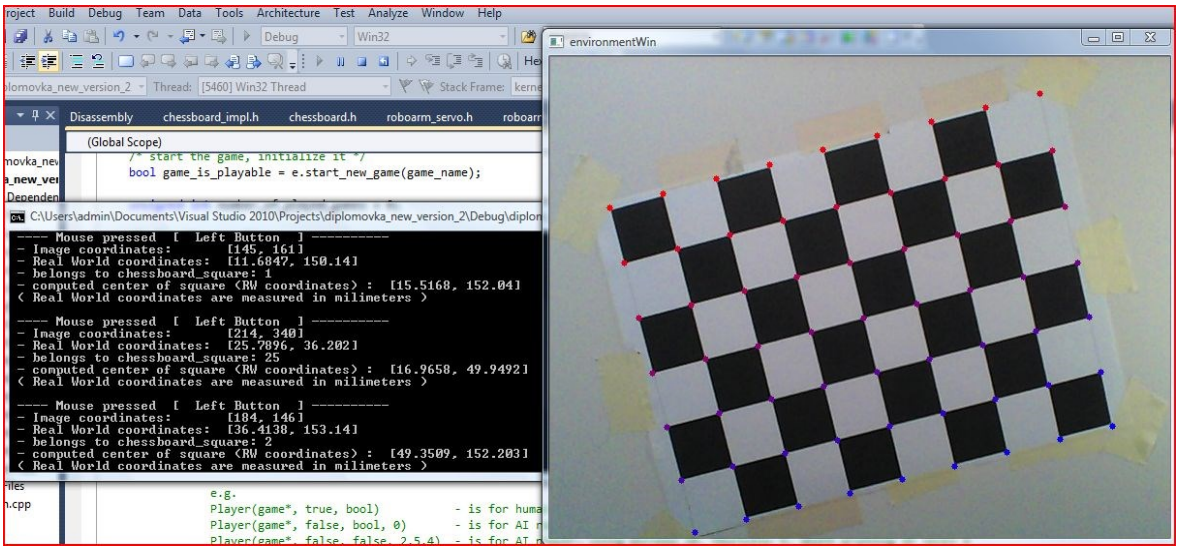
Na obrázku 4.4 je znázornená grafická analýza robotického ramena v 3D priestore reálneho sveta, na základe ktorej sme dopočítali pomocou kosínusovej a Pytagorovej vety všetky potrebné uhly. Do vzniknutých rovníc treba umelo doplniť uhol Θ_5. Algoritmus v aktuálnej verzii systému vyžaduje túto hodnotu ako vstupný parameter. Avšak po neúspešnom pokuse o dosiahnutie daného bodu priestoru so zadanou hodnotu uhla, sa algoritmus následne pokúsi prejsť postupne celý rozsah daného stupňa voľnosti. Takýmto spôsobom zaručí, že ak je rameno schopné dosiahnuť zadaný bod, náš algoritmus nájde aspoň jednu konfiguráciu stupňov voľnosti ramena aby tento bod dosiahol.



Obr. 4.4 Grafická analýza inverznej kinematiky nášho robotického ramena

Realizácia komponentu Hracia plocha (class Chessboard)

Pre komponent Hracej plochy sme implementovali funkcionalitu nájdenia bodov šachovnice. Následne pomocou hodnôt z priestoru reálneho sveta pre niektoré body šachovnice, ktoré získame ako vstupné parametre do komponentu, vypočítame transformačnú maticu pre prechod z priestoru obrázku do priestoru reálneho sveta.

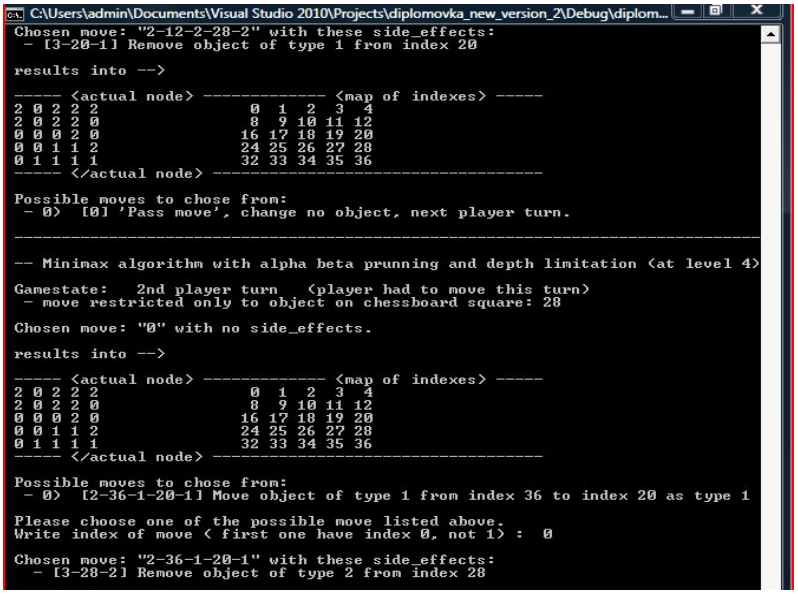


Obr. 4.5 Ukážka funkcionality komponentu Hracia plocha

Na pravej strane obrázku 4.5 vidíme snímku z kamery s vyznačenými bodmi šachovnice a na ľavej strane vidíme textovú informáciu s prepočtom koordinát bodu snímky, na ktorý sme klikli myšou. Tento bod je prepočítaný do priestoru reálneho sveta a je identifikovaná pozícia na šachovnici.

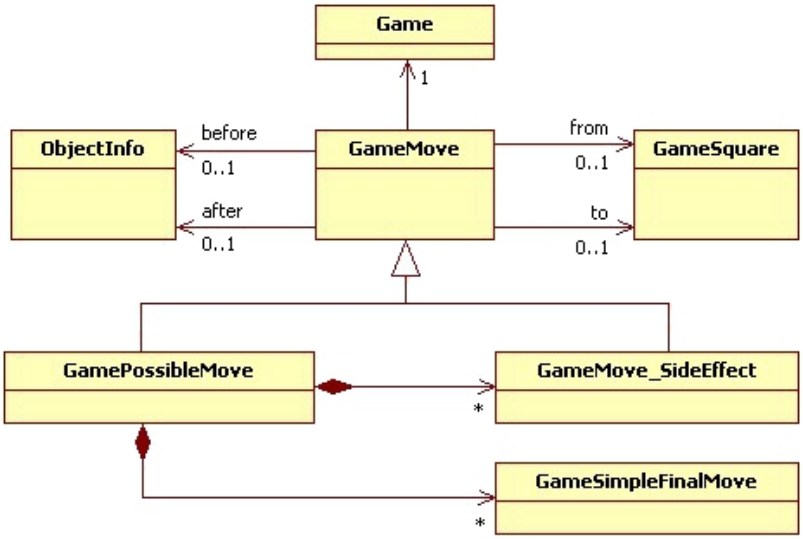
Realizácia komponentu Herné jadro (class Game)

Pre komponent Herné jadro sa nám podarilo implementovať kompletnú funkcionalitu potrebnú pri simulácii hry, či už počítačových agentov voči sebe, alebo ľudskému agentovi.



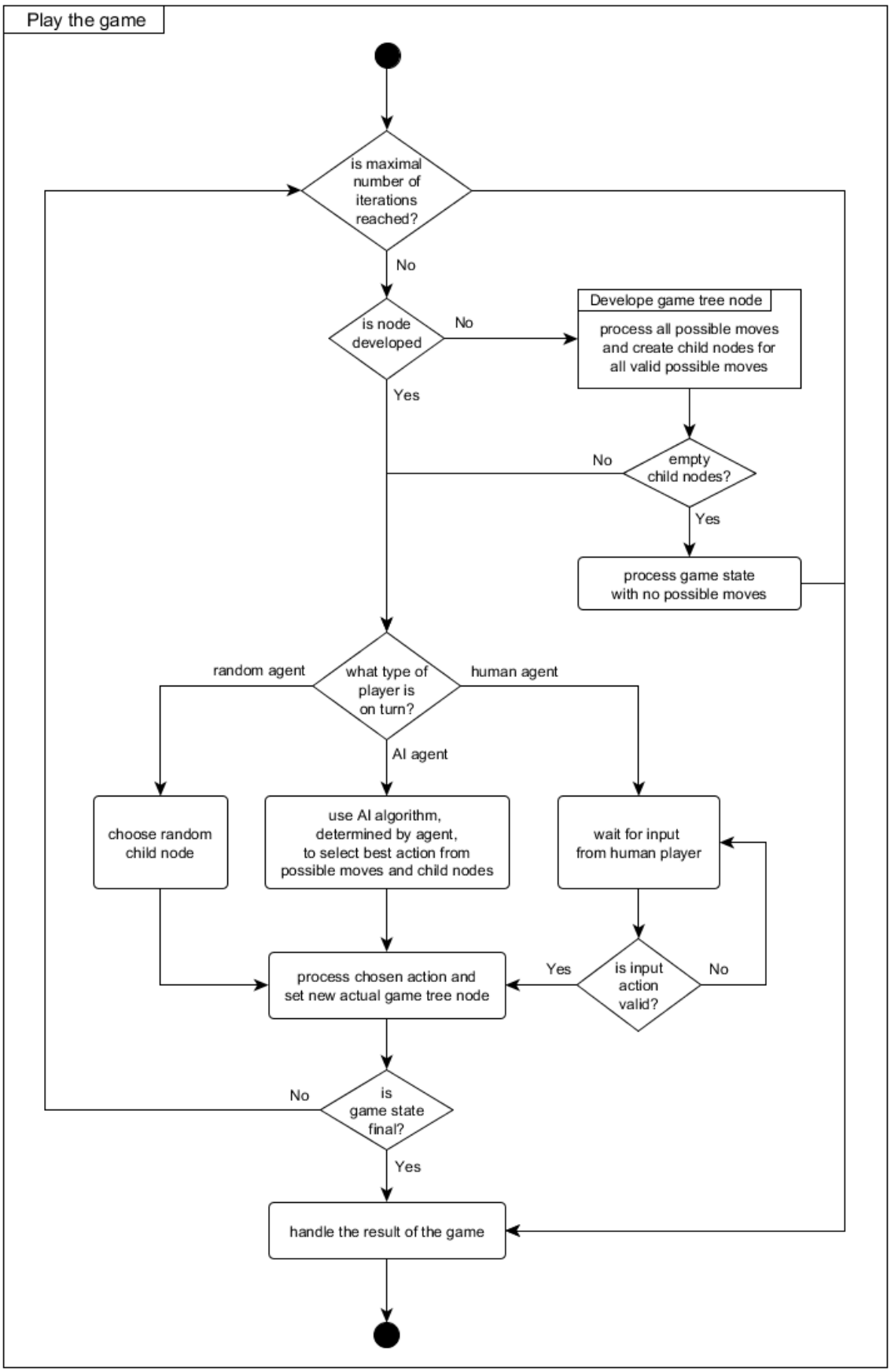
Obr. 4.6 Ukážka textovej informacie pri hre

Akcie agentov sa vnútorne nazývajú možné ťahy. Ako je vidno na obrázku 4.7, možný ťah je hlavná akcia, ktorá sa má vykonať, avšak tieto akcie môžu sprevádzať vedľajšie efekty, ako napríklad vyhodenie preskočeného objektu v dáme.

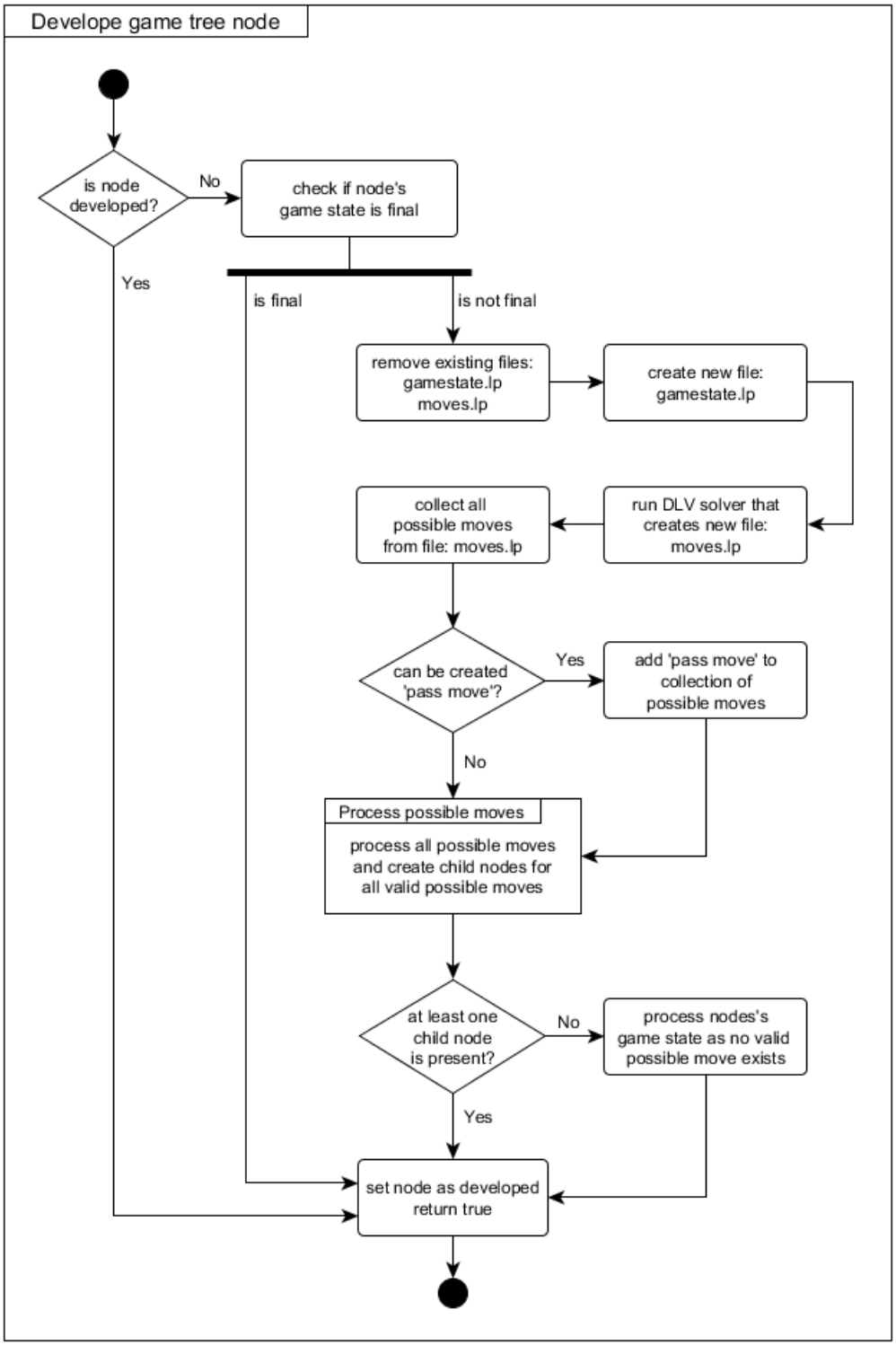


Obr. 4.7 Triedny diagram pre akcie agentov

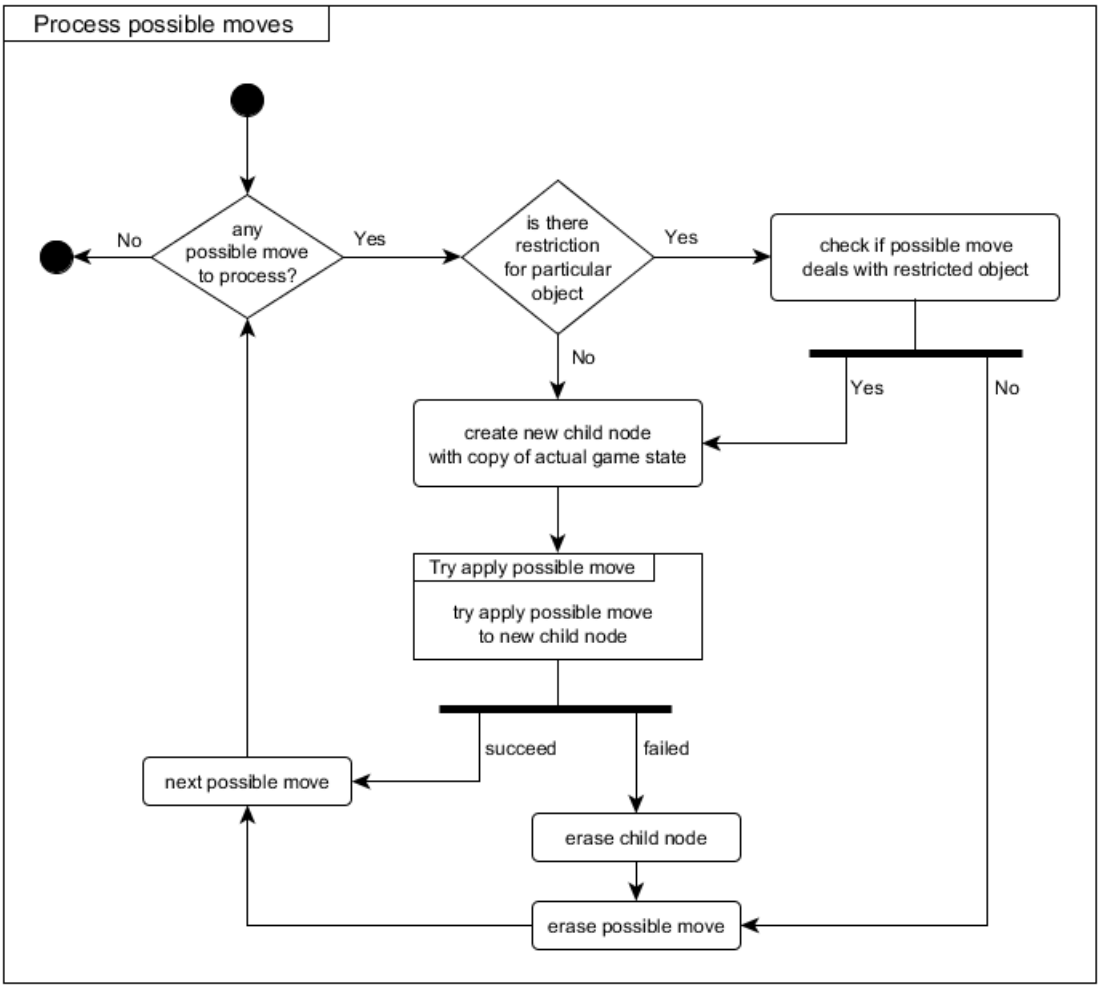
Na nasledujúcich obrázkoch je zobrazený vývojový diagram pri jednotlivých iteráciách herného procesu.



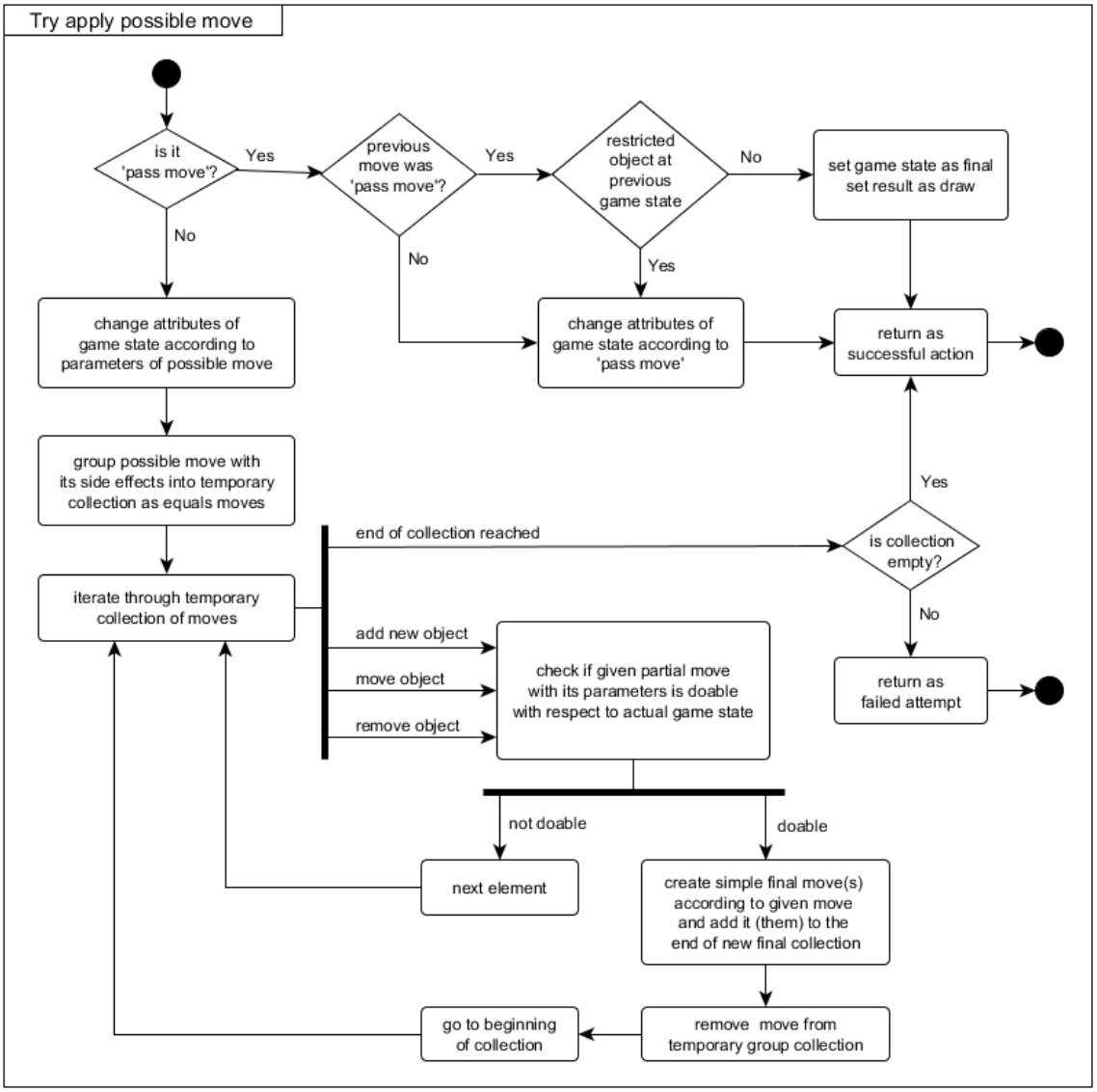
Obr. 4.8 Vývojový diagram pre proces hrania



Obr. 4.9 Vývojový diagram pre rozvoj uzla herného stromu



Obr. 4.10 Vývojový diagram pre spracovanie možných ťahov v stave



Obr. 4.11 Vývojový diagram pre aplikovanie možného ťahu na herný stav

5 ZHODNOTENIE A TESTOVANIE SYSTÉMU

V tejto kapitole si zhodnotíme úspešnosť implementácie a mieru funkčnosti jednotlivých komponentov systému a otestujeme si efektívnosť algoritmov umelej inteligencie pri hraní našich troch zadefinovaných hier.

5.1 Zhodnotenie systému

Systém ako celok sa podarilo implementovať z väčšej časti návrhu systému. (3.2) Prepojenie komponentov je buď kompletne implementované alebo aspoň pripravené na ďalší vývoj. Pozrime sa konkrétnejšie na jednotlivé komponenty.

Kamera

Systém počítačového videnia plne podporuje kalibráciu kamery, pričom je pripravená funkcionalita, ktorá vie po zapojení iného snímacieho zariadenia spustiť kalibračný proces a uložiť výsledné hodnoty k budúcemu použitiu. Pomocou knižnice OpenCV dokáže komponent kamery nájsť na získanom obraze prostredia aj vnútorné body šachovnice ako priestor hracej plochy a posunúť túto informáciu na ďalšie spracovanie do komponentu hracej plochy.

Robotické rameno

Komponent robotického ramena sa podarilo implementovať celý, až na sériovú komunikáciu s obslužným hardvérom ramena. Štruktúra ramena bola analyzovaná podľa štandardu Denavit-Hartenberg (1.5.2) a na jej základe bola implementovaná funkcionalitu doprednej kinematiky. Pri analýze štruktúry ramena sa podarilo aj navrhnúť a implementovať analytický prístup k funkcionalite inverznej kinematiky. Správna funkčnosť oboch kinematík bola simuláciou testovaná a overená aj pre hraničné body manipulačného priestoru ramena.

Hracia plocha

Komponent hracej plochy je implementovaný s obmedzením na aktuálne celkové použitie len na simuláciu hry, nie na použitie v reálnom svete. Komponent vie spracovať výstup z komponentu kamery, pričom si vie doplniť chýbajúce body

šachovnice, bez ohľadu na natočenie kamery, čím nie je obmedzený na použitie len pri presnom kolmom tvare získanej vizuálnej informácii o hracej ploche. Na základe parametrov získaných pri inicializácií, obsahujúcich body šachovnice z priestoru reálneho sveta si vie dopočítať transformačnú maticu medzi priestorom obrázku a priestorom z reálneho sveta. Vďaka tomu vie posunúť presnú informáciu o hľadanom bode priestoru do komponentu robotického ramena, ktoré túto informáciu spracuje pomocou funkcionality inverznej kinematiky. K vykonaniu procesu hry v priestore reálneho sveta chýba komponentu funkcionalita spracovania obrazu za účelom nájdenia hracích objektov na ploche.

Herné jadro

Komponent herného jadra je implementovaný kompletne pre proces simulácie hry, avšak z dôvodu chýbajúcej funkcionality pre proces hrania v priestore reálneho sveta v predošlých komponentoch, nemá ani tento komponent zatiaľ túto funkčnosť. Komponent obsahuje stromovú štruktúru kompletného herného procesu, implementovanú s ohľadom na priestorovú zložitosť. Taktiež ponúka simuláciu herného procesu pomocou náhodného hráča, hráča s využitím algoritmov umelej inteligencie, konkrétne algoritmov Minimax a jeho modifikácie Minimax s *alfa-beta* orezávaním, a samozrejme aj možnosť pre akcie vybrané ľudským hráčom.

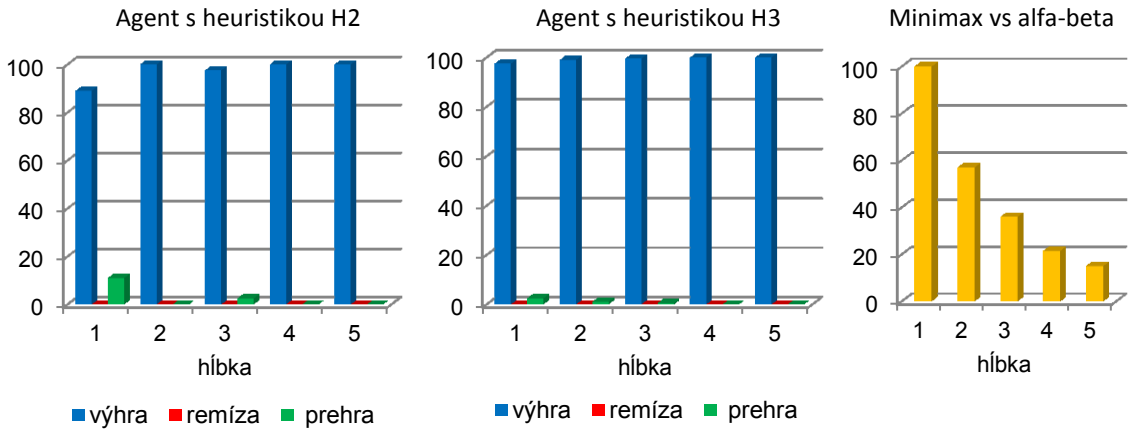
5.2 Výsledky pre testované hry

Hra Žabky

Hra *žabky* bola vybraná za účelom demonštrácie hry pre jedného agenta, čo sa aj podarilo ukázať. Pri testovaní hry náhodným agentom sa nepodarilo ani raz ukončiť hru výhrou, napriek vysokému počtu pokusov, celkovo 10000 krát. Pri hraní agenta s využitým algoritmu Minimax sa agentovi podarilo vyhrať až pri použití hĺbky 6 a väčšej. Takýto zlý výsledok má na svedomí pravdepodobne príliš jednoduchá heuristická funkcia, ktorá nedokázala odhadnúť nevhodný stav. Pri použitý hĺbky 6 vidí agent až za celú najdlhšiu postupnosť ťahov, ktoré je nutné pri tejto hre hneď od druhého ťahu spraviť v presnom poradí.

Hra Connect4

Hru sme testovali na celej hracej ploche 8x6, z čoho vyplýva konštantný faktor vetvenia rovný 8 a veľkosť úplného stromu pri algoritme Minimax $O(8^h)$, kde h je určená hĺbka orezania. Ak by sme neurčili hĺbku orezania, pri hodnote maximálnej hĺbky 48 by sa stal algoritmus nepoužiteľný. Pre testovanie sme zvolili maximálnu hodnotu hĺbky orezania 5 a testovali sme obe zadefinované heuristiky *H2* a *H3* voči náhodným agentom, ako aj voči sebe.



Obr. 5.1 Výsledky pre hru connect4 proti náhodnému agentovi

Na obrázku 5.1 môžeme vidieť percentuálnu úspešnosť agentov s použitým algoritmu Minimax voči náhodnému agentovi, za použitia oboch heuristík, pričom sa agenti striedali kto začínal hru. Tretí graf na tomto obrázku 5.1 vyjadruje mieru efektivity algoritmu Minimax voči Minimax alfa-beta, kde hodnota pre určenú hĺbku orezania označuje koľko percent uzlov spracoval algoritmus Minimax alfa-beta z celkového počtu spracovaných uzlov algoritmom Minimax. Napríklad pri hĺbke 4 už algoritmus Minimax alfa-beta spracoval len 21 % uzlov, ktoré musel spracovať algoritmus Minimax.

		agent 1 H2				
		1	2	3	4	5
agent 2 H3	1	P	V	P	V	V
	2	P	V	P	V	V
	3	P	P	P	V	V
	4	P	P	P	P	V
	5	P	V	P	P	P

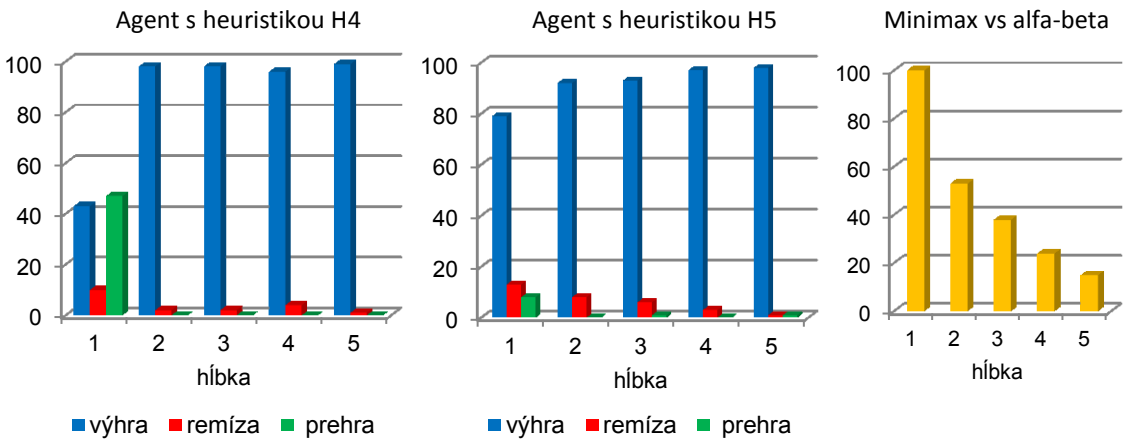
		agent 1 H3				
		1	2	3	4	5
agent 2 H2	1	P	V	V	V	V
	2	P	V	V	V	V
	3	P	V	V	V	V
	4	P	V	P	P	V
	5	P	V	P	V	V

Obr. 5.2 Výsledky pre hru connect4 pre komperatívnych Minimax agentov

Na obrázku 5.2 môžeme vidieť ešte porovnanie heuristík voči sebe za použitia algoritmu Minimax pri rôznych hĺbkach orezania. Agent nachádzajúci hore začínal danú hru a z jeho pohľadu *V* znamená výhra, *P* prehra a *R* remíza.

Hra Alquerque

Hru sme testovali na zmenšenej hracej ploche 5x5, a faktor vetvenia sa pri druhej polovici tejto hre vyšplhal až ku hodnote 12. Ak by sme neurčili hĺbku orezania, hra by mohla ísť teoreticky donekonečna, nakoľko sa dá dostať do stavu kedy už nemôže vyhrať ani jeden agent, ale nie sú to konečné stavy. Vtedy sme hru po istom maximálnom počte akcií zastavili a ukončili ako remízu. Pre testovanie sme zvolili maximálnu hodnotu hĺbky orezania 5 a testovali sme obe zadefinované heuristiky *H4* a *H5* voči náhodným agentom, ako aj voči sebe.



Obr. 5.3 Výsledky pre hru Alquerque proti náhodnému agentovi

Na obrázku 5.3 môžeme vidieť percentuálnu úspešnosť agentov s použitým algoritmu Minimax voči náhodnému agentovi, za použitia oboch heuristík, pričom sa agenti striedali kto začínal hru. Tretí graf na tomto obrázku 5.3 vyjadruje mieru efektivity algoritmu Minimax voči Minimax alfa-beta, kde hodnota pre určenú hĺbku orezania označuje koľko percent uzlov spracoval algoritmus Minimax alfa-beta z celkového počtu spracovaných uzlov algoritmom Minimax. Napríklad pri hĺbke 4 už algoritmus Minimax alfa-beta spracoval len 24 % uzlov, ktoré musel spracovať algoritmus Minimax.

		agent 1 H4				
		1	2	3	4	5
agent 2 H5	1	P	R	R	R	V
	2	P	R	R	R	V
	3	P	R	R	R	V
	4	P	P	P	R	R
	5	P	P	P	R	P

		agent 1 H5				
		1	2	3	4	5
agent 2 H4	1	R	R	V	R	V
	2	R	R	R	R	R
	3	R	R	R	R	R
	4	P	R	R	R	R
	5	P	R	R	P	R

Obr. 5.4 Výsledky pre hru Alquerque pre komperatívnych Minimax agentov

Na obrázku 5.4 môžeme vidieť ešte porovnanie heuristík voči sebe za použitia algoritmu Minimax pri rôznych hĺbkach orezania. Agent nachádzajúci hore začínal danú hru a z jeho pohľadu *V* znamená výhra, *P* prehra a *R* remíza.

ZÁVER

Cieľom našej práce bolo navrhnúť a implementovať robotický herný systém, ktorý by dokázal aplikovať herný proces na ľubovoľnú stolovú hru. Systém má slúžiť ako platforma na následné testovanie funkčnosti implementovaných algoritmov umelej inteligencie, demonštráciu kvality algoritmov pri komparatívnej hre racionálnych agentov voči sebe, ako aj popularizačný prostriedok na možnosť zahrať si človeka v reálnom svete voči agentovi s umelou inteligenciou využívajúceho robotické rameno na vykonávanie akcií.

V prvej fáze sme predložili požiadavky na systém podľa zadaných cieľov práce. V druhej fáze sme na základe týchto požiadaviek predstavili návrh systému a prepojenia jeho komponentov. Snažili sme sa zachovať čo najviac požadovanej funkčnosti a hlavne neobmedzovať požiadavku na možnosť definovania ľubovoľnej stolovej hry. V ďalšej fáze sme postupne implementovali časti funkčnosti do jednotlivých komponentov systému.

Väčšinu hlavných zadaných cieľov sme splnili, podrobnejšie sme ich popísali v kapitole 5. Opakovaním herným procesom sme demonštrovali, že v aktuálnej verzii systému je ľudský agent schopný plnohodnotne odohrať niektorú zo zadefinovaných hier voči agentovi s racionálnym uvažovaním vychádzajúcim z algoritmov umelej inteligencie. Taktiež sme demonštrovali schopnosť simulácie hry dvoch racionálnych agentov voči sebe. Ciele, ktoré sa v súčasnej verzii systému nepodarilo dotiahnuť do kompletnej funkčnosti sú všetky rozpracované a pripravené na doplnenie v nasledujúcej verzii systému.

PRÍLOHY

- Zdrojové kódy programu
- Špecifikácia pre hry Žabky, Connect4 a Alquarque v syntaxe logického programovania

LITERATÚRA

[1] Stuart Rusell, Peter Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition).* Pearson Education, Inc., 2010. ISBN-13: 978-0136042594, ISBN-10: 0136042597

[2] Kvasnicka V. and a spol. *Umelá inteligencia a kognitívna veda III* , FIIT STU v BA, 2010. ISBN 978-80-227-3542-1

[3] Pavol Návrat. a kol. *Umelá inteligencia*. STU, 2007. ISBN: 9788022726290

[4] Duncan Marsh. *Applied Geometry for Computer Graphics and CAD, 2nd edition.* Springer-Verlag London Limited 2005. ISBN 1852338016

[5] Andrej Lúčny. *Architektúra inteligentných programových systémov,*[online] Dostupné na internete 11.8.2014: http://www.microstep-mis.sk/~andy/Kele99.pdf

[6] Ulf Nilsson, Jan Maluszynski. *Logic, Programming and Prolog (2ed).* John Wiley & Sons Ltd, 2000. ISBN-10: 0471959960

[7] Michael Meredith, Steve Maddock. *Real-Time Inverse Kinematics: The Return of the Jacobian* [online] Dostupné na internete 18.2.2015: http://www.dcs.shef.ac.uk/intranet/research/public/resmes/CS0406.pdf

[8] Forward Kinematics: the Denavit-Hartenberg convention. [online] Dostupné na internete 15.1.2015: https://www.cs.duke.edu/brd/Teaching/Bio/asmb/current/Papers/chap3-forward-kinematics.pdf

[9] Scientific Programming, 3D visualisation of robot arm with 3 DOF [online] Dostupné na internete 18.2.2015: http://www.csc.kth.se/~chek/teaching/EL2310/coursework/matlab_project/matlab_project.html

[10] Peter Pukančík. *Riadiaci systém s inverznou kinematikou pre mobilné robotické rameno*. Bakalárska práca, Fakulta matematiky, fyziky a informatiky, UK, 2012.

[11] DLV - User Manual [online] Dostupné na internete 5.11.2014 http://www.dlvsystem.com/html/DLV_User_Manual.html

[12] Minimax Search [online] Dostupné na internete 7.1.2015 http://ai-depot.com/articles/topic-minimax-search/

[13] Checkers, alpha-beta [online] Dostupné na internete 7.1.2015 http://emotion.inrialpes.fr/people/synnaeve/phdthesis/phdthesis.html

[14] OpenCV online documentation. [online] Dostupné na internete 10.9.2014: http://docs.opencv.org/

[15] Camera calibration With OpenCV [online] Dostupné na internete 10.9.2014 http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

[16] Radial Distortion Correction
[online] Dostupné na internete 10.9.2014:
http://www.uni-koeln.de/~al001/radcor_files/hs100.htm

[17] C++ STL Container Efficiency
[online] Dostupné na internete 12.2.2015:
https://www.unetti.com/blog/stl-container-efficiency/

[18] S3 Games project
[online] Dostupné na internete 3.10.2014: https://code.google.com/p/s3games/

[19] SSC-32 Manual Jim Frye, Version: 2.01XE, June 16, 2010.
[online] Dostupné na internete 5.11.2014:
www.lynxmotion.com/images/html/build136.htm