VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

HRANÍ HRY THE DUKE POČÍTAČEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ADRIÁN HORVÁTH**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. FRANTIŠEK ZBOŘIL, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

156884

Ústav:	Ústav inteligentních systémů (UITS)
Student:	**Horváth Adrián**
Program:	Informační technologie
Název:	**Hraní hry The Duke počítačem**
Kategorie:	Umělá inteligence
Akademický rok:	2023/24

Zadání:

1. Nastudujte současné metody pro hraní her dvou hráčů počítačem a seznamte se s pravidly stolní hry 'The Duke'
2. Zvolte vhodné metody pro hraní této hry, jednu z klasických (například Alfa/Beta) a jednu z moderních (například MCTS) metod.
3. Realizujte systémy pro hraní této hry, jeden s klasickými metodami a jeden s modeními.
4. Porovnějte výsledky obou systémů a diskutujte je, včetně nárokům na časové a paměťové zdroje.

Literatura:

1. Norvig, P., Russel, S. :"Artificial Intelligence, A Modern Approach", Prentice Hall, 2020
2. Wang, B.: Monte Carlo Tree Search: An Introduction, Towards Data Science, 2021

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

Vedoucí práce:	**Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu:	Hanáček Petr, doc. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	9.5.2024
Datum schválení:	6.11.2023

Abstrakt

Táto práca sa zaoberá analýzou herných stratégií v hre "The Duke" pomocou algoritmov umeléj inteligencie (AI). Porovnávame tri rôzne prístupy: minimax, alfa-beta orezávanie a Monte Carlo Tree Search (MCTS). Študujeme pravidlá hry, identifikujeme kľúčové faktory ovplyvňujúce stratégiu a vykonávame experimentálne porovnanie výsledkov algoritmov. Zhrňujeme výsledky a diskutujeme o budúcich smeroch výskumu v oblasti herných AI.

Abstract

This thesis deals with the analysis of game strategies in the game "The Duke" using artificial intelligence (AI) algorithms. We compare three different approaches: minimax, alpha-beta pruning and Monte Carlo Tree Search (MCTS). We study the rules of the game, identify key factors affecting strategy, and perform an experimental comparison of the algorithms' results. We summarize the results and discuss future research directions in game AI.

Klíčová slova

The Duke, umelá inteligencia, algoritmy umelej inteligencie, Minimax, Alfa-beta orezávanie, Monte Carlo Tree Search.

Keywords

The Duke, artificial intelligence, artificial intelligence algorithms, Minimax, Alpha-beta pruning, Monte Carlo Tree Search

Citace

HORVÁTH, Adrián. *Hraní hry The Duke počítačem*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. František Zbořil, Ph.D.

Hraní hry The Duke počítačem

Prohlášení

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

.....
Adrián Horváth
6. května 2024

Poděkování

Rád by vyjadril úprimnú vďaku môjmu vedúcemu bakalárskej práce, pánovi doc. Ing. Františkovi Zbořilovi, Ph.D. Veľmi si vážim jeho odborné vedeniea cenné rady počas celého procesu písania práce. Jeho podnetné návrhy prispeli k tomu, aby sa táto práca stala kvalitnou a relevantnou. Som presvedčený, že vďaka jeho pomoci som získal cenné poznatky a zručnosti, ktoré mi budú užitočné v ďalšom štúdiu aj v profesnom živote.

Obsah

1 Úvod **2**

2 Umenlá inteligencia **3**
2.1 Čo je umelá inteligencia 3
2.2 História umelej inteligencie 4
2.3 Umelá inteligencia v hrách 5

3 The Duke **18**
3.1 Pravidlá 18

4 Algoritmy zvolené pre hranie hry *The duke* počítačom **23**
4.1 Minimax 23
4.2 AlphaBeta 26
4.3 Monte Carlo Tree Search 29

5 Implementácia **33**
5.1 Jazyk a nástroje: 33

6 Výsledky a vyhodnotenie **39**
6.1 Porovnanie algoritmov 39

7 Záver **44**

Literatura **45**

A Ikony pohybu figúrok **47**

B Profily pohybu figúrok **48**

C UML Daigram tried **50**

D Obsah priloženého CD **51**

Kapitola 1

Úvod

Svet stolových hier ponúka bohaté pole pre hlboké pochopenie hernej stratégie a jej prepojenia s umelou inteligenciou. Táto práca sa zameriava na formálne skúmanie hernej stratégie v kontexte klasickej stolovej hry "The Duke"[1] s využitím moderných metód AI.

Cieľom tejto práce je preskúmať a porovnať tri rôzne algoritmy k hraniu hry "The Duke"s využitím algoritmov umeléj inteligencie: klasické metódy minimax a alfa-beta pruning a moderný algoritmus Monte Carlo Tree Search (MCTS).

Ciele projektu:

Detailné preskúmanie algoritmov a techník používaných pre hranie hier s AI. Analýza silných a slabých stránok rôznych metód, ako napríklad minimax, alfa-beta pruning, MCTS a neurónové siete. Zhodnotenie vhodnosti rôznych metód pre rôzne typy hier a herné prostredia.

Precízne preštudovanie pravidiel hry "The Duke", vrátane cieľov hry, herných mechanizmov a možných ťahov. Pochopenie komplexnosti hry a identifikácia kľúčových faktorov ovplyvňujúcich stratégiu a víťazstvo. Zváženie rôznych herných stratégií a ich vplyv na priebeh hry.

Na základe analýzy rôznych metód a vlastností hry "The Duke"vybrať vhodné metódy Vysvetliť ich očakávané výhody a nevýhody v kontexte hry "The Duke".

Experimentálne porovnanie výsledkov systémov AI:

Spustiť hru "The Duke"s algoritmami a porovnať ich. Analyzovať výsledky. Diskutovať o silných a slabých stránkach metód v kontexte hry "The Duke". Zhodnotiť vplyv rôznych faktorov na výsledky hry a identifikovať oblasti pre potenciálne vylepšenia.

Zhrnúť výsledky projektu a zdôrazniť kľúčové zistenia týkajúce sa vhodnosti rôznych metód AI pre hranie hry "The Duke". Diskutovať o limitoch a potenciálnych smeroch budúceho výskumu v oblasti hernej AI.

Kapitola 2

Umenlá inteligencia

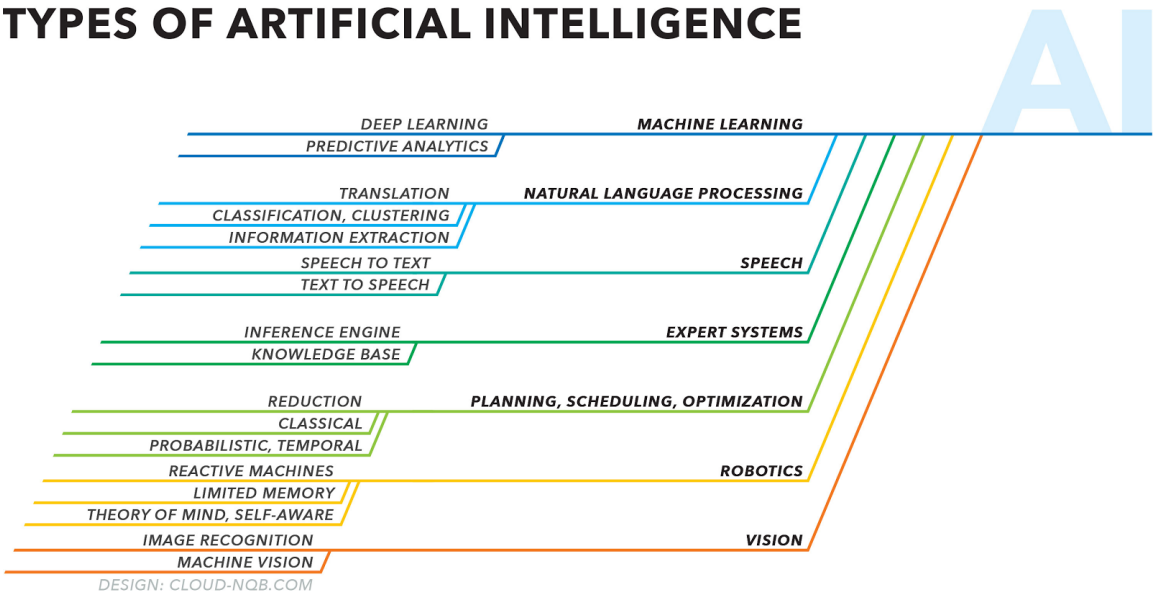
2.1 Čo je umelá inteligencia

Umelá inteligencia (AI) je široký pojem, ktorý sa týka vytvárania inteligentných strojov, schopných vykonávať úlohy, ktoré si zvyčajne vyžadujú ľudskú inteligenciu. Tieto úlohy zahŕňajú:

- **Učenie:** Schopnosť získavať vedomosti a zručnosti z dát alebo skúseností.
- **Riešenie problémov:** Identifikácia a riešenie výziev pomocou analýzy a rozhodovania.
- **Úvaha:** Vytváranie logických záverov na základe dostupných informácií.
- **Adaptabilita:** Úprava správania na základe meniacich sa okolností.

Pomenovanie umelá inteligencia je veľmi obsiahle a preto sa na ňu rôznia aj mnohé definície. Laik by ju mohol opísať ako inteligenciu demonštrovanú strojmi, alebo vykonávanie bežnej ľudskej činnosti bez akéhokoľvek ľudského zásahu. Pravda je taká, že pojem umelá inteligencia má uplatnenie v mnohých sférach:

- **Zdravotníctvo:** Analýza lekárskych snímok, diagnostika chorôb, vývoj liekov.
- **Financie:** Schvaľovanie pôžičiek, detekcia podvodov, personalizované finančné poradenstvo.
- **Doprava:** Autonómne vozidlá, optimalizácia trás, riadenie premávky.
- **Zábava:** Personalizované odporúčania filmov a hudby, vývoj realistických herných protivníkov.
- **Výskum:** Analýza obrovských dátových súborov, hľadanie vzorov a trendov, vývoj nových teórií.
- **Rozpoznávanie a kategorizácia:** Rozpoznávanie a kategorizácia bežných predmetov, farieb a textu.
- **Počítačové hry:** Zohrávanie role klasického oponenta v počítačových hrách.
- **Vizuálne úlohy:** Vykresľovanie grafických častíc okolia a podobných vizuálnych úkonov programov.



Obrázek 2.1: Clenenie umelej inteligencie [7]

Presná definícia AI je predmetom diskusií. Existujú rôzne definície AI, ktoré sa líšia v dvoch hlavných rozmeroch:

- Procesy myslenia/uvažovania alebo správania
- Podla uspešnosti napodobniť ludský výkon alebo napodobniť ideálnu predstavu inteligencie "*racionalita*"

Systém je racionálny, ak robí "správnu vec", s ohľadom na jeho vedomosti.[18]

Týmto spôsobom rozdelenia vzniká model štyroch rôznych spôsobov, ako definovať umelú inteligenciu.[18]

- Prvý spôsob sa zameriava na kognitívne schopnosti AI systémov.
- Druhý spôsob sa zameriava na správanie AI systémov.
- Tretí spôsob sa zameriava na schopnosť AI systémov robiť racionálne rozhodnutia.
- Štvrtý spôsob sa zameriava na schopnosť AI systémov vykonávať úlohy racionálne.

2.2 História umelej inteligencie

Snaha o vytvorenie inteligentných strojov fascinovala ľudstvo celé stáročia.

Semená umelej inteligencie možno nájsť už u starogréckych filozofov, ako bol Aristoteles, ktorý uvažoval o povahe inteligencie a myšlienkových procesoch. Už v antickom Grécku sa objavovali predstavy o mysliacich strojoch. Napríklad grécky boh Hefaistos bol zobrazovaný ako tvorca mechanických sluhov a bronzového muža Talosa.[15]

V 15. a 16. storočí sa začali vytvárať mechanické imitácie zvierat. Napríklad hodinári v tejto dobe dokázali vytvoriť mechanické vtáky, ktoré mohli lietať, alebo mechanické ryby, ktoré mohli plávať.

Tento trend pokračoval aj v 17. a 18. storočí s vynálezom automatov, strojov napodobňujúcich ľudské činnosti. Tieto vynálezy vzbudili zvedavosť a podnietili úvahy o možnostiach replikácie ľudských schopností a inteligencie pomocou mechaniky.

Jedným z významných príkladov z tohto obdobia je kalkulačka Blaise Pascala, ktorú vytvoril medzi rokmi 1642 a 1645. Táto mechanická kalkulačka dokázala vykonávať jednoduché matematické operácie a predstavovala významný krok vpred v oblasti výpočtovej techniky.[17]

V prvej polovici 20. storočia sa objavili prví autonómni roboti. Napríklad v roku 1948 Grey Walter zostrojil roboty Elsie a Elmer, ktoré boli schopné samostatne sa pohybovať a reagovať na svetlo.[16]

V roku 1945 Alan Turing použil šach ako príklad toho, čo by počítač zvládol bez ťažkostí. O štyri roky neskôr naprogramoval prvý šachový program.

V roku 1950 Alan Turing zverejnil svoju prelomovú prácu "Computing Machinery and Intelligence"(Počítacie stroje a inteligencia), v ktorej predstavil Turingov test, meradlo na posúdenie schopnosti stroja prejavovať inteligentné správanie na úrovni človeka.[22]

V roku 1956 John McCarthy prvýkrát použil pojem "Umelá inteligencia". V tom istom roku sa uskutočnil letný výskumný projekt na Dartmouthskej univerzite. Táto epochálna udalosť, ktorú zorganizovali John McCarthy, Marvin Minsky, Nathaniel Rochester a Claude Shannon, sa považuje za oficiálny zrod výskumu umelej inteligencie.[19]

V roku 1957 sa objavil známy General Problem Solver. Tento program bol schopný vyriešiť rôzne logické problémy.[18]

V roku 1963 Thomas Evans dokázal, že počítač je schopný vyriešiť úlohy z bežných IQ testov.

Prvotné nadšenie pre výskum umelej inteligencie viedlo k ambicióznym projektom, no pokrok brzdili obmedzenia výpočtového výkonu a teoretického porozumenia. Do 70. rokov 20. storočia nedostatok významných prelomov spojený so škrtami v rozpočtoch viedol k obdobiu stagnácie známemu ako "zima umelej inteligencie".

AI prešla v priebehu desaťročí dramatickým vývojom, prekonala rôzne etapy a dosiahla pozoruhodný pokrok.

V 90. rokoch došlo k výraznému nárastu záujmu o AI a v tejto oblasti sa objavil rad nových algoritmov. Tieto algoritmy umožnili počítačom vykonávať úlohy, ktoré boli predtým považované za nemožné, čím sa posunuli hranice toho, čo je možné dosiahnuť pomocou AI.

V 21. storočí sa AI stala témou s rastúcou popularitou a praktické využitie AI systémov sa rozšírilo do rôznych oblastí. Dnes sa AI systémy používajú v rôznych odvetviach, ako napríklad: medicína, doprava, výroba a logistika.

2.3 Umelá inteligencia v hrách

AI sa stáva čoraz dôležitejšou súčasťou videohier a prináša do nich nové možnosti a výzvy. Tento text ponúka prehľad o tom, ako AI ovplyvňuje herný dizajn a zážitok.

Implementácia AI vo videohrách

AI sa vo videohrách používa rôznymi spôsobmi, od jednoduchého rozhodovania nehrateľných postáv (NPC) až po komplexné simulácie celých ekosystémov v hernom svete. Medzi najčastejšie implementácie AI patria:

- **Nepriatelia:** AI riadení nepriatelia sa dokážu prispôsobovať hráčovým stratégiám, čím vytvárajú dynamickejší a náročnejší herný zážitok.

- **NPC:** AI umožňuje NPC prejavovať realistickejšie správanie, s jedinečnými osobnosťami a reakciami na hráčove akcie.
- **Procedurálne generovanie obsahu:** AI sa používa na generovanie jedinečných herných prostredí, úloh a výziev, čím sa zaručuje, že každé hranie bude iné.
- **Herní režiséri:** AI režiséri hry dokážu dynamicky upravovať obtiažnosť, spawnovať nepriateľov a spúšťať udalosti na základe hráčovho výkonu, čím zaručujú pútavý a prispôsobený herný zážitok.

Výhody AI vo videohrách

AI prináša do videohier viacero benefitov, vrátane:

- **Realistickejší a pútavejší herný svet:** AI umožňuje vytvárať uveriteľné a dynamické herné prostredia s realisticky sa správajúcimi NPC.
- **Náročnejší a dynamickejší herný zážitok:** AI riadení nepriatelia a dynamické herné svety nútia hráčov neustále sa prispôsobovať a vymýšľať nové stratégie.
- **Prispôsobená obtiažnosť:** AI režiséri hry dokážu upravovať obtiažnosť tak, aby bola zábavná pre hráčov všetkých úrovní skúseností.
- **Širšia dostupnosť:** AI hry robia prístupnejšími pre príležitostných hráčov, ktorí nemusia mať čas alebo zručnosti na zvládanie náročných hier.

Nevýhody AI vo videohrách

Napriek svojim benefitom má AI aj niektoré potenciálne nevýhody:

- **Obmedzená "inteligencia":** Hoci AI dokáže napodobňovať inteligentné správanie, stále jej chýba skutočná kreativita a rozhodovanie podobné človeku. To môže niekedy viesť k nelogickým alebo nemiestnym akciám.
- **Opakované reakcie:** Ak sa AI neimplementovala opatrne, môže sa stať predvídateľnou. Nepriatelia, ktorí opakovane naletia na rovnakú pascu, alebo NPC s obmedzenými možnosťami konverzácie, znižujú zábavnosť.

Budúcnosť AI vo videohrách

Budúcnosť AI vo videohrách sľubuje vzrušujúce možnosti. Algoritmy AI sa neustále vyvíjajú a stávajú sa sofistikovanejšími, čo umožňuje tvorcom hier vytvárať ešte prepracovanejšie a pútavejšie herné zážitky. V budúcnosti môžeme očakávať:

- **Prepracovanejšiu AI:** AI bude schopná robiť komplexnejšie a realistickejšie rozhodnutia, čím sa herné svety stanú ešte dynamickejšími a pútavejšími.
- **AI ako rozprávač:** AI by sa mohla používať na tvorbu dynamických herných príbehov, ktoré sa prispôsobujú hráčovým rozhodnutiam.
- **Hry šité na mieru:** AI by sa mohla používať na prispôsobovanie herných zážitkov individuálnym preferenciám a schopnostiam hráčov.

Prvky riadené AI môžu hry zatraktívniť pre opakované hranie tým, že ponúkajú nepredvídateľné scenáre a vznikajúcu hrateľnosť. Predstavte si, ako sa trasy hliadkovania nepriateľov menia na základe predchádzajúcich stretnutí s hráčom, čím sa pri každom hraní vytvára nový zážitok.

Mnoho populárnych hier naprieč žánrami efektívne využíva AI:

- **Strategické hry:** V hrách ako StarCraft II AI ovláda nepriateľské frakcie, prispôsobuje stratégie a buduje obranu na základe hráčovej taktiky.
- **Open-world hry:** Hry ako Red Dead Redemption 2 využívajú AI pre správanie voľne žijúcich živočíchov, čím vytvárajú dynamický a uveriteľný ekosystém v hernom svete.
- **Športové hry:** AI poháňa tímy súpera v športových hrách, vďaka čomu reagujú a prispôsobujú sa hráčovej stratégii, čím simulujú súťaž v reálnom svete.
- **Pretekárske hry:** AI ovláda ostatných pretekárov v pretekárskych hrách, pričom náročnosť ovplyvňuje ich agresivitu a pretekárske stopy.
- **Dobrodružné hry:** AI môže riadiť správanie spoločníkov v adventúrach, ktorí ponúkajú taktickú pomoc alebo vtipné dialógy, v závislosti od dizajnu hry.

Vývoj AI využíva rôzne nástroje a techniky určujúce, ako sa NPC správajú v hernom svete, ovplyvňuje ich činy, reakcie a celkový proces rozhodovania.

Minimax algoritmus

Predstavuje klasický vyhľadávací algoritmus používaný v hrách na určenie najlepšieho ťahu pre AI. Funguje na princípe vyhodnocovania všetkých možných budúcich stavov hry a ich výsledkov. Cieľom AI je maximalizovať svoj vlastný zisk a minimalizovať zisk súpera.[18]

Podrobnejší popis tohot algoritmu je v sekcii 4.1

Alpha-Beta Pruning

Alpha-Beta Pruning predstavuje optimalizačnú techniku používanú v spojení s Minimaxovým algoritmom. Umožňuje dramaticky znížiť počet stavov, ktoré musí Minimax prehľadať, čím výrazne zlepšuje jeho efektivitu. Vďaka Alpha-Beta Pruning sa tak Minimax stáva použiteľným aj v hrách hraných v reálnom čase. [18]

Podrobnejší popis tohot algoritmu je v sekcii 4.2

Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) predstavuje pokročilý vyhľadávací algoritmus, ktorý sa používa v hrách s neúplnými informáciami (napríklad hmla vo vojnových stratégiách). Na rozdiel od tradičných vyhľadávacích algoritmov, ktoré sa zameriavajú na prehľadávanie celého stromu hry, MCTS funguje na princípe **rovnováhy medzi exploráciou a exploitáciou**.

Explorácia: Spočíva v skúšaní nových, neprebádaných herných možností. Cieľom je objaviť nové stratégie a potenciálne prekvapivé ťahy, ktoré by AI inak nezvážila.

Exploitácia: Znamená využívanie už overených a úspešných herných postupov. AI sa zameriava na ťahy, ktoré sa v predchádzajúcich simuláciách ukázali ako výhodné a s vysokou pravdepodobnosťou vedú k víťazstvu.

Podrobnejší popis tohot algoritmu je v sekcii 4.3

Heuristika

Heuristiky predstavujú metódy alebo funkcie pre informované vyhľadávanie, ktoré pomáhajú AI zamerať sa na sľubné ťahy bez toho, aby musela preskúmať úplne všetky možné možnosti. Heuristické funkcie fungujú ako akési skratky, ktoré vedú AI k dobrým rozhodnutiam v rozumnej časovej lehote. Vďaka heuristike dokáže AI hrať efektívne a plynule v reálnom čase, čím sa herný zážitok stáva dynamickým a pútavým

Príklady Heuristických Funkcií:

- Hodnotenie pozície figúr: V šachových alebo dámových hrách môže heuristika zohľadňovať pozíciu jednotlivých figúr na hracej ploche. Čím lepšia pozícia figúry, tým vyššie skóre pre daný stav hry.
- Materiálna prevaha: V hrách, kde sa berie do úvahy hodnota jednotlivých figúr (napríklad šach), môže byť heuristická funkcia navrhnutá tak, aby uprednostňovala stavy, v ktorých má AI materiálnu prevahu (viac figúr alebo figúr s vyššou hodnotou).
- Kontrola kľúčových oblastí: V niektorých stratégiách môže byť rozhodujúca kontrola určitých oblastí na hernom pláne. Heuristická funkcia môže priradiť vyššie skóre stavom, v ktorých má AI tieto oblasti pod kontrolou.

Výhody:

- Zvýšenie efektivity vyhľadávania: Heuristika výrazne zlepšuje efektivitu hľadania najlepšieho ťahu. AI nemusí skúšať všetky možné kombinácie a dokáže robiť dobré rozhodnutia v reálnom čase. To je obzvlášť dôležité pre komplexné hry s veľkým počtom možných ťahov a herných stavov.
- Umožnenie rozhodovania v reálnom čase: Vďaka heuristike dokáže AI reagovať na jednotlivé ťahy hráča plynule a bez dlhého rozmýšľania. To je kľúčové pre udržanie dynamického a pútavého herného zážitku, najmä v rýchlych strategických hrách.
- Zníženie výpočtovej náročnosti: Heuristika znižuje výpočtovú náročnosť algoritmov AI, čo umožňuje implementáciu komplexných herných mechanizmov a realistickejšieho správania AI aj na zariadeniach s obmedzeným výkonom.

Nevýhody:

- Závislosť na kvalite heuristickej funkcie: Účinnosť heuristiky závisí od toho, ako dobre je navrhnutá heuristická funkcia. Ak funkcia nezohľadňuje všetky dôležité faktory alebo obsahuje chyby, môže viesť k suboptimálnemu hraniu AI (t.j. AI nebude robiť tie najlepšie možné ťahy). To môže mať negatívny vplyv na hernú rovnováhu a zážitok hráča.

- Riziko suboptimálneho hrania: V niektorých prípadoch môže heuristika viesť k tomu, že AI prehliadne prekvapivý, ale výhodný ťah, ktorý by mohol hru zvrátiť v jej prospech. To je obzvlášť problematické v komplexných hrách s nepredvídateľnými hernými scenármi.

Heuristika predstavuje cenný nástroj pre implementáciu AI v hrách, ktorý umožňuje dosiahnuť efektívne a plynulé hranie v reálnom čase. Vďaka heuristike dokáže AI robiť inteligentné rozhodnutia a reagovať na herné situácie v súlade so stratégiou. Kľúčom k úspešnému použitiu heuristiky je však starostlivé navrhnutie heuristickej funkcie, ktorá zohľadňuje všetky dôležité faktory a minimalizuje riziko suboptimálneho hrania.

Heuristika v kombinácii s inými algoritmami AI, ako napríklad minimax alebo Monte Carlo Tree Search, umožňuje dosiahnuť komplexné herné mechanizmy a realistické správanie AI, čím sa herný zážitok stáva pútavým a zábavným.

Behavior Trees (Stromy správania)

Stromy správania (Behavior Trees) predstavujú pokročilý prístup k implementácii AI v hrách. Ponúkajú hierarchickú štruktúru, ktorá umožňuje definovať komplexné správanie NPC s väčšou modularitou a prehľadnosťou. Na základe vopred definovaných podmienok AI sleduje a prechádza vetvami stromu, čím generuje realistické a dynamické reakcie v hernom prostredí.

Základné aspekty stromov správania:

- **Uzly:** Strom pozostáva z uzlov, ktoré predstavujú úlohy, podmienky alebo výsledky.
- **Podmienené vetvenia:** Na základe vopred definovaných podmienok (napr. je hráč nablízku?) si AI vyberie príslušnú vetvu na vykonanie (napr. zaútočiť na hráča alebo utiecť).
- **Flexibilita:** Stromy správania umožňujú komplexnejšie rozhodovanie v porovnaní s FSM (Finite State Machines). Pre bohatšie správanie sa dajú kombinovať s inými technikami, ako je napríklad teória užitočnosti.[13]

Základy stromov správania: Stromy správania pozostávajú z uzlov, ktoré sa vzájomne spájajú a definujú logiku správania AI. Tieto uzly sa delia na tri typy:

- **Akcie:** Konkrétne úkony, ktoré AI vykoná (napr. strieľať, pohybovať sa, liečiť sa).
- **Podmienky:** Pravidlá, ktoré určujú, kedy sa akcia spustí (napr. ak je zdravie nízke, liečiť sa).
- **Podstromy:** Menšie stromy správania, ktoré rozkladajú komplexné úlohy na menšie, zvládnuteľné kroky.

Výhody stromov správania:

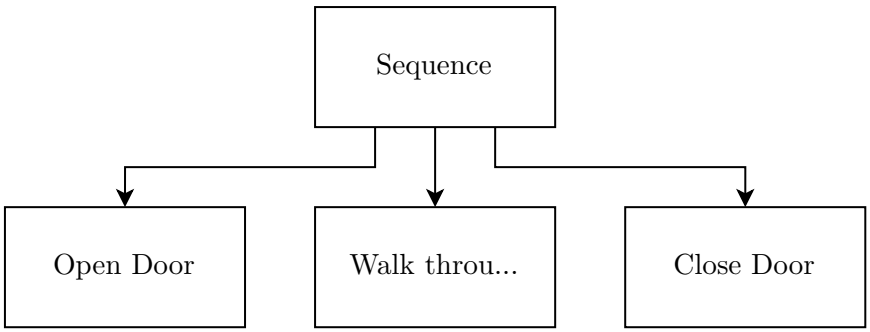
- Modularita a zrozumiteľnosť: Vďaka rozdeleniu na menšie uzly sú stromy správania ľahko pochopiteľné a prehľadné. Dizajnéri tak dokážu budovať zložité správanie AI s väčšou flexibilitou a udržiavateľnosťou.
- Zložité, ale čitateľné správanie: Hierarchická štruktúra umožňuje vytvárať sofistikované rozhodovacie procesy pre AI. Stromy správania dokážu zvládnuť aj náročné herné scenáre vďaka postupnej dekompozícii úloh na menšie časti.

- Dynamické a realistické reakcie: AI riadená stromami správania sa dokáže adaptovať na rôzne herné situácie a generovať realistické reakcie v reálnom čase.

Nevýhody stromov správania:

- Náročnosť ladenia: Zložité stromy správania s mnohými uzlami môžu byť pri ladení chybné a časovo náročné. Proces identifikácie a opravy problémov v zložitých stromoch si môže vyžadovať značné úsilie.
- Návrhárske úsilie: Navrhnutie efektívnych stromov správania si vyžaduje skúsenosti a plánovanie. Vývojári musia starostlivo zvážiť, aké uzly použiť a ako ich navzájom prepojiť, aby dosiahli požadované správanie AI.
- Potenciálna rigidita: V porovnaní s inými prístupmi k AI, ako napríklad neurónové siete, stromy správania obmedzujú flexibilitu a adaptáciu na nepredvídané situácie.

Stromy správania predstavujú cenný nástroj pre implementáciu komplexného a realistického správania NPC v hrách. Ich modularita, zrozumiteľnosť a schopnosť zvládať zložité rozhodovacie procesy z nich robia populárnu voľbu pre vývojárov. Je však dôležité zvážiť aj náročnosť ladenia a potrebu premysleného návrhu pri implementácii stromov správania v herných projektoch.



Obrázek 2.2: Ukážka jednoduchého stromu správania

Fuzzy logika

Fuzzy logika predstavuje odlišný prístup k implementácii AI v hrách. Na rozdiel od doteraz spomínaných metód, ktoré sa spoliehajú na presné pravidlá a stavy, fuzzy logika pracuje s nepresnými informáciami a stupňami pravdivosti. To umožňuje AI robiť rozhodnutia na základe pravdepodobností a odhadov, čím sa približuje k ľudskému spôsobu uvažovania v neistých situáciách[12]

Výhody:

- Nuancovanejšie a realistickejšie správanie: Fuzzy logika umožňuje AI zohľadňovať rôzne stupne pravdivosti a nepresnosti informácií. Namiesto binárneho ”áno” alebo ”nie” dokáže AI vyjadriť aj ”do istej miery”, ”pravdepodobne” alebo ”s vysokou pravdepodobnosťou”. To vedie k realistickejšiemu a menej strojenému správaniu AI v porovnaní s metódami založenými na pevných pravidlách. AI tak dokáže reagovať na rôzne herné situácie s väčšou flexibilitou a adaptabilitou.

- Lepšie zvládnutie neistoty: V hernom prostredí často existuje určitá miera neistoty (napríklad neúplné informácie o pozícii nepriateľa, nepredvídateľné správanie hráčov). Fuzzy logika umožňuje AI lepšie zvládať tieto neisté situácie a robiť informované rozhodnutia aj s neúplnými dátami. AI tak dokáže analyzovať nejasné informácie a zhodnotiť rôzne možnosti, čím zvyšuje šancu na úspech v dynamických herných prostrediach.

Nevýhody:

- Náročnosť návrhu a ladenia: Fuzzy logika si vyžaduje starostlivé navrhnutie a ladenie pravidiel, ktoré definujú stupne pravdivosti a váhy rôznych faktorov. Tento proces môže byť pre vývojárov náročnejší ako pri iných metódach, pretože si vyžaduje hlbšie pochopenie princípov fuzzy logiky a herného prostredia. Nesprávne navrhnuté pravidlá by mohli viesť k nepredvídateľnému alebo neefektívnemu správaniu AI.
- Výpočtová náročnosť: Implementácia fuzzy logiky pre zložité scenáre v hrách môže byť výpočtovo náročná. To môže predstavovať problém pre hry s obmedzenými hardvérovými zdrojmi, najmä pre mobilné platformy alebo hry s vysokou grafickou náročnosťou. V takýchto prípadoch je nutné zvážiť optimalizáciu fuzzy logiky a jej efektívne integrovanie do herného enginu.

Fuzzy logika ponúka alternatívny prístup k implementácii AI v hrách, ktorý umožňuje realistickejšie a flexibilnejšie správanie v neistých herných situáciách. Napriek náročnosti na dizajn a ladenie môže fuzzy logika obohatiť herný zážitok a priniesť do AI komplexnosť a nuansovanosť, ktorá sa priblíži ľudskému uvažovaniu. Vhodným využitím fuzzy logiky a optimalizáciou jej implementácie sa dá prekonať aj výpočtová náročnosť a implementovať ju aj do hier s obmedzenými hardvérovými zdrojmi.

Konečné stavové automaty (FSM)

Konečné stavové automaty (FSM) predstavujú ďalší prístup k implementácii AI v hrách. V tomto modeli sa správanie AI riadi sériou stavov, pričom prechod medzi jednotlivými stavmi je spúšťaný konkrétnou udalosťou.

Ako fungujú FSM:

- **Stavy:** Každý stav určuje, ako sa NPC v danej situácii správa. Napríklad v stave "nečinný", by NPC mohol stáť na mieste a rozhliadať sa, zatiaľ čo v stave "útok", by mohol strieľať na hráča.
- **Prechody:** Prechody spájajú jednotlivé stavy a definujú, kedy a ako sa AI presunie z jedného stavu do druhého. Udalosti, ako napríklad spozorovanie hráča alebo utrpenie poškodenia, slúžia ako spúšťače prechodov. V závislosti od spúšťacej udalosti sa AI presunie do relevantného stavu.[11]

Výhody:

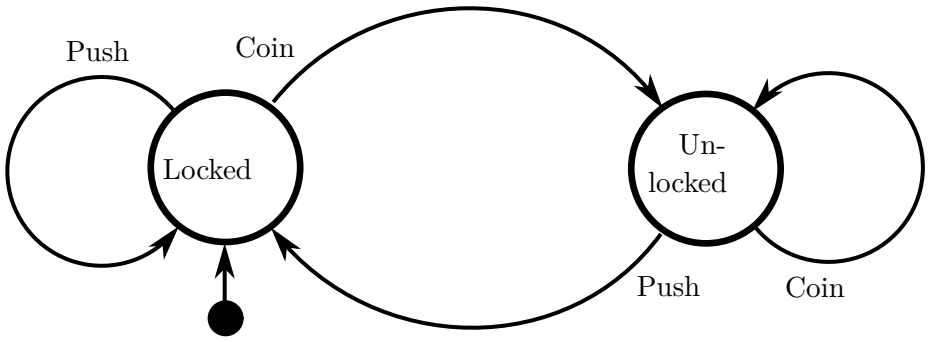
- Zložitejšie rozhodovanie: V porovnaní s jednoduchými pravidlami umožňujú FSM komplexnejšie rozhodovacie procesy. AI môže reagovať na rôzne situácie v závislosti od aktuálneho stavu, v ktorom sa nachádza. To umožňuje realistickejšie a menej predvídateľné správanie.

- Akcie závislé od stavu: Rôzne stavy môžu viesť k rôznym súborom akcií, ktoré môže AI vykonať. To umožňuje AI reagovať na situáciu vhodnejším a kontextovo relevantnejším spôsobom.
- Jednoduchosť a čitateľnosť: FSM ponúkajú jasný a dobre pochopiteľný spôsob modelovania jednoduchého správania NPC. V porovnaní s komplexnejšími technikami AI, ako sú stromy správania, sú FSM vizuálne intuitívne a relatívne ľahko implementovateľné.

Nevýhody:

- Komplexnosť pri viacerých stavoch: S rastúcim počtom stavov a prechodov medzi nimi môže správanie FSM narastať do značnej komplexnosti. To môže byť náročné na dizajn, implementáciu a ladenie, najmä pri rozsiahlych herných svetoch a rôznych herných situáciách.
- Problém škálovateľnosti: Pri hrách s veľkým počtom herných stavov a akcií môže byť obtiažne spravovať a udržiavať enormné množstvo stavov a prechodov v rámci FSM. To môže obmedzovať využitie FSM v komplexnejších hrách s bohatým herným svetom a širokou škálou akcií NPC.

FSM predstavujú užitočný nástroj pre implementáciu základného až stredne komplexného správania NPC v hrách. Ich jednoduchosť, čitateľnosť a flexibilita z nich robia vhodnú voľbu pre hry s menším počtom herných stavov a akcií. Napriek tomu je dôležité zvážiť limity FSM, pokiaľ ide o škálovateľnosť a komplexnosť, pri ich použití v rozsiahlejších herných projektoch.



Obrázek 2.3: Ukážka jednoduchého stavového automatu [9]

Pravidlami riadené systémy

Pravidlami riadené systémy (Rule-Based Systems) predstavujú tradičný prístup k implementácii AI v hrách. V tomto modeli je správanie AI definované súborom predprogramovaných pravidiel a rozhodovacích stromov. Tieto pravidlá určujú, ako sa AI bude správať v rôznych herných situáciách, napríklad ako reagovať na nepriateľov, hľadať cestu alebo plniť úlohy.[8]

Výhody:

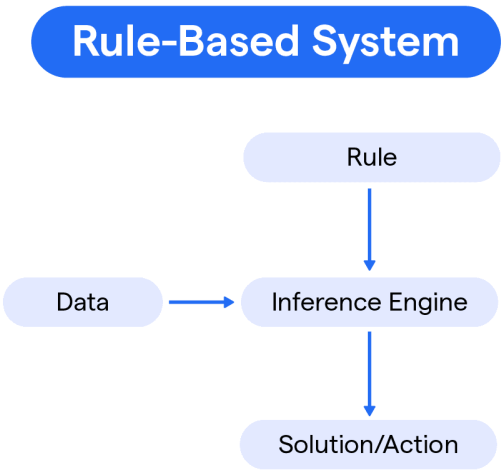
- Jednoduchá implementácia: Pravidlami riadené systémy sa pomerne ľahko vytvárajú a implementujú do herných projektov. Vývojári nepotrebujú hlboké znalosti komplexných algoritmov AI, čím sa znižuje čas a náročnosť vývoja.

- Predvídateľné správanie: Vďaka pevnému súboru pravidiel je správanie AI pre vývojárov aj hráčov ľahko pochopiteľné a predvídateľné. To uľahčuje testovanie a ladenie AI, ako aj pochopenie jej reakcií v rôznych herných situáciách.
- Účinnosť pre jednoduchšie hry: Pre nenáročné hry s menším počtom herných stavov a akcií, ako napríklad puzzle hry alebo plošinovky, môžu byť pravidlami riadené systémy plne postačujúce a efektívne. V takýchto hrách nemusia komplexnejšie techniky AI priniesť značné benefity.

Nevýhody:

- Opakovateľnosť a predvídateľnosť: AI s obmedzenou sadou pravidiel môže v hrách pôsobiť stereotypne a jej správanie sa pre hráčov stane ľahko predvídateľné. To môže znížiť zábavnosť hry, pretože hráči si rýchlo osvoja vzorce správania AI a dokážu ju ľahko prekonať.
- Obmedzená adaptabilita: Pevný súbor pravidiel sťažuje AI reagovať na neočakávané situácie alebo sa prispôsobovať rôznym štýlom hrania hráčov. To môže viesť k tomu, že sa AI stane ľahko prekonateľnou pre skúsených hráčov alebo frustrujúcou pre menej skúsených hráčov, ktorí nevedia predvídať jej reakcie.

Pravidlami riadené systémy predstavujú jednoduchý a efektívny prístup k implementácii AI v hrách s menším počtom herných stavov a akcií. Ich jednoduchosť a predvídateľnosť správania uľahčujú vývoj a testovanie AI. Na druhej strane, obmedzená adaptabilita a stereotypné správanie AI môže znížiť zábavnosť hry pre skúsených hráčov. V komplexnejších hrách s bohatým herným svetom a širokou škálou akcií sa preto odporúča zvážiť použitie pokročilejších techník AI, ako sú stromy správania alebo neurónové siete.



Obrázek 2.4: Rule-Based System [8]

Algoritmy Hľadania Cesty

Algoritmy hľadania cesty sú kľúčovou súčasťou AI v hrách, ktorá umožňuje NPC efektívne sa pohybovať po hernom svete. Tieto algoritmy analyzujú herné prostredie, berúc do úvahy

prekážky, terén a iné faktory, aby určili najefektívnejšiu trasu medzi dvoma bodmi. Vďaka efektívnemu hľadaniu cesty NPC pôsobia prirodzenejšie a realistickejšie, čím sa herný zážitok stáva pútavým a zábavným.

Kľúčové pojmy v hľadaní cesty:

- Vyhľadávanie A:* Patrí medzi najpopulárnejšie algoritmy hľadania cesty, ktoré vyvažujú efektivitu a optimalitu. Dokáže rýchlo nájsť dobrú cestu medzi dvoma bodmi, aj keď nie vždy tú absolútne najkratšiu. Algoritmus A* funguje na princípe priraďovania skóre jednotlivým uzlom v grafe, pričom skóre zohľadňuje vzdialenosť k cieľu a ťažkosti prechodu cez daný úsek. Vďaka tomuto skórovaniu algoritmus uprednostňuje sľubnejšie cesty a vyhýba sa neefektívnym vetveniam.[20]

- Navigačné Meshe: Pre zefektívnenie hľadania cesty sa v herných prostrediach často používajú navigačné meshe. Ide o zjednodušené zobrazenia herného sveta, ktoré rozdeľujú prostredie na menšie polygóny a koridory. Algoritmy hľadania cesty potom pracujú s touto zjednodušenou reprezentáciou, čím sa znižuje výpočtová náročnosť a zrýchľuje sa vyhľadávanie trás.

- Dijkstrav algoritmus: je nástrojom na hľadanie cesty v hrách, ktorý efektívne vypočíta najkratšiu trasu pre postavy alebo objekty. Funguje dobre v statických prostrediach, ako sú prednastavené levely hry, a zaručuje nájdenie najpriamejšej cesty z bodu A do bodu B pre hráča alebo NPC.[14]

Výhody efektívneho hľadania cesty:

- Prirodzenejšie správanie NPC: Vďaka efektívnemu hľadaniu cesty sa NPC dokážu pohybovať plynule a realisticky, čím pôsobia prirodzenejšie a menej strojovo.

- Zvýšená herná plynulosť: NPC, ktoré sa dokážu rýchlo a efektívne pohybovať po hernom svete, prispievajú k plynulej hrateľnosti a minimalizujú frustraciu hráča.

- Zvýšená dynamika hry: Algoritmy hľadania cesty umožňujú implementovať komplexné herné mechanizmy, ako napríklad prenasledovanie hráča, evakuáciu NPC v prípade ohrozenia alebo dynamické scenáre s pohybujúcimi sa prekážkami.

Algoritmy hľadania cesty predstavujú dôležitú súčasť AI v hrách, ktorá umožňuje NPC efektívne sa pohybovať po hernom svete a dodáva hre prirodzenosť a dynamiku. Vďaka neustálemu vývoju a optimalizácii algoritmov hľadania cesty sa môžeme tešiť na stále realistickejšie a pútavejšie herné zážitky v budúcnosti.

Rozhodovanie

Rozhodovanie predstavuje kľúčový aspekt AI v hrách, ktorý umožňuje NPC robiť inteligentné voľby a reagovať na herné situácie realistickým spôsobom. Dva pozoruhodné prístupy k rozhodovaniu v AI sú:

1. Teória Užitočnosti:

Táto teória vníma rozhodovanie NPC ako výber akcie s najvyššou "užitočnosťou". Užitočnosť akcie sa kvantifikuje číselnou hodnotou, ktorá odráža jej želaný prínos pre NPC.[3] Pri výbere akcie NPC zohľadňuje rôzne faktory, ako napríklad:

- Sebazachovanie: NPC sa snaží prežiť a vyhýba sa situáciám, ktoré by ho mohli ohroziť.

- Plnenie cieľov: NPC má rôzne ciele, ako napríklad nájsť predmet, dostať sa na určité miesto alebo poraziť nepriateľa. Teória užitočnosti umožňuje NPC uprednostňovať akcie, ktoré ho priblížia k jeho cieľom.
- Dodržiavanie herných pravidiel: NPC sa riadi pravidlami hry a vyhýba sa neférovému alebo nelegálnemu správaniu.

Výhody Teórie Užitočnosti:

- Flexibilita: Umožňuje NPC robiť rôzne rozhodnutia v závislosti od kontextu hernej situácie.
- Prispôsobivosť: NPC sa dokáže adaptovať na meniace sa herné prostredie a zvoliť si najvhodnejšiu akciu v danej situácii.
- Realizmus: NPC sa správa prirodzenejšie a realistickejšie, keďže jeho konanie je riadené komplexným hodnotením rôznych faktorov.

Nevýhody Teórie Užitočnosti:

- Náročnosť dizajnu: Definovanie a kvantifikácia užitočnosti rôznych akcií môže byť pre dizajnérov hier náročné.
- Výpočtová náročnosť: V komplexných herných prostrediach s veľkým počtom akcií a faktorov môže byť výpočet užitočnosti pre každú akciu náročný na výpočtový výkon.

2. Plánovanie Akcií Zameraných na Cieľ (GOAP): Táto technika sa zameriava na plánovanie sled akcií, ktoré sú potrebné na dosiahnutie konkrétneho cieľa. NPC analyzuje dostupné akcie, ich predpoklady (podmienky, ktoré musia byť splnené, aby bola akcia možná) a dôsledky (výsledok akcie). Na základe tejto analýzy NPC zostavuje plán akcií, ktorý ho dovedie k jeho cieľu.[2]

Výhody Plánovania Akcií Zameraných na Cieľ:

- Štruktúrovaný prístup: Poskytuje jasnú štruktúru pre rozhodovanie NPC a umožňuje mu systematicky uvažovať o krokoch potrebných na dosiahnutie cieľa.
- Prehľadnosť: Dizajnéri hier ľahšie pochopia a ovplyvnia proces rozhodovania NPC, keďže je založený na explicitných plánoch akcií.
- Možnosť optimalizácie: Plány akcií sa dajú optimalizovať pre dosiahnutie maximálnej efektivity a minimalizácie zbytočných krokov.

Nevýhody Plánovania Akcií Zameraných na Cieľ:

- Náročnosť na implementáciu: Implementácia GOAP v hernom engine môže byť náročná a vyžaduje si hlboké znalosti hernej mechaniky a procesov rozhodovania.
- Rigidita: Plány akcií nemusia vždy reagovať flexibilne na nepredvídané herné situácie, čo môže viesť k neefektívnemu alebo nerealistickému správaniu NPC.
- Výpočtová náročnosť: V komplexných herných prostrediach s mnohými cieľmi a akciami môže byť plánovanie GOAP náročné na výpočtový výkon.

Teória Užitočnosti a Plánovanie Akcií Zameraných na Cieľ predstavujú dva odlišné, ale efektívne prístupy k rozhodovaniu NPC v hrách. Výber vhodného prístupu závisí od špecifických požiadakov.

Strojové učenie (ML)

Strojové učenie (ML) otvára v hrách fascinujúce možnosti a posúva hranice herného dizajnu na novú úroveň. Vďaka ML sa NPC stávajú inteligentnejšími a prispôsobivejšími, čím prinášajú do hier hĺbku a dynamiku.

Ako ML transformuje hry:

- Učenie sa z interakcií: Algoritmy ML umožňujú NPC učiť sa z vlastných skúseností a prispôsobovať svoje správanie na základe interakcií s hráčmi a herným prostredím. To vedie k realistickejším a pútavejším herným zážitkom.
- Generovanie kreatívnych stratégií: ML algoritmy dokážu analyzovať herné dáta a vyvíjať nečakané stratégie a taktiky, ktoré by človek len ťažko vymyslel. To prináša do hier prvok prekvapenia a robí ich náročnejšími.
- Neustále zlepšovanie: Vďaka ML sa AI agenty neustále učia a zdokonaľujú svoje herné schopnosti. To znamená, že hráči sa vždy stretávajú s novými výzvami a musia neustále prehodnocovať svoje stratégie.

Typy ML:

- Posilňovacie učenie: AI sa učí prostredníctvom odmien a trestov za svoje herné ťahy. Cieľom je naučiť sa robiť optimálne rozhodnutia, ktoré vedú k maximálnemu úspechu.[21]
- Supervízované učenie: AI sa učí z príkladov, ktoré sú už označené správnym riešením. V hrách sa to môže využiť napríklad na učenie AI hrať hru sledovaním záznamov hrania skúsených hráčov.
- Nesupervízované učenie: AI hľadá vzory a štruktúry v neoznačených dátach. V hrách sa to môže využiť napríklad na rozpoznávanie herných objektov alebo predpovedanie správania hráčov.

Výhody ML:

- Dynamické a pútavé herné zážitky: NPC s ML sa stávajú inteligentnejšími a prispôsobivejšími, čím robia hry dynamickejšími a pútavejšími.
- Zvýšená herná náročnosť: AI agenti s ML neustále zlepšujú svoje herné schopnosti, čím predstavujú pre hráčov stále väčšiu výzvu.
- Nové možnosti herného dizajnu: ML umožňuje dizajnérom hier implementovať komplexné herné mechanizmy a vytvárať sofistikovanejšie herné svety.

Nevýhody ML:

- Náročná implementácia: Implementácia ML do hier môže byť náročná a vyžaduje si hlboké znalosti herného dizajnu a algoritmov ML.
- Požiadavka na veľké množstvo dát: Trénovanie ML modelov si vyžaduje veľké množstvo dát, ktoré nie sú vždy dostupné.
- Nedostatočná transparentnosť: Fungovanie ML modelov môže byť pre ľudí ťažko pochopiteľné, čo môže viesť k nepredvídateľnému správaniu AI agentov.

Strojové učenie otvára v stolných hrách fascinujúce možnosti a posúva hranice herného dizajnu na novú úroveň. Vďaka ML sa stávajú hry dynamickejšími, pútavejšími a náročnejšími, čím lákajú nových hráčov a obohacujú herný svet. Implementácia

AI v stolových hrách

Vplyv AI na stolné hry:

- Inteligentní protivníci a kooperatívni spoluhráči: AI umožňuje vytvárať sofistikovaných protivníkov, ktorí dokážu analyzovať hernú situáciu, robiť strategické ťahy a prispôsobovať sa hernému štýlu hráča. V kooperatívnych hrách môže AI plniť úlohu zradcu alebo náročného bossa, čím do hry pridáva vrstvu nepredvídateľnosti a napätia.
- Zjednodušenie pravidiel a automatizácia: AI môže zjednodušiť zložité pravidlá a automatizovať niektoré aspekty hrania, čím robí stolné hry dostupnejšími pre širšie publikum. Hráči sa tak môžu sústrediť na strategické a sociálne aspekty hry a vychutnať si herný zážitok naplno.
- Nové herné mechanizmy a stratégie: AI umožňuje implementovať do hier komplexné herné mechanizmy a stratégie, ktoré by bez nej neboli možné. To vedie k vzniku inovatívnych herných konceptov a obohacuje herný svet o nové dimenzie.

Výhody AI v stolných hrách:

- Zvýšená dostupnosť: AI spoločníci robia stolné hry dostupnejšími pre sólových hráčov a ľudí, ktorí nemajú vždy možnosť hrať s ostatnými.
- Zlepšenie herných zručností: Hranie s AI protivníkmi umožňuje hráčom zdokonaľovať svoje herné zručnosti, stratégie a taktické myslenie.
- Nové výzvy a prekvapenia: AI protivníci dokážu prekvapiť neočakávanými ťahmi a strategiami, čím udržujú hru vzrušujúcou a nepredvídateľnou.
- Zvýšená zábava a interaktivita: AI môže do hry pridať humor, príbehy a interaktívne prvky, ktoré robia herný zážitok pútavejším a zábavnejším.

Nevýhody AI v stolných hrách:

- Náklady na vývoj a implementáciu: Implementácia AI do stolných hier môže byť náročná a nákladná, najmä pre menšie herné spoločnosti.
- Nedostatok sociálnej interakcie: Hranie s AI protivníkom nenahrádza sociálnu interakciu a emocionálne spojenie, ktoré prináša hranie s reálnymi ľuďmi.
- Možnosť predvídateľnosti: Niektoré AI protivníci nemusia byť dostatočne flexibilní a kreatívni, čím sa stávajú predvídateľnými a znižujú herné napätie.

AI v stolných hrách predstavuje vzrušujúci nový trend, ktorý prináša do herného sveta inovácie a obohacuje herný zážitok o nové dimenzie. Aj keď existujú určité výzvy a obmedzenia, potenciál AI v tejto oblasti je obrovský a sľubuje posunúť stolné hry na novú úroveň zábavy a interaktívnosti.

Kapitola 3

The Duke

V hre *The Duke* hráči pohybujú svoje jednotky (figúrky) po hracej doske a po každom ťahu ich obrátia. Každá strana zobrazuje iný pohybový profil. Ak hráč ukončí pohyb na štvorci obsadenom figúrkou súpera, zajme túto figúrku. Cielom hry je zajať figúrku *Duke* svojho súpera![1]

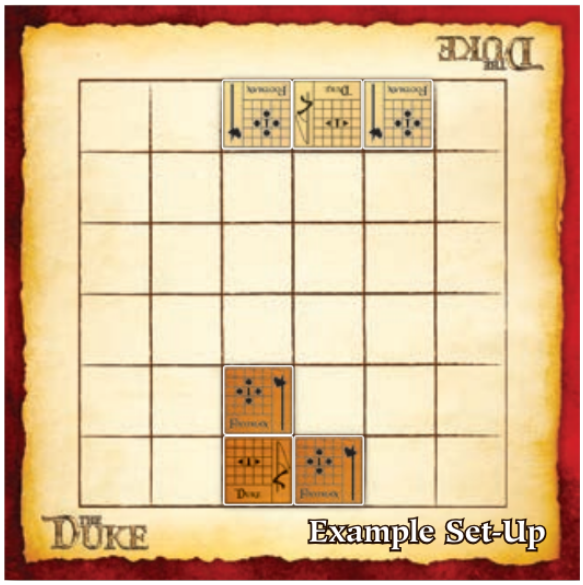
3.1 Pravidlá

Hra pozostáva z hracej dosky (6x6), 2 vakov na figúrky a dvoch setov osemnástich figúrok (Duke, 3xFootman, 3xPikeman, Dragoon Knight, Wizard, Seer, General, Priest, Champion, Marshall, Bowman, Assassin, Longbowman).

Každý hráč si vyberie jedno vrecúško s figúrkami, obaja hráči odložia figúrku *Duke* a dve figúrky *Footman*, všetky ostatné figúrky vložia do vrecúška.

Prvý hráč položí figúrku *Duke* na jedno z dvoch stredových políčok na svojom okraji hracej dosky. Potom umiestni dve figúrky *Footman* na akékolvek dve políčka ktore susedia s figúrkou *Duke*; zdielanie len rohu nieje povolené.

Druhý hráč potom umiestni svoju figúrku *Duke* na jendo z dvoch stredových poolíčok v opačnom okraji hracej dosky a umiestni svoje dve figúrky *Footman* na akékoľvek dve políčka, ktoré susedia s *Duke*. Znova musia byť ich dlhé strany zarovnané. Druhý hráč nemusí umiestniť svojeho *Duke* presne naproti *Duke* svojho súpera. Ukážka ako by to mohlo vyzerať je na ďalšej strane.



Obrázek 3.1: Ukážka ako môže vyzerať počiatočný ťah [1]

Orientácia: Každú figúrku treba umiestniť na hraciu dosku tak, aby jej názov bol orientovaný priamo smerom k hráčovi, ktorý ju ovláda. Bez ohľadu na to, kam sa figúrka pohne alebo kde na hracú dosku sa umiestnia nové figúrky, musia byť vždy orientované smerom k ovládajúcemu hráčovi a toto usporiadanie zostáva nezmenené počas celej hry.

Počiatočná Strana: Každá figúrka má na svojej *počiatočnej* strane počiatočný symbol, s ďalším symbolom na opačnej strane alebo *ne-počiatočnej* strane. Všetky figúrky umiestnené na hracej doske po prvýkrát, či už na začiatku hry alebo pridané počas hry, musia byť umiestnené s viditeľným počiatočným symbolom.

Obrázek 3.2: Počiatočná strana figúrky *Duke*[1]

Obrázek 3.3: Ne-počiatočná strana figúrky *Duke*[1]

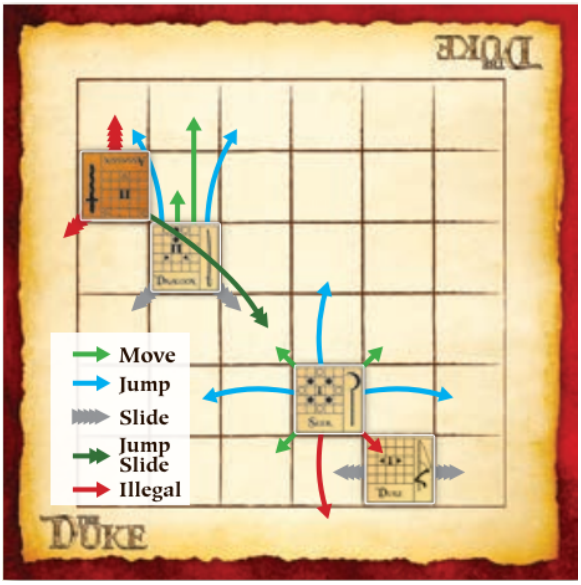
Každý hráč musí počas každého ťahu vykonať jednu a iba jednu z nasledujúcich akcií:

- **Presunúť figúrku:** Hráč presunie figúrku na hracej doske pomocou akéhokoľvek povoleného pohybu, ako je uvedené v jej pohybovom profile, a po dokončení ťahu ju otočte na opačnú stranu.
- **Umiestniť novú figúrku:** Hráč vyberie náhodnú figúrku z vrecúška hráča a umiestni ju na hraciu dosku na neobsadený štvorec, ktorý susedí so štvorcom obsadeným figúrkou *Duke* hráča.

Hráč nikdy nemôže vynechať ťah. V každom ťahu buď pohne existujúcou figúrkou, alebo umiestni novú.

Presun figúrky:

Počas každého ťahu má hráč možnosť presunúť akúkoľvek jednu figúrku, ktorú ovláda na hracej doske, pričom ju otočí na opačnú stranu po dokončení pohybu. Každá figúrka má mriežku pohybu (alebo profil); priestor v strede mriežky zobrazuje aktuálny štvorec figúrky vzhľadom na hraciu dosku, a ďalšie pohybové ikony zobrazujú štvorce, s ktorými môže táto figúrka na hracej doske interagovať. Ikony v danom štvorci v mriežke pohybu určujú, ako figúrka interaguje s touto pozíciou počas pohybu.



Obrázek 3.4: Ukážka ako môže ťahov figuriek [1]

Vo vyššie uvedenom príklade pohybu, má Seer veľa možností pohybu. Môže sa pohybovať o jedno pole diagonálne v troch smeroch alebo skákať o dva štvorčeky dopredu, doľava alebo doprava. Nemôže však zoskočiť z hracej plochy ani sa presunúť na pole, ktoré zaberá jej vlastná figúrka Duke. Medzitým Dragoon má tiež niekoľko možností pohybu. Mohlo by to byť použitie ikony Move na vstup do štvorca priamo nad ním alebo jeho ikony skoku na pohyb hore a doprava, alebo hore a doľava za Assassinom. Mohol by tiež použiť svoje ikony

Slide na presun doľava o jeden štvorec alebo dole a doprava o jeden štvorec; ale nemohol vstúpiť na štvorec kde je Seer. Nakoniec, Assassin má momentálne iba jednu možnosť pohybu, a to dve z možností vedú z hracej plochy. Vďaka svojej ikone Jump Slide mohol preskočiť popri Dragúnovi a buď zajať Seer, alebo sa zastaviť na štvorci medzi Seer a Dragúna, čím sa Duke dostal Guard!

Ikony Pohybu:

Nasledujúce pravidlá definujú, čo umožňujú rôzne ikony pohybu na mriežke pohybu každej figúrky. Hráč môže vo svojom ťahu vybrať iba jeden cieľový štvorec na interakciu a iba jednu ikonu, bez ohľadu na to, s koľkými štvorcami by figúrka mohla potenciálne interagovať, alebo na počet rôznych ikon na mriežke pohybu.

- **Move:** Táto ikona umožňuje figúrke pohnúť sa na označený štvorec, ak existuje jasná a priama cesta k nemu z počiatočného štvorca (štvorca, na ktorom sa kocka nachádza na hracej doske). Ak medzi počiatočným a cieľovým štvorcom existuje priateľská alebo nepriateľská figúrka, pohyb nie je možný. Rovnako pohyb nie je možný, ak v cieľovom štvorci existuje priateľská figúrka. Ak je v cieľovom štvorci nepriateľ, pohyb je možný a hráč zajme nepriateľskú figúrku.

- **Jump:** Táto ikona umožňuje figúrke pohnúť sa na štvorec zobrazený, preskočiac ľubovoľnú figúrku (priateľskú alebo nepriateľskú) po najkratšej ceste od počiatočného štvorca k cieľovému štvorcu. Ak je v cieľovom štvorci priateľská figúrka, pohyb nie je možný. Ak je v cieľovom štvorci nepriateľ, pohyb je možný a zajme sa nepriateľská figúrka (viď Zajímanie Kociek, na ďalšej strane). Figurky, cez ktoré sa skákalo, zostávajú bez zmeny.

- **Slide:** Táto ikona umožňuje figúrke pohnúť sa ľubovoľným počtom štvorcov v zobrazenom smere, pokiaľ existuje jasná cesta. Figúrka nesmie skákať cez žiadne iné kocky. Ak skončí svoj ťah na štvorci obsadenom nepriateľom, ovládajúci hráč zajme túto figúrku (viď Zajímanie Kociek, na ďalšej strane); nemôže skončiť svoj pohyb na štvorci obsadenom priateľskou figúrkou.

- **Jump Slide:** Táto ikona podlieha rovnakým pravidlám ako pre Slide, s nasledujúcim doplnením: táto ikona umožňuje figúrke pohnúť sa smerom k ikone Jump Slide a úplne ignorovať akúkoľvek figúrku, ktorá by mohla byť susedná, podľa označeného smeru.

- **Strike:** Táto ikona umožňuje figúrke zajímať z diaľky. Odstráňte nepriateľskú figúrku v jednom zo štvorcov označených touto ikonou (viď Zajímanie Kociek), ale ponechajte figúrku na jej pôvodnom mieste; použitie tejto ikony nemení polohu figúrky, ale figúrka sa stále otočí. Táto ikona nemá žiadny vplyv na priateľskú figúrku.

- **Command:** Figúrka s týmito symbolmi môže pohybovať iné figúrky okolo seba. Namiesto toho, aby sa pohybovalo touto figúrkou, hráč presúva jednu z figúrok, ktoré ovláda, z akéhokoľvek štvorca označeného jedným z týchto symbolov na iný štvorec označený jedným z týchto symbolov; hoci táto figúrka nevykonáva pohyb, stále sa otočí, ako vždy. Ak figúrka, ktorou sa pohybovalo, skončí svoj ťah na štvorci obsadenom nepriateľom, zajme túto kocku (viď Zajímanie Kociek); figúrka nemôže skončiť svoj pohyb na štvorci obsadenom priateľskou figúrkou.

Otočenie Kocky:

V každom z vyššie uvedených prípadov sa na konci ťahu hráča (po použití ikony pohybu) figúrka, ktorá použila ikonu, otočí na opačnú stranu. Ak sa figúrka presunie na nový štvorec, otočí sa na novú stranu po dosiahnutí nového štvorca. Ak sa použijú ikony Strike alebo Command, hráč stále otočí figúrku, aj keď sa figúrka nepohla. Pri použití ikony Command sa figúrka, ktorou sa pohybovalo, neotočí; otočí sa len figúrka, ktorá iniciovala pohyb (t.j. figúrka, ktorá použila ikonu Command).

Zajímanie Kociek:

Vždy, keď sa figúrka dostane na koniec svojho pohybu na štvorec obsadený figúrkou súpera, protihráčova figúrka sa odstráni z hracej dosky. Vždy, keď hráč pohne figúrkou, ktorá mu umožní zajímať *Duke* jeho súpera na jeho nasledujúcom ťahu, musí povedať *Guard*.

Umiesťovanie figúrky *Duke* do Nebezpečenstva:

Hráč nesmie presunúť figúrku, ak by to umožnilo súperovi zajímať jeho figúrku *Duke* na ďalšom ťahu.

Umiestnenie novej figúrky:

Počas každého ťahu hráča, namiesto pohybu figúrkou, ktorú ovláda, môže umiestniť novú kocku vojska na hernú dosku. Náhodne vyťahujte figúrku z príslušného vrecúška a umiestnite ju lícovou stranou nahor (označenú začiatočným ikonou) na akékoľvek prázdne políčko, ktoré susedí s hráčovou figúrkou *Duke* (zdieľanie len rohu nie je povolené); pamätajte na to, že ju umiestnite s správnou orientáciou. Ak je *Duke* ovládajúceho hráča obklopený (buď figúrkami ovládajúceho hráča alebo súpera, alebo okrajom herného plátna), nová figúrka nemôže byť umiestnená; hráč musí namiesto toho presunúť figúrku, ktorá už je na hernej doske. Keď je kocka vyňatá z tašky, musí byť umiestnená na dosku, aj keď je automaticky v ohrození, že bude chytená v ďalšom ťahu. Ak sa hráčova taška vyprázdni, tento hráč už nemôže umiestniť nové kocky počas tejto hry; hráč musí každý ťah presúvať kockou.

Hru vyhrá hráč okamžite po zajatí figúrky *Duke* súpera.

Kapitola 4

Algoritmy zvolené pre hranie hry *The duke* počítačom

Stolové hry, kedysi čisto preverenie ľudského dôvtipu a stratégie, sa stali bojiskom pre inteligentné stroje. Srdcom týchto prefíkaných protivníkov s umelou inteligenciou sú algoritmy, sofistikované súbory inštrukcií, ktoré riadia rozhodovanie počítača. Tento úvod skúma tri kľúčové algoritmy, ktoré poháňajú umelú inteligenciu v stolových hrách: Minimax, AlphaBeta Prunning a Monte Carlo Tree Search.

4.1 Minimax

Tento klasický algoritmus skúma všetky možné budúce ťahy pre umelú inteligenciu aj jej súpera a priradí skóre každému výslednému stavu hry. Minimax potom zvolí ťah, ktorý vedie k najvyššiemu skóre pre umelú inteligenciu, čím zabezpečí najvýhodnejší dlhodobý výsledok. [18]

Algoritmus Minimax je základným konceptom umelej inteligencie v stolových hrách a funguje ako strategická cestovná mapa pre rozhodovanie počítača. Funguje tak, že prehľadne prehľadáva stromovitú štruktúru nazývanú strom hry, kde každá úroveň predstavuje možný ťah v hre a vetvy predstavujú rôzne možnosti, ktoré nasledujú po každom ťahu.

Strom hry: Mapa možností

Predstavte si hru Piškvorky. Koreň stromu hry predstavuje aktuálny stav hracej plochy. Každá vetva vychádzajúca z koreňa zobrazuje možný ťah, ktorý by mohla umelá inteligencia urobiť. Z každého ťahu sa ďalej rozvetvujú ďalšie vetvy, ktoré predstavujú potenciálne odpovede súpera a tak ďalej. Strom hry sa neustále rozrastá a zahŕňa všetky možné sledy ťahov pre oboch hráčov, až kým hra nedosiahne svoj koniec (výhra, prehra alebo remíza).

Prehľadávanie s prioritou hĺbky: Navigácia vo vetvách

Minimax využíva stratégiu prehľadávania s prioritou hĺbky. Začína sa na koreni (aktuálny stav hracej plochy) a prechádza jednotlivé vetvy jednu po druhej. Pre každý zvažovaný ťah Minimax simuluje možné odpovede súpera a pokračuje v zostupnom smere po zodpovedajúcich vetvách v strome. Tento proces pokračuje, kým sa nedosiahne vopred stanovená hĺbka alebo sa hra prirodzene neskončí.

Sila hodnotenia: Priraďovanie skóre budúcnosti

Akonáhle Minimax dosiahne koniec vetvy (preddefinovaná hĺbka alebo záver hry), použije funkciu hodnotenia. Táto funkcia priradí výslednému stavu hry skóre na základe cieľa umelej inteligencie. V Piškvorkách by cieľom mohlo byť maximalizovanie počtu "X"na hracej ploche (čo predstavuje výhru pre umelú inteligenciu) alebo minimalizovanie počtu "O" (zabránenie súperovi vo výhre). Toto skóre odráža, aký priaznivý by bol daný výsledok pre umelú inteligenciu.

Spätná propagácia

Minimax sa vracia späť z listov (koncové stavy) preskúmaných vetiev a postupne prenáša skóre nahor. Na každej úrovni si vyberie ťah, ktorý vedie k najpriaznivejšiemu skóre podľa funkcie hodnotenia. Pre hráča, ktorý maximalizuje (umelá inteligencia), si vyberie ťah s najvyšším skóre. Naopak, pre minimalizujúceho hráča (súper) si vyberie ťah s najnižším skóre. Tento proces pokračuje, kým sa nedosiahne koreňový uzol a ťah s najlepším skóre pre umelú inteligenciu, berúc do úvahy všetky možné budúce výsledky, sa stane zvolenou akciou.

Algoritmické vysvetlenie fungovania Minimax algoritmu

Algoritmus minimax (stav, hĺbka):

1. Ak je stav koncový stav hry:
   - Vráť hodnotiacu funkciu(stav).
2. Ak je hĺbka = 0:
   - Vráť hodnotiacu funkciu(stav).
3. Ak je maximalizujúci hráč na rade:
   - nastav premennú hodnotu na $-\infty$ (negatívne nekonečno)
   - pre každý nasledujúciStav:
     - hodnota = Max(hodnota, Minimax(nasledujúciStav, hĺbka-1))
   - Vráť hodnotu
4. Inak (minimalizujúci hráč na rade):
   - nastav premennú hodnotu na $+\infty$ (pozitívne nekonečno)
   - pre každý nasledujúciStav:
     - hodnota = Min(hodnota, Minimax(nasledujúciStav, hĺbka-1))
   - Vráť hodnotu

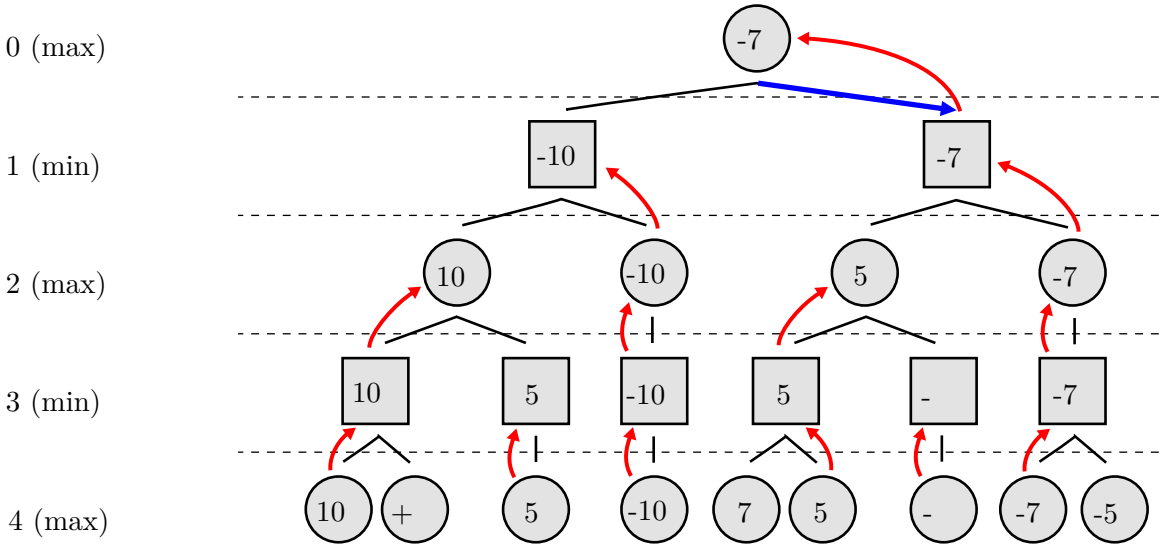
Algoritmus sa volá rekurzívne pre každý stav hry, pričom sa udáva aj aktuálna hĺbka prehľadávania.

Ak je stav koncový, vráti sa jeho hodnota vyhodnotená hodnotiacou funkciou.

Ak je dosiahnutá maximálna hĺbka prehľadávania, použije sa tiež hodnotiaca funkcia.

Pre maximalizujúci hráča sa pre všetky možné nasledujúce stavy (po jeho akcii) rekurzivne zavolá Minimax a zo získaných hodnôt sa vyberie maximum.

Pre minimalizujúceho hráča sa postupuje podobne, ale vyberie sa minimum.



Obrázek 4.1: Ukážka jednoduchého stromu Minimax algoritmu [5]

Limity Minimaxu

Minimax, hoci je výkonný algoritmus, má určité limity, hlavne pri riešení zložitých hier.

- Exponenciálny rast stromu hry: V hrách s veľkým rozvetvovacím faktorom (mnoho možných ťahov v každom ťahu) strom hry exponenciálne narastá s hĺbkou prehľadávania. Vyhodnotenie všetkých možných vetiev sa stáva výpočtovo náročným, najmä pre hry v reálnom čase.
- Nepraktické pre zložité hry: V komplexných hrách ako šach alebo Go je úplné prehľadávanie pomocou Minimaxu nepraktické kvôli obrovskému počtu možných ťahov. Čas a zdroje potrebné na prehľadanie celého stromu hry sa stávajú neúnosnými.

Na prekonanie týchto obmedzení sa používa technika AlphaBeta pruning. Táto technika strategicky odstraňuje irelevantné vetvy zo stromu prehľadávania, čím znižuje počet stavov, ktoré je potrebné hodnotiť. Vďaka tomu je Minimax efektívnejší a praktickejší pre zložitejšie hry. [18]

Silné stránky Minimaxu

Napriek svojim limitom, minimax zostáva základným algoritmom v teórii hier a umelej inteligencii.

- Garantovaná optimalita: Za predpokladu dokonalých informácií (poznajúcich všetky budúce ťahy) a neobmedzených výpočtových prostriedkov Minimax zaručene nájde pre umelú inteligenciu optimálny ťah, čím zabezpečí najlepší možný výsledok na základe vyhodnocovacej funkcie. To z neho robí cenný nástroj na analýzu hier a pochopenie strategického rozhodovania.
- Základ pre ďalšie algoritmy: Základné koncepty Minimaxu, ako je prehľadávanie stromu hry a hodnotenie stavu, slúžia ako stavebné kamene pre mnoho ďalších algoritmov

umelej inteligencie v stolových hrách. Pochopenie Minimaxu poskytuje pevný základ pre objavovanie pokročilejších techník.

Minimax je silným algoritmom pre hry s menším rozvetvovacím faktorom, ale pre zložitejšie hry s veľkým počtom možných ťahov je nutné ho vylepšiť technikami ako AlphaBeta pruning, aby sa dosiahla efektívnosť a praktickosť.

4.2 AlphaBeta

Hoci je Minimax výkonný, pre zložité hry s mnohými vetvami sa môže stať výpočtovo náročným. Tu prichádza Alpha-Beta Pruning. Pôsobí ako výkonná optimalizačná technika, zjednodušujúca vyhľadávanie Minimax a výrazne redukujúca počet stavov, ktoré treba vyhodnotiť. Ponoríme sa hlbšie do toho, ako Alpha-Beta Pruning zvláda túto výzvu efektívnosti.

Zatiaľ čo Minimax poskytuje silný prístup k hľadaniu optimálneho ťahu, jeho preskúmanie celého herného stromu môže byť výpočtovo náročné pre zložité hry. Alpha-Beta Pruning rieši túto výzvu dynamickým eliminovaním irelevantných vetiev počas hľadania Minimax.[18]

Jadrová myšlienka

Predstavte si záhradníka dôkladne orezávajúceho strom. Alpha-Beta Pruning sa správa ako strategický záhradník, sústrediac sa na najľubozvučnejšie vetvy (ťahy) v hernom strome a odstránením tých, ktoré nemôžu priniesť ovocie (vedú k horším výsledkom). Toto orezanie výrazne znižuje počet stavov, ktoré musí Minimax vyhodnotiť, čo robí rozhodovanie AI efektívnejším.

Alfa a Beta: Strážcovia Vyhľadávania:

- **Alfa:** Táto hodnota predstavuje najvyššie skóre, ktoré maximalizujúci hráč (AI) garantuje dosiahnuť pri zohľadnení všetkých doteraz preskúmaných budúcich ťahov. Pôvodne nastavená na negatívnu nekonečno , Alfa sa aktualizuje počas vyhľadávania, keď Minimax narazí na vetvy s lepším skóre pre AI.
- **Beta:** Táto hodnota predstavuje najnižšie skóre, do ktorého môže byť minimalizujúci hráč (súper) nútený zohľadniť všetky doteraz preskúmané budúce ťahy. Pôvodne nastavená na pozitívnu nekonečno , Beta sa aktualizuje, keď Minimax narazí na vetvy s horším skóre pre súpera.

Orezávanie s Účelom:

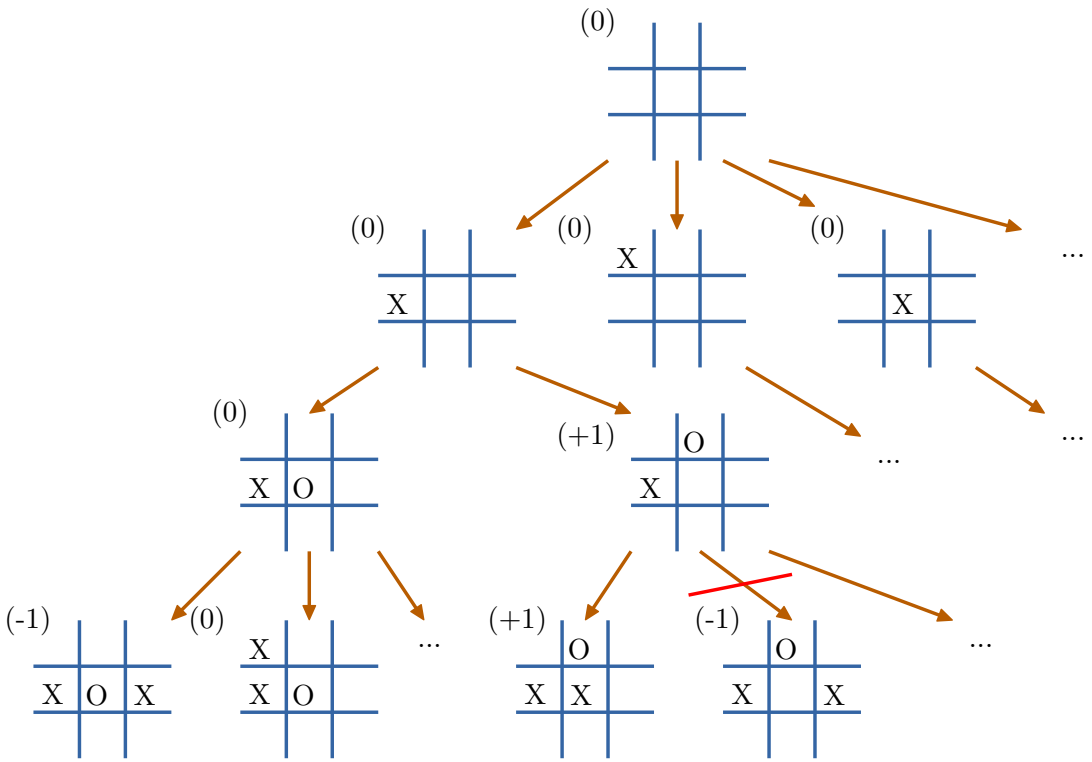
Alpha-Beta Pruning využíva Alfu a Betu na identifikáciu a elimináciu irelevantných vetiev. Tu je kľúčový princíp:

Ak počas vyhľadávania Minimax narazí na ťah pre maximalizujúceho hráča, ktorý dosahuje nižšie skóre ako aktuálna hodnota Alfa, celá vetva môže byť odrezaná. Prečo? Pretože AI nikdy nevyberie ťah, ktorý je garantovaný horší, ako to, čo už videla.

Podobne, ak Minimax narazí na ťah pre minimalizujúceho hráča, ktorý dosahuje vyššie

skóre ako aktuálna hodnota Beta, vetva môže byť odrezaná. Súper by nikdy nevybral ťah, ktorý ho núti do horšieho výsledku, ako už zaznamenal.

Ukážka Alpha Beta algoritmu v hre piškvorky



Obrázek 4.2: Ukážka Alpha Beta algoritmu v hre piškvorky [6]

- Po každom prvom ťahu X bude výsledkom stále remíza, ak obaja hráči hrajú správne
- V poslednom riadku na obrázku vyššie je na rade O. Jeden uzol má minimax hodnotu +1, čo znamená, že X už vyhral hru, za predpokladu, že X bude od tohto bodu hrať optimálne. Vedľa neho je uzol ktory ma hodnotu -1, nakolko tento uzol ma menšiu hodnotu ako Alpha(+1), môžeme celu cestu odrezať a tento uzol sa ďalej nerieši.
- V druhom rade na obrázku vyššie je na ťahu O. Preto minimax hodnota každého uzla v tomto riadku je minimom minimax hodnôt uzlov pod ním.
- V treťom rade na obrázku vyššie je na ťahu X. Preto minimax hodnota každého uzla v tomto riadku je maximom minimax hodnôt uzlov pod ním.

Algoritmické vysvetlenie fungovania Alpha-Beta Pruning algoritmu

1. Inicializácia:
    - Nastavte alpha na $-\infty$ (negatívne nekonečno) - predstavuje najhoršiu možnú hodnotu pre maximalizujúceho hráča.

- Nastavte beta na $+\infty$ (pozitívne nekonečno) - predstavuje najhoršiu možnú hodnotu pre minimalizujúceho hráča.

2. Rekurzia Alpha-Beta:

- Funkcia Alpha-Beta(stav, hĺbka, maximalizujúci hráč):
  - Ak je stav koncový stav hry:
    - Vráťte hodnotiacu funkciu(stav).
  - Ak je hĺbka = 0:
    - Vráťte hodnotiacu funkciu(stav).
  - Ak je maximalizujúci hráč na rade:
    - pre každý nasledujúciStav:
      - alpha = Max(alpha, Alpha-Beta(nasledujúciStav, hĺbka-1, False))
      - Ak alpha >= beta:
      - Preruš cyklus (Alpha-Beta Pruning)
      - Vráť alpha
  - Inak (minimalizujúci hráč na rade):
    - pre každý nasledujúciStav:
      - beta = Min(beta, Alpha-Beta(nasledujúciStav, hĺbka-1, True))
      - Ak alpha >= beta:
      - Preruš cyklus (Alpha-Beta Pruning)
      - Vráť beta

Algoritmus sa volá rekurzívne pre každý stav v stromčeku hry.

Funkcia prijíma aktuálny stav hry, hĺbku prehľadávania a informaciu, či je na rade maximalizujúci hráč.

Ak je stav koncový, vráti sa jeho hodnota vyhodnotená hodnotiacou funkciou.

Ak je dosiahnutá maximálna hĺbka prehľadávania, použije sa tiež hodnotiaca funkcia.

Pre maximalizujúceho hráča sa pre všetky možné nasledujúce stavy (po jeho akcii) spočíta Alpha-Beta hodnota a nastaví sa alpha na maximum z pôvodnej hodnoty a novej Alpha-Beta hodnoty.

Ak dôjde k bodu, kedy je alpha už väčšie alebo rovné beta, vieme, že minimalizujúci hráč dokáže prekaziť dosiahnutú hodnotu a ďalej sa v tejto vetve neprehľadáva (Alpha-Beta Pruning).

Pre minimalizujúceho hráča sa postupuje podobne, ale minimalizuje sa beta hodnota.

Výhody Alpha-Beta Pruning

Tu sú kľúčové výhody Alpha-Beta Pruning algoritmu:

- Dramatické Zrýchlenie: Najvýznamnejšou výhodou Alpha-Beta Pruning spočíva v jeho schopnosti dramaticky znížiť počet stavov preskúmaných algoritmom Minimax. Strategickým orezaním irelevantných vetiev sa Alpha-Beta Pruning zameriava na najpriaznivejšie ťahy, čo výrazne znižuje priestor pre vyhľadávanie a výpočetný čas. To sa prejavuje rýchlejším rozhodovaním pre AI, čo ju robí vhodnejšou pre hry v reálnom čase.

- Hrateľnosť v Reálnom Čase: Sám osebe je Minimax nepoužiteľný pre hry s veľkým počtom vetiev, ktoré by boli v reálnom čase nevypočítateľné. Alpha-Beta Pruning robí Minimax praktickejším pre takéto hry tým, že umožňuje rýchlejšie preskúmanie a výber ťahu v obmedzenom časovom rámci.

Nevýhody Alpha-Beta Pruning

Aj keď Alpha-Beta Pruning výrazne zlepšuje účinnosť algoritmu Minimax, má určité nevýhody:

- Stále výpočtovo náročné: Aj napriek optimalizácii Alpha-Beta Pruning zostáva Minimax výpočtovo náročným algoritmom, najmä pre hry s veľmi zložitými pravidlami a veľkým počtom možností. V takýchto hrách môže byť nutné zvážiť alternatívne algoritmy s nižšou náročnosťou, aj keď s nižšou presnosťou.

Efektívnosť a Implementácia:

Usporiadanie Ťahov: Účinnosť Alpha-Beta Pruning závisí od poradia, v ktorom sa ťahy vyhodnocujú v stromovej štruktúre. Vyhodnotenie sľubných ťahov najskôr umožňuje skoršie orezanie irelevantných vetiev. Avšak určenie "najlepšieho"poradia môže byť náročné a závisí od konkrétnej hry. Ak by sme mali tahy usporiadane od najhoršieho po najlepší Alpha Beta by nám nepomohla lebo by sme nemali uzol, ktorý by sme mohli odrezať.

Komplexita Implementácie: Starostlivá implementácia Alpha-Beta Pruning je kľúčová pre zabránenie zbytočného orezania, ktoré by mohlo viesť k suboptimálnym ťahom. Techniky ako nulové orezanie môžu ďalej zvýšiť efektívnosť.

Minimax vs. Alpha-Beta Pruning:

Minimax:

Zaručené nájdenie optimálneho ťahu pri dokonalých informáciách a výpočtových zdrojoch. Výpočtovo náročné pre hry s veľkým počtom vetiev. Nepraktické pre hranie v reálnom čase v komplexných stolných hrách.

Alpha-Beta Pruning:

Optimalizuje vyhľadávanie Minimax odstránením irelevantných vetiev, čím výrazne znižuje výpočtový čas. Alpha-Beta Pruning neobetuje optimálnosť. Zaručuje nájdenie najlepšieho ťahu, ak sú k dispozícii dostatočný čas a zdroje na úplné vyhľadávanie Minimax (aj keď v praxi to nemusí byť možné pre komplexné hry).

Alpha-Beta Pruning pôsobí ako silná optimalizačná technika, oživuje Minimax. Tým, že zjednodušuje proces vyhľadávania a znižuje výpočtové náklady, Alpha-Beta Pruning otvára cestu k rýchlejšiemu, efektívnejšiemu a nakoniec aj zaujímavejšiemu protivníkovi umelej inteligencie vo svete stolných hier.

4.3 Monte Carlo Tree Search

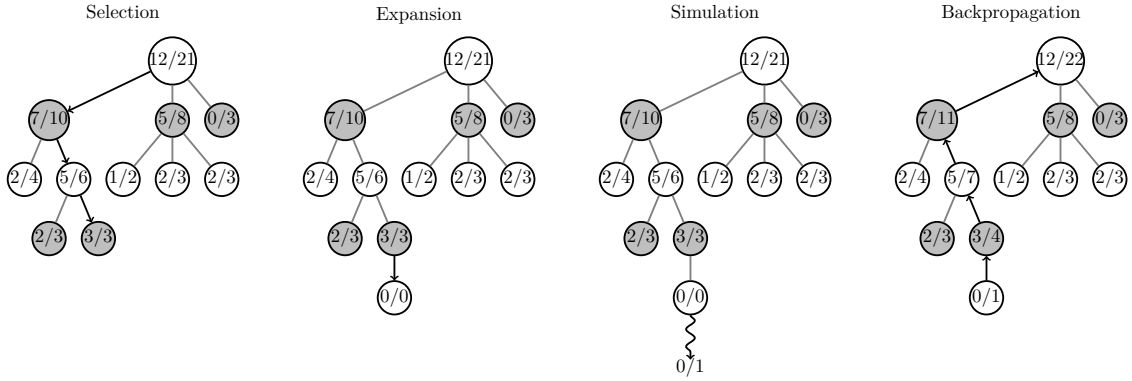
Tento pravdepodobnostný prístup sa zameriava na objavovanie sľubných herných stavov pomocou simulácií. Odohráva náhodné hry z aktuálnej pozície a učí sa z výsledkov, postupne uprednostňuje ťahy, ktoré vedú k väčšiemu počtu víťazstiev pre umelú inteligenciu.

Táto metóda vyvažuje prieskum (Exploration), vyskúšanie nových stratégií, s rozširovaím (Expansion), držaním sa osvedčených úspešných ťahov.

V oblasti stolných hier dosiahla umelá inteligencie impozantné úspechy. Algoritmy ako Minimax, so svojím stratégickým preskúmaním herného stromu, položili základy pre inteligentných protivníkov. Avšak deterministické vyhľadávacie algoritmy ako Minimax sa stretávajú s obmedzeniami v reálnych scenároch. Pri hraní hier s obrovskými vetvovitými faktormi (veľké množstvo možných ťahov v každej partii) exploduje vyhľadávací strom exponenciálne. Minimax, napriek svojej sile, sa môže stať výpočtovo nezvládnuteľným, čím je nepraktický pre rozhodovanie v reálnom čase v komplexných hrách.

To je miesto, kde vstupuje Monte Carlo Tree Search (MCTS). MCTS ponúka pravdepodobnostný prístup, ktorý sa oslobodzuje od obmedzení deterministického vyhľadávania. Využíva chytrú stratégiu, ktorá vyvažuje preskúmanie a exploatáciu, umožňujúc umelej inteligencii efektívne navigovať rozsiahlymi hernými priestormi. Ponorme sa hlbšie do jadrových princípov MCTS a ako prekonáva obmedzenia deterministických vyhľadávacích algoritmov.

Minimax, hoci silný, má problémy s obrovskosťou komplexných herných priestorov. Monte Carlo Tree Search (MCTS) rieši túto výzvu unikátnym prístupom. Spoľahne sa na simulácie a štatistiky, čím sa stáva pravdepodobnostnou vyhľadávacou technikou. Poďme preskúmať štyri jadrové fázy MCTS:



Obrázek 4.3: Ukážka 4 fáz MCTS algoritmu [4]

Výber(Selection):

Predstavte si rozľahlý les (herný strom) s mnohými cestami (možné ťahy). MCTS nevyberá smer naslepo. Namiesto toho používa výberovú politiku, ako je napríklad Upper Confidence Bound Applied to Trees (UCT), aby stratégicky vybral "najlákavejšiu"cestu. UCT vyvažuje prieskum (skúšanie nových ťahov) a rozširovanie (zameranie sa na ťahy, ktoré ukázali úspech v minulých simuláciách).

$UCT = \overline{X_j} + \sqrt{\frac{2 \ln n}{n_j}}$ [10]

Kde $\overline{X_j}$ je priemerný vysledok simulacii uzlu j, n_j je počet simulácii uzlu j a n celkový počet simulácii rodičovského uzlu

Rozšírenie(Expansion):

Po nasledovaní vybranej cesty (na základe výberu) môže MCTS naraziť na neobjavenú oblasť (neexpandovaný uzol v strome). Tu MCTS rozširuje strom. Pridáva nové detské uzly

reprezentujúce možné ťahy, ktoré doteraz neboli zvážené. To rozširuje pochopenie AI o stave hry.[21]

Simulácia(Simulation):

MCTS prehráva rýchlu a hrubú simuláciu z nového expandovaného uzla (alebo z akéhokoľvek sľubného uzla) do konca hry (terminálneho stavu). Tieto simulácie sú náhodné a často zjednodušené v porovnaní s skutočnou hrou. Cieľom je získať informácie o potenciálnych výsledkoch rôznych ťahov.[21]

Spätná Propagácia(Backpropagation):

Po simulácii sa MCTS vracia späť po vybranej ceste (z uzla simulácie späť k základu). Na každom uzle aktualizuje štatistiky výhier/prehier na základe výsledku simulácie. Ak simulácia skončila výhrou pre AI (alebo priaznivým výsledkom), uzly na ceste získavajú "výhru". Naopak, prehry sa odrážajú v štatistikách. Tento proces informuje budúce výbery, usmerňuje AI k viac sľubným ťahom v čase.[21]

Príklad MCTS v hre:

Predstavte si hru, kde AI je na rade zvoliť ťah.

Výber: S využitím UCT by MCTS mohol vybrať preskúmať vetvu, kde AI umiestnilo kameň(hra Go) alebo figúrku(šach) na konkrétnom mieste. Rozšírenie: Ak toto miesto ešte nebolo preskúmané, MCTS rozširuje strom pridaním detských uzlov reprezentujúcich možné odpovede súpera. Simulácia: Z každého nového detského uzla MCTS prehrá rýchlu simuláciu (napríklad niekoľko náhodných ťahov zo strany oboch hráčov), aby videl, ako by sa hra mohla vyvinúť. Spätná Propagácia: Na základe výsledkov simulácie (výhra/prehra pre AI) MCTS aktualizuje štatistiky výhier/prehier pre uzly po celej ceste späť k základu. To pomáha AI naučiť sa, ktoré oblasti herného stromu sú výhodnejšie a usmerňuje budúce výbery ťahov. Tým, že neustále iteruje cez tieto štyri fázy, MCTS efektívne preskúma rozsiahly herný priestor, učí sa zo simulácií a časom zdokonaluje svoje rozhodovanie. To MCTS robí obzvlášť vhodným pre komplexné hry s obrovskými vetvovitými faktormi.

MCTS sa ukazuje ako sľubná alternatíva k deterministickým vyhľadávacím algoritmom ako Minimax. Zatiaľ čo Minimax exceluje v hľadaní optimálneho ťahu s dokonalými informáciami, má problémy s obrovskými komplexnými hernými priestormi. MCTS rieši túto výzvu pravdepodobnostným prístupom, čo ho robí vhodným pre reálne scenáre s neúplnými informáciami. Poďme sa ponoriť do síl MCTS, jeho úvah a porovnania s Minimaxom.

Sily MCTS

MCTS sa ukázal ako alternatíva k tradičným algoritmom. Tu sú niektoré z jeho kľúčových výhod:

- Navigácia v neistote: Na rozdiel od Minimaxu, ktorý vyžaduje dokonalé poznanie budúcich stavov, MCTS vyniká v hrách s neúplnými informáciami. Dokáže sa učiť a prispôsobovať prostredníctvom simulácií, aj keď nepozná vopred všetky možné výsledky.
- Šampión preskúmavania: MCTS aktívne skúma nové ťahy počas simulácií. To môže viesť k objavovaniu nových stratégií, ktoré by mohli byť prehliadnuté deterministickými algoritmami fixovanými na preddefinované cesty.

- Dynamická adaptácia: MCTS sa dokáže prispôsobovať meniacim sa herným situáciám. Prioritizáciou ťahov, ktoré preukázali úspech v minulých simuláciách, sa umelá inteligencia neustále zdokonaľuje počas hry.

Obmedzenia

Hoci MCTS ponúka významné výhody pre komplexné hry, má aj niektoré nevýhody, ktoré treba zvážiť:

- Výpočtové nároky: Spúšťanie mnohých simulácií má svoju cenu. Pre komplexné hry môže byť MCTS výpočtovo náročný, a preto je dôležité starostlivo zvážiť hĺbku simulácie pre dosiahnutie optimálneho výkonu.
- Ladenie motora: Účinnosť MCTS závisí od výberovej politiky (ako napríklad UCT) a parametrov simulácie. Často je potrebné experimentovať, aby sa našla najlepšia konfigurácia pre konkrétnu hru.

MCTS vs. Minimax:

Minimax

- Sily: Zaručene nájde optimálny ťah s dokonalými informáciami a dostatkom výpočtových zdrojov. Poskytuje pevný základ pre pochopenie ďalších algoritmov umelej inteligencie.
- Obmedzenia: Výpočtovo náročný pre hry s veľkými vetvovitými faktormi. Nepraktický pre hranie v reálnom čase v komplexných stolných hrách.

MCTS:

- Rieši obmedzenia: Zameriava sa na preskúmavanie a adaptáciu, čím sa stáva vhodným pre hry s obrovskými vetvovitými faktormi a neúplnými informáciami. Efektívne preskúmava herný priestor prostredníctvom simulácií.
- Kompromisy: Nezaručuje optimálny ťah, ale poskytuje dobrú rovnováhu medzi preskúmaním a exploatáciou. Vyžaduje starostlivé ladenie parametrov pre optimálne výkony.

Kapitola 5

Implementácia

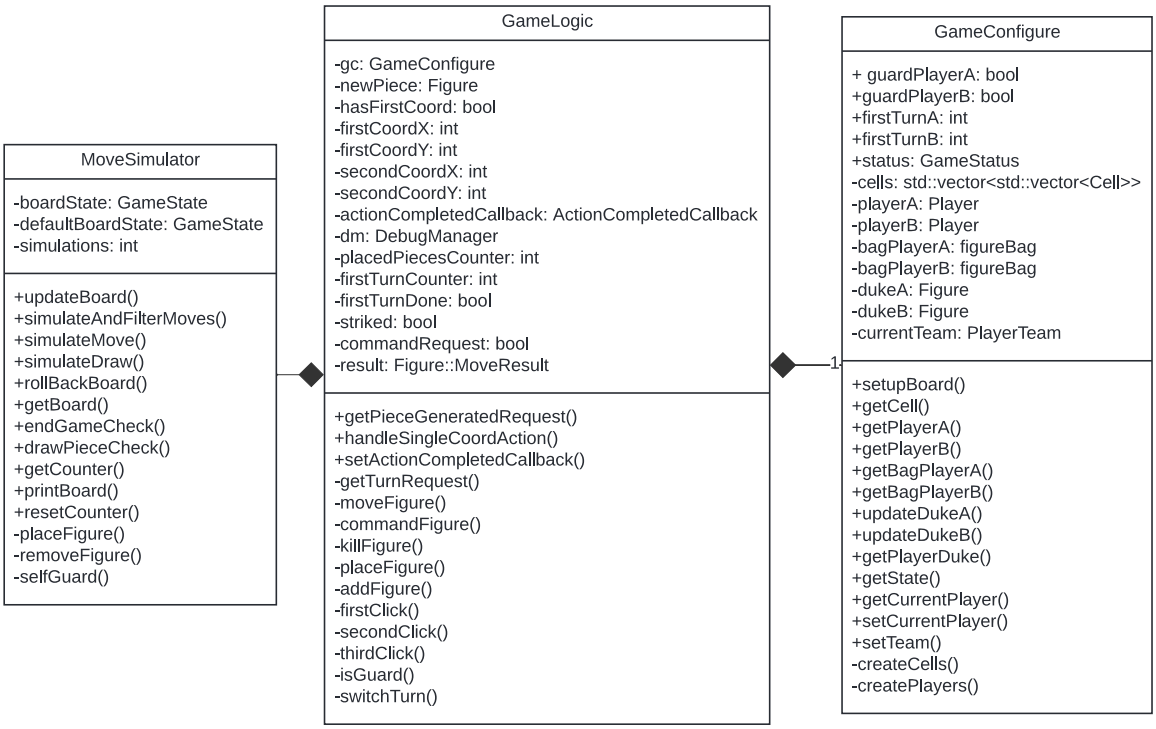
5.1 Jazyk a nástroje:

Na implementáciu hry *The Duke* a aj samotné algoritmy bol použitý programovací jazyk C++, kvoli jeho lepšiemu výkonu oproti pythonu s využitím frameworku QT na užívatelské rozhranie

Architektúra

Projekt je navrhnutý s dôrazom na modularitu a objektovo orientované princípy s cieľom zabezpečiť škálovateľnosť, udržateľnosť a opätovné použiteľnosť. Každá časť programu je zapuzdrená v rôznych triedach, čo podporuje abstrakciu a zapuzdrenie.

Štruktúra tried programu zahŕňa niekoľko kľúčových komponentov.



Obrázek 5.1: UML GameLogic

Trieda GameLogic:
Toto je jadro hry, ktoré spravuje celkový stav a pravidlá hry. Stav hry je uložený v triede gameconfigure. Trieda GameLogic je zodpovedná za základnú inicializáciu hry, spracovávanie herných udalostí a vynucovanie pravidiel hry.

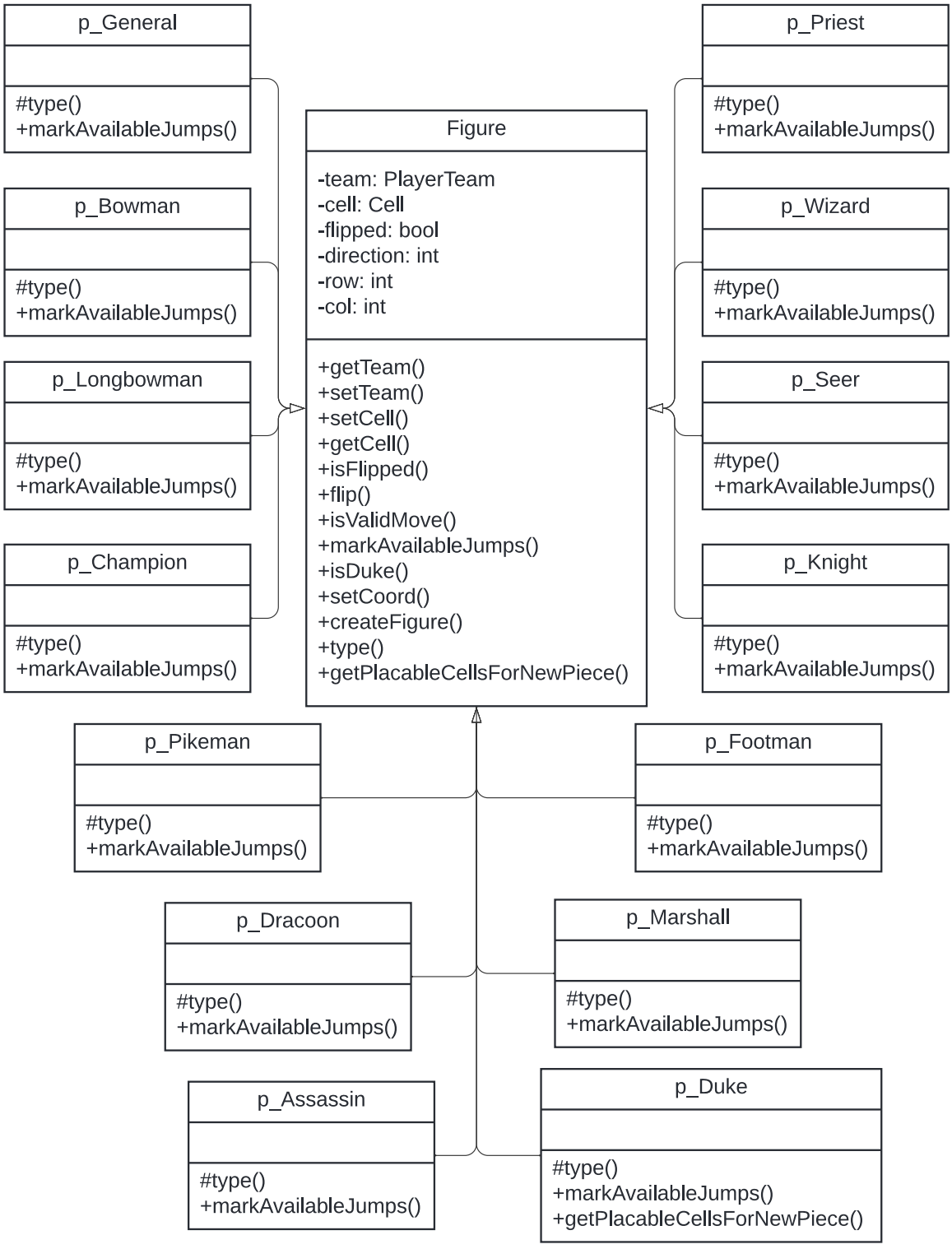
Uchováva si informáciu o súradniciach ktoré boli zvolené a podľa toho reaguje (urobí ťah alebo čaká na dalšie súradnice), v jeden moment môže mat informáciu až o 3 zvolených súradniciach (pri použití tahu Command)

Pre zachovanie konzistencie je táto trieda jediná ktorá priamo upravuje stav hry, ostatné triedy používaju iba štruktúru GameState, ktorá obsahuje jednoduchý momentálny stav hry

```
struct GameState {
    std::vector<std::vector<std::tuple<PieceType, PlayerTeam, bool>>> board;
    PlayerTeam currentPlayer;
    std::vector<PieceType> playerABag;
    std::vector<PieceType> playerBBag;

    bool playerA_UnderGuard;
    bool playerB_UnderGuard;

    std::pair<int, int> dukeCoordA;
    std::pair<int, int> dukeCoordB;

    int firstTurnA;
    int firstTurnB;

    GameStatus status = InProgress;
};
```

Na simulovanie ťahu figúrky z vrecka je použitá trieda figureBag, ktorá pri inicializácii vytvorý QList ktory obsahuje objekty figúrok, na náhodne "ťahanie z vrecúška"sa využíva QRandomGenerator

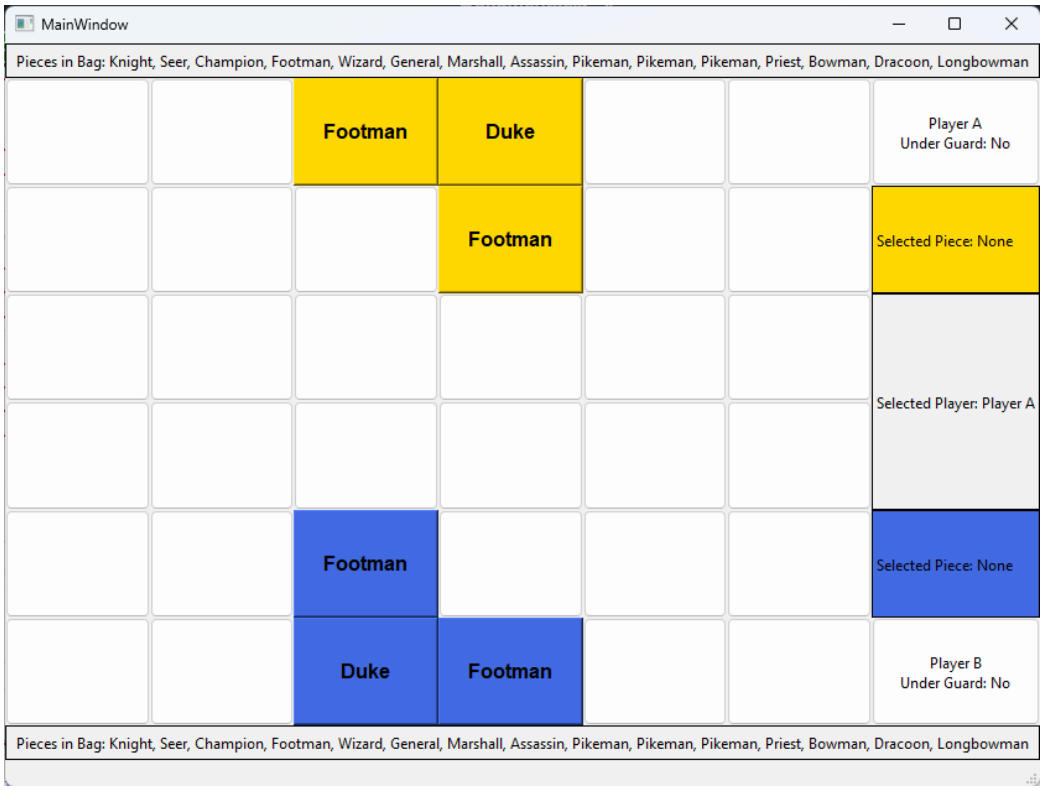
Trieda Figure:
Základná trieda pre všetky figúrky je trieda Figure, ktorá sa stará o vytvorenie správneho objektu pre každú figúrku s použitím triedy *Figure* createFigure(PlayerTeam team, PieceType type)*, o správne zvolenie políčok kde sa môže daná figúrka pohnúť sa stara funckia *MoveResult markAvailableJumps(GameState state)*, ktorú implementuje každá trieda figúrky pre seba, táto funkcia ale používa iba statické informácie o stave dosky a preto sa priamo táto funkcia nepoužíva v GameLogic ale namiesto toho je použitá trieda MoveSimulator ktorá obsahuje metódy na filtrovanie platných tahov, kontrolu podmienok konca hry a je použitá aj pri simulovaní tahov algoritmov



Obrázek 5.2: UML Figure

GUI (Qt): Na vytvorenie jednoduchého užívatelského rozhrania (hlavne kvoli testovaniu) je použitá iba jedina trieda mainwindow, ktorá obsahuje hraciu dosku, a základné informácie o hre (Duke ktorého hráča ma Guard, figúrky vo vrecku, vytiahnutá figúrka

ktorú treba umiestnit...), doska je pre jednoduchosť vytvorená z tlačítok (QPushButton), ktorým sa iba upravuje farba pozadia a text podla toho aká figúrka je na danej pozícii

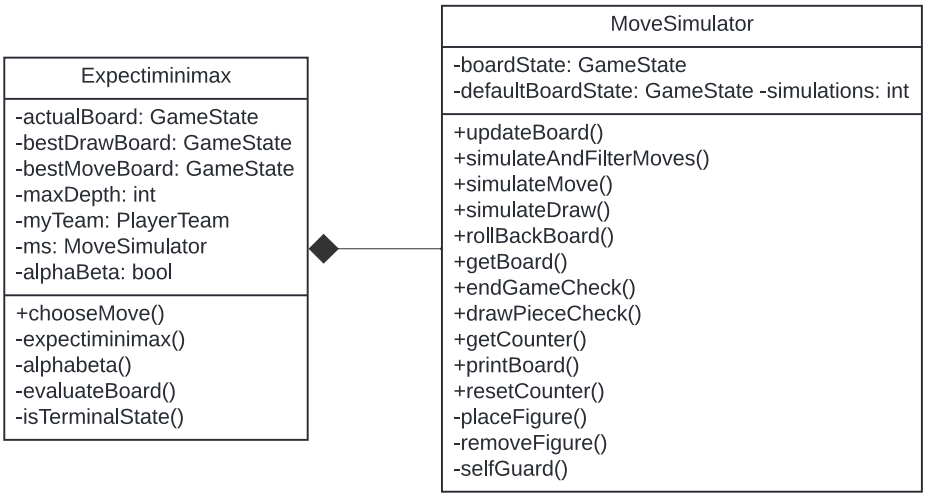


Obrázek 5.3: Ukážka GUI

Implementácia algoritmov: Implementované sú 3 algoritmy MCTS a minimax a Alpha Beta, minimax a Alpha Beta sú implementované v jednej triede, ktorá ich používa, podľa prepínača zvoleného pri jej vytváraní

Trieda Expectiminimax Táto trieda implementuje metódy algoritmov minimax a alphabeta. Metóda minimax funguje tak, že rekurzívne vola samú seba s dekrementovanou hlbkou a zmeneným prepínačom na Maximalizovanie/Minimalizovanie bodov pre daného hráča. Ak zistí koniec hry pred dosiahnutím požadovanej hĺbky vráti hodnotu nekonecno/-nekonečno podla toho aký hráč je na tahu. Ak dosiahne požadovanú hlbku tak sa vypočíta "hodnota dosky"podla vopred definovanej funkcie. Metóda alphabeta funkuje skoro rovnako ako minimax až na to že každá iterácia propaguje aj hodnoty Alpha/Beta, ak pri hľadaní tahu zistí, že dalši tah by mal horší výsledok ako Alpha/Beta tak sa preskočí rozbalovanie toho uzla.

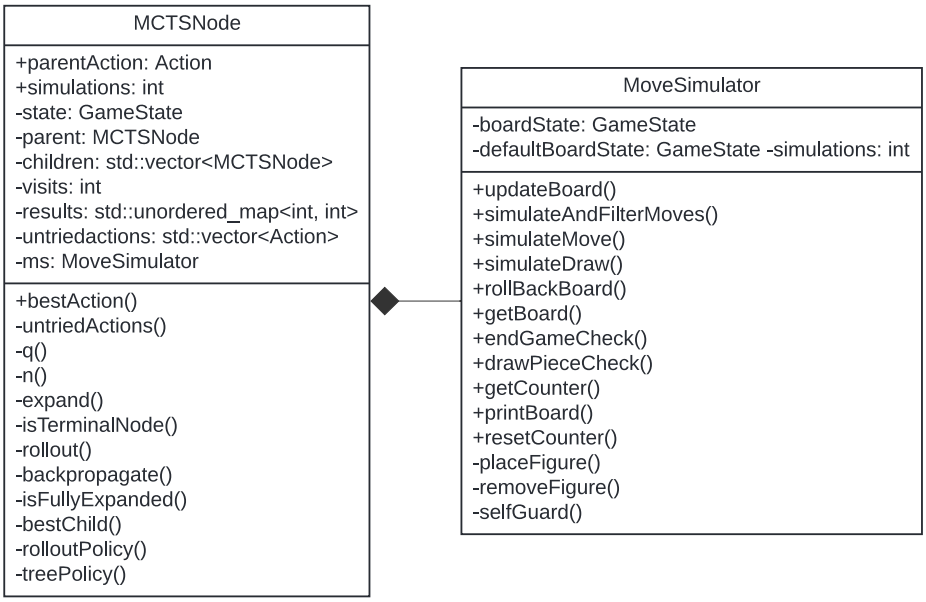
```
bestMoveScore = std::min(bestMoveScore, alphabeta(actualBoard, \
                  depth - 1, alpha, beta, false));
    beta = std::min(beta, bestMoveScore);
    if (beta <= alpha) {
        break; // Alpha cut-off
    }
```



Obrázek 5.4: UML Expectiminimax

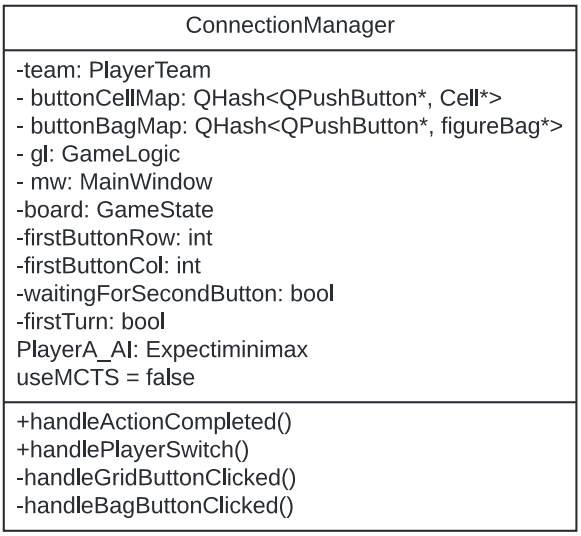
Trieda MCTSNode Táto trieda implementuje algoritmus MCTS, obsahuje viac metód ako predchádzajúca trieda, na začiatku sa vytvorí prvá instancia triedy, ktorá načíta všetky možné validné tahy pre momentálny stav hry Potom sa zavolá trieda bestAction(), ktorá vdaka funkcii bestChild zvolí najlepší uzol na simulovanie, simulovanie využíva náhodné tahy a prebieha až pokial nieje koniec hry, výhra niektorého z hráčov alebo remíza Na zvolenie dalšieho uzla je využitý vzorec UCT ktorý bol popísaný vyžšie.4.3

```
for (const auto& child : children) {
    double weight = (child->q() / child->n()) + \
    cParam * std::sqrt((2 * std::log(this->n())/ child->n()));
    if (weight > maxWeight) {
        maxWeight = weight;
        bestChild = child;
    }
}
```



Obrázek 5.5: UML MCTSNode

ConnectionManager Všetky časti projektu sú prepojende pomocou triedy ConnectionManager, táto trieda sa stará o komunikáciu medzi GUI (mainwindow), jadrom hry (GameLogic) a algoritmamy, čo zabezpečuje modulárny dizajn na jednoduchšiu modifikáciu, či už pravidiel hry, gui alebo pridávanie algoritmov



Obrázek 5.6: UML ConnectionManager

Kapitola 6

Výsledky a vyhodnotenie

V tejto kapitole sa zameriame na porovnanie výsledkov troch implementovaných algoritmov (minimax, AlphaBeta pruning a MCTS) a na celkové zhodnotenie projektu.

6.1 Porovnanie algoritmov

Minimax vs AlphaBeta

V tejto sekcii porovnáme dva základné algoritmy vyhľadávania v hrách: minimax a alphabeta, v kontexte našej hry. Zameriame sa na to, ako sa algoritmy líšili.

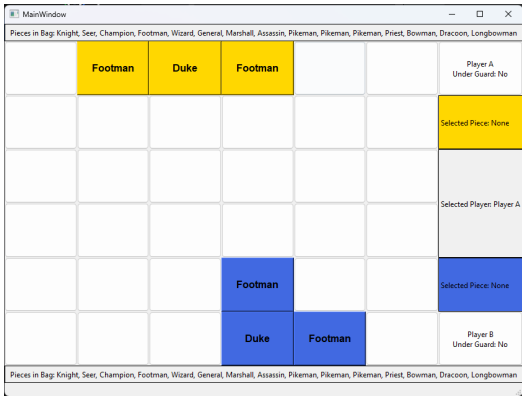
Náročnosť na implementáciu a pochopenie:

Algoritmus minimax je pomerne jednoduchý na na implementáciu a pochopenie, vyžaduje len základné znalosti teórie hier. Zatial čo algoritmus alphabeta je mierne zložitejší na implementáciu a pochopenie, pretože vyžaduje dodatočnú logiku pre prerezávanie vetiev stromu vyhľadávania

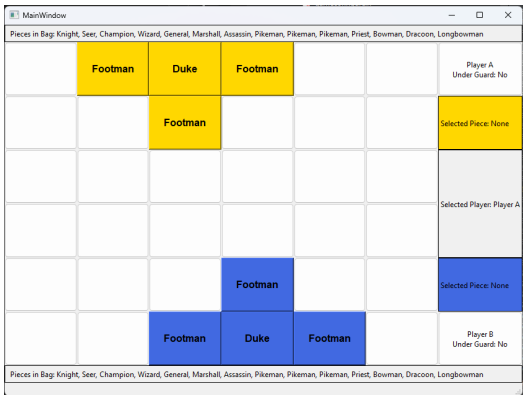
Výkonnosť:

Pre účely porovnania budeme používať rovnaké rozloženie dosky pri prvých 2 ťahoch Aj keď množstvo skúmaných uzlov sa môže mierne meniť podla náhodného zvolenia figúrky.

V prvom ťahu pri hĺbke 5 minimax vykonal 78710 simulácii za 1779 milisekúnd a v druhom ťahu bolo pri hĺbke 5 103453 za 2423 milisekúnd. Zatial čo v prvom ťahu pri hĺbke 6 vykonal 799804 simulácii za 17457 milisekúnd a v druhom ťahu bolo simulácii 1109030 za 28784 milisekúnd. Čo je rozdiel viac ako 10 násobok.



Obrázek 6.1: Prvý ťah



Obrázek 6.2: Druhý ťah

Vďaka tomu môžeme vypočítať priemerný faktor vetvenia $O(b^d)$ pre prvé ťahy, kde b je faktor vetvenia a d je zvolená hĺbka vyhladávania.[18]

Faktor vetvenia pre prvý ťah 9,6.

Faktor vetvenia pre druhý ťah 10,1.

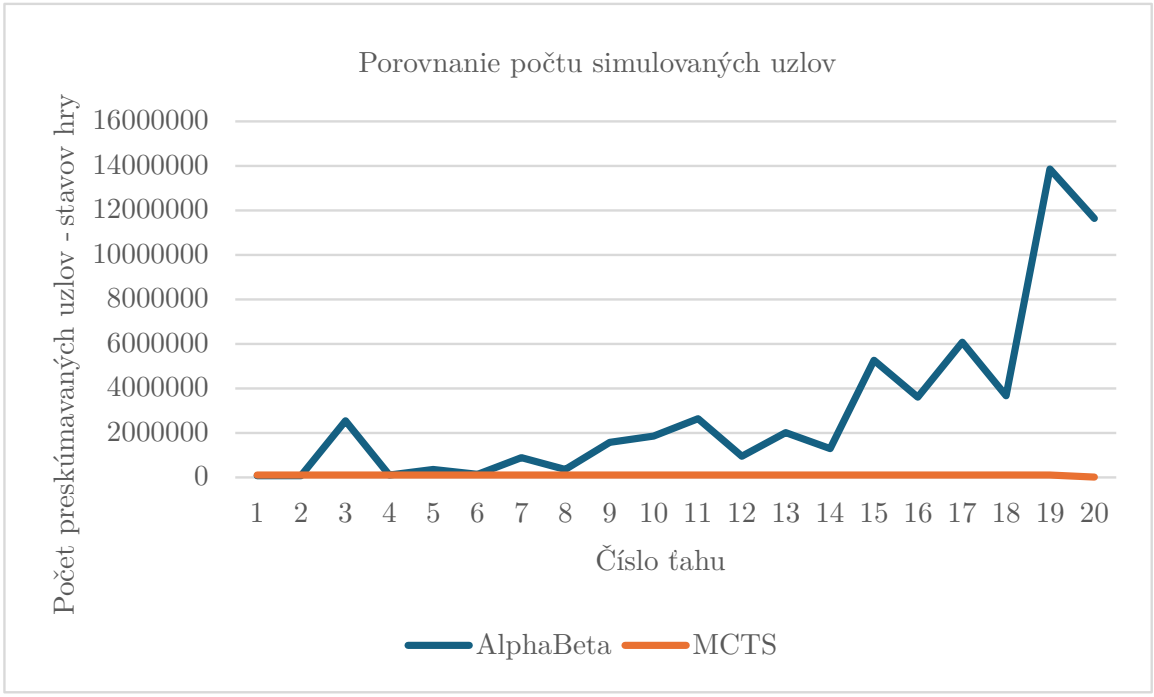
V prvom ťahu pri hĺbke 5 alphabeta algoritmus vyhodnotil 19989 uzlov za 488 milisekúnd a v druhom ťahu bolo pri hĺbke 5 15546 za 448 milisekúnd. Zatial čo v prvom ťahu pri hĺbke 6 vykonal vyhodnotenie 139878 uzlov za 3758 milisekúnd a v druhom ťahu bolo vyhodnotených 88313 uzlov za 2494 milisekúnd. Zaujímavosťou je, že pri porovnávaní alphabeta mala v druhom ťahu menej vyhodnotení ako v prvom, tento jav sa opakoval aj vo väčšej hĺbke a rozdiel v počte vyhodnotení medzi hĺbkou 5 a 6 bol v priemere menej ako 6 nasobok. Dôvodom je, spôsob ako je tento algoritmus implementovaný a to je ten, že najprv sa skontroluje a vyhodnotí sa možnosť pre ťahanie novéj figúrky a až potom sa prevyhodnocujú pohyby, nakolko v prvom ťahu môžu obaja hráči ťahať tak sa vyhodnocuje vetva pre ťahanie oboch hráčov, dalším dôvodom je naša vyhodnocovacia funkcia, ktorá dáva + body za každú našu figúrku a - body za figúrku ktorú ovláda oponent, nakoľko pri našom rozložení dosky v druhom ťahu nemôžme zajať žiadnu figúrku oponenta, vetva kde budeme ťahať novú figúrku bude mať stále väčšie skóre, vdaka AlphaBeta algoritmu tak možeme preskočiť vyhodnocovanie značného množstva stavov.

AlphaBeta vs MCTS

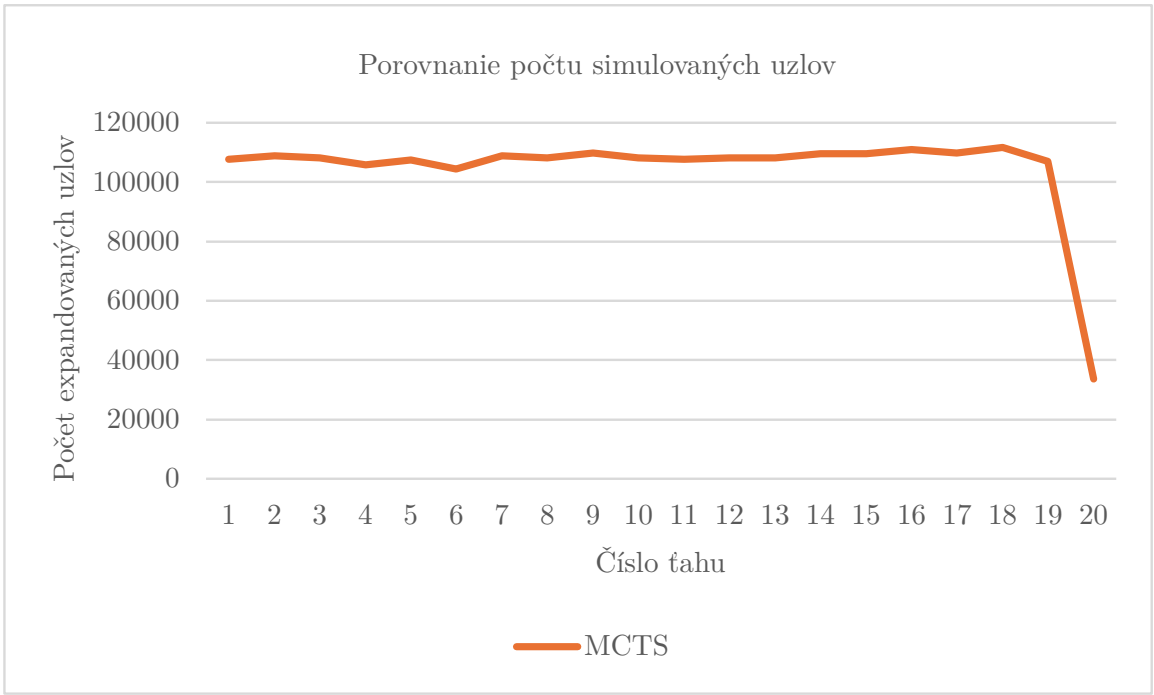
Pre testovanie, necháme hrať AlphaBeta(Hráč A) algoritmus proti MCTS(Hráč B). Nastavená hĺbka prehľadávania AlphaBeta je 6, počet simulácii pre MCTS(koľko krát musí odsimulovať hru do konca) je nastavený na 1000, nakoľko ale hladal simulácie, ktoré skončili až pri niekoľko tisíc tahov, použijeme obmedzenie, že ak simulácia neskončí do 100 ťahov, tak to automaticky zaráta ako remízu.

Počas celého testovania budeme považovať za prvý ťah až ťah ktorý nasleduje po prvotnom rozložení figúrky *Duke* a dvoch *Footmanov*

Ako je možné vidieť z grafov, ani s alphabeta algoritmom by hra s takýmto veľkým faktorom vetvenia nebola hratelná a to sme použili iba hĺbku 6, a už sme pri väčšom počte figúrok na doske išli do rádov niekoľko miliónov simulácii, ako je možné vidieť na konci hry kde bolo na doske najviac figúrok počet simulácii stúpol skoro až na 14 milionov.



Obrázek 6.3: Graf počtu simulovaných uzlov AlphaBeta algoritmu



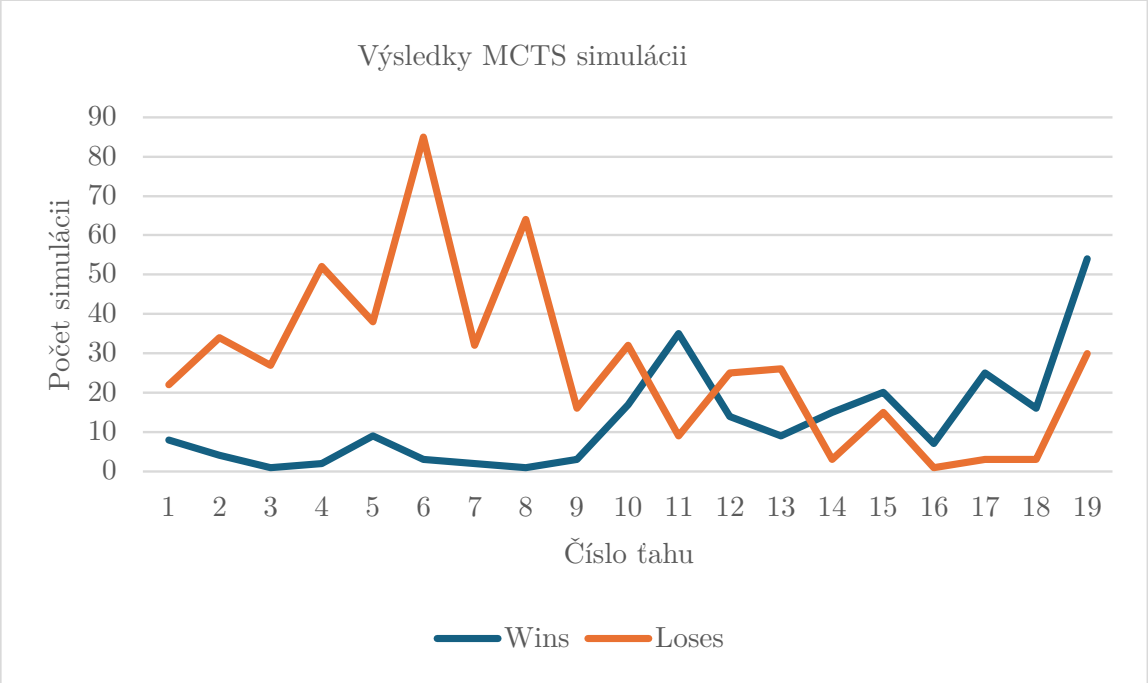
Obrázek 6.4: Graf počtu simulovaných uzlov MCTS

MCTS algoritmus v porovnaní s AlphaBeta má obrovský rozdiel v počte simulácii, aj vďaka spomínanému obmedzeniu na 100 ťahov. Na konci je možné vidiet masívny pokles simulácii, to je sposobené tým že sa algoritmus blížil ku koncu hry.

Číslo ťahu	1	2	3	4	5	6	7	8	9	10
AlphaBeta	88370	84453	2560946	105105	347559	142667	895546	355030	1596070	1867591
MCTS	107634	108812	108159	105759	107449	104539	108842	108169	109859	108268

Číslo ťahu	11	12	13	14	15	16	17	18	19	20
AlphaBeta	2634783	954332	2020535	1288969	5262909	3612259	6089500	3682119	13873606	11634178
MCTS	107715	108225	108125	109491	109571	111036	109738	111826	106972	33588

Tabulka 6.1: Tabuľka počtu simulovaných uzlov AlphaBeta a MCTS

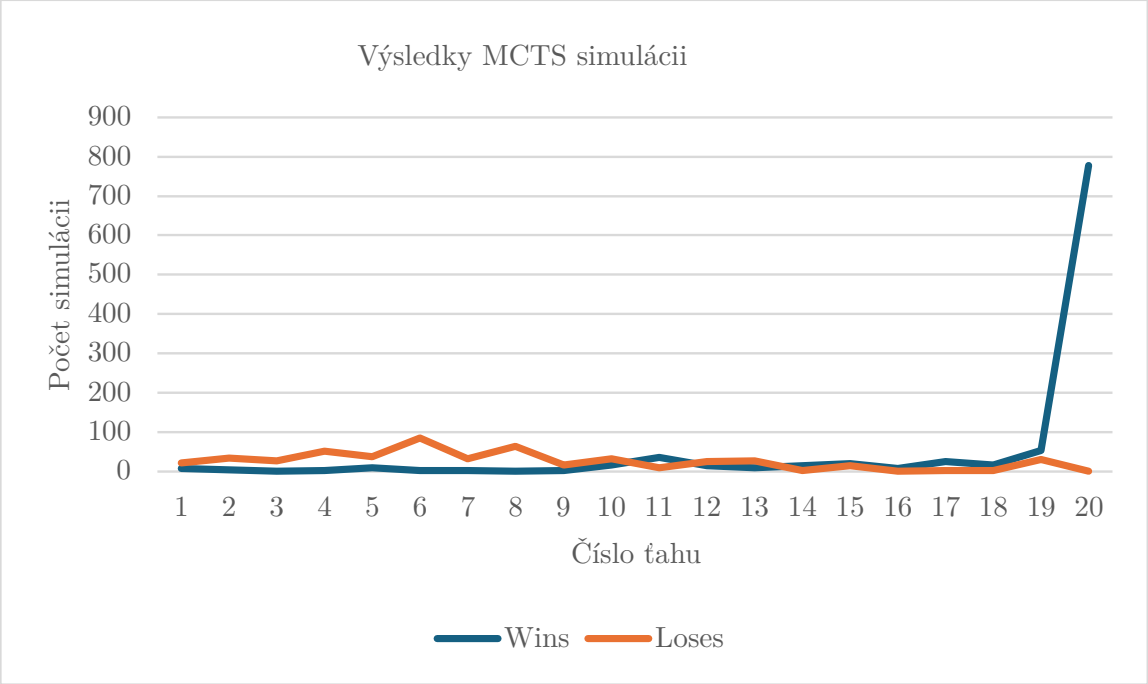


Obrázek 6.5: Štatistiky simulácii MCTS

Na tomto grafe môžeme vidiet kolko krát z tých 1000 simulácii ktoré MCTS algoritmus robil každý ťah vyhral/prehral. Na začiatku je vidieť značnú výhodu pre oponenta, dôvodom je veľká miera náhodnosti v tejto hre, kde pri veľkom štastí je možné vyhrať na 3. ťah (ak rátame prvotné nastavenie figúrok ako 1 ťah). Neskôr počas hry je vidieť ze sa šance vyrovnávajú. Ak by sme zmenili nastavenia na viac simulácii a zrušili naše obmedzenie na maximálny počet tahov tak by to vylepšilo naše výsledky ale to by zvýšilo aj požiadavky na výkon a hlavne na čas, čo by ovplivnilo hratelnosť.

Tu je graf, ktorý znázorňuje posledný ťah MCTS predtým ako vyhrá hru, počet výher sa rapídne zvýšil, skoro na 800, ale stále bol algoritmus schopný najsť spôsob ako by mohol protivník vyhrať (aj keď prehral iba 1 simuláciu z 1000).

Počas testovania bolo vidieť značný vpliv "štastia", náhodnosti na výsledok hry, preto pre lepšie výsledky by bolo treba vykonať ovela väčšie množstvo testov.



Obrázek 6.6: Štatistiky simulácii MCTS posledný ťah

Táto tabulka ukazuje výsledky kedy hral MCTS (500 simulácii) vs AlphaBeta(hĺbka 5). Testovali sme tiež aj vpliv rovnováhy medzi prieskumom a rozširovaním, ktorú poskytuje UCT4.3, úpravou cParam hodnoty v metóde bestChild.5.1

Čím je hodnota cParam väčšia tým má väčší vplyv prieskumom, pre dodržanie rovnováhy podľa UCT by mala byť hodnota nastavená na 1.

Ako je vydieť v tabulke, či už sme kládli väčší dôraz na prieskum alebo rozširovanie, MCTS mal najlepšie výsledky keď bola medzi nimi rovnováha.

	cParam = 1	**cParam = 0,5**	**cParam = 2**
Výhra	9	7	7
Prehra	1	3	3

Tabulka 6.2: Porovnanie výsledkov simulácií

Zhodnotenie

Aj keď by sa rozhodovanie nášho AlphaBeta algoritmu dalo zlepšiť úpravou vyhodnocovacej funkcie, stále by s obmedzeným množstvom času nedosahoval rovnakých výsledkov ako MCTS.

Vďaka tomuto testovaniu je možné vidieť, že aj keď MCTS nedokáže vždy najsť ten najlepší ťah, je pre hry s vysokým faktorom vetvenia lepšou voľbou ako AlphaBeta.

Kapitola 7

Záver

Tento projekt skúmal herné stratégie a umelú inteligenciu v stolových hrách. Implementovali sme tri algoritmy (minimax, alphabeta pruning, MCTS) a analyzovali sme ich a ich vplyv na herný zážitok. Algoritmus MCTS sa ukázal ako najvhodnejší pre našu hru, pretože ponúka optimálnu rovnováhu medzi výpočtovou efektivitou a presnosťou rozhodovania.

Projekt sa ukázal ako mimoriadne úspešný a priniesol hlboké a cenné poznatky v oblasti hernej AI. Dosiahnuté výsledky otvárajú fascinujúce dvere pre ďalší rozsiahly výskum a vývoj v tejto oblasti.

Projekt priniesol hlbšie pochopenie princípov a fungovania rôznych algoritmov AI v kontexte stolových hier. Implementácia a analýza troch algoritmov (minimax, alphabeta pruning a MCTS) poskytli cenné poznatky o ich silných a slabých stránkach, ako aj o ich vhodnosti pre rôzne typy hier.

Vďaka tomuto projektu sa herná AI posúva o krok vpred a otvára cestu pre rozvoj pútavejších, interaktívnejších a strategicky náročnejších hier. Algoritmy AI by sa mali dokázať učiť z herných dát a prispôsobovať sa rôznym typom protivníkov a herným štýlom. To by viedlo k realistickejším a pútavejším herným zážitkom. Princípy a algoritmy hernej AI by sa dali aplikovať aj v iných oblastiach, ako napríklad v robotike, autonómnom riadení a strojovom učení.

V budúcnosti by sa táto hra dala rozšíriť o oficiálne rozšírenia ktoré sú spomenuté v pravidlách hry[1] na ešte lepší a pútavejší herný zážitok. Taktiež by sa mohli pridať dalšie algoritmy, a upraviť tie stávajúce. Tiež by bolo dobré vytvoriť online verziu hry aby sa dalo získať väčšie množstvo testovacích dát na porovnávanie alebo napríklad na trénovanie pokročilejších AI.

Literatura

[1] *The Duke Rules - ©2013 Catalyst Game Labs* [Online]. Dostupné z: https://gamers-hq.de/media/pdf/18/13/e1/The-Duke-Rulebook-Lo-Res_FINAL.pdf. [Accessed on 20.11.2023].

[2] *Goal Oriented Action Planning* [Online]. Dostupné z: https://medium.com/@vedantchaudhari/goal-oriented-action-planning-34035ed40d0b. [Accessed on 30.4.2024].

[3] *An Introduction to Utility Theory* [Online]. Dostupné z: https://www.gameaipro.com/GameAIPro/GameAIPro_Chapter09_An_Introduction_to_Utility_Theory.pdf. [Accessed on 30.4.2024].

[4] *MCTS (English) - Updated 2017-11-19.svg* [Online]. Dostupné z: https://commons.wikimedia.org/wiki/File:MCTS_%28English%29_-_Updated_2017-11-19.svg. [Accessed on 27.4.2024].

[5] *Minimax* [Online]. Dostupné z: https://en.wikipedia.org/wiki/Minimax#/media/File:Minimax.svg. [Accessed on 27.4.2024].

[6] *The minimax algorithm and alpha-beta pruning.* [Online]. Dostupné z: https://ksvi.mff.cuni.cz/~dingle/2020-1/prog_2/notes_11.html. [Accessed on 27.4.2024].

[7] *Preparing for Artificial Intelligence* [Online]. Dostupné z: https://becominghuman.ai/preparing-for-artificial-intelligence-d0b644087537. [Accessed on 27.4.2024].

[8] *Rule-Based System* [Online]. Dostupné z: https://botpenguin.com/glossary/rule-based-system. [Accessed on 27.4.2024].

[9] *Turnstile state machine* [Online]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/6/68/Turnstile_state_machine.svg. [Accessed on 27.4.2024].

[10] Auer, P.; Cesa Bianchi, N. a Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning*. Springer, 2002, sv. 47, s. 235–256.

[11] Ben Ari, M. a Mondada, F. *Elements of Robotics*. Springer International Publishing, 2017. ISBN 9783319625331. Dostupné z: https://books.google.sk/books?id=itpCDwAAQBAJ.

[12] Buckley, J. a Eslami, E. *An Introduction to Fuzzy Logic and Fuzzy Sets*. Physica-Verlag HD, 2002. Advances in Intelligent and Soft Computing. ISBN 9783790814477. Dostupné z: https://books.google.sk/books?id=IzN93YlrKlwC.

[13] Colledanchise, M. a Ögren, P. *Behavior Trees in Robotics and AI: An Introduction*. CRC Press, 2018. Chapman & Hall/CRC Artificial Intelligence and Robotics Series. ISBN 9780429950896. Dostupné z: https://books.google.sk/books?id=QhBmDwAAQBAJ.

[14] Cormen, T.; Leiserson, C.; Rivest, R. a Stein, C. *Introduction To Algorithms*. MIT Press, 2001. Mit Electrical Engineering and Computer Science. ISBN 9780262032933. Dostupné z: https://books.google.sk/books?id=NLngYyWFl_YC.

[15] Homer. *Iliada*. 800-750 Bc. ISBN 9780140445923.

[16] Norman, J. M. *Grey Walter Constructs the First Electronic Autonomous Robots; the Origin of Social Robotics* [Online]. 2024. Dostupné z: https://historyofinformation.com/detail.php?id=668. [Accessed on 27.4.2024].

[17] O'Connor, J. J. a Robertson, E. F. *Blaise Pascal* [Online]. 1996. Dostupné z: https://mathshistory.st-andrews.ac.uk/Biographies/Pascal/. [Accessed on 27.4.2024].

[18] Russell, S. a Norvig, P. *Artificial Intelligence: A Modern Approach*. CreateSpace Independent Publishing Platform, 2016. ISBN 9781537600314. Dostupné z: https://books.google.sk/books?id=PQI7vgAACAAJ.

[19] Sabry, F. *Dartmouth Proposal: Fundamentals and Applications*. One Billion Knowledgeable, 2023. Artificial Intelligence. Dostupné z: https://books.google.sk/books?id=HVbJEAAAQBAJ.

[20] Sabry, F. *A Star: Fundamentals and Applications*. One Billion Knowledgeable, 2023. Artificial Intelligence. Dostupné z: https://books.google.sk/books?id=Tb3IEAAAQBAJ.

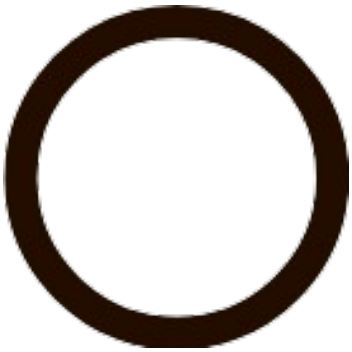
[21] Sutton, R. a Barto, A. *Reinforcement Learning, second edition: An Introduction*. MIT Press, 2018. Adaptive Computation and Machine Learning series. ISBN 9780262352703. Dostupné z: https://books.google.sk/books?id=uWV0DwAAQBAJ.

[22] Turing, A. *Computing Machinery and Intelligence*. Blackwell for the Mind Association, 1950. Mind: a quarterly review. Dostupné z: https://books.google.sk/books?id=Oq0rtAEACAAJ.

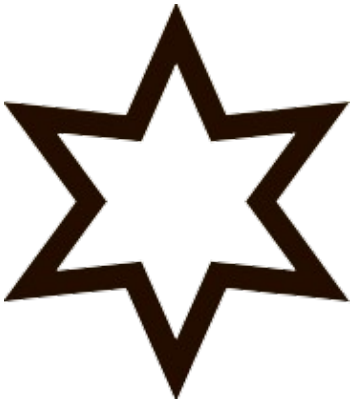
Příloha A

Ikony pohybu figúrok

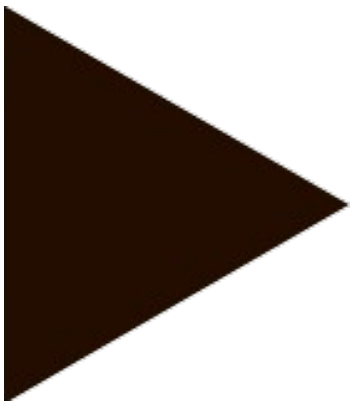
Obrázek A.1: Ikona *Move*[1]



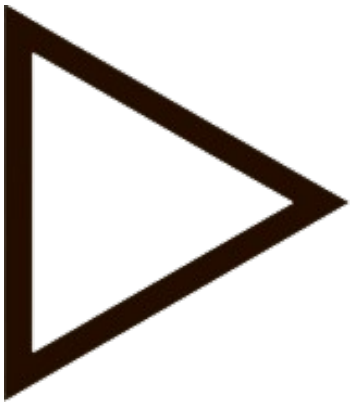
Obrázek A.2: Ikona *Jump*[1]



Obrázek A.3: Ikona *Strike*[1]



Obrázek A.4: Ikona *Slide*[1]



Obrázek A.5: Ikona *Jump Slide*[1]

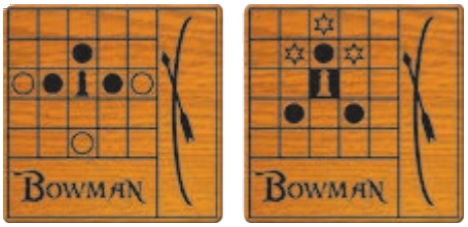
Obrázek A.6: Ikona *Command*[1]

Příloha B

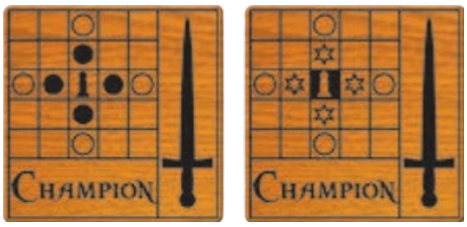
Profily pohybu figúrok

Obrázek B.1: Profil *Duke*[1]

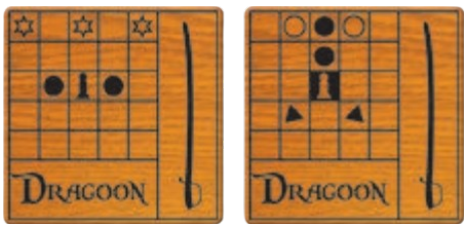
Obrázek B.2: Profil *Assassin*[1]



Obrázek B.3: Profil *Bowman*[1]

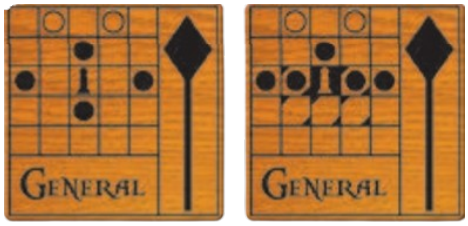


Obrázek B.4: Profil *Champion*[1]

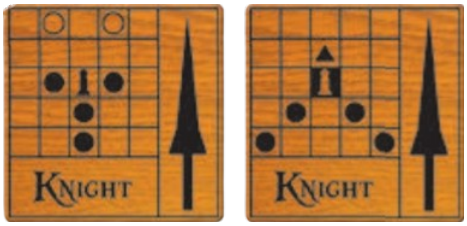


Obrázek B.5: Profil *Dragoon*[1]

Obrázek B.6: Profil *Footman*[1]

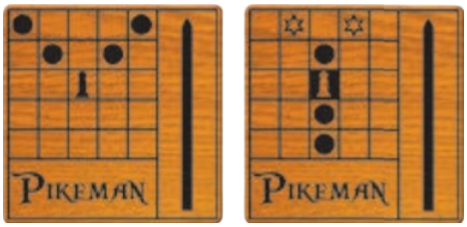


Obrázek B.7: Profil *General*[1]

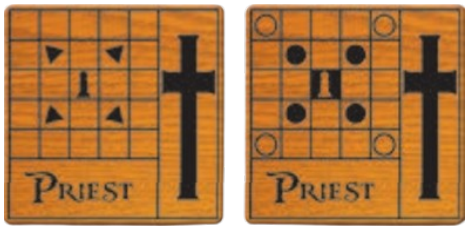


Obrázek B.8: Profil *Knight*[1]

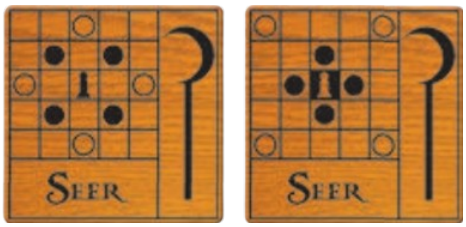
Obrázek B.9: Profil *Marshall*[1]



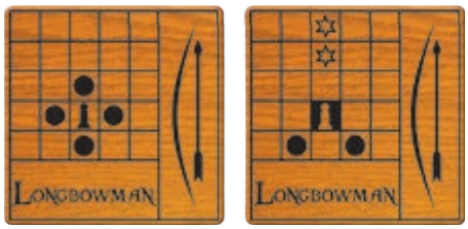
Obrázek B.10: Profil *Pikeman*[1]



Obrázek B.11: Profil *Priest*[1]



Obrázek B.12: Profil *Seer*[1]

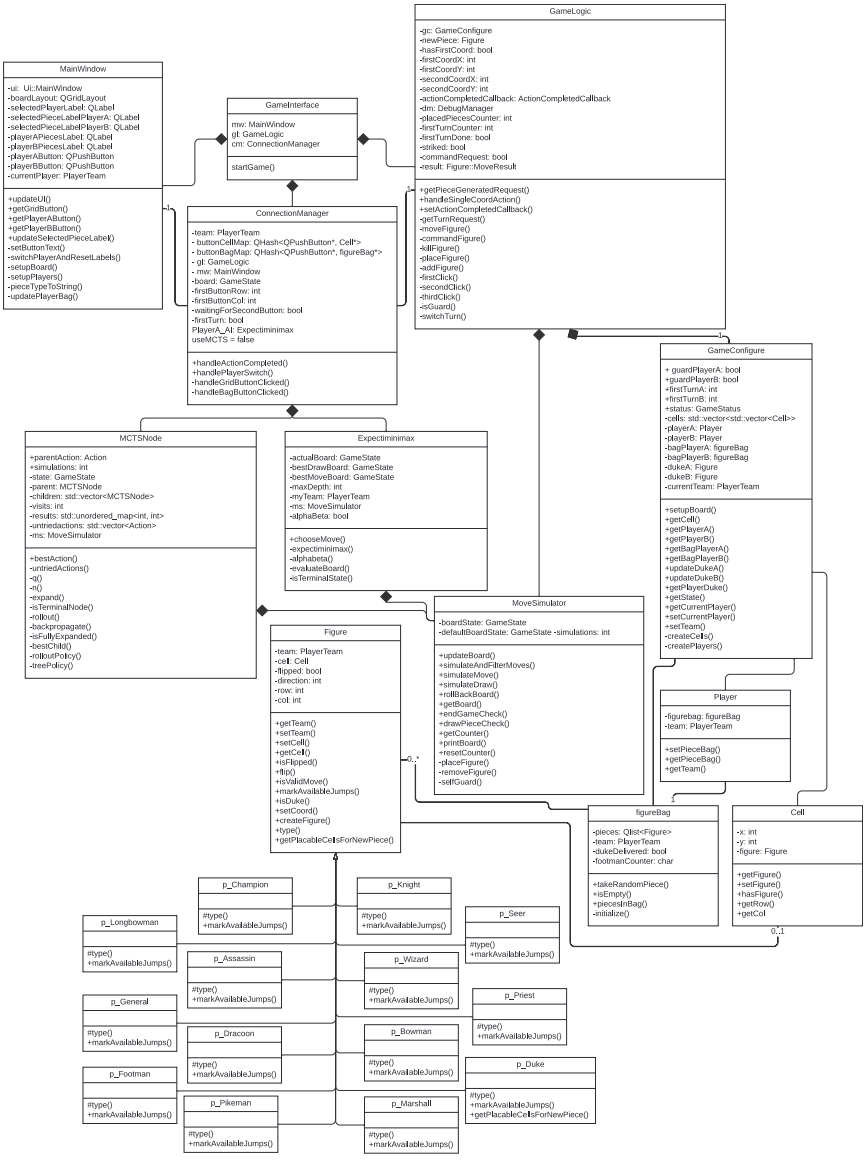


Obrázek B.13: Profil *Longbowman*[1]

Obrázek B.14: Profil *Wizard*[1]

Příloha C

UML Daigram tried



Obrázek C.1: UML Diagram tried

Příloha D

Obsah priloženého CD

```
/
├── BP Adrián Horváth.pdf ..... Elektronická verzia práce
├── LaTex/ ..... Zložka obsahujúca zdrojove súbory LaTex
├── UML.svg ..... UML diagram tried
├── The-Duke-Rulebook-Lo-Res_FINAL.pdf ..... Stiahnutá verzia pravidiel hry
├── Game/ ..... Zloška obsahujúca zdrojové a binárne subory k hre
│   ├── build-DukeGame-Desktop_Qt_6_6_1_MinGW_64_bit-Debug/ ... Binárne subory
│   └── DukeGame/ ..... Implementácia hry
│       ├── Algorithms/ ..... Zložka obsahujúca zdrojové subory k algoritmom
│       │   ├── ExpMinMax/ ..... Implementácia Minimax a Alphabeta algoritmu
│       │   └── MCTS/ ..... Implementácia MCTS algoritmu
│       └── Game/ ..... Zložka obsahujúca zdrojové súbory k hre
│           └── GameComponents/ ..... Implementácia herných komponentov
│               └── Figures/ ..... Zložka obsahujúca jednotlivé figúrky
```