Implementace deskové hry s prvky UI v jazyce Python

Ondřej Lukáš

Bakalářská práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Ondřej Lukáš**
Osobní číslo:	**A20311**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Implementace deskové hry s prvky UI v jazyce Python**
Téma práce anglicky:	**Board Game Implementation with AI Elements in Python**

Zásady pro vypracování

1. Vypracujte literární rešerši na dané téma.
2. Rámcově popište historii využití metod umělé inteligence pro simulaci stolních her.
3. Stručně představte několik algoritmů, které se pro tyto účely využívají, přičemž alespoň jeden vybraný, dále použitý v praktické části práce, popište podrobněji.
4. V jazyce Python realizujte vybranou stolní hru.
5. Implementuje v ní zvolený algoritmus umělé inteligence tak, aby byla schopna simulovat protihráče.
6. Vytvořenou hru řádně otestujte a kriticky zhodnoťte dosažené výsledky.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. ROZTOČIL, O. Evoluce umělé inteligence pro tahovou strategickou hru. Praha, 2020. Bakalářská práce. Matematicko-fyzikální fakulta, Univerzita Karlova.
2. KOLAŘÍK, J. Teorie her a její aplikace. Pardubice, 2019. Bakalářská práce. Fakulta elektrotechniky a informatiky, Univerzita Pardubice.
3. NEKVINDA, M. Umělá inteligence a herní strategie v deskové hře Carcassonne. Praha, 2018. Bakalářská práce. Matematicko-fyzikální fakulta, Univerzita Karlova.
4. KURENKOV, A. A 'Brief' History of Game AI Up To AlphaGo. In: Andrey Kurenkov [online]. 2016 [cit. 2022-11-24]. Dostupné z: https://www.andreykurenkov.com/writing/ai/a-brief-history-of-game-ai/
5. RABIN, S. Game AI Pro 3: collected wisdom of game AI professionals. Boca Raton: Taylor & Francis, 2017. ISBN 978-1498742580.
6. DAGRACA, M. Practical Game AI Programming. Birmingham: Packt Publishing, 2017. ISBN 978-1787122819.
7. KAPOOR, A., GULLI, A., PAL, S. Deep Learning with TensorFlow and Keras: Build and deploy supervised, unsupervised, deep, and reinforcement learning models. Birmingham: Packt Publishing, 2022. ISBN 978-1-80323-291-1.

Vedoucí bakalářské práce: **doc. Ing. František Gazdoš, Ph.D.**
Ústav řízení procesů

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Jméno, příjmení: Ondřej Lukáš
Název bakalářské/diplomové práce:
Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.

že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou

Ve Zlíně, dne 13.05.2024

Ondřej Lukáš v. r.
.....
podpis diplomanta

ABSTRAKT

Předkládaná bakalářská práce “Implementace deskové hry s prvky UI v jazyce Python” se v teoretické části zabývá rámcovým popisem historie využití metod umělé inteligence ve hrách s důrazem na ty deskové. Praktickou část této práce tvoří dva hlavní body: 1) implementace mírně upravených pravidel deskové hry “Minutová říše” v jazyce Python za využití knihovny Pygame, 2) vlastní implementace algoritmu, který je schopen simulovat kompetentního protivníka. Pro dosažení tohoto cíle bylo využito přístupu využívajícího Monte Carlo simulace, přičemž v závěru jsou kriticky zhodnoceny dosažené výsledky.

Klíčová slova: umělá inteligence, desková hra, Mnutová Říše, Python, Pygame, Monte Carlo

ABSTRACT

The theoretical part of the thesis "Board Game Implementation with AI elements in Python" deals with a general description of the history of the use of artificial intelligence methods in games with an emphasis on board games. The practical part of this thesis consists of two main points: 1) the implementation of slightly modified rules of the board game "Eight Minute Empire" in Python using the Pygame library, 2) the actual implementation of an algorithm that is able to simulate a competent opponent. A Monte Carlo simulation approach was used to achieve this goal, and the results are critically evaluated in the conclusion.

Key words: artificial intelligence, board game, Eight Minute Empire, Python, Pygame, Monte Carlo

Chtěl bych poděkovat vedoucímu této práce panu doc. Ing. Františkovi Gazdošovi za jeho ochotu, rady i trpělivost. Také bych chtěl poděkovat svým rodičům za veškerou podporu i za neustále vyptávání se “Jak jsi na tom s tou bakalářkou?” Dále děkuji kolegům Magdaléně Nevrlé a Bc. Machalu Martinovy za pomoc při testování. Nesmím ani zapomenout poděkovat kolegovi Patriku Lehnerovi, jehož schopnost stíhat věci na poslední chvíli mi dodávala naději, že též tuto bakalářskou práci stihnu zhotovit a odevzdat včas.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Prohlašuji, že při tvorbě této práce jsem použil/a nástroj generativního modelu AI ChatGPT 4,0; https://chatgpt.com za účelem asistence při tvorbě kódu. Po použití tohoto nástroje jsem provedl/a kontrolu obsahu a přebírám za něj plnou zodpovědnost.

OBSAH

ÚVOD

Moderní snaha vytvořit počítačem řízené protivníky, kteří by byli schopni vyzvat k hraní deskových her ty nejschopnější lidské hráče sahá ke konci druhé světové války. Nejznámější z těchto experimentů se soustředily spíše na tzv. “tradiční stolní hry.” Tedy takové, které předchází vynálezu moderních osobních počítačů často i o několik tiscíciletí. Hry jako šachy a jejich varianty, dáma, go, vrhcáby a další jim podobné byly historicky cílem mnohých pokusů o jejich “vyřešení”. V teoretické části této práce jsou popsány významné úspěchy (a do jisté míry i neúspěchy) některých z těchto počinů.

V dnešní době však na trhu existuje nespočet “moderních” stolních her, vydávaných komerčně různými vydavateli a vyvíjené profesionálními herními vývojáři. Tato bakalářské práce je snaha o elektronickou implementaci jedné z takových her za využití programovacího jazyka Python a knihovny Pygame, včetně kompetentních počítačem simulovaných protivníků. Zvolená implementovaná pravidla značně vychází z deskové hry ‘Minutová říše” prodávané v České republice pod záštitou společnosti MINDOK. Pro potřeby snadnější implementace byla některá pravidla mírně upravena.

V závěru této práce dojde jak k otestování kompetence simulovaného protivníka, tak i zjištění výpočetního času, který simulovaní protivníci vyžaduji k vykonání implementovaného algoritmu a celkovému zhodnocení realizované implementace včetně možných vylepšení.

I. TEORETICKÁ ČÁST

1 LITERÁRNÍ REŠERŠE

Moderní práce na umělé inteligenci v této oblasti začaly již před koncem druhé poloviny 20. století. Článek „Programming a Computer for Playing Chess“ (česky: Programování počítače pro hraní šachů), který v roce 1949 sepsal Cluade Shannon se dá považovat za počátek seriózního vývoje programů herní umělé inteligence: [1]

„Šachový stroj je pro začátek ideální, jelikož: (1) úloha je ostře definována, jak v oblasti povolených operací (tahů) tak v oblasti hlavního cíle (šachmat); (2) není natolik triviální ani natolik obtížná pro úspěšné vyřešení; (3) obecně se považuje, že šachy vyžadují schopnost 'myšlení' aby je šlo zručně hrát; vyřešení této úlohy nás buď přinutí uznat možnost mechanizovaného myšlení, nebo přehodnotit náš koncept 'myšlení'; (4) diskrétní struktura šachů se dobře hodí k digitální povaze moderních počítačů. (...) Je tedy jasné, že problém nespočívá v navrhování stroje, který hraje dokonalé šachy (což je zcela nepraktické), ani stroje, který hraje pouze šachy dle definovaných pravidel (což je triviální). Cílem by měla být zručná hra, snad srovnatelná s hrou dobrého lidského hráče.“

Krom stanovení těchto cílů a překážek, Shannonův článek též popsal dva typy přístupů k umělé inteligenci Umělá inteligence typu A, představovala přístup hrubou silou, ve kterém se program pokouší projít co nejvíce nadcházejících tahů a vybrat z nich ty nejlepší. Umělá inteligence typu B naopak představovala intuitivní přístup, založený na vytipování si relativně malého množství tahů, které jsou následně vyhodnoceny. [1]

Ve stejném období byly vyvinuty i různé „paper machines“, tedy programy, jejichž implementace existovala pouze na papíře. Nejznámější z těchto počinů je Turochamp (1948), za kterým stojí Alan Turing a David Champernowne. Přestože původní kód je již ztracen historii, šlo o své podstatě o minimax algoritmus s hloubkou dva.[2][3]

První program herní umělé inteligence, spustitelný na moderním elektronickém počítači naprogramoval pravděpodobně Arthur Samuel. Jeho program byl schopen simulovat protivníka ve hře dáma a zároveň představoval první sebevzdělávací umělou inteligenci.[4]

První kompletní program umělé inteligence pro hraní šachů byl představen v roce 1957. Za tvorbou této umělé inteligence stál IBM tým pod vedením Alexandra Bernsteina. Tento program byl příkladem implementace Shannonovy strategie typu B. [5]

Algoritmy založené na přístupech Monte Carlo se poprvé pro potřeby herní umělé inteligence začaly používat pro karetní hry, jako je např. poker. První nalezené použití u deskové hry je program Monte Carlo Go (1993, Bernd Brügmann).[6][15]

Na začátku 21. století došlo ke kombinaci Monte Carlo přístupu s klasickým procházením stromů. Tento přístup se nazývá Monte Carlo Tree Search a byl poprvé představen ve vědeckém článku „Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search.“, jehož autorem je Rémi Coulom.[7]

Co se moderní sebevzdělávací inteligence týče hraje na tomto poli důležitou roli společnost DeepMind, založena v roce 2010. V druhé dekádě 21. století se její zaměstnanci podíleli na výzkumu umělé inteligence pro klasickou deskovou hru go. Výsledkem jejich snažení byla uměláa inteligence AlphaGo. Po nich následovali umělé inteligence AlphaGo Zero, AlphaZero a MuZero.[8][9][10][11][12]

Co se studentských prací týče, zmíním hlavně bakalářskou práci Ondřeje Motlíčka z Matematicko-fyzikální fakulty Univerzity Karlovy na téma “Umělá inteligence pro hru Carcassonne – Objevitelé”. V dané práci jsou aplikovány a porovnány tři různé metody pro simulaci protivníka: Expectiminimax, Monte Carlo simulace a strojové učení.[13]

Podobná práce „Metody umělé inteligence pro hraní sběratelských her“ zpracovaná Patrikem Březinou z Fakulty elektronické ČVUT se zabývá tvorbou umělé inteligence pro počítačovou karetní hru „Hearthstone“. Tato práce využívá různé variace metody Monte Carlo Tree Search.[14]

Pro potřeby popisu historie vývoje programů umělé herní inteligence bylo využito zejména internetového článku „A 'Brief' History of Game AI Up To AlphaGo“, jehož autorem je Andrey Kurenkov, doktor ze Stanford University. Další podstatný zdroj v tomto ohledu je web „Chess Programming Wiki“, což je zdroj informací zabývající se programováním umělé inteligence pro hru šachy. Zakladatelem této wikipedie je Mark Lefler, absolvent Virginia Polytechnic Institute and State University a autor několika šachových programů. [15] [16][17][18]

2 HISTORIE VYUŽITÍ METOD UMĚLÉ INTELIGENCE PRO HRANÍ STOLNÍCH HER

Snahy o vytvoření umělého inteligentního protivníka předchází vynálezu moderních elektronických počítačů. Ať už jde o hoax v podobně tzv. „mechanického Turka“,[19] nebo opravdový automat *El Ajedrecista*.[20][21]

Tato kapitola je rozdělena na dvě hlavní části. První část se věnuje programům herní umělé inteligence pro klasické hry jako šachy, dáma či go, které předcházely mainstreamovému nasazení moderní sebevzdělávací inteligence založené na neuronových sítích. Herní umělé inteligence založené na sebevzdělávacích neuronových sítích jsou pak stručně popsány ve druhé části této kapitoly.

U některých podkapitol jsou stručně zmíněny různé specifické algoritmy. Ty jsou následně v detailu popsány ve třetí kapitole (VYBRANÉ PŘÍSTUPY K TVORBĚ HERNÍ UMĚLÉ INTELIGENCE).

2.1 Před příchodem moderní sebevzdělávací umělé inteligence

Základy herní sebevzdělávací umělé inteligence byly položeny již v roce 1956, když Arthur Samuel představil svůj program pro hraní dámy. Přesto více než půl století „dominovaly“ deskovým hrám umělé inteligence, u kterých sebevzdělávací funkce zcela chyběly, nebo byly velmi omezené.[15]

2.1.1 Šachy

První snahy vývoje umělé inteligence započaly právě u šachů. Již v roce 1912 španělský inženýr Leonardo Torres y Quevedo, stvořil velmi jednoduchý automat, který dokázal řešit dílčí úlohu matem, kdy se bílý hráč (jemuž zbyl jen král a věž), snaží dát mat člověkem ovládanému osamělému králi. Tento stroj, zvaný *El Ajedrecista* (doslova přeložitelné jako šachista), rozhodoval o pohybu bílých figurek na základě šesti podmínek, tykajících se pozice černého krále. Tento přístup nebyl nijak optimální, ale dokázal vždy dosáhnout matu do 50 tahů či méně. [20][21]

První kompletní program pro hru šachů zhotovil právě Turing a David Champernowne v roce 1948, dnes známý jako *Turochamp*. Turochamp byl v základě postaven na minimax algoritmu, který většinou procházel možné herní stavy do hloubky dva (tedy hráčův tah a protivníkova odpověď. Byl však schopen zajít i hlouběji v případě nucených pohybů a braní

figurek. Výhodnost pozice byla odvozena z faktorů jako je počet zbývajících figurek, mobilita figurek a krále, ochrana krále a ochrana figurek a tak dále. Turochamp však nebyl počítačový program, šlo o tzv. „paper machine“ (doslova papírový stroj), tedy program, který existoval pouze na papíře a vyžadoval člověka co provádí výpočty a simuluje logiku programu.[2][3][15]

Přestože Turochamp nebyl sám o sobě úspěšný (jediná zaznamenaná hra s programem Turochamp skončila jeho prohrou), principy, které byly při jeho tvorbě použité tvořily základ mnoha šachových programů, které následovaly.[2][3]

V roce 1949 Claude Shannon popsal dva přístupy k tvorbě umělé inteligence pro hru šachů. Dnes jsou tyto přístupy známé jako strategie typu A a strategie typu B. Strategie typu A v podstatě popisuje Turcohamp a jemu podobné programy, tedy minimax algoritmus, který prochází všechny možné stavy a vybírá ty nejlepší možné tahy. Strategie typu B, kterou Shannon preferoval, pak měla představovat program, který si je schopen vytipovat pár slibných tahů, které následně projde a vybere ten nejlepší. Cílem tohoto přístupu bylo ušetřit co nejvíce výpočetní kapacity a zároveň napodobit intuitivní myšlení člověka.[1][15]

První kompletní umělá inteligence schopná úspěšně hrát šachy byla dílem IBM týmu, jenž vedl Alexandr Bernstein, a využívala strategii typu B. Jeho programu, který byl představen v roce 1957, sice předcházely různé experimentální počiny, ty však byly schopny řešit jenom dílčí problémy (např. program *Mate-in-Two*, který sestrojil Dietrich Prinz v roce 1951), nebo využívaly podstatně zjednodušená pravidla hry (např. *Los Alamos Chess* pro MANIAC I, ten byl zase dílem týmu kolem von Neumanna).[3][15][22][23]

Bernsteinův program využíval minimax algoritmus s hloubkou čtyři a šířkou sedm. To znamená, že na každém uzlu si vytipoval sedm nejslibnějších tahů, a to na základě osmi různých priorit.[5][15]

V průběhu následujících let však docházelo jak k nárůstu výpočetních kapacit počítačů, tak k vývoji nových metod, jako například Alfa-Beta ořezávání, které dokázaly s výpočetní kapacitou pracovat ekonomičtěji. Tento rozvoj umožnil programům založených na strategii typu A překonat tehdejší programy typu B, které, podobně jako člověk, byly schopné přehlédnout optimální tahy.[15][24]

Deep Thought, program vyvinutý v roce 1986 týmem, který vedl americko-tchajwanský inženýr Feng-hsiung Hsu byl v principu vyvinut kolem strategie typu A. Byl to právě

šachový počítač *Deep Blue*, odvozený od programu Deep Thought, který v roce 1997 porazil tehdejšího šachového světového šampióna Garryho Kasparova.[15][22]

2.1.2 Dáma

Přesto že to byly šachy, které inspirovaly první snahy o tvorbu herní umělé inteligence, byla to dáma, jenž předehnala vývoj šachové inteligence ve dvou důležitých prvenstvích. První kompletní počítačový program, který dokázal odehrát plnou hru dámy byl stvořen v roce 1956, tedy dřív než šachový program Alexandra Bernsteina. Zároveň lze tento program považovat za první sebevzdělávací umělou inteligenci. Tvůrce tohoto programu, Arthur Samuel, zhodnotil svůj úspěch následovně:[4][15]

"Dva postupy strojového učení pro hraní dámy byly do jisté míry prozkoumány. Dostatečné úsilí bylo vynaloženo k ověření schopnosti naprogramovat počítač tak, aby dokázal hrát hru lépe než člověk, co program sepsal. Počítač je navíc schopen této úrovně dovednosti dosáhnout ve velmi krátkém čase (8 až 10 hodin strojového učení) a to čistě na základě znalosti pravidel hry, prvotnímu nasměrování a seznamu parametrů, o kterých se předpokládá, že mají něco společného se hrou, ale jejichž správná znaménka a relativní váhy jsou neznámé a nespecifikované. Principy strojového učení ověřené těmito experimenty jsou samozřejmě použitelné v mnoha dalších situacích."[4]

Dva postupy zmíněné na začátku této citace jsou:[4][15]

a) "rote-learning" - schopnost uložit si již prozkoumané stavy, aby je nebylo třeba příště zcela od začátku zhodnocovat

b) "learning-by-generalization" - upravení důležitosti různých herních parametrů, na základě porovnání předpokládané užitečnosti tahu během hry se zhodnocením reálné užitečnosti tahu po konci hry.

2.1.3 Go před AlphaGo

Jedním z důvodů, proč programy umělé inteligence pro dámu byly schopny předehnat ve svých schopnostech podobné programy pro šachy, je i rozdílná komplexnost dvou her. Zatímco co faktor větvení šachů (tedy kolik větví mají v průměru uzly v rozhodovacím stromě) se odhaduje na 35, faktor větvení dámy se pohybuje mezi šesti a sedmi.[15]

Na druhé straně tohoto extrému je starověká hra čínského původu zvaná *go*. Go se standardně hraje na herní ploše 19x19, ale existují také varianty v odlišných velikostech.

Během hry dva hráči (většinou bílý a černý) pokládají na herní plochu kameny své barvy. Cílem je dosáhnout kontroly nad herní plochou a zároveň zajmout co nejvíce nepřátelských kamenů. Toho se dosáhne obklíčením daného nepřátelského kamene.[15][25]

Faktor větvení go se pohybuje kolem 250, tedy asi sedmkrát více než u šachů. Krom toho, jedno sezení v průměru zabere asi 150 tahů. Partie šachů v průměru zabere jen něco kolem čtyřiceti tahů. Této situaci nenapomáhá nutnost komplexnějších evaluačních metod, které zabírají další výpočetní kapacitu.[15]

Z těchto důvodů trvalo až do roku 1968, než vznikla umělá inteligence schopna porážet nezkušené hráče. Program, který vytvořil Alfred Zobrist, nevyužíval rozhodovacího stromu. Místo toho umělá inteligence ohodnotila důležitost každého herního políčka na základě deterministický funkcí. Podobně neortodoxní přístup byl použit i v roce 1979, u umělé inteligence, kterou sestrojil Bruce Wilcox. Jeho program nahlížel na celou herní plochu jako na několik menších ploch, a následně využil databázi her odehraných zkušenými hráči pro generování vhodných tahů.[15][26][27]

Během devadesátých let se mezi nejúspěšnější umělé inteligence pro hraní go řadil program *Many Faces of Go*, za nímž stál David Fotland. Ten kombinoval tradiční minimax algoritmus s alfa-beta ořezáváním a různými ‚rule-based' technikami, které byly do té doby vyvinuty. Přesto herní zdatnost tohoto programu byla omezená ve srovnání se zkušenými hráči.[15][28]

Monte Carlo, přístup, který je detailněji popsaný v další kapitole, byl již nějakou dobu využívaný u karetních her jako je např. poker. Program umělé inteligence *Monte Carlo Go*, který sestrojil Bernd Brügmann v roce 1993, poprvé využil tuto techniku pro go. Přestože Monte Carlo Go nebylo nutně lepším hráčem než Many Faces of Go, dostatečně se mu dovedností přibližovalo, a to i přesto že jeho algoritmus byl podstatně jednodušší.[15][6]

Trvalo však nějakou dobu, než se Monte Carlo přístup ujal v širší komunitě. V první dekádě 21. století však vyšlo několik stěžejních vědeckých článků, zabývajících se touto tématikou. Výsledkem byly dva hlavní průlomy:[15]

1) Monte Carlo Tree Search metoda (MCTS), která kombinovala Monte Carlo metodu s tradičním průzkumem herního stromu. Poprvé použit u umělé inteligence *CrazyStone* (vyvinul Rémi Coulom).[7]
2) Technika UCT (Upper Confidence Bounds for Trees), která umožnila efektivní rozhodování během fáze selekce u MCTS. Za touto technikou stáli Levente Kocsis a

Csaba Szepesvári a byla poprvé použitá u umělé inteligence *MoGo*, kterou vyvinul Sylvain Gelly.[29][30]

Ke konci první dekády 21. století se herním inteligencím, založených na metodě Monte Carlo začalo dařit porážet profesionální hráče (byť s penalizacemi). Šlo o průlomové období na poli programů umělé inteligence pro Go.[15]

2.2 Moderní sebevzdělávací inteligence

Pro potřeby této bakalářské práce se za moderní sebevzdělávací inteligenci považují programy umělé inteligence využívající umělé neuronové sítě (zkráceně ANN – artificial neural network). Výzkum umělých neuronových sítí započal již ve čtyřicátých letech minulého století. Teprve ale až v 80. a 90. letech došlo k důležitým technologickým pokrokům, které umožnili jejich praktickou aplikaci. Je to právě v tomto období, kdy došlo k uplatnění ANN i u programů herní umělé inteligence.[15][31]

2.2.1 Vrhcáby

První umělé inteligence využívající umělých neuronové sítí nevznikly ani pro šachy ani pro Go, ale pro zcela jinou klasickou hru: vrhcáby. Podobně jako Go, vrhcáby mají příliš velký faktor větvení (>400), aby mohlo být efektivně využito klasických rozhodovacích stromů. To je způsobeno tím, že u vrhcábů se hází dvěma šestistěnnými kostkami, jejichž výsledek značně ovlivňuje proveditelné tahy.[15][32]

Hra vrhcáby, jako všechny doposud zmíněné hry, je určena pro dva hra hráče. Existují různé regionální varianty, ale princip zůstává stejný. Zjednodušeně: na herní ploše se nachází několik poliček, na kterých se může nacházet několik žetonů jednoho z hráčů. Cílem je po těchto políčkách přesunout jako první své žetony mimo herní plochu. Kostky určují o kolik políček se lze přesunout a existují různé limitace, kam lze žetony přesunout.[15][32]

Zdá se však, že oproti go měla hra vrhcáby jednu podstatnou výhodu, bylo totiž jednoduší vytvořit evaluační funkci. Již v roce 1980 se umělé inteligenci *BKG 9.8* podařilo porazit (skóre 7-1) tehdejšího světového šampióna, kterým byl Luigi Villa, byť BKG měl větší štěstí na hody kostkou. Program BKG, za kterým stál Hans J. Berliner, ještě nevyužíval neuronových sítí. Místo toho využíval evaluačního algoritmu, jehož parametry byly založeny na Berlinerových herních zkušenostech.[15][33]

První úspěšný ANN program pro hraní vrhcáb byl *Neurogammon*. Neurogammon byl vycvičen na sérii vzorových pozic, připravených zkušeným hráčem. Neurogammon 1.0 úspěšně zvítězil ve své kategorii (skóre 5-0) na první počítačové olympiádě v Londýně v roce 1989.[15][34]

Neurogammon byl překonán programem *TD-Gammon*, TD-Gammon byl sestrojen IBM týmem, který vedl Gerald Tesauro, a poprvé představen v roce 1992. Využíval techniky strojového učení založené na principu zpětnovazebného učení. TD-Gammon se naučil hrát vrhcáby na základě velkého množství her, které hrál proti sobě.[15][35][36]

2.2.2 Následující vývoj

Úspěch přístupu programu TD-Gammon se nepodařilo okamžitě replikovat u šachů i go. Programy NeuroChess(1995, Sebastian Thurn) a NeuroGo(1996, Markus Enzenberger) nebyly schopné překonat tehdejší programy využívající standardní přístupy.[15][31]

Dle článku z roku 2020, který sepsala Swasti Khurana:[36]

„Na rozdíl od šachů, [ve Vrhcábách] nikdy nedojde k remíze, a tedy hra hraná necvičenou sítí, která dělá náhodné tahy, vždy dosáhne úplného konce (i když to může trvat podstatně déle, než u hry mezi kompetentními hráči). Navíc, náhodnost kostek vede sebevzdělávací inteligenci do mnohem větší části prozkoumatelného prostoru, než která by byla navštívena u deterministické hry. “ [36]

Technologie a teorie v této době jednoduše ještě nedosáhla dostatečného stupně vývoje, aby mohl být ANN přístup pro šachy a go efektivně implementován. Na začátku první dekády 21. století tedy zůstával přístup umělých neuronových sítí k hraní stolních her upozaděný. Obecný výzkum však pokračoval dál. [15][24][36]

2.2.3 AlphaGo a nástupci

V roce 2014 Chrisopher Clark and Amos Storkey publikovali svůj výzkum, ve kterém se jim podařilo sestrojit umělou inteligenci typu ANN, které se podařilo porazit standardně programovanou umělou inteligenci GNU Go v 85 % případů, i když proti komplexnější programům využívající MCTS algoritmu stále zaostával. Neuronové síť byla naučena na základě 16,5 miliónu tahů odehraných expertními hráči a zároveň neměla žádné další znalosti týkají se hry. Nedlouho později, podobný výzkum, na kterém pracovalo několik zaměstnanců společnosti DeepMind a její mateřské společnosti Google, dosáhli ještě lepšího výsledku. Jejich umělá inteligence dokázala porazit GNU Go v 97 % případu. Obě práce pak

doporučily v budoucnu kombinovat MCTS algoritmus s umělými neuronovými sítěmi. Právě na tomto základu staví umělá inteligence *AlphaGo,* vyvinutá právě společností DeepMind. [8][15][37]

Program AlphaGo byl nejprve vycvičen na několika amatérských hrách, aby si osvojil lidský přístup ke hře. Poté program trénoval výhradně na hrách, které hrál sám proti sobě. Výsledkem byla umělá inteligence, která určovala strategie na základě intuice, podobně jako toho měla docílit strategie typu B, poprvé popsány Claudem Shannonem v roce 1949.

V roce 2015 program AlphaGo vyhrál proti tehdejšími evropskému šampiónovy (AlphaGo vs FanHui, 5-0). V roce 2016 pak proti AlphaGo prohrál tehdejší světový šampión, Lee Sedol (AlphaGo vs LeeSedol, 5-0). Během hry se prokázalo, že AlphaGo dokáže vynalézat své vlastní tahy a strategie. [9][15]

V roce 2017 byl vyvinut následovník AlphaGo, *AlphaGo Zero*. Hlavním rozdíl mezi AlphaGo a AlphaGo Zero je, že AlphaGo Zero nevyužíval žádné člověkem odehrané hry jako počáteční vstupy. Umělá inteligence AlphaGo Zero předčila původní AlphaGo během tří dnů od svého spuštění. [10]

AlphaZero, je generalizovanější program umělé inteligence, který byl představen v roce 2018. Ten je zbaven jakýchkoliv pomocných funkcí, a vybaven pouze pravidly hry. Program AlphaZero úspěšně předčil do té doby nejsilnější šachový program *Stockfish*, kterým nebyl ani jednou poražen. [11]

Program *MuZero*, představen v roce 2019, disponuje schopností hrát různorodé stolní hry, bez nutnosti předem znát jejich pravidla.[12]

V současnosti nejnovější počin společnosti DeepMind týkající se stolních her, je výzkum zabývající se tvorbou umělé inteligence schopné hrát hru Diplomacy, v jejímž jádru je vyjednávání (a následné zrazování) mezi hráči. Výsledkem je umělá schopná vytvářet smlouvy, provádět zrady a reagovat nedůvěrou vůči hráčům, kteří mají tendenci zrazovat častěji než ostatní.[38]

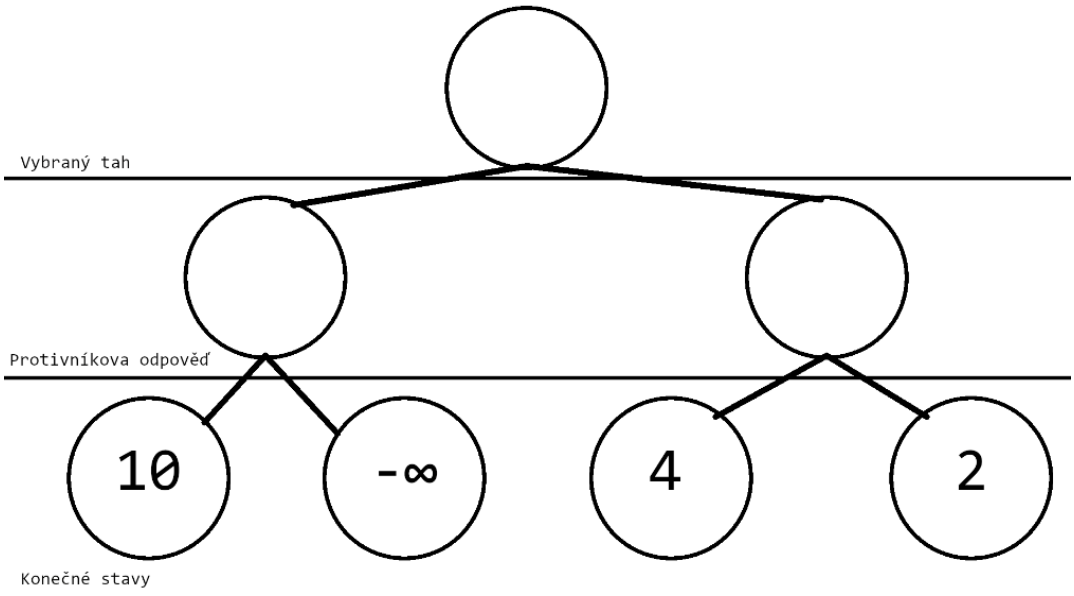
3 VYBRANÉ PŘÍSTUPY K TVORBĚ HERNÍ UMĚLÉ INTELIGENCE

Skrze historii herní umělé inteligence vznikly různorodé metody její aplikace. Dále uvedené metody představené v této části, byly vybrány na základě jejich rozšířenosti.

3.1 Minimax

Algoritmus minimax funguje na principu vytvoření stromu ze všech možných tahů, a protitahů, a z něj pak vybírat ty tahy, které by vedly k nejrychlejšímu a nejbezpečnějšímu vítězství. Teoreticky ideální implementace algoritmu by prošla všechny stavu od začátku, až do všech jejich konců. Realisticky je taková implementace nemožná, kvůli výpočetní kapacitě, která by byla vyžadována. Například u šachů se celkový počet všech možných dosažitelných herních stavů během jedné partie odhaduje na 4.8 * 10^44. Proto se u umělé inteligence postavené na základě algoritmu minimax omezuje hloubka, a v některých případech i šířka průzkumu.[15][39]

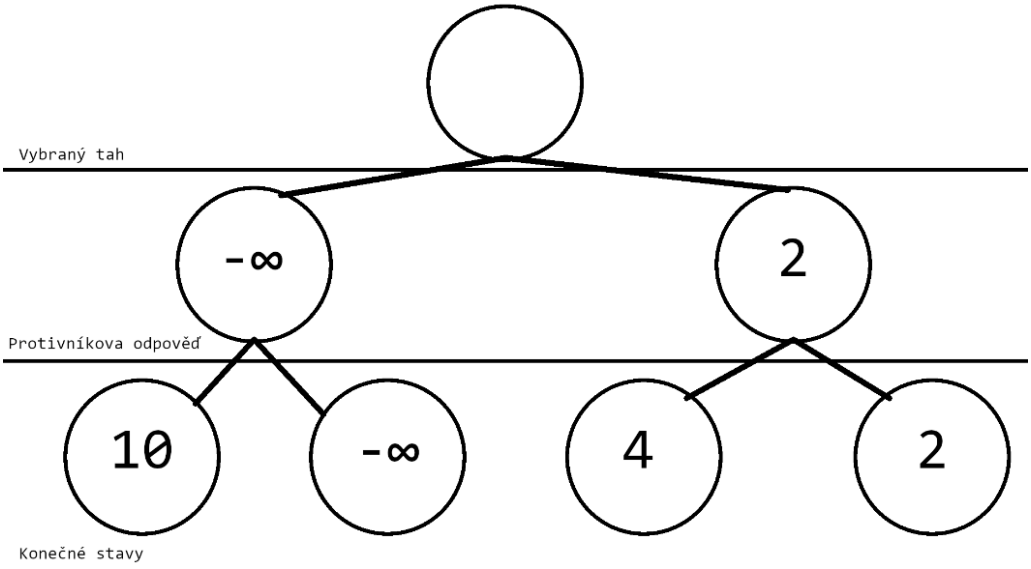
Následující obrázky (obrázek 1, obrázek 2, obrázek 3) ilustrují, jak funguje Minimax algoritmus s hloubkou.



Obrázek 1: Příklad algoritmu minimax část 1.

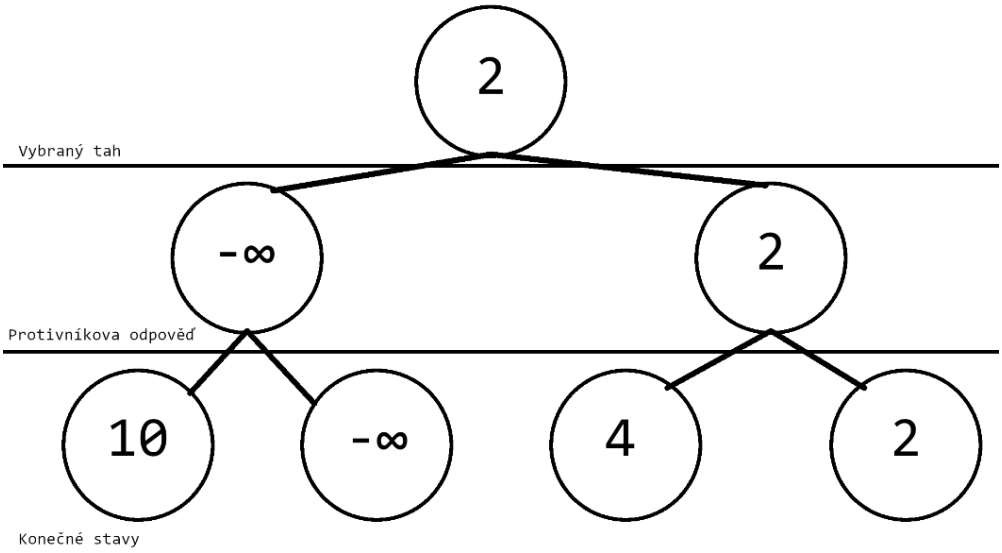
Předpokládejme situaci, kdy hráč má na výběr jeden ze dvou tahů a protivník má na každý tah dvě možné odpovědi. Na konci průzkumu do hloubky dvou tahů, se tedy herní plocha

může nacházet v jednom ze čtyř stavů, vyobrazených na ilustraci výše. Z leva doprava: hráč má skóre 10, hráč okamžitě prohrává, hráč má skóre 4 a hráč má skóre 2.



Obrázek 2: Příklad algoritmu minimax část 2.

Algoritmus předpokládá, že protivník při svém tahu vybere odpověď, která dostane hráče do co nejméně výhodné pozice. V tomto případě lze předpokládat, že na první hráčovu možnost tahu vybere protivník tah, který vede k okamžité prohře hráče. U druhé možnosti se předpokládá, že protivník si vybere tah, který dostane hráče do situace, kdy má jen 2 body skóre, místo 4 bodů.



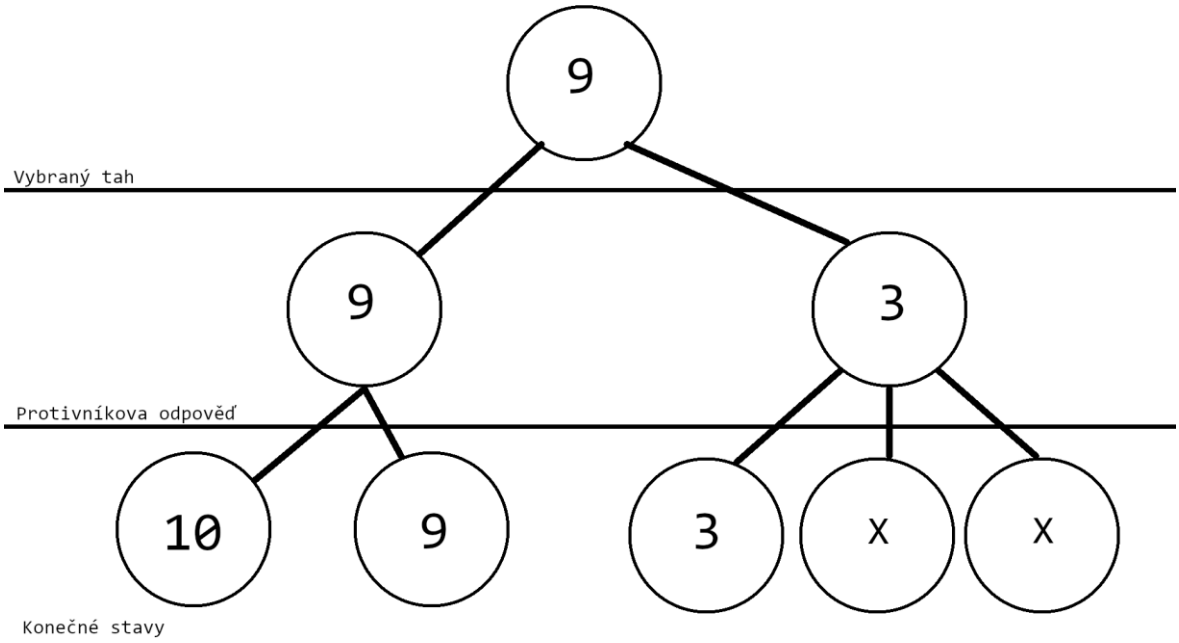
Obrázek 3: Příklad algoritmu minimax část 3.

V závěru algoritmus vybere možnost, kdy je hráč v nejvýhodnější pozici i po optimálních odpovědích protivníka. V tomto případě tedy stav, kdy má hráč skóre 2.

3.1.1 Alfa-Beta ořezávání

Skrze historii byly vyvinuty různé optimalizační techniky pro minimax algoritmus. Za tu nejvýznamnější lze považovat tzv. Alfa-Beta ořezávání. Při této technice algoritmus pracuje s dvěma pomocnými proměnnými: Alfa a Beta. Alfa je hodnota dosavadně nejlepšího tahu, který může hráč provést. Beta je nejnižší možná hodnota, kterou si může protivník vybrat v daném minimalizačním uzlu. Pokud je hodnota Beta stejná nebo nižší než Alfa, nepokračuje se v dalším průzkumu uzlu, a zbývající větvě jsou „ořezány“[41]

Jak vypadá implementace Alfa-Beta ořezávání můžeme vidět na obrázku 4: jelikož hodnota 3 (Beta), je nižší než 9 (Alfa), je jasné, že lepšího výsledku již nepůjde dosáhnout. Další stavy proto nejsou vůbec prozkoumány.

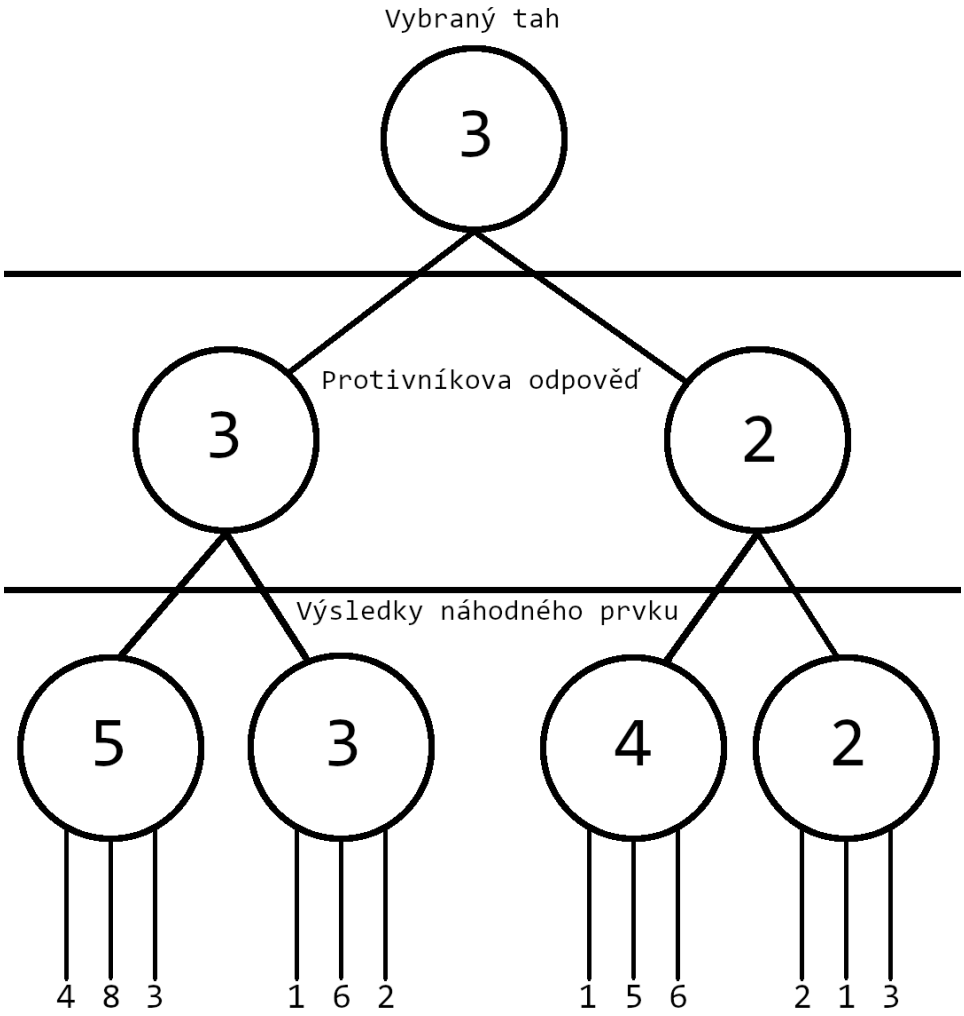


Obrázek 4: Alfa-Beta ořezávání

K ideální situaci dochází při odhalení nejlepšího tahu hned v prvním uzlu, to umožní ořezání největšího možného počtu větví. Z tohoto důvodu se společně s Alfa-Beta ořezáváním používají různé algoritmy pro vytipování dobrých tahů. Například u šachů útok pěšce proti figurce s vyšší hodnotu má velkou šanci být nejlepším tahem, a proto se hodí prozkoumat jako první.[41]

3.1.2 Expectiminimax

U her s prvkem náhody (např. hod kostkou), nelze standardní minimax přístup jednoduše aplikovat. Proto byl vytvořen algoritmus *expectiminimax*. Ten mimo fází maximalizační a minimalizační obsahuje i fázi náhodnou. V této fázi každý uzel zprůměruje možné výsledky náhodného prvku. Příkladem, jak expectiminimax algoritmus může fungovat je obrázek 5.[40]



Obrázek 5: Expectiminimax

3.1.3 Max^n algoritmus

Standardní minimax algoritmus není vhodné použít pro hru s více než dvěma hráči. Pokud by byl totiž každý protivník považován za minimalizační funkci, umělá inteligence by předpokládala, že všichni protivníci spolupracují za účelem ji společně porazit. Realisticky lze ale předpokládat, že každý hráč se bude spíše soustředit na maximalizaci své vlastní šance na vítězství. Na tomto předpokladu staví algoritmus *Max^n*.[42]

U algoritmu Max^n je zhodnocení stavu hry uloženo jako matice, která reprezentuje poměr výhod všech hráčů. Místo střídání minimalizační a maximalizační funkce se pak střídá maximalizační funkce každého hráče, která se snaží pro dané hráče vybírat ten nejvýhodnější poměr.[42]

3.2 Monte Carlo

Zatímco minimax algoritmus je čistě deterministický, a lze předpokládat, že minimax algoritmus ve stejné situaci, se stejnou hloubkou průzkumu a řídící se stejnými parametry, vždy dosáhne stejného výsledku (s výjimkou případných programů, které rozhodují v případě remízy stavů náhodně), Monte Carlo přístup využívá při svém rozhodování náhodný průzkum.[43][44]

Implementace Monte Carlo pro potřebu hraní deskových her by mohla vypadat následovně:

U každého možné tahu, který hráč může provést dojde k několika simulacím, při kterých jsou následující tahy hráče i protihráče vybírány zcela náhodně až do konce hry. Ke každému možnému tahu je následně přiřazeno ke kolika vítězstvím a porážkám daný tah vedl. Následně se vybere tah s nejpříznivějším poměrem vítězství a proher.

Nevýhoda přístupu Monte Carlo, oproti minimax algoritmu a od něj odvozených algoritmů, je neschopnost zaručit výběr optimálního tahu, dokáže ho pouze odhadnout. Prakticky se budou tahy vybrané Monte Carlo přístupem k optimálnímu tahu pouze přibližovat.

Krom tohoto nedostatku, má však tento přístup i jasné výhody, pro potřeby této práce ty nejdůležitější jsou:

1) není třeba vytvářet vlastní funkce pro zhodnocení stavu - vzhledem k tomu, že se výhodnost tahu určuje na základě toho, jak často vede k vítězství nebo prohře, není třeba vymýšlet deterministické funkce pro zhodnocení situace na herní ploše. To je obzvlášť výhodné u komplexních her, u kterých není jednoznačné, které parametry jsou důležité pro formulaci vhodné strategie. Zároveň využitím této metody lze vytvořit kompetentní umělou inteligenci i bez toho, aniž by sám programátor byl zkušený v pravidlech a hraní dané hry.
2) schopnost pracovat s náhodou – Monte Carlo přirozeně zahrnuje náhodné elementy hry do svých simulací. To umožňuje algoritmu efektivně strategizovat, a do jisté míry i zajímavě ‚riskovat' ve hrách, založených na náhodných prvcích, jako jsou karty, nebo kostky.

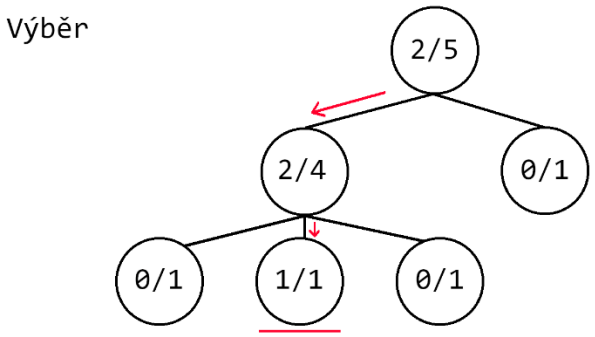
Hlavně z těchto důvodů byl algoritmus Monte Carlo zvolen jako vhodný přístup pro tuto bakalářskou práci.

3.2.1 Monte Carlo Tree Search

Monte Carlo Tree Search, zkráceně MCTS, je varianta Monte-Carlo algoritmu, která kombinuje Monte Carlo přístup s klasickým prohledáváním stromu. K tomuto užitku MCTS postupuje v následujících čtyřech krocích:[7][44]

1) VÝBĚR (SELEKCE):

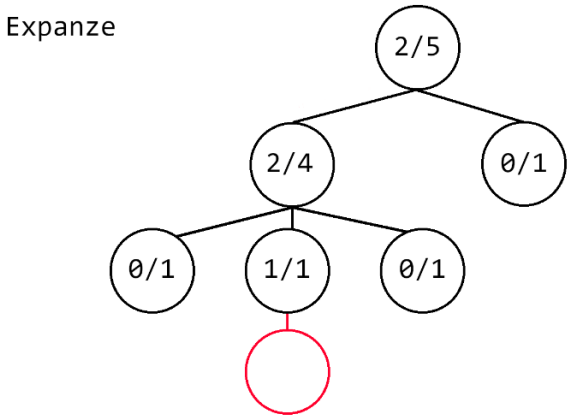
   Probíhá od kořene stromu směrem hlouběji, dokud se nedorazí k uzlu, jehož možnosti nebyly ještě zcela prozkoumány, nebo jde o končený uzel. Viz obrázek 6 (informace v uzlech: počet vítězství dosažených danou větví / kolikrát byla daná věta navštívena). Pro určení, které uzly prozkoumat se dnes užívá převážně UCT algoritmu, popsaného v další části.



Obrázek 6: MCTS - výběr

2) ROZŠÍŘENÍ (EXPANSION):

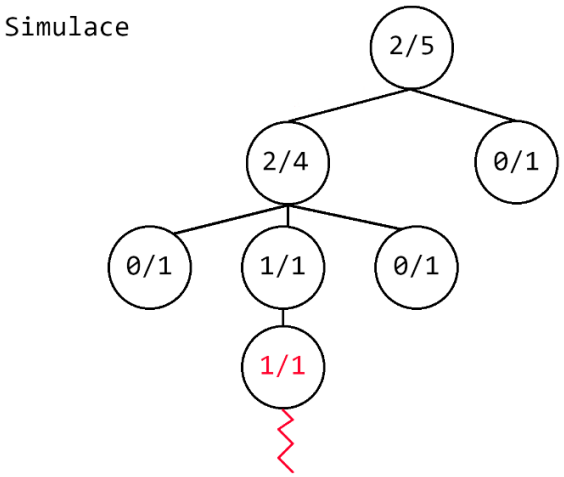
Pokud navštívený uzel není konečný stav, dojde k vytvoření nového uzlu, který reprezentuje jeden z možných tahů v daném stavu hry. Viz obrázek 7.



Obrázek 7: MCTS - expanze

3) SIMULACE (SIMULATION)

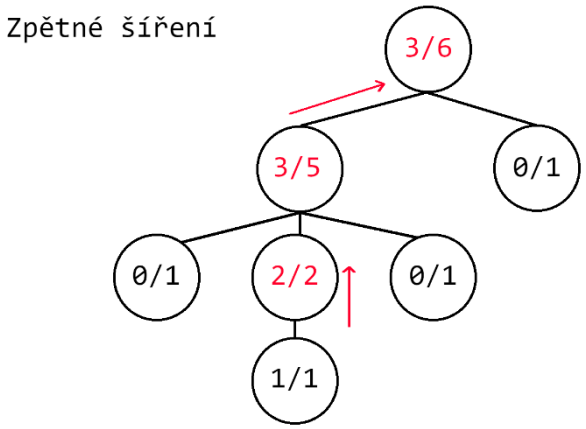
Z expandovaného uzlu se provede Monte Carlo simulace. Viz obrázek 8.



Obrázek 8: MCTS - simulace

4) ZPĚTNÉ ŠÍŘENÍ (BACKPROPAGATION):

   Aktualizuje se poměr výher a proher u všech uzlů vedoucích od současného uzlu, až ke kořenu stromu. Viz obrázek 9.



Obrázek 9: MCTS - zpětné šíření

3.2.2 Upper Confidence Bounds for Trees

Upper Confidence bounds applied to Trees (UCT) je algoritmus vyvinutý pro fázi rozšíření u algoritmu Monte Carlo Tree Search. Cílem bylo zajistit balanc mezi prozkoumání nových větví s rozvíjením větví, které se zdají slibné. UCT je řízen na základě vzorce UCB1. Zde je jeden ze zápisů:[29][44]

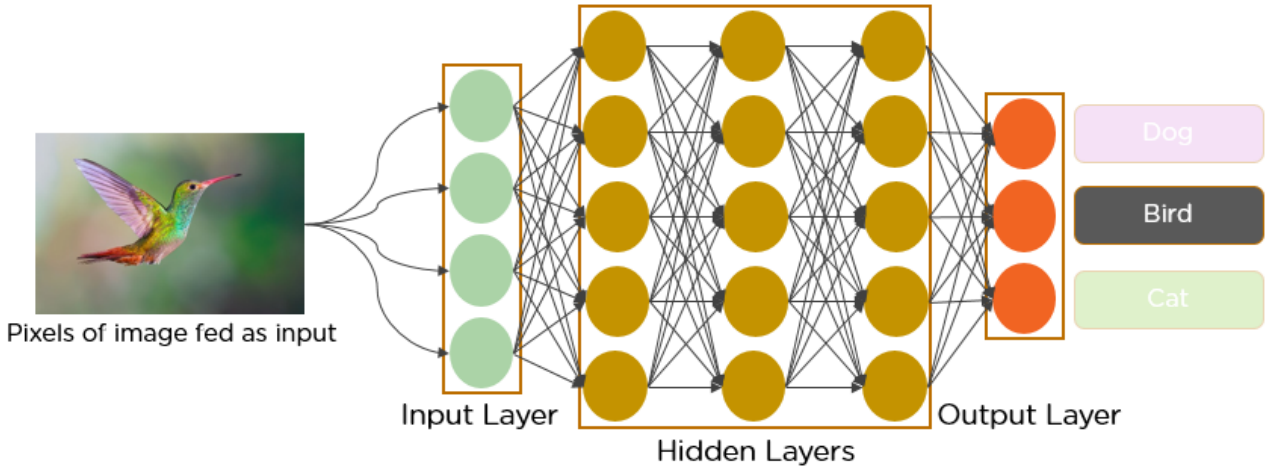
$$UCT_j = X_j + C * \sqrt{\frac{\ln(n)}{n_j}}$$

- UCT_j je hodnota uzlu j, která se používá k výběru uzlů pro další prozkoumání
- X_j je je výsledek (např. poměr vítězství/porážek) simulací procházejících uzel j
- n je celkový počet simulací, které prošly rodičem uzlu
- n_j je počet simulací, které prošly uzlem
- C je konstanta, jejíž nastavení určuje, zda se algoritmus bude přiklánět spíš k průzkumu nových větví, nebo rozvíjení slibných ale již do větší míry prozkoumaných větví (vyšší hodnoty dávají přednost průzkumu, zatím co nižší hodnoty dají přednost rozvíjení)

3.3 Umělé neuronové sítě a strojové učení

Umělé neuronové sítě se pokouší, jak jméno napovídá, napodobit strukturu lidského nervového systému. Základem umělé neuronové sítě, podobně jako člověka, jsou neurony. Umělý neuron lze implementovat jako objekt s určitým počtem vstupů, které mají nastavené určité váhy, matematickou funkcí, která sčítá dané vstupy, aktivační funkci, která nelineárně zpracuje daný součet a výstup, který předá tyto informace dále.[45]

Spojením několika neuronů do sebe vznikají neuronové sítě. Existují různé architektury neuronových sítí, každá určená pro jiné úkony. Ty se většinou skládají z jedné vstupní vrstvy, jedné výstupní vrstvy a jedné až několika skrytých vrstev, které vstupní a výstupní vrstvy propojují.[45]



Obrázek 10: Příklad struktury neuronové sítě [48]

U současných herních programů využívajících umělé inteligence (např. AlphaGo a jeho nástupci) se obzvlášť používají konvoluční neuronové sítě (Convolutional Neural Network – CNN). CNN jsou specializované pro zpracování dat s mřížkovou topologiím a díky tomu jsou schopné efektivně rozpoznávat vizuální vzory. [45]

Na obrázku 10 můžeme vidět příklad CNN sítě, která jako vstup bere barevné hodnoty pixelů zkoumaného obrázku a jako výstup vrací rozpoznané zvíře.

3.3.1 Strojové učení

Byť neuronové sítě nejsou jediným typem umělé inteligence využívající strojového učení, jsou svou strukturou vhodné pro strojové učení. U strojového učení program sám upravuje váhy mezi neurony na základě předchozích zkušeností. K strojovému učení dochází ve třech

hlavních podobách: učení s učitelem, učení bez učitele a zpětnovazební učení. Pro potřeby stolních her se využívá hlavně učení s učitelem:[46]

Učení s učitelem – umělé inteligenci jsou předány plné výstupy k porovnání, na základě kterých, se poté zdokonaluje. Příklad takového vstupu může být databáze tahů šachových šampiónů. [46]

Zpětnovazebné učení – umělá inteligence interaguje s prostředím a učí se na základě této interakce. Jednoduchý příklad je umělé inteligence, která hraje velké množství her a zlepšuje se na základě tahů, které vedou k vítězství. [46]

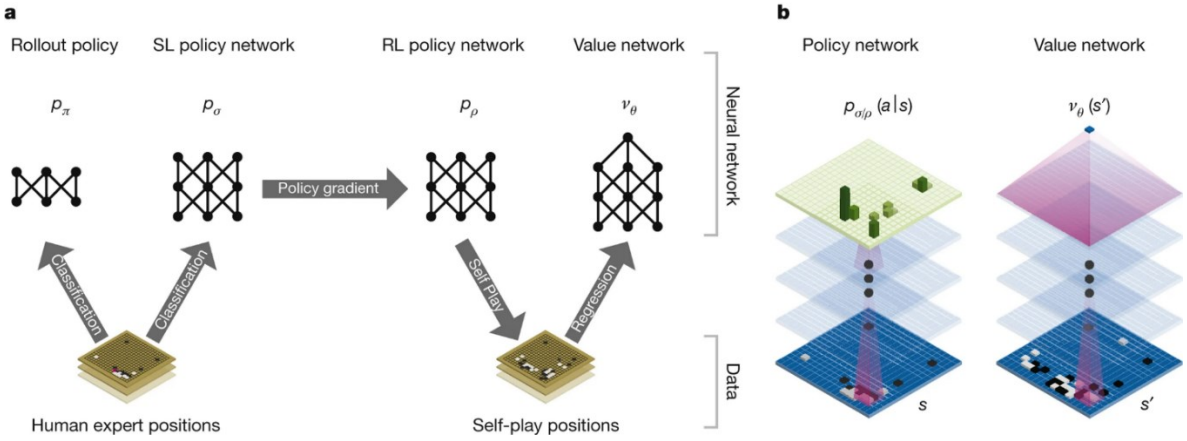
3.3.2 Přiklad sebevzdělávací neuronové sítě: AlphaGo

Zjednodušeně se umělá inteligence v AlphaGo skládá ze dvou sítí: „Policy Network“ a „Value Network“, jak lze vidět na obrázku 11. [8][47]

Policy Network je vytvořen využitím učení s učitelem a využíval jako vstupu databázi her zkušených hráčů. Na základě této databáze určovala umělá inteligence AlphaGo pravděpodobnost zahrání různých pozic v danem stavu zkušeným lidským hráčem. AlphaGo pak využívala tyto informace k ořezání tahů, které by tedy pravděpodobně nikdo nezahrál.[8][47]

Value Network pak slouží k vytvoření evaluační funkce na základě her odehraných proti sobě. Této evaluační funkce potom AlphaGo dále využívá pro navigaci v Monte Carlo Tree Search.[8][47]

Kombinací těchto dvou neuronových sítí do jednoho programu, vznikl program umělé inteligence, schopný přicházet s promyšlenými herními strategiemi v rozumném časovém rozmezí.[8][47]



Obrázek 11: Znázornění principu neuronových sítí použitých u programu AlphaGo.[8]

II. PRAKTICKÁ ČÁST

4 POUŽITÉ NÁSTROJE

V této kapitole jsou stručně představeny hlavní nástroje použité pro tuto bakalářskou práci. Jmenovitě: programovací jazyk Python a vybrané moduly z jeho standardní knihovny, komunitní Python knihovnu PyGame, integrované vývojové prostředí PyCharm, a expertní program ChatGPT.

4.1 Programovací jazyk Python

Programovací jazyk Python je vysokoúrovňový, interpretovací programovací jazyk, který v roce 1991 navrhnul nizozemský programátor Guido van Rossum. V současnosti je Python vyvíjen jako Open-Source projekt a patří mezí nejpopulárnější programovací jazyky.[49][50]

Python se vyznačuje svou čitelnosti a minimalistickým přístupem k syntaxi. Prvním zřetelným rozdílem oproti jiným programovacím jazykům je použití odsazení k organizaci kódu. V Pythonu je odsazení totiž nejen stylizační prvek, ale má syntaktický význam, který ovlivňuje běh programu.[49][50]

Python podporuje různé programovací přístupy, včetně objektově orientovaného, imperativního, a v omezené míře i funkcionálního programování. Samotný python je vybaven obsáhlou standardní knihovnou, která je dále podpořena velkým množstvím dále rozšiřujících knihoven, které jsou tvořeny a spravovány rozsáhlou komunitou vývojářů.[49][50]

Další vlastností, která dělá z Pythonu populární nástroj mezi programátory je jeho portabilita, tedy schopnost běžet na různých operačních systémech bez větších úprav.[49][50]

Samozřejmě, jsou i důvody kvůli kterým nemusí být Python k některým projektům vhodný. Mezi tři hlavní faktory patří:

a) rychlost provádění operací – jelikož Python patří mezi rodinu interpretovacích jazyků, provádí operace pomaleji než tzv. kompilované jazyky (jako např. C a C++).[50]

b) využití paměti – podobně, jako co se týče rychlosti, využívá Python více paměti než některé jiné programovací jazyky, obzvlášť více než výše zmíněně kompilované jazyky.[50]

c) jedna z vlastností Pythonu je tzv. ‘Global Interpreter Lock’ (GIL). Zjednodušeně, GIL je zámek, který zajišťuje, že v jednom okamžiku běží pouze jedno vlákno provádějící tzv. „bajtkód“ programovacího jazyka Python. To znamená, že standardní implementace jazyku Python nemůže plně využívat výhody multiprocesorových systémů pro paralelní zpracování.[51]

Python byl vybrán pro potřeby této bakalářské práce hlavně kvůli předchozím zkušenostem s jeho použitím. Verze Pythonu, použitá pro tuto bakalářskou prací je Python 3.10, vydaná 4. října 2021.

Python moduly použité v této bakalářské práce jsou:

a) random – tento modul slouží ke generování pseudonáhodných čísel v různém rozmezí. Součást standardní Python knihovny.[52]

b) enum – umožňuje vytvářet symbolická jména představující specifické hodnoty. Součást standardní Python knihovny.[52]

c) sys – umožňuje manipulovat s runtime prostředí programovacího jazyka Python. Součást standardní Python knihovny.[52]

d) math – obsahuje různé matematické funkce, včetně logaritmických funkcí, trigonometrických funkcí atd. Součást standardní Python knihovny.[52]

e) copy – obsahuje funkce pro kopírování objektů. Součást standardní Python knihovny.[52]

d) pygame – knihovna specializovaná pro tvorbu videoher. Nejde o součást standardní Python knihovny, více informací v následující části.[53]

4.2 PyGame

PyGame je open-source, komunitou tvořena knihovna, zaměřena na tvorbu videoher využívající dvoudimenzionální grafické prostředí a umožňuje i tvorbu videoher využívající trojdimenzionálního prostředí. Mezi hlavní funkcionality knihovny patří správa oken, vykreslování grafických objektů a zpracovávání uživatelských vstupů. Dále obsahuje funkce pro načítání a práci se zvukovými soubory jako např. MP3 soubory nebo WAV soubory.

PyGame je postavena na základě knihovny SDL (Simple DirectMedia Layer), která byla vytvořena za podobným účelem.

Knihovna PyGame je populární mezi amatérskými a nezávislými vývojáři. Pro potřeby této bakalářské práce byla využita verze PyGame 2.5.2, vydaná 18. září 2023.[53]

4.3 Vývojové prostředí PyCharm

PyCharm je integrované vývojové prostředí vydané firmou JetBrains, vyvinuté primárně pro programovací jazyk Python. Existují dvě hlavní verze:

Community Edition – Základní open-source verze, dostupná zdarma.

Professional Edition – Placená verze, která nabízí rozšířené funkce.

První veřejnosti dostupná verze vyšla v roce 2010.

Pro tuto bakalářskou práci byla využita verze PyCharm Community 2021.2.2.[54]

4.4 Expertní systém ChatGPT

ChatGPT je chatbot a expertní systém, který sám sebe popsal následovně: [55][56]

> "ChatGPT je jazykový model vyvinutý společností OpenAI, založený na architektuře GPT (Generative Pre-trained Transformer). Tento model byl speciálně navržen tak, aby rozuměl a generoval lidský jazyk, což mu umožňuje odpovídat na otázky, vést konverzace, psát texty a mnoho dalšího. ChatGPT byl trénován na rozsáhlé množině textových dat, aby mohl poskytnout informace, simulovat dialogy a reagovat na různé příkazy v rámci konverzace. Výsledkem je, že se může zdát užitečný, informovaný a často překvapivě přesný v generování odpovědí na nejrůznější témata."

ChatGPT byl využit pro tvorbu následujících metod:

endgame_handler - slouží k vyhodnocení vítěze hry

create_board - slouží k vytvoření hrací plochy

explore_and_assign - slouží k přiřazování regionů ke kontinentům

Zároveň služba ChatGPT značně pomohla vytvořit počáteční struktury, grafické třídy a byla často používána k debuggingu a k vytváření deepcopy definice.

Verze ChatGPT použitá při tvorbě kódu této bakalářské práce je ChatGPT 4.0.

5 IMPLEMENTOVANÁ DESKOVÁ HRA: MINUTOVÁ ŘÍŠE

Desková hra, která byla vybrána se nazývá "Minutová Říše". Tato hra byla vydána v roce 2013 vydavatelstvím MINDOK. Jejím autorem je designer amerického původu Ryan Laukat. Jed určena pro 2 až 5 hráčů. Herní doba se odhaduje na patnáct minut. Na internetovém portálu "Zatrolené hry" je následující popis:[57]

"Minutová říše je převahová hra, která se odehrává na hrací desce představující mapu smyšleného světa, tvořeného několika kontinenty rozdělenými na území. Hráči na těchto územích vytvářejí a přesunují své armády a snaží se během stanoveného počtu kol získat převahu na co nejvíce územích."

5.1 Jádro deskové hry

Hra se odehrává na herní ploše, která je rozdělena na několik regionů a kontinentů. Jeden z regionů je označen jako hlavní region. [58]

5.1.1 Příprava hry

Na hlavní provincii si každý z účastněných hráčů umístí tři armády. Armády jsou ve fyzické verzi reprezentovány kostičkami a jeden hráč může mít čtrnáct aktivních armád na herní ploše. Krom armád se můžou na herní ploše ocitnout i města, které jsou ve fyzické verzi označeny kulatými žetony.[58]

Připraví se balíček karet, počet celkových karet je určen počtem hráčů. Pokud hraje méně jak pět hráčů, hraje se s 37 kartami, v případě že hraje všech pět hráčů, přidá se do balíčku dalších pět karet.[58]

Z balíčku se vytáhne šest karet, které jsou následně rozloženy do jedné řady, a to v pořadí ve kterém byly z balíčku vytaženy.

Následně se každému hráči rozdají mince, počet mincí, které dostane každý hráč, záleží na počtu hráčů:[58]

Pět hráčů: 8 mincí

Čtyři hráči: 9 mincí

Tři hráči: 11 mincí

Dva hráči: 14 mincí

Mince se během hry nijak nezískávají, hráč musí pracovat s omezeným počtem, co dostal na začátku. [58]

5.1.2 Průběh herního kola

Herní kolo deskové hry "Minutová říše" začíná tím, že si hráč s řady šesti aktivních karet vybere jednu z nich. Cena je určena pozicí karty, první karta (ta která je úplně nalevo) stojí nula mincí, druhá a třetí karta stojí jednu minci, čtvrtá a pátá karta stojí dvě mince, šestá karta stojí tři mince. [58]

Po výběru jedné z karet, se všechny karty za ní posunou o pozici doleva a na poslední pozici, se vybere nová karta. Toto pravidlo v principu znamená, že čím déle je karta ve výběru, tím je levnější. [58]

Každá karta se skládá ze dvou hlavních složek, ze zboží, kterou reprezentuje a z akce, kterou hráči umožňuje. Zboží je jedním ze zdrojů vítězných bodů (viz. závěr a bodování). Zatímco akce umožňují hráči interagovat s herní plochou různými způsoby. Mezi akce patří: [58]

Přesun armád: karta umožňuje hráči přesunout armádu z jednoho regionu do sousedního. Kolikrát tak může být provedeno, je vyznačeno počtem kostiček na kartě.

Přesun armád i s případným přesunem přes moře: karta umožňuje hráči stejné akce jako karta přesun armád. K tomu navíc umožňuje i přesun po vyznačených námořních linií.

Postavit město: hráč může postavit město v kterémkoliv regionu, ve kterém se nacházejí jeho armády.

Umístění nových armád na herní plochu: Hráč může umístit na herní plochu nové armády své barvy. A to buď na startovní region, nebo na region, kde má hráč město. Kolik armád může být na herní plochu umístěno, je vyznačeno počtem kostiček na kartě.

Zničení armády: umožňuje z herní plochy odstranit armádu jednoho z protihráčů.

Některé karty umožňuji provést víc jak jednu akci. Takové karty využívají buď symbolu "/" k označení karet, u kterých si hráč musí vybrat jen jednu akci, nebo symbolu "+" v případě že hráč může provést obě akce během daného kola.

5.1.3 Závěr a bodování

Po určitém počtu kol, hra automaticky končí a dochází k bodování. Počet kol závisí na počtu hráčů. [58]

Dva hráči 13 kol

Tři hráči 10 kol

Čtyři hráči 8 kol

Pět hráčů 7 kol

Hráči dostávají jeden bod za každý region a každý kontinent, který ovládají. Aby šlo považovat region či kontinent za ovládaný, musí na něm hráč mít více armád než kterýkoliv z jeho protivníků (město se počítá jako jedna armáda).[58]

Zároveň hráči dostávají body za zboží, které získaly z nasbíraných karet. Na základě kvantity zboží hráči můžou dosáhnout jedné ze čtyř bodovacích úrovní. Za první tři úrovně hráči dostanou po jednom bodě, a za čtvrtou úroveň dostanou další dva body navíc. Viz tabulka 1. Některé karty mohou dávat dvojnásobek zboží. [58]

Zároveň existuje karty s žolíkem, které se počítají jako zboží dle hráčova výběru. [58]

Tabulka 1: Seznam zboží a kvantity nutné k dosažení bodových úrovní

Zboží	1. Úroveň (+1 bod)	2. Úroveň (+1 bod)	3. Úroveň (+1 bod)	4. Úroveň (+2 body)
Jídlo	3	5	7	8+
Dřevo	2	4	5	6+
Uhlí	2	3	4	5+
Drahokamy	1	2	3	4+
Železo	2	4	6	7+

V případě remízy, vítězí hráč, kterému zbývá nejvíce mincí. Pokud remíza stále přetrvává, vyhrává hráč, který kontroluje nejvíce provincií. Pokud i nadále panuje stav remízy, vyhrává hráč s největším počtem armád. [58]

5.2 Úprava pravidel pro potřeby vlastní programové implementace

Některé elementy hry jsou implementovány jinak, než jak jsou popsány ve standardních pravidlech hry. Tyto změny byly provedeny za účelem zjednodušení implementace. Testování neodhalilo, že by tyto změny vedly k výrazným změnám ve strategiích lidských hráčů.

5.2.1 Určení pořadí hráčů

Ve standardní verzi se pořadí hráčů vybírá dražbou. Tato počítačová verze používá volitelné pravidlo, kdy je pořadí hráčů vybráno zcela náhodně.

5.2.2 Tvar provincií

V současné implementaci jsou všechny provincie čtvercové políčka stejné velikosti. Nejpodstatnější dopad na hru je, že každé provincie může mít maximálně čtyři sousedy.

5.2.3 Námořní trasy

V současné implementaci se nevyskytují vyznačené námořní trasy, místo toho přesun přes moře umožňuje přesun přes jeden vodní region.

Ostatní pravidla zůstávají v nezměněné podobě.

6 POPIS VLASTNÍ IMPLEMENTACE

Vysvětlení termínů:

Fáze – Jeden ze stavů, ve kterém se hra může nacházet, v dané fázi můžou být provedeny jen určité tahy

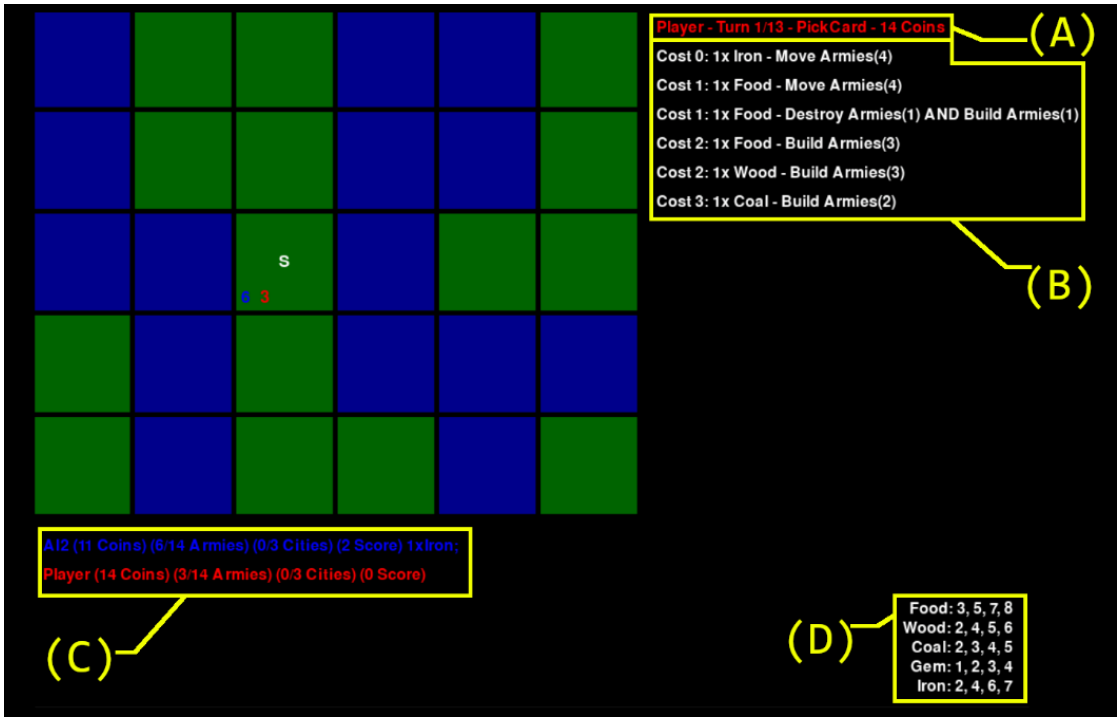
Tah – jedna akce provedená hráčem (postavení armády, přesun armády, výběr karty atd.)

Kolo – sekvence tahů, od fáze výběr

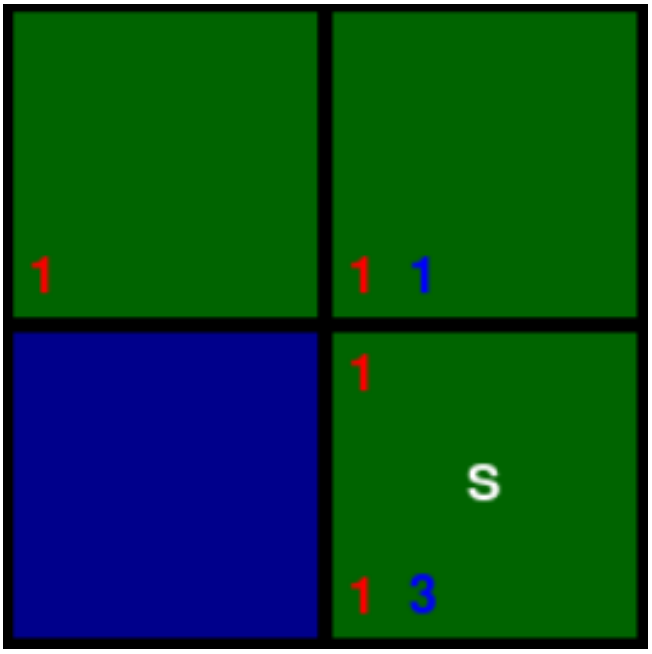
Manévr – omezení počtu tahů v dané fázi

6.1 Hraní hry

K ovládání hry se primárně používá myš. Tou se vybírají karty, regiony, a schopnosti. Klávesa mezerník slouží k ukončení tahu. Na obrázku 12 je vysvětleno uživatelské rozhraní: (A) nadpis pravého ovládacího panelu, obsahuje informace o současném hráči, současné fázi, počtu kol, počtu mincí/manévrů; (B) pravý ovládací panel; (C) seznám hráčů a informace o počtu armád, měst, mincí, skóre a získaném zboží; (D) ukazatel bodových úrovní zboží.



Obrázek 12: Grafické rozhraní hry



Obrázek 13: Znázornění jednotlivých políček hry – detail

Na obrázku 13 je dále v detailu vysvětleno grafické znázornění herní plochy: zelená políčka představují regiony, modré políčka představují moře oddělující kontinenty, čísla v horní části políčka značí počet měst daného hráče, čísla v dolní části políčka značí počet armád daného hráče, písmeno ‚S' značí startovní region.

6.1.1 Herní fáze

Během hry, se program může nacházet v jedné z deseti fází, které určují akce, jenž, jsou zrovna proveditelné.

PickCard – fáze výběru karty

PickAbilityAND – fáze výběru schopností, lze použít obě schopnosti v daném kole

PickAbilityOR – fáze výběru schopností, lze použit jen jednu ze schopností v daném kole

BuildArmy – umožňuje umístit armádu na herní plochu

BuildCity – umožňuje umístit město na herní plochu

MoveArmy – umožňuje přesun armád po souši

SailArmy – umožňuje přesun armád jak po souši, tak i přes jedno vodní políčko

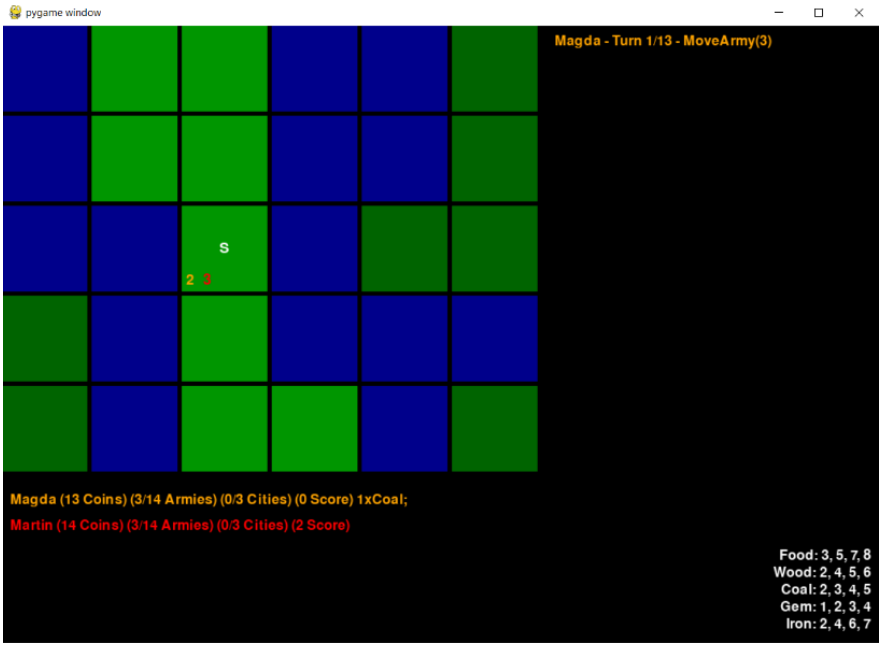
DestroyArmy – umožňuje odstranit armádu jednoho z protivníků

JokerAssignment – slouží k přiřazení žolíků k zboží na konci hry

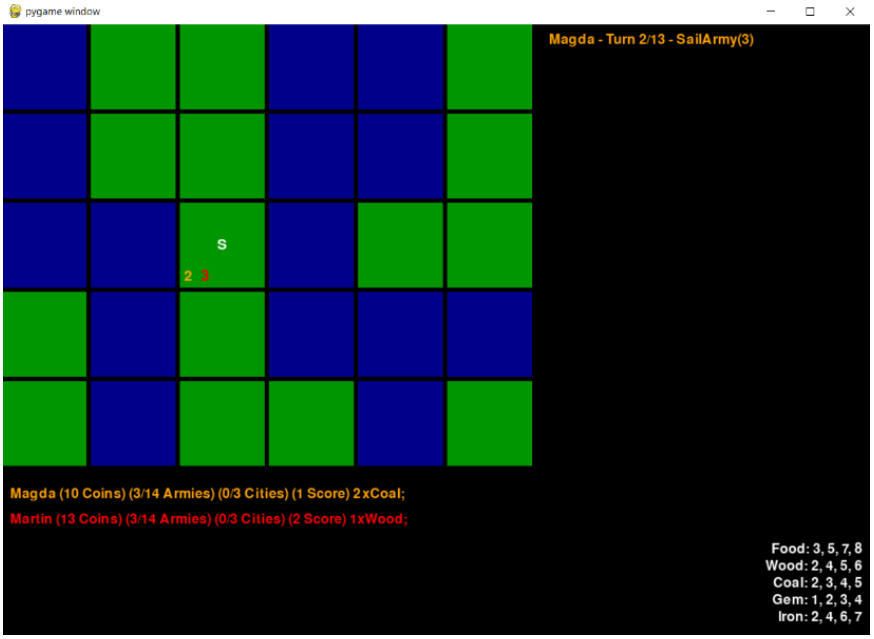
EndGame – tato fáze značí konec hry, během ní dojde k vybrání a zobrazení vítěze

6.1.2 Přesun armád

K přesunu armád může dojít buď ve fázi MoveArmy nebo SailArmy. Prvním kliknutím si hráč vybere aktivní region, ze kterého následně vybírá armády. Opakovaným kliknutí na aktivní region může hráč vybrat více armád. Vybrané armády pak může hráč přesunout kliknutím na jeden ze zvýrazněných regionů. Jak vypadá použití schopnosti MoveArmy je znázorněno na obrázku 14, použití schopnosti SailArmy je znázorněno na obrázku 15. V obou případech je vybrána jedna armáda a aktivní hráč má k dispozici tři manévry.



Obrázek 14: Znázornění jednotlivých políček hry – detail



Obrázek 15: Ukázka přesunu armád - SailArmy

6.1.3 Ukončení tahu

Jak již bylo řečeno, tah lze ukončit mezerníkem. Pokud se hra nachází v jedné z fází představující schopnost karty, a hráči nezbývají k použití žádné další schopnosti v případě zahrání karty se dvěma schopnostmi spojené spojkou AND, dochází po ukončení tahu i k ukončení kola a na řadě je další hráč.

6.2 Popis implementace programu

Vytvořený program je popsán v následujícím pořadí: inicializace programu, cyklus programu, popsání herní logiky, simulace protivníka a nakonec fungování grafické stránky programu.

6.2.1 Inicializace

Na úplném začátku dojde k inicializaci funkcionality modulu pygame. Následuje vytvoření několika globálních proměnných. Jmenovitě: *SCREEN_WIDTH* a *SCREEN_HEIGHT*, určující velikost obrazovky při spuštění; slovník *COLORS*, který obsahuje RGB zápis různých barev použitých během hry; a proměnné typu integer *STARTING_ARMIES*, *MAX_CITIES* a *MAX_CITIES*, což jsou konstanty vycházející z pravidel hry; *AI_ACTION_EVENT* je vlastní pygame event, jehož použití je vysvětleno v následující části.

V druhé části inicializace, která začne po definici všech tříd, dochází k vytváření objektů. První vytvořené objekty vychází ze třídy *Player*, představující hráče, tyto objekty jsou následně uloženy do listu *Players*. Následuje slovník *default_goods* obsahující objekty *Good* a slovník *ABILITIES* obsahující objekty vycházející dědící třídu *Ability*. Podobně jsou vytvořeny i listy *bonuscards* a *defaultcards*. První z těchto dvou listů obsahuje objekty typu *Card*, které se objeví v každé variaci hry. Druhý z těchto obsahuje objektu typu *Card*, které se objeví jen v případě, že hraje všech pět hráčů.

Listy *default_goods*, *bonuscards* a *defaultcards* jsou následně použity k vytvoření objektu *TheDeck*, který uchovává informace týkající se používaných karet. Následně dojde k vytvoření objektů *TheTileManager* (slouží k provádění některých komplexnější herních operací, např. pohybu armád po herní ploše) a *TheAIManager* (má na starosti simulované protivníky).

Dále je vytvořen objekt *TheGame*, který má na starosti fungování většiny herní logiky. Jedna z metod třídy, ze které je tento objekt vytvářen, slouží i k vytvoření herní plochy. K tomu je potřeba list proměnných typu string, který udává, jak bude herní plocha vypadat. Obrázek 16 představuje, jak takový list vypadá. W zde představuje vodní políčka, G představuje klasické regiony a S představuje startovní region.

```
board_layout = [
    "WGGWWG",
    "WGGWWG",
    "WWSWGG",
    "GWGWWW",
    "GWGGWG"
]
```

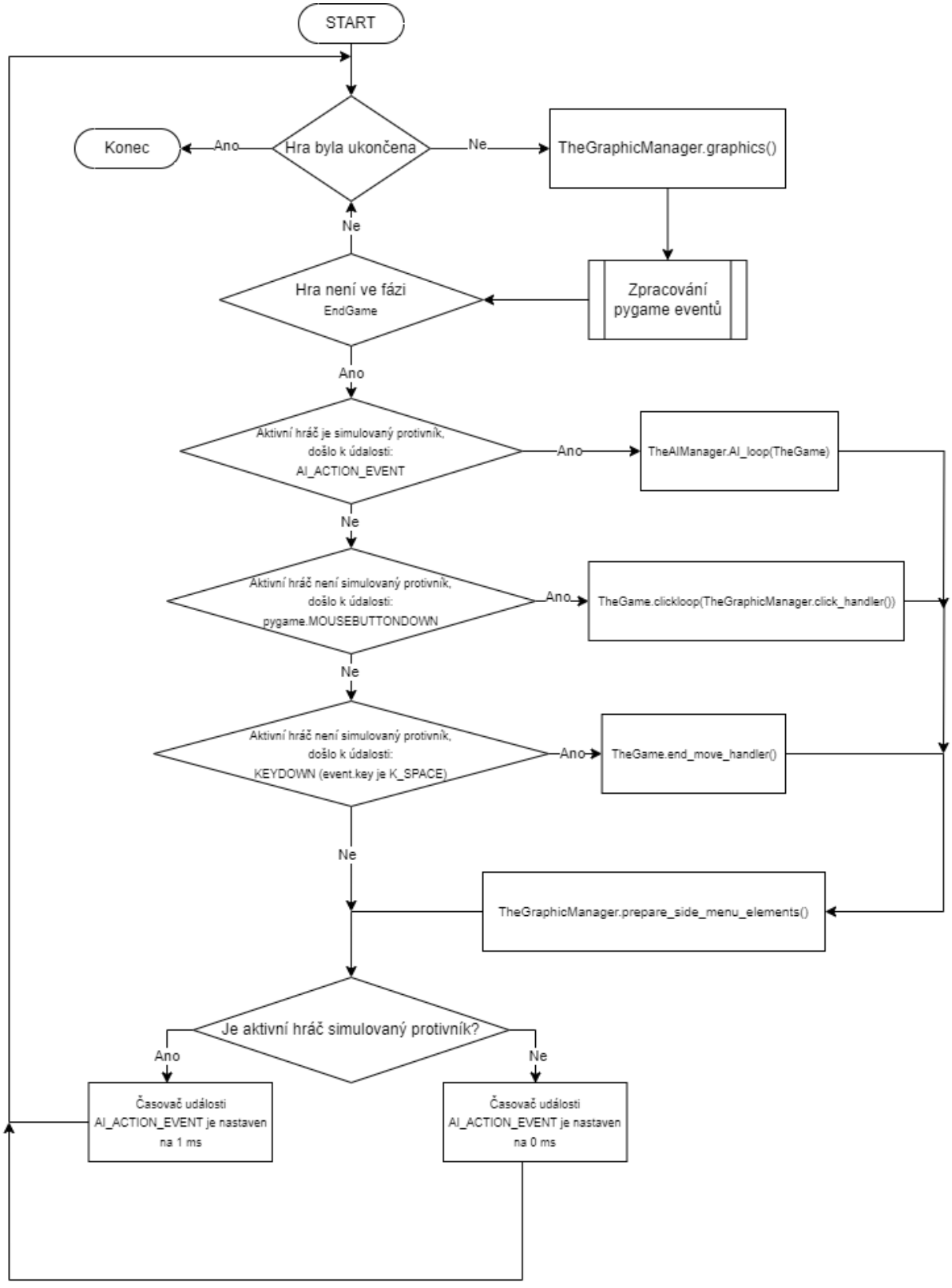
Obrázek 16: Board layout

Jako poslední věc, v případě, že první hráč je simulovaný protivník, dojde k nastavení hodnoty *AI_ACTION_EVENT* na 1. Což znamená, že k tomuto eventu bude docházet každou milisekundu.

6.2.2 Hlavní cyklus

Obrázek 17 vizualizuje hlavní cyklus, který slouží ke spuštění jedné ze čtyř metod. Pokud má hrát simulovaný protivník, dojde k zavolání metody *AI_loop* objektu *TheGraphicManager*. Pokud hraje reálný hráč a dojde ke kliknutí, je zavolána metoda *clickloop* objektu *TheGame*. Pokud hraje reálný hráč a dojde ke zmáčknutí mezerníku, je zavolána metoda *end_move_handler* objektu *TheGame*. Každá tato metoda je podrobněji popsána v patřičné části této kapitoly.

Cyklus při svém rozhodování využívá Pygame eventů. Eventy představují různé akce, které jsou při splnění daných podmínek zařazeny do fronty. Z této fronty jsou pak postupně volány. V případě že je očekávaný čas časovaného eventu nastaven na 0, daný event je neaktivní a nikdy k němu nedojde.



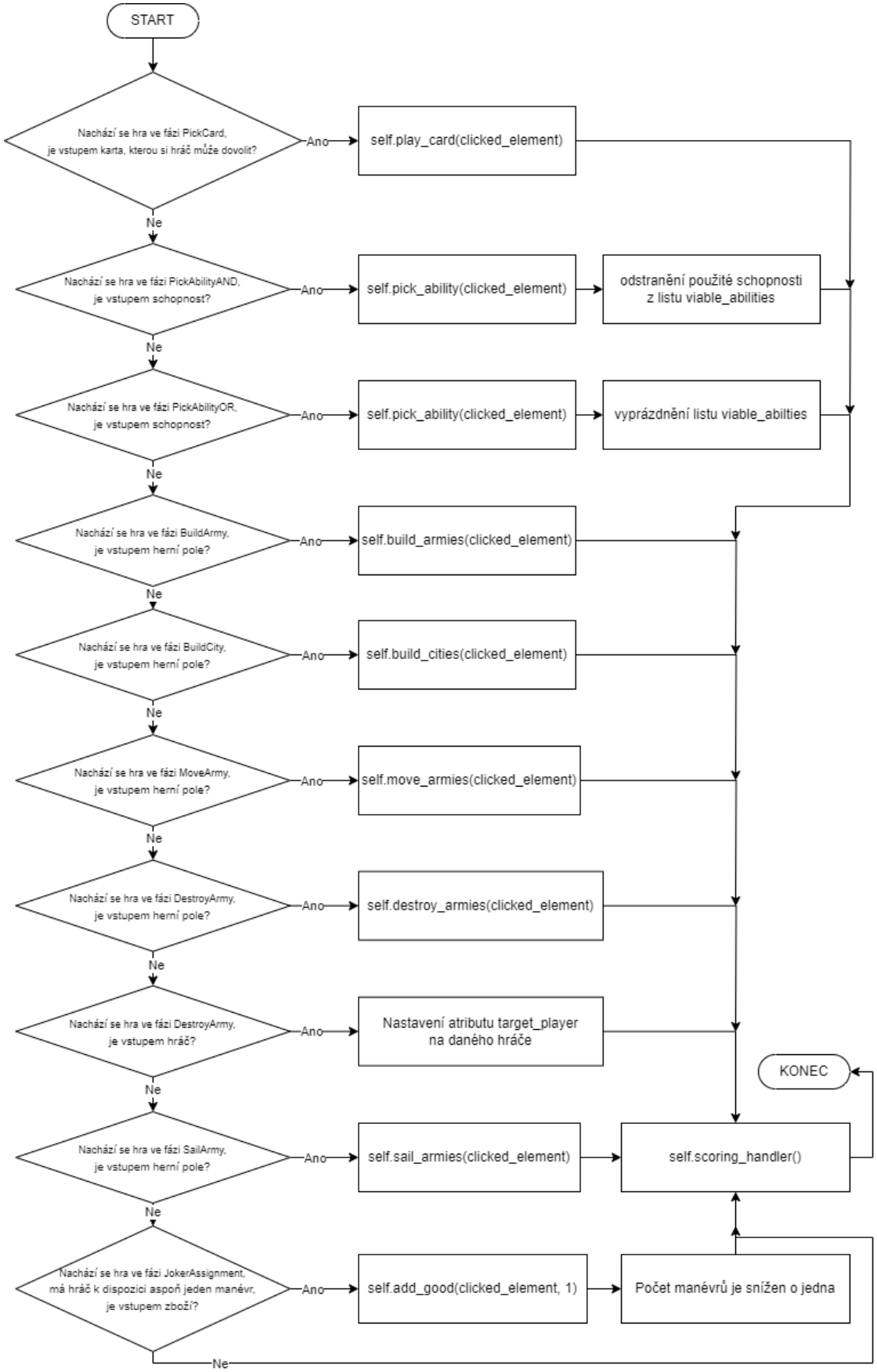
Obrázek 17: Hlavní cyklus

6.2.3 Herní logika

Třída *Game*, a obzvlášť jeden z její objektů (*TheGame*) představuje jádro programu a má na starosti většinu herní logiky. Tato třída zajišťuje vytvoření a správu herní plochy, určuje fázi, ve které se hra nachází, hráče, který je na tahu, počet hráčů a jejich pořadí, zbývající manévry v současné fázi, list použitelných schopností, list aktivních karet a propojení s třídami *Deck*, *TileManager*, *GraphicManager* a *AI_manager*.

Během hlavního cyklu se s objektem této třídy (nazvaný *TheGame)* interaguje primárně skrze dvě metody: *clickloop* a *end_move_handler*.

Metoda *cklickloop* vyžaduje jako vstup odkaz na jeden z těchto objektů: *Tile* (představující region), *Card* (představující herní kartu), *Ability* (představující jednu ze schopností), *Player* (představující hráče) a *Good* (představující zboží). Tato informace se buďto získá zavoláním metody *click_handler* objektu *TheGraphicManager* nebo zavoláním metody *pass_real_instruction* objektu *TheAIManager*. Tyto metody jsou podrobněji popsány v dalších částech. Na obrázku 18 je vizualizována struktura této metody. Tabulka 2 poskytuje popis dalších metod, které metoda *clickloop* během svého průběhu volá.

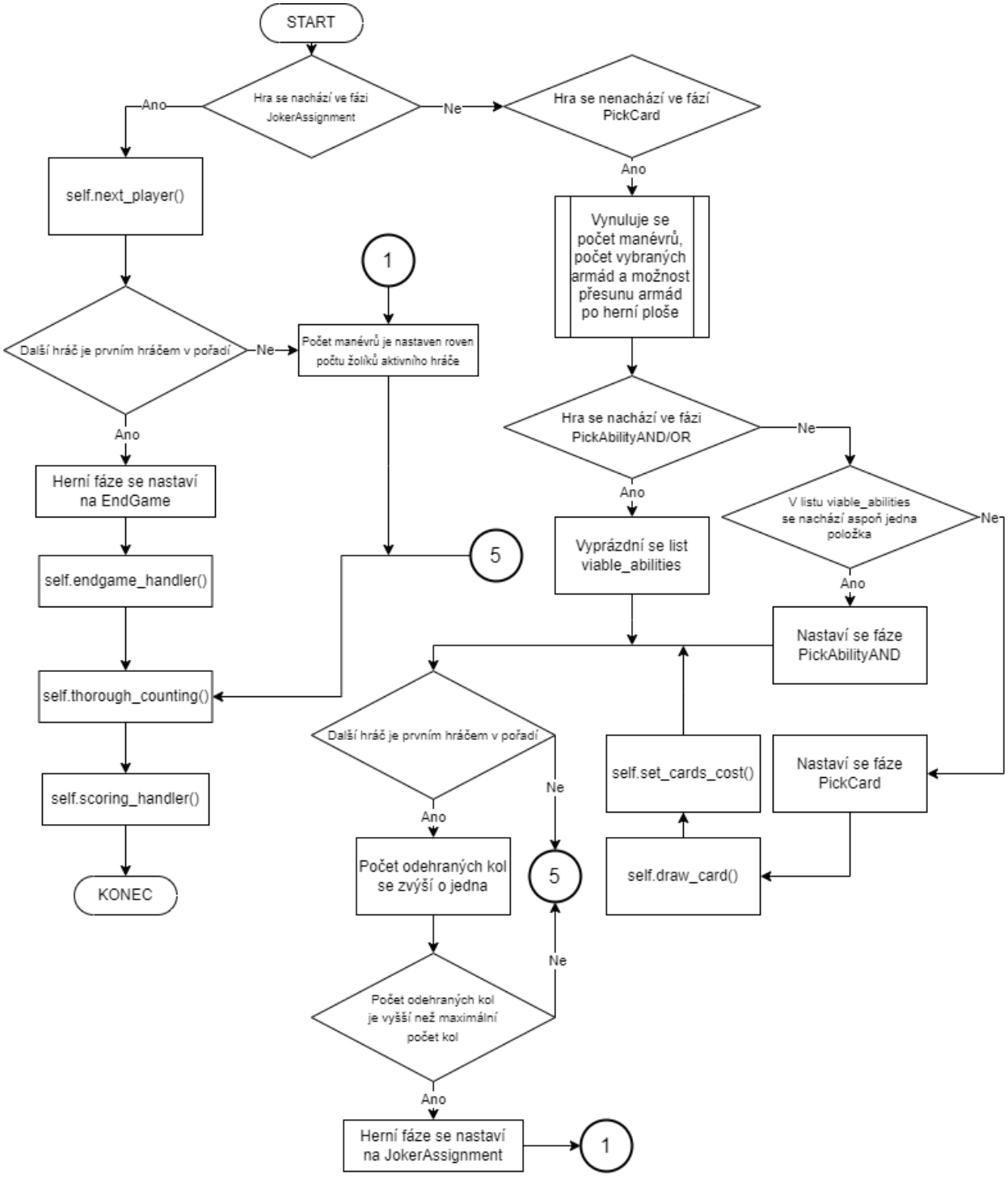


Obrázek 18: Clickloop

Tabulka 2: Clickloop metody

Jméno metody	**Popis**
play_card	Přiřadí hráči patřičné zboží pomocí metody *add_good*, na základě informací karty buď přímo zavolá metodu *pick_ability* nebo uloží schopnost do listu *viable_abilities* a přejde do fáze PickAbillityAND/OR, odebere kartu ze hry.
pick_ability	Nastaví patřičnou fázi a přiřadí počet manévrů.
build_armies	Pokud je herní pole startovním regionem, nebo regionem, kde má hráč město a zároveň má hráč méně než maximum armád, vytvoří se nová armáda za cenu jednoho manévru.
build_cities	Pokud je herní pole regionem, kde má hráč armádu/y a zároveň má hráč méně než maximum měst, postaví se nové město za cenu jednoho manévru.
move_armies/sail_armies	Pokud se na vybrané poli nachází aspoň jedna hráčova armáda a hráč nemá vybrané žádné armády, či jde o pole, ze kterého si již jednu armádu vybrán, dojde k výběru jedné armády. Pomocí metody *movable_tiles*/*sailable_tiles* zavolané z přiřazené instance třídy *TileManager*. Ta na základě počtu manévrů nastaví některé regiony jako možné k přesunu. Po kliknutí na takový regiony se zde vybrané armády přesunou za patřičnou cenu manévrů.
destroy_armies	Pokud je v daném poli aspoň jedna armáda cíleného hráče (atribut *target_player*) dojde k odstranění jedné z jeho armád za cenu jednoho manévru.
add_good	Přiřadí aktivnímu hráči určitou kvantitu zboží.
scoring_handler	Aktualizuje skóre všech hráčů na základě kontrolovaných regionů a kontinentů, a také na základě zboží hráče.

Metoda *end_move_handler* se buď volá v rámci hlavního cyklu, nebo v rámci metody *AI_loop* objektu *TheAIManager*. Tato metoda slouží k ukončení tahu, případně kola, a její struktura je vizualizována na obrázku 19 a další metody v ní použité jsou dále popsané v tabulce 3.



Obrázek 19: end_move_handler

Tabulka 3: end_move_handler metody

Jméno metody	**Popis**
next_player	Předá hru dalšímu hráči na řadě.
endgame_handler	Určí vítěze, popřípadě hráče, kteří remízovali na základě priorit: skóre > zbývající mince > ovládané regiony > počet armád.
draw_card	Do aktivních karet se přidá nová karta z balíčku.
set_cards_cost	Aktualizuje cenu karet, na základě jejich pořadí.
thorough_counting	Spočítá a aktualizuje počet všech armád a měst ve hře.
scoring_handler	Aktualizuje skóre všech hráčů na základě kontrolovaných regionů a kontinentů, a také na základě zboží hráče.

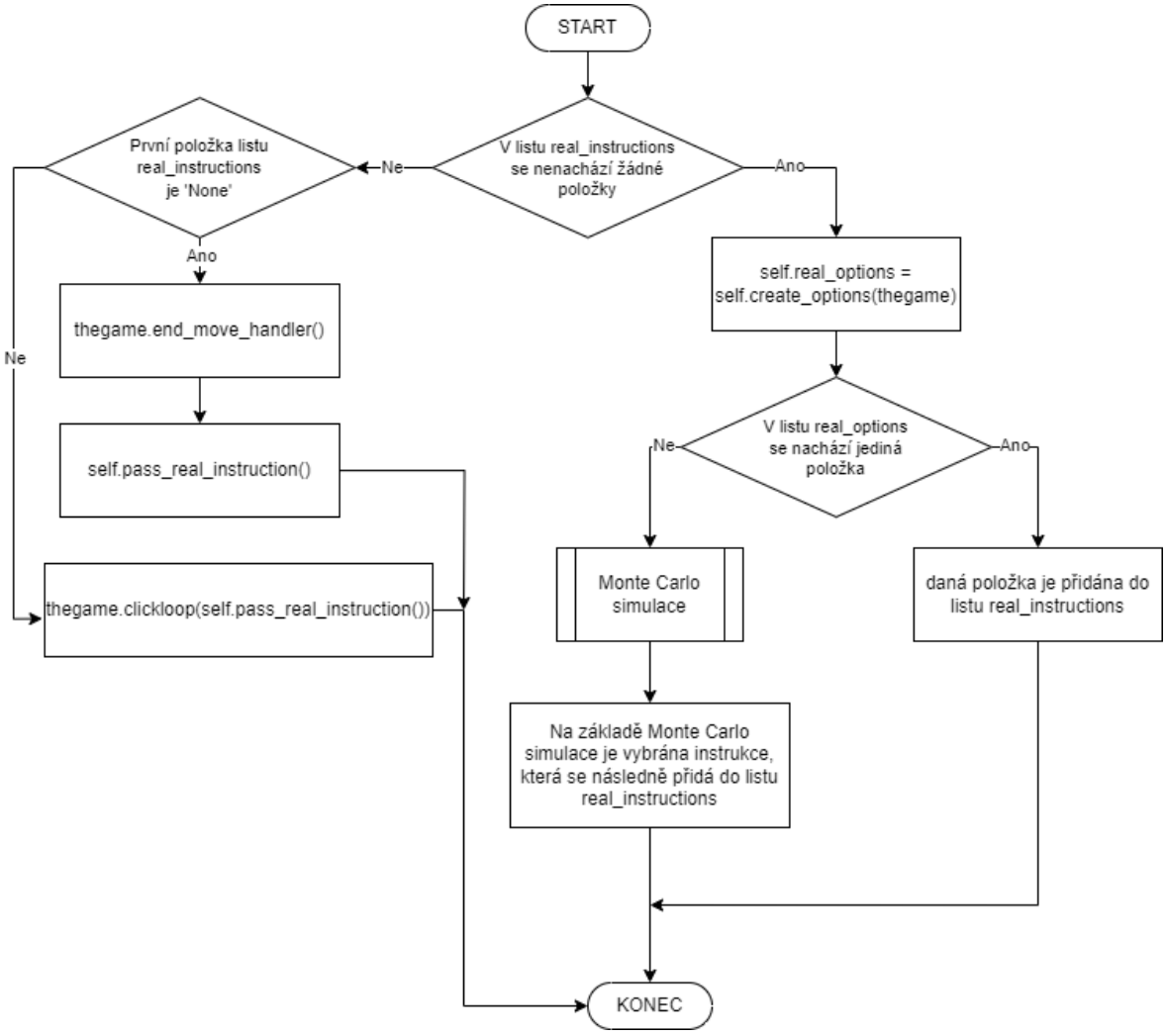
V závěru této části je třeba ještě zmínit funkcionalitu třídy *TileManager*. Jak již bylo zmíněno výše, tato třída pomáhá vykonat instancím třídy *Game* některé komplexnější herní akce. Tabulka 4 představuje seznam nejdůležitějších metod této třídy.

Tabulka 4: TileManager

Jméno metody	**Popis**
reset_armies	Slouží k vynulování vybraných armád a k jejich „přesunu“ na jejich původní pole v případě předčasného ukončení tahu.
count_armies	Vrátí počet armád vybraného hráče na herní ploše.
continent_army_count	Vrátí počet armád vybraného hráče na daném kontinentě.
count_cities	Vrátí počet měst vybraného hráče na herní ploše.
continent_city_count	Vrátí počet měst vybraného hráče na daném kontinentě.
movable_tiles	Rekurzivní funkce. Pokud na to stačí manévry, je vybrané pole nastavené jako „pohybovatelné“ a přiřadí se mu cena za pohyb na základě vzdálenosti od původního pole. Následně je tato funkce zavolaná u sousedních regionů.
sailable_tiles	Funguje podobně jako movable_tiles, s tím rozdílem, že je zavolána i na sousedících vodních polích navštíveného pole, pokud navštívené pole samotné není vodním pole. Vodní pole zároveň nejsou nastavené jako „pohybovatelné“, není jim přiřazena cena za pohyb a ani se nepočítají do vzdálenosti pro potřeby vypočítání ceny za pohyb u ostatních polí.
reset_movable_tiles	Slouží k řádnému resetování „pohybovatelnosti“ hrací plochy.

6.2.4 Simulace protivníka

AI_manager je třída, jejíž instance mají na starosti chování počítačem ovládaných protivníků. Základem této třídy je metoda *AI_loop*. Její fungování je vizualizováno v následujícím diagramu (obrázek 20). Další metody třídy jsou zase popsány v tabulce 5.



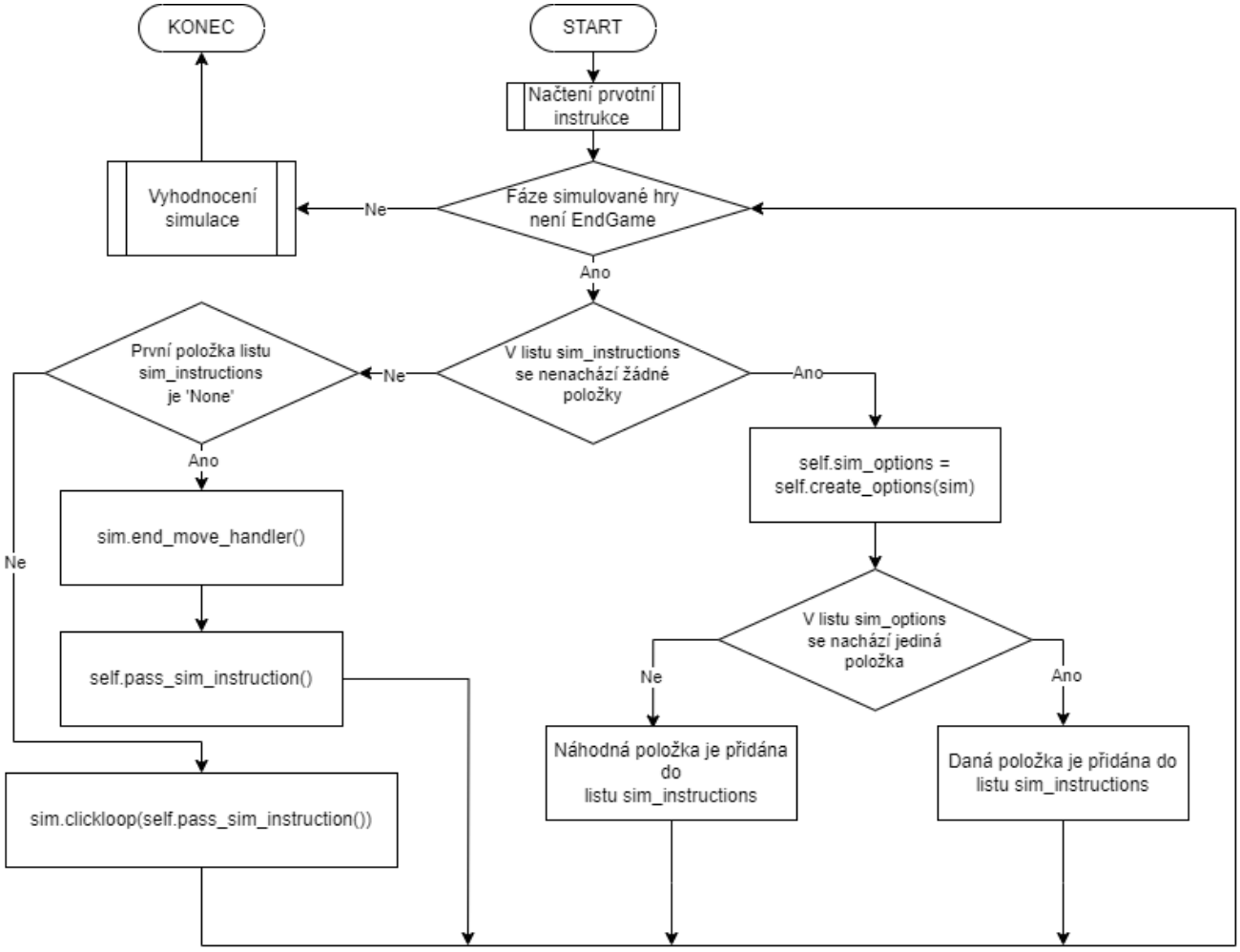
Obrázek 20: ai_loop

Tabulka 5: *AI_loop* metody

Jméno metody	**Popis**
self_create options	Vrací možné akce, které daný lze tah zahrát. Viz tabulka 6.
pass_real_instruction	Vrací vybranou akci. Zároveň danou akci odstraní z listu real_instructions.

Monte Carlo simulace je provedena u každé z možných akcí v daný tah za pomocí metody *SimulateGame*. Ta v principu funguje podobně jako výše zmíněná metoda *AI_loop.* Pracuje však s kopii objektu *TheGame*. Třída *Game* pro tyto potřeby disponuje vlastní definicí metody *_deepcopy_* a také metodou *clone_game* za účelem vytvoření řádné kopie stavu hry. Tato kopie je v následujícím diagramu (obrázek 21) označená jako *sim*.

Po dokončení Monte Carlo simulace se do patřičného indexu listu *weights*, který zrcadlí list *real_options*, přičtou buďto 3 body v případě vítězství, 1 bod v případě remízy, 0 bodů v případě prohry.



Obrázek 21: SimulateGame

Tabulka 6: Možné instrukce vytvářené třídou create_options

Formát instrukce	Poznámky
Card	Používaná ve fázi PickCard. V této fázi se jako možné instrukce uloží reference na všechny aktivní karty, které si simulovaný hráč může dovolit.
Ability	Používaná ve fázích PickAbilityAND a PickAbilityOR. V těchto fázích se jako možné instrukce uloží všechny schopnosti v listu *viable_abilities*.
Tile	Samotné herní pole se jako instrukce používá ve fázích BuildArmy a BuildCity. Aby bylo herní pole přidáno jako možná instrukce, musí splňovat podmínky pro použití dané schopnosti.
[Tile, Tile]	List dvou herních polí se jako instrukce používá ve fázích MoveArmy a SailArmy. Za každé herní pole, na kterém se nachází aspoň jedna armáda simulované hráče, se vytvoří tolik těchto instrukcí, kolik má dané pole sousedů, do kterých se dá přesunout. První herní pole v listu představuje startovní pole, to druhé představuje destinaci.
[Player, Tile]	Tento typ instrukce se vytváří jen během fáze DestroyArmy. Do možných instrukcí jsou přidány všechny kombinace protivníků a regionů, ve kterém mají armády.
Good	Tento typ instrukce se vytváří jen během fáze JokerAssignment. Do možných instrukcí je přidáno každé zboží, které lze v dané hře získat.
None	Tato instrukce představuje konec tahu. Přidává se do listu možných instrukcí ve fázích MoveArmy a SailArmy. Zároveň se automaticky vybírá v případě, že nelze vytvořit žádnou jinou instrukci.

6.2.5 Třída GraphicManager

Tato třída má na starosti jak grafické zobrazení programu, tak interakci s grafickým uživatelským prostředím. „Hlavní metodou“ této třídy je metoda *graphics*.

Na začátku této metody dojde k zavolání metody *fill* pro hlavní obrazovku, která je součástí knihovny pygame. Tato metoda zaplní obrazovku barvou uloženou ve slovníku Colors pod klíčem *background*.

Následuje volání další metody z knihovny pygame: *pygame.mouse.get_post*. Tato metoda má na starosti sledování polohy myši.

Poté následuje metoda *draw_board*, která má na starosti vykreslení všech Tile_Graphic objektů.

Další zavolaná metoda je *draw_side_menu*, která vykresluje grafické objekty zobrazeny na pravém ovládacím panelu.

Metoda *draw_player_list* vykresluje seznam hráčů, jejich skóre, zboží počet armád a měst.

Metoda *good_list_draw* vykresluje seznam zboží, včetně bodových úrovní.

Nakonec je zavolána další metoda z knihovny pygame: *display.flip*. Ta slouží k aktualizaci displeje.

Další důležitá metoda této třídy je *clickhandler*. Jak název napovídá, tato metoda má na starosti zpracovávání klikání hráče. Po každé, tato metoda projde všechny Tile_Graphic objekty a všechny objekty, které jsou v té chvíli zobrazeny v pravém ovládacím panelu. Pokud procházený objekt je dědicem třídy Clickable_Element, a zároveň metoda *clicked* takového objektu vrací hodnotu True, vrátí tato metoda referenci na patřičný objekt.

Na závěr je třeba ještě zmínit metody *prepare_tile_graphics* a *prepare_side_menu_element*. První zmiňovaná metoda vytvoří Tile_Graphic objekty, propojí je s patřičnými Tile objekty a uloží je do atributu *tile_graphics*. Druhá z těchto metod vytvoří objekty, které se zobrazí v pravém ovládacím panelu a uloží je do atributu *side_menu_elements*. Objekty vytvořené touto metodou odpovídají fázi, ve které se hra nachází.

6.2.6 Grafické třídy

V této části jsou detailněji popsané třídy, se kterými pracuje již výše popsaná třída GraphicManager.

Clickable_Element

Třída *Clickable_Element* je v podstatě interface, děděný veškerými grafickými třídami, na které může uživatel kliknout. Má dva atributy.

Atribut *graphic_manager* je určený pro propojení s objektem GraphicManager, který s tímto grafickým objektem pracuje.

Atribut *rect* představuje plochu, kterou daný grafický objekt bude zabírat na obrazovce.

Metoda *clicked*, což je zároveň jediná metoda definovaná v tomto objektu, vrací *True* v případě že atribut *clicked_pos* objektu GraphicManager se překrývá s atributem *rect*. Pro zjištění tohoto pokrytí se používá metoda *collidepoint* z knihovny pygame.

Tile_Graphic

Objekty této třídy se používají pro grafické vykreslení objektů *Tile*. Každý tento objekt je přiřazený k patřičnému *Tile* objektu, který reprezentuje. Zároveň má informace o své poloze na obrazovce a velikosti vykresleného herního pole.

Metoda *draw* na základě těchto informací vykreslí dané herní pole, včetně počtu armád a města, a označení startovního regionu.

Tato třída je dědicem třídy *Clickable_Element*.

Card_Button

Objekty této třídy se používají pro vykreslení textu reprezentující aktivní karty, který je následně zobrazen v pravém ovládacím panelu během fáze PickCard. Objekty této třídy mají přiřazený patřičný *Card* objekt a obsahují informace o vyžadované poloze.

Metoda *draw* na základě těchto informací vykreslí daný text. Zároveň pomocí metody *collidepoint* z knihovny pygame zajišťuje zvýraznění textu, pokud na něj zrovna míří kurzor.

Tato třída je dědicem třídy *Clickable_Element*.

Ability_Button

Objekty této třídy se používají pro vykreslení textu reprezentující možné schopnosti, který je následně zobrazen v pravém ovládacím panelu během fáze PickAbilityOR nebo PickAbilityAND. Objekty této třídy mají přiřazený patřičný *Ability* objekt a obsahují informace o vyžadované poloze.

Metoda *draw* na základě těchto informací vykreslí daný text. Zároveň pomocí metody *collidepoint* z knihovny pygame zajišťuje zvýraznění textu, pokud na něj zrovna míří kurzor.

Tato třída je dědicem třídy *Clickable_Element*.

JokerAssignmentButton

Objekty této třídy se používají pro vykreslení textu reprezentující zboží, který je následně zobrazen v pravém ovládacím panelu během fáze JokesAssignment. Objekty této třídy mají přiřazený patřičný Good objekt a obsahují informace o vyžadované poloze.

Metoda *draw* na základě těchto informací vykreslí daný text. Zároveň pomocí metody *collidepoint* z knihovny pygame zajišťuje zvýraznění textu, pokud na něj zrovna míří kurzor.

Tato třída je dědicem třídy *Clickable_Element*.

Target_Button

Objekty této třídy se používají pro vykreslení textu reprezentující hráče, který je následně zobrazen v pravém ovládacím panelu během fáze DestroyArmy. Objekty této třídy mají přiřazený patřičný Player objekt a obsahují informace o vyžadované poloze.

Metoda *draw* na základě těchto informací vykreslí daný text. Zároveň pomocí metody *collidepoint* z knihovny pygame zajišťuje zvýraznění textu, pokud na něj zrovna míří kurzor.

Tato třída je dědicem třídy *Clickable_Element*.

Player_List_Element

Tato třída se používá pro vykreslení nadpisu pravého kontrolního panelu. Ten slouží k předání následujících informací: jméno hráče na tahu, současné kolo, zbývající kola, fáze, počet manévrů (v případě fáze, která odpovídá jedné ze schopností), počet mincí (v případě fáze PickCard). Nadpis zároveň mění barvu, podle toho, který hráč je zrovna na řadě.

Pokud je hra ve fázi EndGame, je objekt této třídy použit k vypsání vítězného hráče, popřípadě hráčů, kteří dosáhli remízy.

7 TESTOVÁNÍ

K otestování implementovaného algoritmu Monte Carlo a jeho schopnosti simulovat kompetentního protivníka bylo odehráno několik her. Testování se zúčastnili tři studenti Fakulty aplikované informatiky UTB ve Zlíně, konkrétně:

Ondřej Lukáš, autor této bakalářské práce, dál uveden v tabulkách jako LH1,

Magdaléna Nevrlá, dále uvedená v tabulkách jako LH2,

Bc. Martin Machala, dále uvedený v tabulkách jako LH1,

Všichni hráči jsou znalí pravidel dané hry a občasně ji hrají proti sobě.

Počet Monte Carlo simulací prováděných „umělými“ protivníky je nastaven na 200. Tato hodnota byla vybrána po několika zkušebních hrách, při kterých takto simulovaný protivník (dále označen jako „SH“) dokázal konzistentně vyhrávat proti reálným hráčům.

Vítěz každé hry je vyznačen tučnými písmy, rozhodující faktor je podtržen. Pro připomenutí, pokud dojde k remíze, porovná se nejprve počet zbývajících mincí hráče, následně se porovnávají kontrolované regiony a jako poslední se porovná počet armád na herní ploše.

7.1 Scénář č. 1: jeden reálný hráč, jeden simulovaný hráč

Celkem bylo odehráno pět her, ve kterých jeden lidský hráč čelil jednomu „umělému“ protivníku. Simulovaný hráč byl schopen zvítězit ve třech těchto hrách. Při těchto hrách si šlo povšimnout dvou věcí:

1) Priorita simulovaného hráče byla maximalizovat svůj počet armád na herním poli.
2) Simulovaný hráč se nijak nepokoušel šetřit své mince. Často došlo k situacím, kdy simulovaný hráč všechny své mince vypotřeboval již před dosažení poloviny herních kol.

Pro připomenutí: Ve hře dvou hráčů se hraje do 13 kol a každý hráč má k dispozici 14 mincí.

Tabulka 7: LH1 vs SH, SH vyhrál na skóre

Hráč	Skóre	Zbývající mince	Kontrolované regiony	Konečný Počet armád
LH1	15	0	5	14
SH	**16**	**0**	**7**	**11**

Tabulka 8: LH1 vs SH, SH vyhrál na skóre

Hráč	Skóre	Zbývající mince	Kontrolované regiony	Konečný počet armád
LH1	14	0	7	10
SH	**15**	**0**	**6**	**14**

Tabulka 9: LH2 vs SH, LH2 vyhrál na skóre

Hráč	Skóre	Zbývající mince	Kontrolované regiony	Konečný počet armád
LH2	**14**	**0**	**6**	**14**
SH	12	0	6	12

Tabulka 10: LH2 vs SH, SH vyhrál na skóre

Hráč	Skóre	Zbývající mince	Kontrolované regiony	Konečný počet armád
LH2	15	1	6	13
SH	**17**	**0**	**7**	**14**

Tabulka 11: LH3 vs SH, LH3 vyhrál na skóre

Hráč	Skóre	Zbývající mince	Kontrolované regiony	Počet armád
LH3	**17**	**2**	**8**	**10**
SH	10	0	3	14

7.2 Scénář 2: jeden reálný hráč, dva simulovaní hráči

Celkem se podařilo odehrát tři hry, ve kterých jeden lidský hráč stojí proti dvěma „umělým“ protivníkům. Z těchto tří her se jednomu ze simulovaných hráčů podařilo zvítězit ve dvou případech. Za zmínku stojí říct, že se zvýšením počtu hráčů, znatelně klesá priorita simulovaných hráčů vytvářet nové armády za každou cenu, zároveň „umělí“ hráči déle šetří své mince.

Pro připomenutí: Ve hře tří hráčů se hraje do 10 kol a každý hráč má k dispozici 11 mincí.

Tabulka 12: LH1 vs SH1 vs SH2, LH1 vyhrál na mince

Hráč	Skóre	Zbývající mince	Kontrolované regiony	Konečný počet armád
LH1	**10**	**2**	**4**	**9**
SH1	7	0	3	13
SH2	10	0	3	10

Tabulka 13: LH2 vs SH1 vs SH2, SH2 vyhrál na kontrolované regiony

Hráč	Skóre	Zbývající mince	Kontrolované regiony	Konečný počet armád
LH2	10	0	4	8
SH1	10	0	3	10
SH2	**10**	**0**	**5**	**14**

Tabulka 14: LH1 vs SH1 vs SH2, SH2 vyhrál na skóre

Hráč	Skóre	Zbývající mince	Kontrolované regiony	Konečný počet armád
LH1	9	1	3	14
SH1	4	0	1	12
SH2	**11**	**0**	**3**	**7**

7.3 Scénář č. 3: dva reální hráči, jeden simulovaný hráč

Her, při kterých dva hráči stáli proti jednomu „umělému“ protivníku, se podařilo odehrát pouze dvě. V obou případech zvítězil jeden z lidských hráčů. Temperament simulovaného protivníka nebyl výrazně rozdílný oproti předchozímu scénáři.

Tabulka 15: LH1 vs LH2 vs SH, LH1 zvítězil na mince

Hráč	Skóre	Zbývající mince	Kontrolované regiony	Konečný počet armád
LH1	**11**	**2**	**3**	**10**
LH2	8	2	3	13
SH	11	0	6	13

Tabulka 16: LH1 vs LH2 vs SH, LH3 zvítězil na skóre

Hráč	Skóre	Zbývající mince	Kontrolované regiony	Konečný počet armád
LH2	9	0	3	10
LH3	**10**	**0**	**3**	**8**
SH	8	0	4	11

7.4 Scénář č. 4: dva reální hráči, dva simulovaní hráči

Pro potřeby testování se podařilo odehrát jen jednu hru, ve kterém dva lidští hráči čelili dvou „umělým“ protivníkům. V tomto případě se podařilo zvítězit jednomu z lidských hráčů, nedá se ale říct, že by simulovaní protivníci hráli nekompetentně. Šetřivost umělé inteligence byla znatelně zvýšena, což je podtrhnutu faktem, že jde o první scénář, při kterém jeden ze simulovaných protivníků dokončil hru alespoň s jednou mincí v rezervě.

Pro připomenutí: Ve hře čtyř hráčů se hraje do 8 kol a každý hráč má k dispozici 9 mincí.

Tabulka 17: LH2 vs LH3 vs SH1 vs SH2, LH3 zvítězil na skóre

Hráč	Skóre	Zbývající mince	Kontrolované regiony	Konečný počet armád
LH2	7	0	2	12
LH3	**8**	**0**	**3**	**7**
SH1	7	1	3	13
SH2	7	0	2	3

7.5 Scénář 5: tři reální hráči, dva simulovaní hráči

Scénář, při kterém tři lidští hráči hráli se dvěma simulovanými protivníky se podařilo otestovat jen jednou. Cílem toho scénáře bylo hlavně otestovat funkcionalitu hry při plném obsazení. Každopádně v tomto scénáři zvítězil jeden ze simulovaných hráčů. Zajímavostí je, že jde o jediný scénář, při kterém „umělý“ hráč zvítězil ve hře, který byla rozhodnuta zbývajícím počtem mincí.

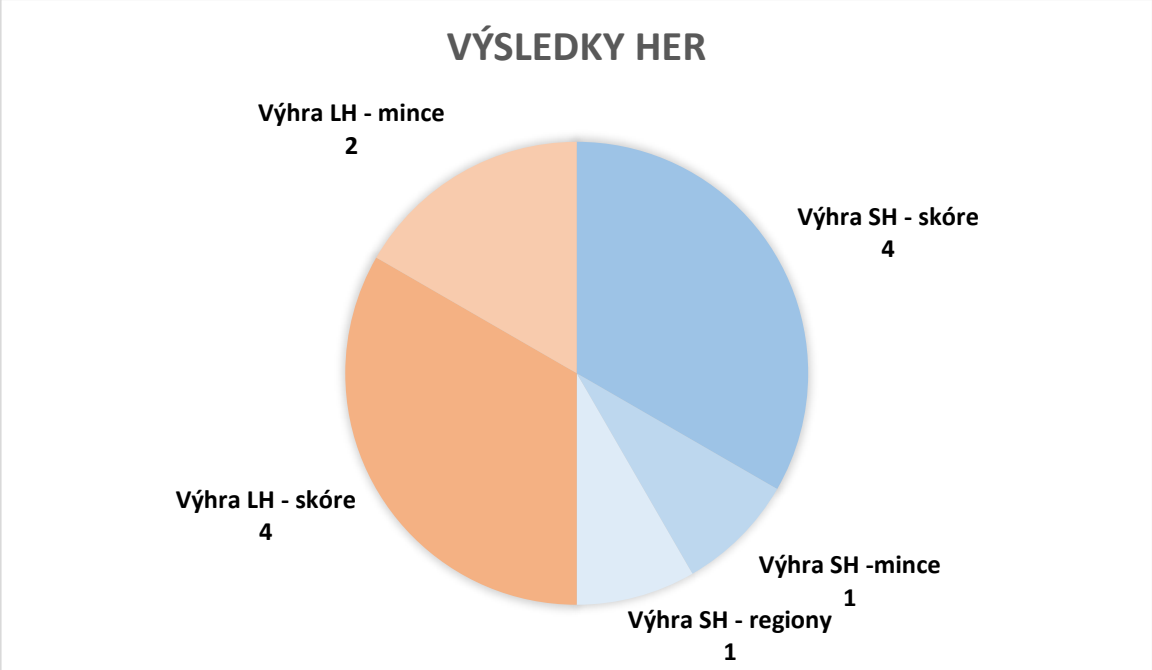
Pro připomenutí: Ve hře pěti hráčů se hraje do 7 kol a každý hráč má k dispozici 8 mincí. Zároveň jsou do hry přidány karty navíc, které se ve hrách s nižším počtem hráčů neobjevují.

Tabulka 18: LH1 vs LH2 vs LH3 vs SH1 vs SH2, SH1 zvítězil na mince

Hráč	Skóre	Zbývající mince	Kontrolované regiony	Konečný počet armád
LH1	6	0	1	5

LH2	8	0	3	11
LH3	8	0	3	8
SH1	**8**	**1**	**4**	**12**
SH2	6	0	3	10

7.6 Celkové zhodnocení výkonu simulovaného protivníka



Obrázek 22: Graf výsledků her

Z dvanácti celkových her v šesti z nich zvítězil simulovaný protivník. Čtyři tyto hry byly vyhrány na základě získání nejvyššího skóre, jedna hra byla vyhrán na základě počtu kontrolovaných regionů a jedna hra byla vyhrán na základě zbývajících mincí. Tyto informace jsou zobrazeny v celkovém grafu výsledků her (Obrázek 22).

Celkově se dá říct, že implementovaný algoritmus relativně efektivně simuluje kompetentního hráče, k jehož porážce je třeba jistá zkušenost se hrou a plánovaní dopředu. Zde jsou některé poznatky a závěry ze všech provedených experimentů:

- simulovaní hráči jsou schopni umisťovat města na strategicky výhodná místa (např. na jiný kontinent než ten, na kterém se nachází startovní region)

- simulovaní hráči jsou schopni soustředit se na vybrané suroviny, které chtějí sbírat (např. se často soustředí na sběr karet uhlí a jídla)

- simulovaní hráči nejsou schopni přiměřeně odhadnout hodnotu žolíka. To je způsobeno tím, že k přiřazování žolíků dochází až na konci hry. Praktickým řešením by bylo vytvoření deterministické metody pro efektivní přiřazování žolíků, která by byla zároveň použita u Monte Carlo simulací.

- simulovaní hráči nejsou schopni přiměřeně odhadnout užitečnost karet, které mají dvě schopnost spojené spojkou OR. To je způsobeno tím, že výběr schopnosti je vyhodnocován separátně od výběru karty. Díky tomu je užitečnost těchto karet zkreslena. Jednoduchým řešením by bylo rozdělit takové karty na dvě různé možnosti ve fázi vyhodnocení výběru karet, místo toho, aby k výběru schopností docházelo v separátním zavolání metody *AI_loop*.

7.7 Měření výpočetní rychlosti

Pro potřeby měření výpočetní rychlosti programu, bylo celkem odehráno 16 her, ve kterých byli všichni hráči simulováni, tedy hraní se neúčastnil ani jeden lidský hráč: Pět her se dvěma hráči, pět her se třemi hráči, tří hry se čtyřmi hráči a tři hry s pěti hráči.

Důvod, proč hry se čtyřmi a pěti hráči byly simulovány pouze třikrát, je kvůli značně vyššímu vyžadovanému času.

Specifika stroje, na kterém byly provedeny testy.

Model notebooku: Ideapad Gaming 3-15IMH05 Laptop - Type 81Y4

Procesor: Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz

Počet jader: 4

Nainstalovaná paměť: RAM 8,00 GB (použitelné: 7,87 GB)

Typ systému: 64bitový operační systém, procesor pro platformu x64

Čas, který simulovaní hráči vyžadovali na zahrání jednoho kola je zaznamenán pomocí Python modulu time a je dále uveden v sekundách, vždy zaokrouhlen na celé číslo. V levém sloupci je celková suma stráveného času od začátku do konce všech AI_ACTION_EVENT. Průměrný čas vyžadovaný na provedení jednoho kola je vypočítán jednoduchou rovnicí (čas na jedno kolo = celkový čas / počet hráčů / počet kol). Následně je vypočítán i celkový průměr na jedno kolo, vydělením součtu všech průměrů počtem spuštěných her v dané kategorii.

Jasný „bottleneck“ programu tvoří akce pohybu. Což je dáno velkým množství možných pohybů, obzvlášť případě, kdy je možný i přesun po moři. Jsou to právě tyto, které způsobují náhlé navýšení výpočetního času ve hře se čtyřmi hráči – viz obrázek 23 níže. Ve hře čtyř hráčů je totiž garantované, že se všechny karty dostanou do aktivního řádku. To znamená, že budou neodvratně zahrány skoro všechny pohybové karty. U hry pěti hráčů náročnost zase klesá, kvůli přidáním nových karet do hry.

Tabulka 19: Výpočetní čas – hra dvou hráčů

Celkový čas [s]	**Průměrný čas na jedno kolo [s]**
494	19
656	25
621	24
624	24
497	19
Celkový průměrný čas na jedno kolo[s]:	22

Tabulka 20: Výpočetní čas – hra tří hráčů

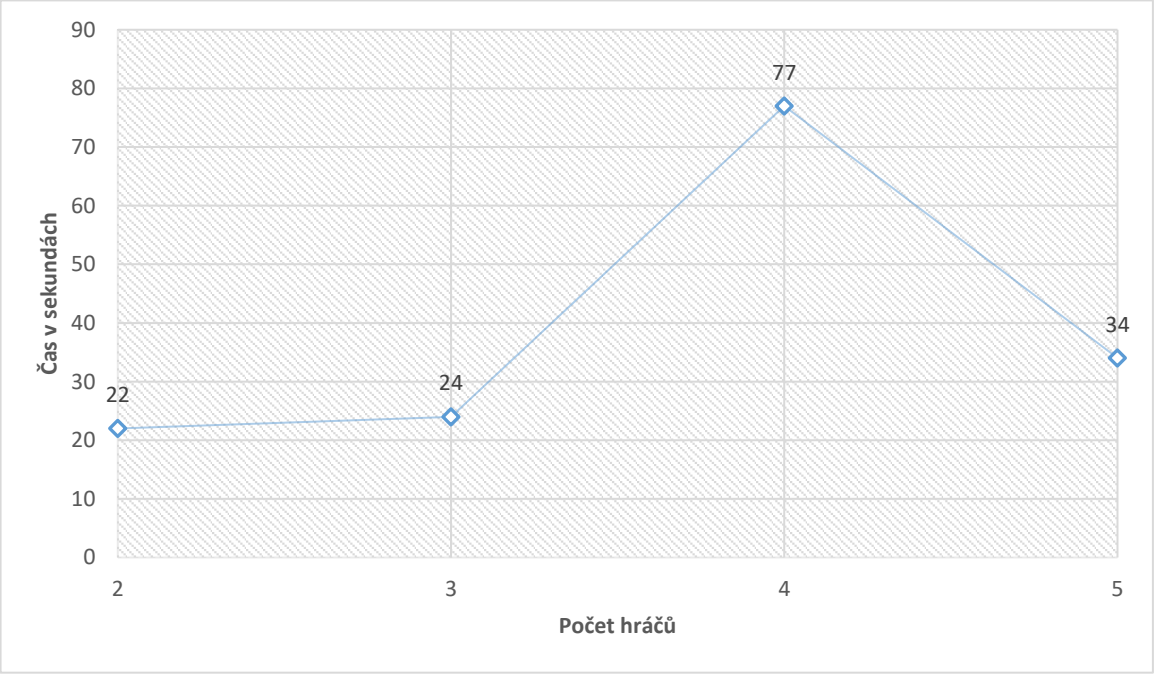
Celkový čas [s]	**Čas na jedno kolo [s]**
699	23
735	25
806	27
631	21
785	26
Celkový průměrný čas na jedno kolo[s]:	24

Tabulka 21: Výpočetní čas – hra čtyř hráčů

Celkový čas [s]	Čas na jedno kolo [s]
2211	69
2991	93
2230	70
Celkový průměrný čas na jedno kolo[s]:	77

Tabulka 22: Výpočetní čas – hra pěti hráčů

Celkový čas [s]	Čas na jedno kolo [s]
1065	30
968	28
1146	33
Celkový průměrný čas na jedno kolo[s]:	30



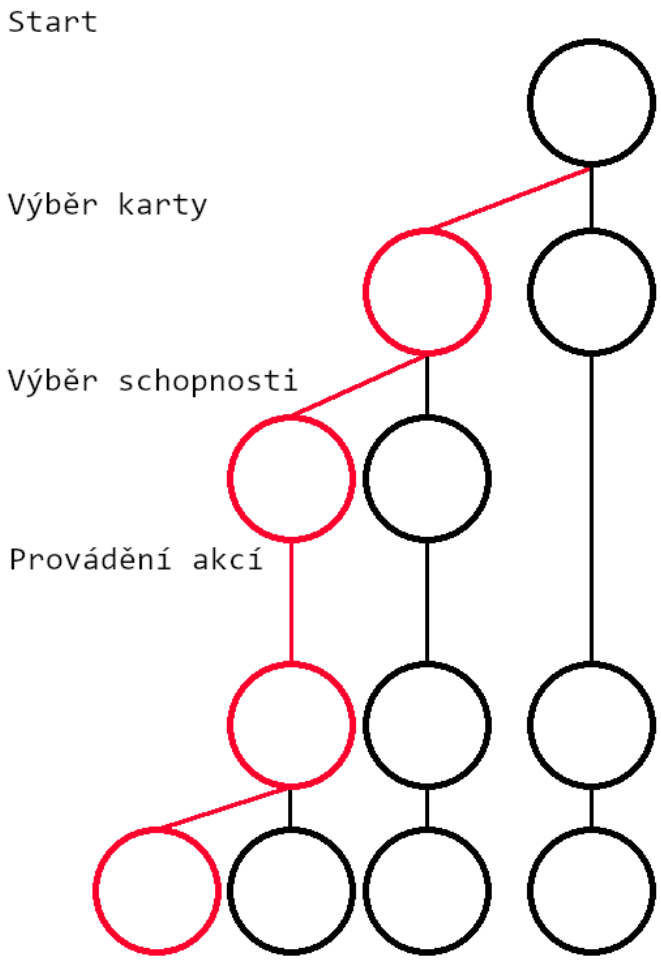
Obrázek 23: Průměrný výpočetní čas simulovaných protivníků na 1 kolo dle počtu hráčů

Plánované vylepšení programu

Vzhledem k omezením současného řešení bylo v plánu implementovat dvě úpravy, který by měl, jak zásadně snížit časovou náročnost programu, tak zlepšit zručnost simulovaného protivníka. Bohužel na jejich implementaci a řádné otestování již nezbyl čas, přesto je zde pro úplnost uvedu.

- Deterministický algoritmus pro přiřazování žolíků, aby simulovaný protivník mohl lépe odhadnout hodnotou žolíků.

- Implementace Monte Carlo Tree Search (MCTS) algoritmu, hloubkou omezený do konce tahu simulovaného hráče, z jehož výsledku by se mohl vytvořit jeden dlouhý řetěz instrukcí. Hlavní výhodou tohoto plánu, by krom zručnější hry, bylo několikanásobné omezení počtu provedených Monte Carlo Simulací během jednoho tahu. Na obrázku 24 je ilustrované, jak by takový algoritmus mohl fungovat.



Obrázek 24: Plánovaný MCTS algoritmus

ZÁVĚR

V rámci této bakalářské práce došlo k úspěšné implementaci vybraných pravidel stolní hry. Hra je plně hratelná a jakékoliv "gamebreaking" chyby byly během testování odstraněny. Hru lze hrát ve všech možných kombinací hráčů a simulovaných protivníků.

I přes některé úpravy pravidel, přístup lidských hráčů nebyl nijak viditelně rozdílný od jejich přístupů hraní fyzické verze. Simulovaní protivníci taktéž úspěšně napodobovali strategie hráčů, které šlo vypozorovat u deskové hry (např. maximalizace počtu armád).

Co se schopností „umělých" protivníků týče, simulovaní hráč dokázali porazit lidské protivníky v polovině testovaných případů. To se navíc podařilo za použití velmi jednoduchého algoritmu.

Bohužel ne-existuje žádná organizovaná komunita pro tuto hru jako např. u šachů nebo go, která by umožnila formální zhodnocení schopností implementované „umělé inteligence" i proti velmi zkušeným hráčům. Největší slabinou implementovaného přístupu zůstává vyžadovaný čas pro provedení výpočtů. Byť snesitelný, byl hlavním zdrojem stížností testujících hráčů.

Jak již bylo zmíněno v poslední kapitole, šlo by jak značně snížit vyžadovaný výpočetní čas, tak i zvýšit kompetenci simulovaných protivníků implementací alespoň mělkého Monte Carlo Tree Search algoritmu.

SEZNAM POUŽITÉ LITERATURY

[1] SHANNON, Claud E. *Programming a Computer for Playing Chess*. 1949.

[2] ISENBERG, Gerd. Turochamp. LEFLER, Mark. *Chess Programming Wiki* [online]. 2007 [cit. 2024-03-18]. Dostupné z: https://www.chessprogramming.org/Turochamp

[3] COPELAND, B. Jack. The essential Turing: seminal writings in computing, logic, philosophy, artificial intelligence, and artificial life, plus the secrets of Enigma. Oxford: Clarendon Press, c2004. ISBN 978-0-19-825080-7.

[4] SAMUEL, Arhur. Some Studies in Machine Learning Using the Game of Checkers. IBM Journal of Research and Development. 1959, 3(3), 211-229.

[5] BERNSTEIN, Alex a Michael DE V. ROBERTS. COMPUTER V. CHESS-PLAYER. 1958.

[6] BRÜGMANN, Bernd. *Monte Carlo Go* [pdf]. Max-Planck-Institute of Physics.

[7] COULOM, Rémi. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search [pdf]. 2006.

[8] SILVER, David, Aja HUANG, Chris MADDISON, et al. Mastering the game of Go with deep neural networks and tree search. Nature. 2016, 529(7857), 484-513.

[9] GOOGLE DEEPMIND. AlphaGo. GOOGLE DEEPMIND. *Google DeepMind* [online]. [cit. 2024-04-22]. Dostupné z: https://deepmind.google/technologies/alphago/

[10] SILVER, David a Demis HASSABIS. AlphaGo Zero: Starting from scratch. In: GOOGLE DEEPMIND. *Google DeepMind* [online]. [cit. 2024-04-22]. Dostupné z: https://deepmind.google/discover/blog/alphago-zero-starting-from-scratch/

[11] SILVER, David, Thomas HUBERT a Julian SCHRITTWIESER. AlphaZero: Shedding new light on chess, shogi, and Go. In: GOOGLE DEEPMIND. Google DeepMind [online]. [cit. 2024-04-22]. Dostupné z: https://deepmind.google/discover/blog/alphago-zero-starting-from-scratch/

[12] GOOGLE DEEPMIND. AlphaZero and MuZero. GOOGLE DEEPMIND. Google DeepMind [online]. [cit. 2024-04-22]. Dostupné z: https://deepmind.google/technologies/alphago/

[13] MOTLÍČEK, Ondřej. Umělá inteligence pro hru Carcassonne - Objevitelé. Praha, 2020. Bakalářská práce. Univerzita Karlova, Matematicko-fyzikální fakulta, Katedra teoretické informatiky a matematické logiky. Vedoucí práce Hric, Jan.

[14] BŘEZINA, Patrik. Metody umělé inteligence pro hraní sběratelských karetních her. Praha, 2019. Bakalářská práce. České vysoké učení

technické v Praze, Fakulta elektronická, Katedra počítačové grafiky a interakce. Vedoucí práce Lisý Viliam

[15] KURENKOV, Andrey. A 'Brief' History of Game AI Up To AlphaGo. In: KURENKOV, Andrey. Andrey Kurenkov [online]. 2014, Last updated: Devember 29th 2023 [cit. 2024-04-22]. Dostupné z: https://www.andreykurenkov.com/writing/ai/a-brief-history-of-game-ai/

[16] GOOGLE. Andrey Kurenkov. GOOGLE SCHOLAR. Google Scholar [online]. [2004] [cit. 2024-04-22]. Dostupné z: https://scholar.google.com/citations?user=mmiHOS4AAAAJ&hl=en

[17] LEFLER, Mark. Chess Programming Wiki [online]. 2007 [cit. 2024-04-22]. Dostupné z: https://www.chessprogramming.org

[18] ISENBERG, Gerd. Mark Lefler. In: LEFLER, Mark. Chess Programming Wiki [online]. 2007 [cit. 2024-04-22]. Dostupné z: https://www.chessprogramming.org/Mark_Lefler

[19] SHWARTZ, Oscar. Untold History of AI: When Charles Babbage Played Chess With the Original Mechanical Turk. In: INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. IEEE Spectrum [online]. [2020] [cit. 2024-04-23]. Dostupné z: https://spectrum.ieee.org/untold-history-of-ai-charles-babbage-and-the-turk

[20] ISENBERG, Gerd. El Ajedrecista. In: LEFLER, Mark. Chess Programming Wiki [online]. 2007 [cit. 2024-04-22]. Dostupné z: https://www.chessprogramming.org/El_Ajedrecista

[21] NORMAN, Jeremy. Torres y Quevedo Invents El Ajedrecista, the First Decision-Making Automaton. In: NORMAN, Jeremy. Jeremy Norman's HistoryOfInformation.com [online]. 2004 [cit. 2024-04-23]. Dostupné z: https://www.historyofinformation.com/detail.php?id=569

[22] ISENBERG, Gerd. History. In: LEFLER, Mark. Chess Programming Wiki [online]. 2007 [cit. 2024-04-22]. Dostupné z: https://www.chessprogramming.org/History

[23] Chess Playing Programs and the Problem of Complexity. IBM Journal of Research and Development. 1958, 4(2), 320-335.

[24] BRUDNO, Michael. *Competitions, Controversies, and Computer Chess*. 2000.

[25] MASTERS TRADITIONAL GAMES. The Rules of Go or Wei Chi. MASTERS TRADITIONAL GAMES. *Masters Traditional Games* [online]. 1999 [cit. 2024-04-25]. Dostupné z: https://www.mastersofgames.com/rules/go-rules.htm

[26] ZOBRIST, Albert. COMPUTER SCIENCES DEPARTMENT, UNIVERSITY OF WISCONSIN. Feature extraction and representation for pattern recognition and the game of GO. 1970.

[27] WILCOX, Bruce. Reflections on building two Go programs. *ACM SIGART Bulletin*. 1985, (94), 29-43.

[28] SMART GAMES LLC. The Many Faces of Go, Version 12. SMART GAMES LLC. Smart Games [online]. [cit. 2024-04-25]. Dostupné z: https://smart-games.com/manyfaces.html

[29] KOCSIS, Levente a Csaba SZEPESVÁRI. COMPUTER AND AUTOMATION RESEARCH INSTITUTE OF THE HUNGARIAN ACADEMY OF SCIENCES. *Bandit based Monte-Carlo Planning*. 2006.

[30] GELLY, Sylvain, Yizao WANG, Rémi MUNOS a Olivier TEYTAUD. INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE. *Modification of UCT with Patterns in Monte-Carlo Go*. HAL Open Science, 2006.

[31] A Concise History of Neural Networks. In: MEDIUM. *Medium* [online]. [2012] [cit. 2024-04-25]. Dostupné z: https://towardsdatascience.com/a-concise-history-of-neural-networks-2070655d3fec

[32] Rules of Backgammon. KEITH, Tom. Backgammon GALORE! [online]. 1995 [cit. 2024-04-25]. Dostupné z: https://www.bkgm.com/rules.html

[33] BERLINER, Hans. Computer Backgammon. *Scientific American*. 1980, **242**(6), 64-72.

[34] TESAURO, Gerald. IEEE. *Neurogammon: A Neural-Network Backgammon Program*. 1990.

[35] Temporal Difference Learning and TD-Gammon. Communications of the ACM. 1995, 38(3), 58-68.

[36] KHURANA, Swasti. TD-Gammon algorithm. In: MEDIUM. *Medium* [online]. 2012 [cit. 2024-04-25]. Dostupné z: https://medium.com/clique-org/td-gammon-algorithm-78a600b039bb

[37] CLARK, Christopher a Amos STORKEY. *Teaching Deep Convolutional Neural Networks to Play Go*. 2.0. 2015.

[38] BACHRACH, Yoram a János KRAMÁR. AI for the board game Diplomacy. In: GOOGLE DEEPMIND. *Google DeepMind* [online]. [cit. 2024-04-25]. Dostupné z: https://deepmind.google/discover/blog/ai-for-the-board-game-diplomacy/

[39] Minimax Algorithm in Game Theory | Set 1 (Introduction). GEEKSFORGEEKS. *GeeksforGeeks* [online]. 2008 [cit. 2024-04-25]. Dostupné z: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

[40] Expectimax Algorithm in Game Theory. GEEKSFORGEEKS. *GeeksforGeeks* [online]. 2008 [cit. 2024-04-25]. Dostupné z: https://www.geeksforgeeks.org/expectimax-algorithm-in-game-theory/

[41] ISENBERG, Gerd. Alpha-Beta. In: LEFLER, Mark. *Chess Programming Wiki* [online]. 2007 [cit. 2024-04-22]. Dostupné z: https://www.chessprogramming.org/Alpha-Beta

[42] LUCKHARDT, Carol a Keki IRANO. UNIVERSITY OF MICHIGAN, ELECTRICAL ENGINEERING AND COMPUTER SCIENCE. *AN ALGORITHMIC SOLUTION OF N-PERSON GAMES*. 1986.

[43] What is Monte Carlo Simulation? IBM. *IBM* [online]. [cit. 2024-04-26]. Dostupné z: https://www.ibm.com/topics/monte-carlo-simulation

[44] ISENBERG, Gerd. Monte-Carlo Tree Search. In: LEFLER, Mark. *Chess Programming Wiki* [online]. 2007 [cit. 2024-04-22]. Dostupné z: https://www.chessprogramming.org/Monte-Carlo_Tree_Search

[45] IBM. What is a neural network? IBM. IBM [online]. [cit. 2024-04-30]. Dostupné z: https://www.ibm.com/topics/neural-networks

[46] IBM. Https://www.ibm.com/topics/machine-learning. IBM. *IBM* [online]. [cit. 2024-04-30]. Dostupné z: https://www.ibm.com/topics/machine-learning

[47] From AlphaGo to MuZero. In: Youtube [online]. 5. 4. 2021 [cit. 2024-06-05]. Dostupné z: https://www.youtube.com/watch?v=lVMgxtm5L-U&t=1006s. Kanál uživatele Harvard CMSA.

[48] MANDAL, Manav. Introduction to Convolutional Neural Networks (CNN). ANALYTICS VIDHYA. *Analytics Vidhya* [online]. [cit. 2024-05-06]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/

[49] PYTHON SOFTWARE FOUNDATION. *Welcome to Python.org* [online]. [cit. 2024-04-30]. Dostupné z: https://www.python.org

[50] *C++ vs. Python for Machine Learning: Which Language Should You Choose?* [online]. [cit. 2024-05-09]. Dostupné z: https://www.linkedin.com/pulse/c-vs-python-machine-learning-which-language-should-you/

[51] GlobalInterpreterLock. In: *Python Wiki* [online]. [cit. 2024-05-09]. Dostupné z: https://wiki.python.org/moin/GlobalInterpreterLock

[52] The Python Standard Library. *Python Docs* [online]. [cit. 2024-05-09]. Dostupné z: https://docs.python.org/3/library/index.html

[53] [online]. [cit. 2024-05-09]. Dostupné z: https://www.pygame.org

[54] JETBRAINS. *PyCharm* [online]. [cit. 2024-05-09]. Dostupné z: https://www.jetbrains.com/pycharm/

[55] OPENAI LP. *ChatGPT* [online]. [2022] [cit. 2024-05-10]. Dostupné z: https://chatgpt.com

[56] ChatGPT: Model jazykových odpovědí. OPENAI LP. *ChatGPT* [online]. [2022] [cit. 2024-05-10]. Dostupné z: https://chat.openai.com/share/d0316e59-5014-40c6-9d69-cc513f483566

[57] Minutová říše. In: *Zatrolené hry* [online]. [cit. 2024-05-10]. Dostupné z: https://www.zatrolene-hry.cz/spolecenska-hra/minutova-rise-3400/

[58] MINDOK. *Oficiální pravidla Mindok* [online]. Dostupné také z:

https://files.zatrolene-hry.cz/11f76f5a3da93eca2854def41f92c315.pdf

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AI	Artificial inteligence
UI	Umělá inteligence
MTCS	Monte Carlo Tree Search
UCT	Upper Confidence bounds applied to Trees
UCB1	Upper Confidence Bound, version 1
LH	Lidská hráč
SH	Simulovaný hráč

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

Příloha P I: USB flash disk obsahující zdrojové soubory kódu + PDF soubor bakalářské práce

PŘÍLOHA P I: STRUKTURA PŘÍLOŽENÉHO USB FLASH DISKU

Přiložený USB flash disk osahuje:

- Bakalářskou práci ve formátu .pdf: BP_LukasOndrej_2024.pdf
- Zdrojový kód projektu ve formátu .zip BP-MinutovaRise.zip