VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

HRANÍ STOLNÍ HRY STRATEGO POČÍTAČEM

PLAYING THE BOARD GAME STRATEGO BY COMPUTER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **DOMINIK IROVSKÝ**
AUTHOR

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

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

147966

Ústav:	Ústav inteligentních systémů (UITS)
Student:	**Irovský Dominik**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Hraní stolní hry Stratego počítačem**
Kategorie:	Umělá inteligence
Akademický rok:	2022/23

Zadání:

1. Seznamte se s pravidly stolní hry Stratego a s metodami počítačového hraní her.
2. Navrhněte metody na základě známých metod hraní her s neúplnou informací pro hraní této hry.
3. Implementujte zvolené metody a ověřte fungování takto vytvořeného systému.
4. Vyhodnoťte schopnost tohoto systému při hře proti lidskému hráči nebo proti jiným počítačovým realizacím, pokud takové systémy naleznete.

Literatura:

- Smith, S.: Learning to Play Stratego with Convolutional Neural Networks, Stanford University, 2015
- Stratego programming, online [3.10.2021] https://home.hccnet.nl/jabcwolf/stratego/links.html

Při obhajobě semestrální části projektu je požadováno:
První dva body zadání

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

Vedoucí práce:	**Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu:	Hanáček Petr, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	31.7.2023
Datum schválení:	3.11.2022

Abstrakt

Tématem této práce je desková hra s neúplnou informací Stratego. Cílem je průzkum a zhodnocení dosavadních řešení hraní počítačem, návrh, implementace a testovaní vlastního řešení hraní počítačem. Pro vlastní řešení byly zvoleny modifikovaný algoritmus Monte Carlo Tree Search, algoritmus alfa-beta a expectimax. Řešení bylo realizováno jako konzolová aplikace s možností rozšíření. Funkcionalita implementace byla validována a otestována pomocí experimentů. Efektivita výsledného algoritmu byla uspokojivá.

Abstract

The topic of this thesis is the board game of Stratego. This game features incomplete information. The goal of this thesis is research of existing game playing algorithms and, design and implementation of new solution. For the new solution modified version of Monte Carlo Tree Search as well as alfa-beta algorithm and expectimax were used. The solution was implemented as a console application with possibility of future expansion. Functionality of the solution was validated and tested using experiments. Effectivity of the final algorithm was satisfying

Klíčová slova

Stratego, MCTS, Monte Carlo, Monte Carlo Tree Search, alfa-beta, expectimax, umělá inteligence, teorie her, heuristika, hry s neúplnou informací, neúplná informace, Maršál a špión

Keywords

Stratego MCTS, Monte Carlo, Monte Carlo Tree Search, alfa-beta, expectimax, artificial intelligence, game theory, heuristics, games with incomplete information, incomplete information

Citace

IROVSKÝ, Dominik. *Hraní stolní hry Stratego počítačem.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. František Zbořil, Ph.D.

Hraní stolní hry Stratego počítačem

Prohlášení

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

.....
Dominik Irovský
31. července 2023

Poděkování

Rád bych poděkoval vedoucímu práce, Doc. Ing. Františku Zbořilovi, Ph.D., za odborné konzultace, zpětnou vazbu a vstřícný přístup.

Obsah

Kapitola 1

Úvod

Náplní této práce je desková hra Stratego v Česku lépe známá jako Maršál a špión. Jedná se o deskovou hru pro dva hráče s maticovým herním polem a různými typy figurek. Jeden z prvků této hry je neúplná informace. Neúplnou informaci si můžeme představit tak jako hru šachu, kde nevidíme hodnoty oponentových figurek dokud na ně nezaútočíme. To vytváří v této hře element nejistoty, možnost blafování či jiných strategií.

Pro toto téma jsem se rozhodl, protože inteligentní systémy jsou fascinujícím oddílem informatiky a ve volném čase rád hraji jak klasické deskové hry i počítačové hry. Rozhodl jsem se tyto dvě témata tedy spojit. Vybral jsem si hru Stratego, protože není tolik známá, má jednoduchá pravidla, ale zároveň poskytuje velký prostor pro rozvoj složitějších strategií a zlepšování se ve hraní.

Oblast hraní deskových her je populárním tématem různých prací a výzkumů, většina se ale zabývá hrami s úplnou informací tj. kde v jakýkoli moment hry můžeme zhodnotit stav hry pohledem na herní desku. Hry s neúplnou informací nejsou tolik populární. Proto jsem chtěl jít proti tomuto trendu a přispět do této oblasti. Jak k obecnému hraní her s neúplnou informací tak Strategu.

Cílem této práce je vytvořit algoritmus, který bude schopný samostatné hry proti lidskému hráči. Pro vytvoření úspěšného algoritmu je potřeba se seznámit s pravidly hry, taktikami, průzkum již existujících řešení, návrh vlastních algoritmů a implementaci tak, aby bylo možné proti vytvořenému programu hrát a otestovat sílu vytvořeného řešení.

Několik řešení hraní Stratega počítačem již existuje, jsou různě pokročilé a používají různé metody, od klasických herních algoritmů po strojové učení. Existující řešení budou podrobně probrána v kapitole 4.

Kapitola 2

Stratego

V této kapitole bude podrobně popsána desková hra Stratego, její pravidla, výskyt neurčitosti, taktiky a strategie. Stratego je strategická desková hra pro dva hráče vzdáleně se podobající šachům. Na rozdíl od šachů ale vyžaduje dobrou paměť kvůli zakrytým hodnotám figurek, Stratego má pravděpodobně větší potenciál pro pokročilé strategie 'high skill ceiling' než dříve zmíněné šachy, protože tato desková hra není tolik populární a je možné, že dosud existují neobjevené strategie hry.

2.1 Deskové hry s rozsáhlými stavovými prostory

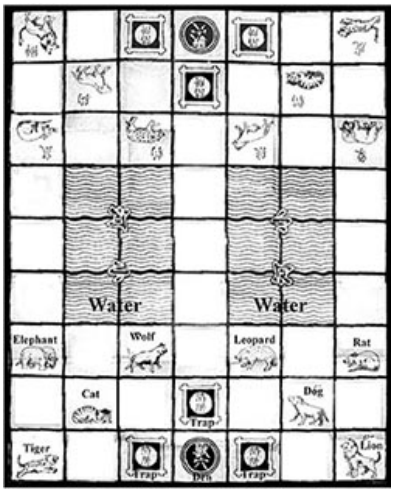
¨ Stratego není jedinou hrou s rozsáhlým stavovým prostorem. O tento typ her se v poslední době zvětšuje zájem v oblasti umělé inteligence a tvorby herních algoritmů. Tyto hry poskytují výzvu pro dosavadní algoritmy a implementace umělé inteligence. Jednou z takovýchto her je Go.

Go

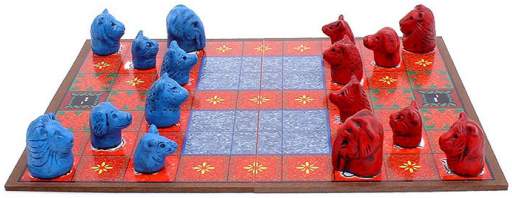
Go je abstraktní strategická desková hra pro dva hráče, kde cílem je zabrat co největší teritorium a nenechat se obklopit oponentem. Přestože hra má velmi jednoduchá pravidla, tak jako Stratego, je velmi komplexní. Hra Go je v oblasti umělé inteligence známá, protože dlouho neexistoval efektivní algoritmus pro hraní této hry strojem. Velké popularity a medializace se dočkal v roce 2017 projekt AlphaGo vedený firmou Google DeepMind. Tento projekt využil kombinací deep learningu (typ strojového učení) a algoritmu Monte Carlo Tree Search. Tento projekt byl schopen porazit nejlepšího lidského hráče v čtyřech z pěti her a popularizoval umělou inteligenci širšímu publiku. [2]

Scotland Yard

Scotland Yard je desková hra s neúplnou informací, kde dva týmy hrají proti sobě. Agenti a zločinec (Mr. X), agenti se snaží chytit zločince. Neúplná informace v této hře se vyskytuje v podobě, skrytí lokace mistra X a jeho lokace je odhalena pouze periodicky. Herní figurky se hýbou po herní desce, která se skládá ze zastávek veřejné dopravy v Londýně, pro pohyb hráči používají omezený počet jízdenek. Hra končí dopadením mistra X, stoupnutím na stejné pole nebo zablokováním všech jeho únikových cest. Mistr X vítězí v momentě kdy agentům dojdou jízdenky. [14]



Obrázek 2.1: Zvířecí šachy diagram. Obrázek převzán z [13]



Obrázek 2.2: Zvířecí šachy. Obrázek převzán z [13]

2.2 Historie Stratega

Desková hra Stratego byla poprvé vytvořena před rokem 1942 v Nizozemí a toho roku byla registrována pod ochrannou známkou Nizozemskou společností Van Perlestein & Roeper Bosch NV. Práva na tuto hru byla poté různě prodávána jak rostla její popularita a nakonec skončila v USA ve vlastnictví firmy Hasbro od roku 1984. Nejedná se ale o naprosto originální nápad, velmi podobné deskové hry lze najít mnohem dříve, jednou z nejstarších jsou japonské vojenské šachy, které lze datovat do roku 1895. Liší se v několika aspektech, ve hře je potřeba rozhodčí, který vyhodnocuje útoky na figurky, vlajka je umístěna ve velitelství, které je potřeba okupovat, aby hráč zvítězil, figurka průzkumník (scout) neexistuje a další. [13]

Další podobnou hrou je tradiční čínská desková hra Džungle (Jungle) také známá pod názvem Zvířecí šachy (Animal chess), obrázky 2.1, 2.2 v originále Dou Shou Qi. Herní figurky v této hře nejsou vojáci nýbrž zvířata, startovní pozice figurek je fixní, tedy každou hru stejná a hodnoty figurek jsou po celou dobu hry odhalené. [13]

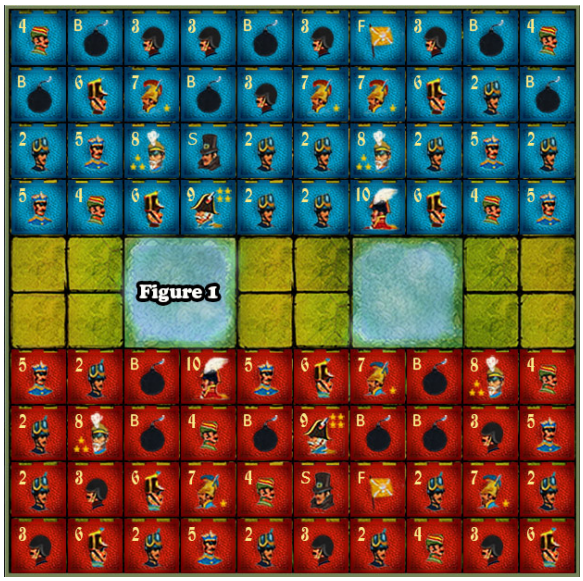
Jako poslední hru podobnou Strategu, která se objevila dříve zmíním francouzskou deskovou hru L'Attaque, viz obrázek 2.3, 2.4. Hra se poprvé objevuje okolo roku 1910 a je nejvíce podobná Strategu. Jediným větším rozdílem je herní pole o rozměru 9 x 10 a počet figurek každého hráče 36, jinak tak jako Stratego mají figurky zakryté hodnoty, odhalí se pouze při útoku a hra končí zabráním oponentovy vlajky. První počítačová verze této hry byla uvedena v roce 1990 firmou Accolade, hra byla založena na Microsoft DOS. Hra obsahovala velmi jednoduchou umělou inteligenci proti které mohl hráč hrát, dělala ale velké taktické chyby. Druhá počítačová verze vyšla roku 1998 pro Windows 95/98, vyvinuta firmou Hasbro a umožňovala hru dvou hráčů přes internet.[13]

2.3 Pravidla

Hra obsahuje jednu herní plochu o velikosti 10 x 10, 40 figurek pro červeného hráče a 40 figurek pro modrého hráče. Herní plocha obsahuje dvě jezera o rozměru 2 x 2 na které nelze vstoupit ani je překročit, fungují jako bariéra. Před začátkem samotné hry je potřeba

Obrázek 2.3: L'Attaque figurky. Obrázek převzán z [13]

Obrázek 2.4: L'Attaque herrní deska. Obrázek převzán z [13]



Obrázek 2.5: Počáteční rozestavění. Obrázek převzán z [15]

figurky rozestavět, hráč může libovolně naskládat svých 40 figurek na do prvních čtyř řad ze své strany desky a vytvořit tak 4 x 10 útvar. Výsledné rozestavění pak může vypadat takto, viz obrázek 2.5. Hru začíná červený hráč. Každý hráč táhne jednou figurkou a předává tah protihráči. Hráč musí během svého tahu táhnout figurkou neexistuje možnost přeskočení tahu. Hra končí pokud hráčova vlajka je zabrána, vlajku může zabrat jakákoli pohyblivá figurka nebo pokud hráčovi dojdou pohyblivé figurky a nemůže tedy provést tah. [15]

Všechny pohyblivé figurky se mohou pohybovat o jedno pole směrem nahoru, dolu, doleva nebo doprava. Diagonální pohyb není možný, viz obrázek 2.6. Výjimkou je figurka

Obrázek 2.6: Pohyb figurky. Obrázek převzán z [15]

Jméno	Jméno česky	Hodnota	Počet	Vlastnosti
Marshal	Maršál	10	1	Může být poražen špiónem pokud špión zaútočí první
General	Generál	9	1	
Colonel	Plukovník	8	2	
Major	Major	7	3	
Captain	Kapitán	6	4	
Lieutenant	Poručík	5	4	
Sergeant	Seržant	4	4	
Miner	Minér	3	5	Může zneškodnit bomy / miny
Scout	Skaut / Průzkumník	2	8	Může se pohybovat o libovolný počet polí, přičemž nesmí přeskočit žádnou figurku
Spy	Špión	1	1	Může porazit Maršála pokud zaútočí první
Bomb	Mina / Bomba	B	6	Nepohyblivá figurka, vyhrává proti všem kromě Minéra
Flag	Vlajka / Prapor	F	1	Nepohyblivá figurka, jejím zabráním končí hra, Zabrat ji může jakákoli pohyblivá figurka

Tabulka 2.1: Počet a typ herních figurek

průzkumníka (scout), který se může pohybovat ve stejných směrech jako ostatní, ale o libovolný počet polí. Figurka nemůže na pole vstoupit pokud je okupované figurkou stejné barvy. Vstoupením na pole s figurkou opačné barvy se útočí. Oba hráči odhalí hodnost své figurky, vyšší hodnost vyhrává a nižší je odstraněna z herní desky. Pokud útočící figurka vyhrála postaví se na pole na které útočila, pokud vyhrál obránce, zůstává stát na svém místě. V případě kdy obě figurky mají stejnou hodnost jsou obě odstraněny z herní desky. Dále jsou ve hře dvě pravidla která mají předejít zbytečnému prodlužování herní doby a to že žádná figurka se nesmí pohnout tam a zpět mezi dvěma stejnými poli více jak 3 tahy po sobě. Druhé pravidlo je, že se figurky nesmí nekonečně honit po herní desce.

Hodnoty a počet figurek každého hráče jsou v tabulce 2.1:

Existují různé variace na pravidla nebo úpravy pro turnaje například výhoda agrese (aggressor advantage) kdy pokud se potkají dvě figurky stejné hodnoty vyhrává ta co útočila místo prohry obou, dalším pravidlem je tichá obrana kdy při útoku odhalí hodnotu figurky pouze útočník a obránce rozhodne zda vyhrál nebo prohrál s výjimkou průzkumníka který odhalí hodnotu obránce. Posledním turnajovým pravidlem je záchrana (rescue), pokud se jakoukoli figurkou, vyjma průzkumníka, dostanu ze svého pohledu do poslední řady herní plochy mohu navrátit do hry libovolnou vyhozenou figurku.

Na závěr této kapitoly bych dodal, že pravidla hry Stratego nejsou nijak pevně stanovena, liší v závislosti kdy daná edice vyšla a která firma ji vydala. Pro lepší přehlednost a snazší pochopení budou použita pravidla z klasického Stratega.

2.4 Taktiky a strategie hraní

Tempo hry Stratego je poměrně pomalé a herní pole velké, k tomu 12 různých typů figurek a není divu, že se v této hře otevírá prostor pro velký počet různých strategií a taktik. V tom dále napomáhá neurčitost, která otevírá možnost blafování a šanci oklamat oponenta. Tato sekce se bude zabývat nejčastějšími taktikami a tipy na výhru.

První taktika která se používá v drtivé většině her je umístění vlajky do poslední řady. Není to povinné, ale je rozumné vlajku bránit za každou cenu, protože pokud si ji hráč nechá zabrat okamžitě prohrál hru.

Další velmi používanou taktikou ve Strategu je obklopení své vlajky bombami, jak vidno na obrázcích 2.7 a 2.8. Tímto způsobem je zajištěno, že pro zabraní vlajky je nezbytně nutný minér, tím pádem pokud se jedné straně během hry podaří vyřadit všechny oponentovy miněry muže již oponent vyhrát pouze zabraním všech pohyblivých figurek. Také se tímto způsobem dá předejít nečekanému zabraní vlajky z velké vzdálenosti pomocí průzkumníka. Dvě metody jak rozmístit bomby a vlajku je buď do rohu, kde nám stačí dvě bomby na obklopení nebo kamkoli jinam do poslední řady herního pole, kde budou třeba bomby tři. Dobré rozprostření hodností na desce je velmi dobrou taktikou, není dobré mít figurky stejných hodností pohromadě, není to takticky nějak výhodné a hrozí poté, že pokud se do této oblasti dostane nepřátelská figurka vyšší hodnosti může velmi rychle zneškodnit celou skupinu.

Schovávání špióna. Je důležité skrývat pozici svého špiona co nejdéle. Pokud soupeř správně odhadne pozici špióna nebo hůře, podaří se mu špióna zabrat, stane se jeho maršál mnohem větším nebezpečím. V této situaci může maršála ohrozit už jen další maršál.

Ponechání několik minérů v zadních liniích pro pozdější fázi hry. Důvodem pro tento postup je fakt, že v pozdější fázi hry jsou pravděpodobně identifikovány pozice většiny bomb a je snazší zabrat miněry, proto není špatný nápad nechat většinu minérů v zadních dvou řadách.

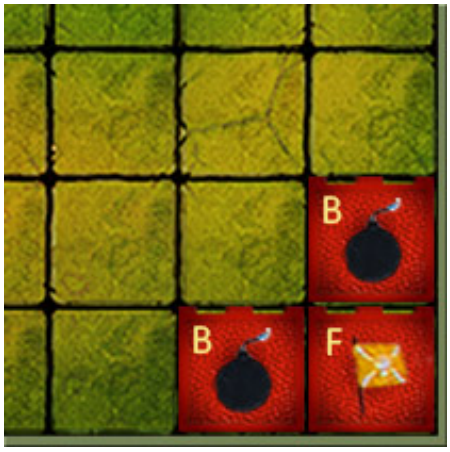
Postavení maršála a generála dále od sebe na herní ploše, není potřeba mít obě figurky s nejvyšší hodností blízko u sebe ale raději je rozprostřít ať na každé straně desky je jedna velmi silná figurka

Sledovaní pohybu oponentových figurek, po poměrné krátké době lze dobře odhadnou kde nepřítel ukrývá svoji vlajku a bomby, jednoduše všímáním si, které figurky se ještě nepohnuly, tady se také naskýtá možnost blafování a schválně skupinou některých figurek nehýbat a předstírat tak, že jsou vlajkou a bombami.

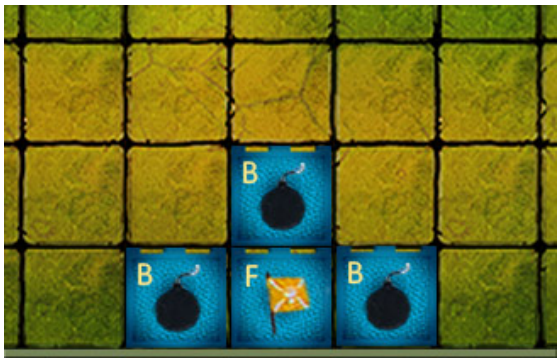
Průzkumníci v první linii, je jedna z taktik která obětuje většinu průzkumníků na začátku hry pro rychlé odhalení oponentových figurek a získaní vitální informace. Je třeba ale myslet na pozdější fáze hry a nějaké průzkumníky si ponechat v záloze. Průzkumníci jsou tehdy silnější, díky více volného místa mohou začít naplno využívat svůj potenciál pohybu o libovolný počet polí. Také se může stát, že oponent si bude šetřit ke konci hry silné figurky a bránit jimi vlajku, v této situaci se průzkumníci hodí pro odhalení přesných hodností figurek.

Maršál, generál a částečně plukovník jsou důležité figurky a není rozumné o ně přijít v počátečních fázích hry. Těmito figurkami by mělo být útočeno pouze v případě, kdy víme že, nepřátelská figurka se již pohnula, tím pádem víme, že se nejedná o bombu. Nebo pokud jsme si téměř jistí, že figurka na kterou útočíme má nižší hodnotu. Toto byly běžné taktiky a tipy pro hru Stratega, ale existují i méně tradiční, pro zajímavost je jich pár uvedeno níže.

Blokování cest bombami. Čtyřmi bombami lze zablokovat levou a pravou cestu tak, že jediná průchozí zůstane ta prostřední, viz obrázek 2.10. Tímto způsobem se hráč nemusí



Obrázek 2.7: Vlajka obklopena minami v rohu. Obrázek převzán z [16]



Obrázek 2.8: Vlajka obklopena minami. Obrázek převzán z [16]

Obrázek 2.9: Alternativní obrana vlajky. Obrázek převzán z [12]

Obrázek 2.10: Blokování cest bombami. Obrázek převzán z [12]

obávat útoku ze stran. Na střed hrací desky se umístí figurky s vysokou hodností a po začátku hry hráč rychle postupuje touto zúženou cestou a napadne nic nečekajícího nepřítele.

Obrana vlajky pomocí Seržantů, tato taktika spočívá v obklopení vlajky seržanty a poté následně minami, viz obrázek 2.9. Seržant je poměrně nízká hodnost a proto fakt, že si jej hráč zablokuje nemá velký impakt na hru. Pointa tohoto rozestavení je, že jakmile protihráč identifikuje naší vlajku, pošle minéra, aby zneškodnil její obranu, jakmile zničí první minu z bariéry nebude očekávat, že za bariérou není vlajka ale figurka s přesně o jednu vyšší hodnost a můžeme tak minéra protiútokem zneškodnit. [16] [12]

Kapitola 3

Teorie hraní her počítačem

V této kapitole bude obsažena obecná teorie ke hraní her dvou hráčů. Budou prozkoumány různé metody hraní her, jejich výhody a nevýhody. Po prvních dvou kapitolách bychom měli mít, dobrý přehled o Strategu a obecně o hrách s neúplnou informací. Informace při studiu teorie hraní her a informace o Stratego jsem čerpal z [19] [11]. Nyní je potřeba najít způsob, jakým program bude hrát tuto hru. Nejdříve v této kapitole budou podrobně probrány klasické herní algoritmy a jejich potenciál pro řešení našeho problému. V následující kapitole bude nahlédnuto na již existující řešení a na metody použité v nich.

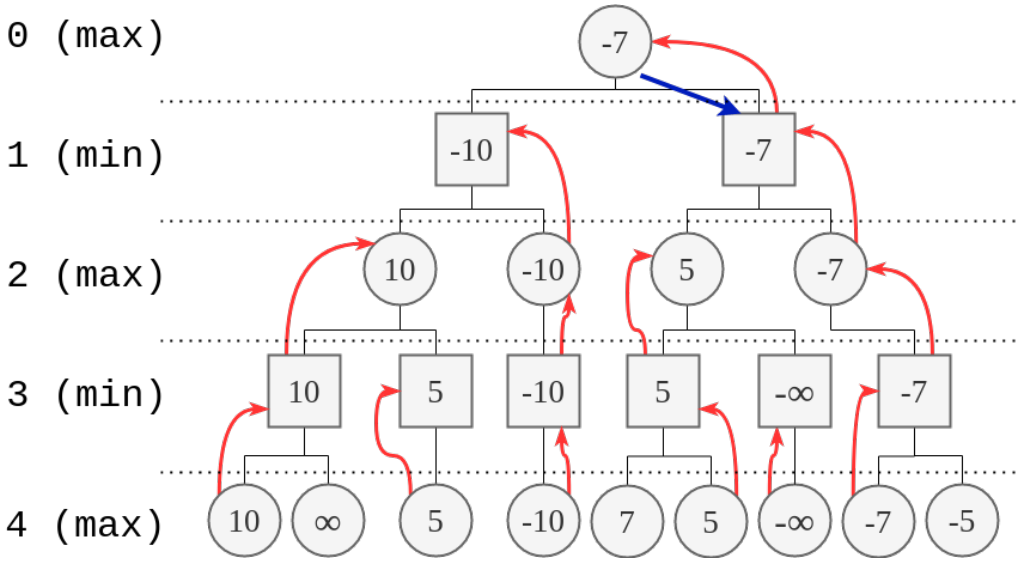
Hraní dvou hráčů lze zjednodušit na prohledávání stavového prostoru. Stavový prostor je množina všech možných kombinací ve kterých se může systém nacházet. Reprezentuje se jako orientovaný graf, uzly představují jednotlivé stavy hry a hrany představují přechod mezi dvěma stavy. Formálně je stavový prostor definovaný jako čtveřice N, A, S, G, kde N představuje množinu všech stavů, A představuje množinu hran, S je neprázdná podmnožina množiny N obsahující počáteční stavy a G je neprázdná podmnožina množina N obsahující výherní stavy. [20]

Cílem metod hraní her je určit tah hráče, který je na řadě tak, aby jeho tah vedl k výhře nebo se přesunul do stavu, který je blíže výhernímu stavu.

3.1 Minimax

Tento odstavec čerpá z opory IZU [20, 10]. Minimax je metoda, jejíž základem je procedura se stejným jménem. Minimax je rekurzivní nebo používá metodu backtracking tj. metoda zpětného vyhledávání, používá se v problémech vyžadujících rozhodování nebo v teorii her, nabízí výběr optimálního tahu s předpokladem, že náš oponent také zahraje optimálně, tj. zahraje nejlepší možný tah. Minimax používá rekurzi k prohledávání stavové prostoru v našem případě herního stromu, obrázek 3.1. Vstupními hodnotami algoritmu jsou aktuální stav hry, maximální hloubka do které budeme prohledávat a informace který hráč je nyní na tahu. Algoritmus minimax je nejčastěji používán při vývoji umělé inteligence pro hry, typicky byl použit na hry jako šachy, piškvorky nebo dáma, Go a jiné hry pro dva hráče s nulovým součtem. V tomto algoritmu tedy hrají dva hráči, jeden je MIN a druhý je MAX, oba dva hráči se snaží zahrát takový tah, aby z něj oponent měl minimální benefit zatím co oni měli maximální benefit. Vlastnosti algoritmu Minimax jsou:

- Úplnost - pokud existuje řešení, algoritmus jej určitě najde
- Optimálnost - minimax je optimální pokud oba hráči hrají optimálně



Obrázek 3.1: Minimax algoritmus, hráč na tahu je uveden vlevo, uvnitř uzlů jsou jejich ohodnocení

- Časová náročnost - minimax provádí v podstatě prohledávání do hloubky proto jeho časová náročnost je $O(b^m)$, kde b je faktor větvení (počet dětí nelistových uzlů), a m je maximální hloubka stromu

Postup procedury minimax je následující:

1. Pokud je uzel listem, to znamená, je koncovým stavem hry nebo pokud jsme dosáhli maximální hloubky prohledávání stanovené vstupním parametrem, vrať ohodnocení tohoto uzlu
2. Pokud je na tahu hráč Max tak pro všechny jeho možné stavy volá Minimax pro hráče Min a vrací maximální z navrácených hodnot a tah, který vede k nejlépe ohodnocenému následníkovi
3. Pokud je na tahu hráč Min tak pro všechny možné tahy volá Minimax pro hráče Max a vrací minimální navrácenou hodnotu

Výhody a nevýhody algoritmu

Hlavní nevýhodou algoritmu Minimax je fakt, že prohledává zbytečně mnoho stavů/uzlů, algoritmus prohledává všechny možné stavy, v menších hrách jako piškvorky či dáma to není problém, ale u jakékoli hry co má velký větvící faktor to působí potíže. Tato limitace lze vyřešit pomocí metody alfa-beta ořezávání. Výhodou algoritmu je jeho jednoduchost.

3.2 Alfa beta

Tento odstavec čerpá z opory IZU [20, 10]. Alfa-beta ořezávání je modifikace minimax algoritmu, jedná se o optimalizační techniku. Alfa-beta řezy zabraňují zbytečnému prohledávání všech uzlů herního stromu, viz obrázek 3.2. Tím že se jedná o modifikaci minimaxu jsou

si tyto dvě metody velmi podobné. Vstupními parametry jsou opět stav hry, maximální hloubka do které prohledáváme herní strom a informace který hráč je právě na tahu, navíc jsou dva nové parametry, alfa a beta. Tyto dva parametry lze definovat jako:

1. Alfa je nejlepší (má nejvyšší hodnotu) volba jakou jsme zatím během hry našli pro hráče Max, počáteční hodnota je -∞
2. Beta je nejlepší (má nejnižší hodnotu) volba jakou jsme zatím během hry našli pro hráče Min, počáteční hodnota je +∞

Alfa-beta ořezávání vrací stejné hodnoty jako klasický minimax s tím rozdílem, že alfa-beta pouze eliminuje uzly, které nemají na finální výsledek vliv. Snížením počtu kontrol uzlů se tento algoritmus výrazně urychluje.

Procedura alfa-beta

1. Pokud je uzel listem, tedy konečným stavem nebo uzlem v maximální hloubce, vrať ohodnocení uzlu
2. Tah hráče A
   a) Dokud je $\alpha<\beta$, tak pro první nebo další tah volá proceduru alfa-beta s aktuálními hodnotami α a β. Po každém vyšetření tahu nastaví hodnotu parametru α na maximum z aktuální a navrácené hodnoty.
   b) Pokud je $\alpha \geq \beta$ nebo pokud uzel nemá žádné děti, vrať hodnotu parametru α a tah vedoucí k nejlepšímu následníkovi
3. Tah hráče B
   a) Dokud platí $\alpha<\beta$ tak pro další tah volá proceduru alfa-beta s aktuální hodnotou parametrů α a β. Po každém vyšetření tahu nastaví hodnotu β na minimum z aktuální a navrácené hodnoty
   b) Pokud je $\alpha \geq \beta$ nebo pokud uzel nemá žádné děti, vrať aktuální hodnotu β

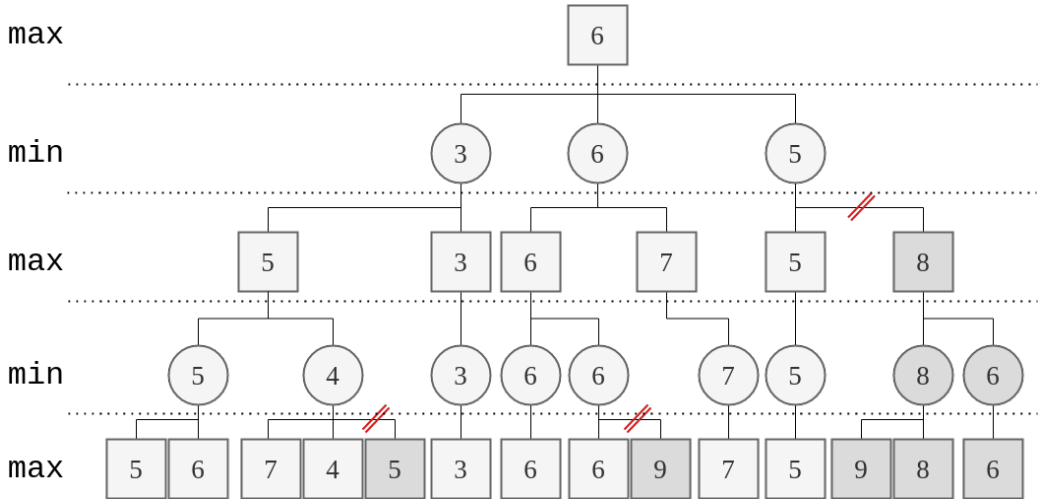
Klíčové informace na které během provádění tohoto algoritmu je dobré pamatovat. Hráč Max může aktualizovat pouze hodnotu α, hráč Min může aktualizovat pouze hodnotu β a hlavní podmínka pro ořez je $\alpha \geq \beta$

Efektivita alfa-beta algoritmu silně závisí na pořadí v kterém jsou uzly vyšetřovány. V nejlepším případě, kdy nastává velké množství ořezů a nejlepší tahy jsou v levé části herního stromu je možné dosáhnout časové náročnosti $O(b^{m/2})$ exponent je poloviční oproti klasickému minimaxu.

Naopak pokud bude náš herní strom mít nejhorší možné seřazení, stane se, že alfa-beta neořeže žádné uzly a bude pracovat stejně jako minimax. Tento algoritmus bude dokonce pomalejší než minimax protože musí pracovat s více proměnnými a operacemi. V tomto případě jsou nejlepší tahy na pravé straně herního stromu. Časová náročnost takového algoritmu bude $O(b^m)$.

3.3 AO* algoritmus

Tento odstavec čerpá z opory IZU [20, 10] . Algoritmus AO* je informovaná vyhledávací metoda a jedná se o modifikaci algoritmu AO, který je neinformovaný. Algoritmus AO*



Obrázek 3.2: příklad Alfa-beta ořezávání, ořezané uzly jsou v šedé

je založený na dekompozici problému tj. rozložení problému na menší části. Když problém lze rozdělit na podproblémy, kde každý z těchto menších problémů má své vlastní řešení, lze je vyhodnotit a poskládat z nich celkové řešení problému. Pro reprezentaci těchto typů problému se používají AND-OR stromy. Pokud lze podproblémy ohodnotit pomocí nějaké heuristické funkce, lze poté v uzlech přepínat mezi mezi řešenými problémy (OR funkce) a vyřešit tak nejsnáze daný podstrom. Řešitelné uzly se označuji 'Solved' a neřešitelné uzly jako 'Futility'. Algoritmus pracuje následujícím způsobem:

1. Sestroj prázdný strom a vlož do něj uzel INIT spolu s jeho ohodnocením. Stanov Hodnotu Futility, která určuje maximální cenu řešení
2. Pokud je uzel INIT označen jako Solved, ukonči řešení jako úspěšné, pokud je hodnota INIT větší nebo rovna hodnotě Futility, ukonči řešení jako neúspěšné, jinak pokračuj
3. Procházej nejslibnějším podstromem dokud nenajdeš neexpandovaný uzel, tento uzel označ NODE
4. Expanduj uzel NODE, pokud NODE nemá žádné děti, přiřaďte mu hodnotu Futility a jděte na bod 7, jinak pokřačuj bodem 5
5. Pokud jsou nějaké děti elementární úlohy, označ je jako Solved, u zbývajících hodnoty odhadněte
6. Připoj děti uzlu NODE ke stromu a přenes jejich ohodnocení směrem nahoru k uzlu INIT tímto způsobem:
   - ohodnocení uzlů AND je rovno součtu ohodnocení všech potomků
   - ohodnocení uzlů OR je rovno nejnižšímu ohodnocení ze všech potomků
7. Ve všech uzlech OR označ nejnadějnější podstromy = hrany s nejlépe ohodnoceným bezprostředním potomkem
8. Běž zpět na krok 2

Výhodami algoritmu AO* je úplnost a menší nároky na paměť, nevýhodou je neoptimálnost, jakmile najde jedno řešení dále už nehledá.

3.4 Monte Carlo Tree Search

Monte Carlo Tree Search je heuristický vyhledávací algoritmus, kombinuje znalosti z klasické prohledávání stavového stromu a principy ze strojového učení, konkrétně posilované učení. Jeho časté užití je právě v programech hrajících deskové hry. V posledních letech tato metoda narostla na popularitě díky úspěchům v oblasti hraní deskové hry Go a potažmo projektu AlphaGo, kde v minulosti běžné herní algoritmy jako minimax nebo alfa-beta ztroskotaly, zvláště kvůli vysoké náročnosti předpovězení výhry z počátečního neterminálního stavu. Výhodami oproti klasickému minimaxu jsou například: [17, 18]

- Aby hloubkově omezený minimax rozeznal dobrý tah od špatného potřebuje funkce k vyhodnocení stavu herní desky. Tato heuristická funkce může být složitá na sestavení ve složitých hrách jako Go. Monte Carlo Tree Search tyto ohodnocovací funkce ke správnému fungování nepotřebuje, jediné co MCTS potřebuje znát jsou pravidla hry, aby mohlo generovat validní tahy
- MCTS se dokáže efektivně vypořádat s deskovými hrami, které mají vysoký faktor větvení. Čím déle algoritmus pracuje a čím více informací nasbírá tím častěji se rozhoduje pro tahy, které se zdají být dobré
- Minimax musí doběhnout až do konce aby nám vrátil nejlepší řešení, což jej dělá časové neflexibilním, pro hry s velkým stavovým prostorem jako šachy nebo Go může být toto prohledávání nekontrolovatelné. Monte Carlo Tree Search nemusí prohledávat celý stavový prostor, iterace se dá kdykoli zastavit a algoritmus poskytne nejlepší možné řečení jaké do té doby našel, proto čím déle běží tím silnější řešení poskytne. Díky tomuto je MCTS velmi flexibilní.

MCTS je poměrně nový algoritmus, jeho aplikace byla poprvé popsána v roce 2006 Rémi Coulomem. [5] Metoda Monte Carlo jejíž teorie se v tomto algoritmu využívá se poprvé začala objevovat v 40. letech minulého století.

Procedura Monte Carlo Tree search

Principem metody MCTS je vyhledávání co možná nejsilnějšího řešení pomocí expanze herního stromu na základě náhodného vzorkování stavového prostoru. Implementace MCTS se zakládá na vysokém počtu iterací (rollouts, playouts). V každé iteraci se hra hraje až do úplného konce náhodnými pohyby, odtud plyne pojmenování Monte Carlo. Konečný výsledek hry je poté použit na ohodnocení uzlů v herním stromu takovým způsobem, aby budoucí iteraci častěji volily uzly s dobrým ohodnocením. Jedna iterace metody MCTS se dělí na čtyři základní části. Selekce, expanze, simulace a zpětná propagace, viz obrázek 3.3. [9, 4]

1. Selekce

Prvním krokem při běhu MCTS je selekce. Selekce začíná vždy od kořene stromu. Je nutno zdůraznit, že nyní nemluvíme o herním stromu reprezentujícím celý stavový prostor hry, ale o stromu který si MCTS během iterací samo staví. Kořenem tohoto stromu je tedy stav v kterém nám protihráč po svém tahu herní desku 'odevzdal'. Nemá smysl aby MCTS drželo v paměti celý herní strom, protože by obsahoval velké množství stavů do kterých se již nelze dostat. Od kořene se postupuje směrem k listovým uzlům tak, že se vybírají nejvíce

atraktivní uzly. Pokud na se narazí na uzel, který nemá žádné potomky a nejedná se o terminální uzel, přidáme mu náhodně jednoho z potomků procesem expanze. To které uzly jsou nejvíce atraktivní rozhoduje balancující funkce. Tato funkce řeší problém Exploration vs. Exploitation, objevování versus zužitkování, kdyby selekce pokaždé vybírala pouze uzly s nejlepším ohodnocením tj. uzly dobrý tah např. zabrání figurky, je možné že si nechá ujít tah, který se na první pohled nezdá tak dobrý, ale v později vede k lepšímu stavu hry. Funkce tedy balancuje prohledávání nových uzlů či částí stromu a zužitkování uzlů o kterých jíž víme, že jsou hodnotné. Konkrétně u MCTS se používá funkce UCT - Upper Confidence Bound 1 aplikovaný na stromy která je založená na funkci UCB1. Hodnota uzlu za použití UCT se vypočítá pomocí tohoto vzorce [7]

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$

kde w_i je počet simulací spuštěných z daného uzlu které skončily výhrou, n_i je celkové počet simulací spuštěných z uzlu, N_i je celkový počet simulací spuštěných na rodičovském uzlu a C je konstanta objevování, teoreticky rovna $\sqrt{2}$, prakticky její hodnota je zvolena podle potřeby, některé deskové hry mohou vyžadovat více objevování jiné více zužitkování. Část vzorce nalevo od sčítání představuje komponentu zužitkování, hodnota je vysoká pro stavy s vysokou výhernostní, napravo je komponenta objevování hodnota komponenty je vysoká pokud proběhlo velké množství celkových simulací a uzel byl navštíven málo. Konstantou C lze algoritmus ladit, čím vyšší C zvolíme tím více se bude objevovat. Lze měnit i v průběhu hry, například v pozdějších fázích hry lze hodnotu C snížit aby se dané zdroje více soustředily na zužitkování.

2. Expanze

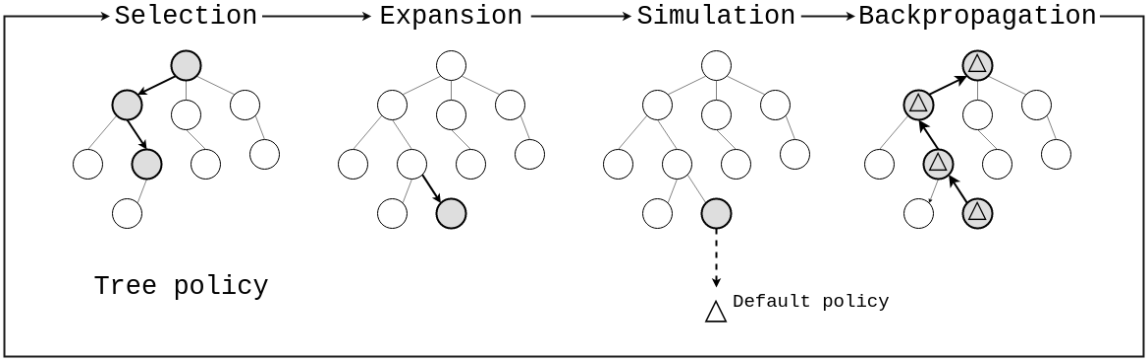
Poté co Selekce v předchozím kroku vybrala vhodný uzel je potřeba tento uzel expandovat. Expanze uzlu je provedena jako vybrání náhodného potomka uzlu ve kterém se nacházíme, na tomto novém uzlu se provede simulace. Náhodný potomek představuje jeden náhodný tah ze stavu v rodičovském uzlu. Vždy expandujeme bezprostředního potomka, tj. o jednu úroveň hloubky stromu níže. Pokud se jedná o uzel terminální (nemá žádné potomky) provede se simulace s tímto uzlem, přičemž se jedná o konec hry takže simulace skončí okamžitě.

3. Simulace

Třetím krokem algoritmu MCTS je simulace. Simulace se provádí na expandovaném uzlu z druhé fáze. Opakovaně jsou voleny náhodné tahy a hraje se až do úplného konce, kde se zjistí kdo vyhrál. Jednoduše se zjistí všechny možné platné tahy, jeden se náhodně vybere a stav hry se posune o jednu hlouběji ve vyhledávacím stromu. V této fázi se nevytváří žádné nové uzly. Pro připomínku uzly ve vyhledávacím stromu odpovídají jednu ke jedné se uzly herního stromu. Žádná část simulace se neukládá důležité je jen kdo na konci vyhrál.

4. Zpětná propagace

Po dokončení simulační fáze a zjištění kdo vyhrál je potřeba aktualizovat hodnoty na všech navštívených uzlech. Propagace probíhá směrem nahoru po vyhledávacím stromě ke kořeni a zastaví se na něm. "Po cestě"u každého navštíveného uzlu se inkrementuje hodnota atributu n_i a podle toho kdo vyhrál a kterému hráči uzel patří se inkrementuje atribut w_i. Toto



Obrázek 3.3: Diagram jedné iterace MCTS

ohodnocování i protihráčových uzlů není bezdůvodné, tímto způsobem se MCTS snaží najít nejlepší možné tahy pro oba hráče tak jako algoritmus Minimax.

Tyto čtyři fáze tedy poběží dokud neprovedou požadovaný počet iterací, nedojde vyhrazený čas nebo zdroje. MCTS poté navrátí uzel s největším počtem simulací n_i a to je nejlepší tah který MCTS doposud našlo.

Výhody a nevýhody

Hlavní výhodou algoritmu MCTS je, že nepotřebuje žádné vyhodnocovací funkce pro herní stavy, pouhá znalost pravidel stačí na efektivní prohledávání stavového prostoru. Druhá velká výhoda je, že se dokáže dobře vypořádat s hrami, které mají vysoký faktor větvení protože růst vyhledávacího stromu je asymetrický díky UCT funkci.

Nevýhodou je, že v některých situacích mohou existovat tahy, které se tváří silně, ale vedou k prohře pokud hra pokračuje velmi specifickým způsobem, těmto situacím se říká "trap states"stavové pasti a profesionální hráči jich mohou být schopni zneužít.

Kapitola 4

Existující řešení

Tato kapitola se bude zbývat již existujícími řešeními, tedy programy hrajícími Stratego, jejich poznatky, použitými postupy a výsledky, které tyto práce dosáhly. Na základě poznatků z předchozích prací a herních algoritmů v předchozí kapitole bude jednodušší navrhnout vlastní řešení. Stratego není velmi známou hrou, ale i tak se stala předmětem zájmu předchozích prací, které se pokusily tuto zdánlivě nenáročnou hru s neúplnou informací efektivně hrát.

4.1 Invicible a Stratego Bot

Tato sekce čerpá informace z práce [3]. Invincible je název umělé inteligence pro hrání Stratega z referencovaného článku. Jedná se o rozsáhlou diplomovou práci, kde auto vytvořil silnou umělou inteligenci kombinací originálních přístupů a ověřených algoritmů. Algoritmus této umělé inteligence používá statistické informace získané hrou lidských hráčů pro tvorbu startovních rozestavení figurek na herní desce. Pro hraní hry používá předpovídaní hodností oponentových figurek pomocí 2D pole, kde sloupce představují jednotlivé figurky a řádky jednotlivé hodnosti. Tato matice je poté naplněna pravděpodobnostmi u každé hodnosti tak, aby součet u každé figurky byl roven jedné. Rozhodování o nejlepším možném tahu je rozděleno na několik plánů. Každý plán je samostatně vyhodnocen a nejlepší tah je zvolen kombinací hodnot jednotlivých plánů. Plánem může být například rozhodnutí o zabrání odhalené figurky, nebo obrana vlajky. Kombinace takovýchto malých úkolů vede k celkovému vítězství.

4.2 Multi agentní Stratego

Tato sekce čerpá z práce [6]. V této práci se autor rozhodl implementovat umělou inteligenci pro hru Stratego pomocí agentních systémů. V této implementaci každá figurka ovládaná počítačem je agent. Figurky autor rozděluje na vysoce a nízce ohodnocené agenty, Nízce ohodnocení agenti jsou figurky nízkých hodností a jejich jediným úkolem zabrat nepřátelskou vlajkou, zatímco vysoce ohodnoceni agenti mají za úkol útočit jak na vlajku tak na nepřátelské figurky. Agenti pozorují prostředí kolem sebe a vypočítávají nejlepší možnou akci. Všichni agenti v jednom týmu komunikují s centrálním rozhodovačem a posílají mu jejich individuální nejlepší tah. rozhodovač poté jeden z nich vybere. Agenti v této implementaci vidí dvě pole od své pozice a mají čtyři možné akce, útok, ústup, bloudění a nečinnost.

4.3 DeepNash

Tato sekce čerpá informace z vědecké práce [8]. DeepNash je projekt týmu DeepMind, spadající pod společnost Google, který se zabývá tématem umělé inteligence. Tento tým se v minulosti již vyznamenal projektem AlphaGo, umělé inteligence pro hrání deskové hry Go. Jméno DeepNash je složeninou dvou termínů, deep learning a Nash equilibrium, dvou hlavních metod, které tento tým při implementaci umělé inteligence pro hraní hry Stratego využil. Konkrétní typ deep learningu použit byl "model-free deep reinforcement learning method". To znamená kombinaci hlubokého učení (deep learning) a zpětnovazebního učení (reinforcement learning) bez modelu. Deep learning je obecně kategorie strojového učení, která využívá učení umělých neuronových sítí, inspirovaných zpracováním informací a jejich distribucí v reálných biologických systémech, tj. neurony v mozku. Zpětnovazební učení je oblast strojového učení zabývající se inteligentními agenty a tím jak se mají rozhodovat v daném prostředí. Cílem je naučit agenta co nejlepšímu chování, které maximalizuje užitek (optimal policy) v daném prostředí za pomocí systému odměn a trestů za každý provedený krok. Zpětnovazební učení bez modelu poté znamená, že tento algoritmus si nevytváří explicitní model svého prostředí, ale operuje na principu pokus a omyl, kdy agent zpočátku podniká náhodné kroky a zaznamenává si velikost odměny nebo trestu. Příkladem takového algoritmu je Q-learning.

Druhou hlavní metodou toho projektu je již zmiňovaný Nash equilibrium. Jedná se o koncept v teorii her, který popisuje situaci, kdy všichni hráči si vybrali nejlepší možnou strategii v závislosti na strategiích zvolených oponenty. V této situaci, ekvilibriu, nemá smysl pro hráče měnit strategii, protože by nic nezískal. Je možné, že existuje obecně lepší strategie, ale strategie z Nash ekvilibria je momentálně ta nejlepší a maximalizuje zisk proti oponentovi. Je možné, že u dané hry existuje ekvilibrií více, ale i žádné.

Použitím těchto praktik, se týmu DeepMind podařilo vytvořit dosavadně nejlepší umělou inteligenci pro hru Stratego. jedinou která dokáže hrát na expertní úrovni. Při porovnávání DeepNash s ostatními algoritmy a lidskými hráči na on-line platformě pro hraní Stratega Gravon. Proti lidským hráčům na Gravonu měl DeepNash výhernost 84% a umístil se jako třetí nejlepší hráč na platformě. Pří porovnání s ostatními algoritmy měl průměrnou výhernost 99.1 procenta, z toho nejhorší byla 97.1 % proti algoritmu Demon of Ignorance. Jednalo se o neočekávaný výsledek, doposud nikdo nevěřil, že takto schopná umělá inteligence pro hru Stratega je možná, protože Stratego je hra výrazně komplexnější než Go nebo jiné hry s neúplnou informací. Výsledkem tohoto projektu byl překvapen i spoluautor Vincent de Boer, který sám je trojnásobný mistr světa ve Strategu a aktuálně hodnocen jako čtvrtý nejlepší hráč na světě.

Kapitola 5

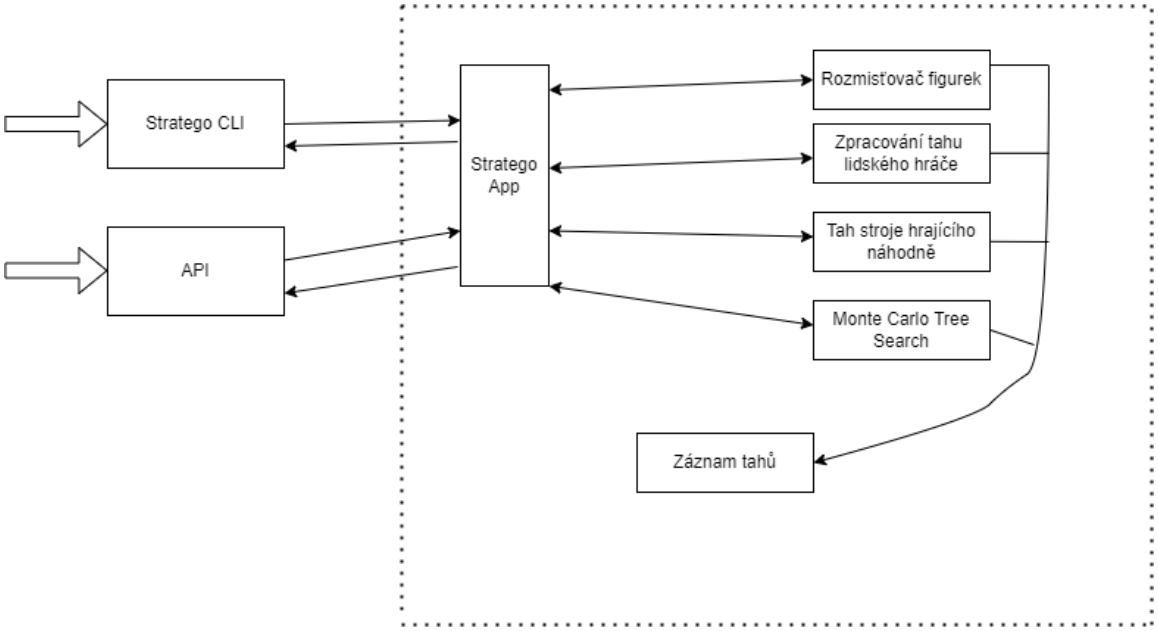
Návrh aplikace

Stratego App bude vyvinuta jako konzolová aplikace. Bude se jednat o implementaci hry Stratego s klasickými pravidly a bude umožňovat různé kombinace her člověka proti počítači. Hlavním důvodem pro tuto volbu je rozšiřitelnost, aplikace bude mít plně modulární design. Aplikace bude plně ovladatelná z konzole, ale její návrh bude umožňovat vytvoření samostatného uživatelského rozhraní. Veškerá logika bude v separátním modulu. Modularita programu také bude umožňovat budoucí přidávání nových algoritmů pro hraní hry. Dalším důvodem pro volbu implementace bez grafického uživatelského rozhraní je, že Stratego je po vizuální stránce jednoduchá hra. Herní deska je matice 10 x 10 a kromě osmi polí s vodou jsou všechny pole stejná. Hrají pouze 2 hráči, červený a modrý, prostředí konzole je tedy dostačující pro hraní hry, to stejné platí pro tažení figurkami, které se bude provádět vybráním figurky první dvojicí souřadnic a tahem pomocí druhé dvojice. Jakékoli neplatné stavy budou ošetřeny a uživatel bude vyzván k opravě vstupních údajů.

Aplikace bude tedy rozdělena do tří hlavních částí, viz obrázek 5.1. StrategoCLI (Command Line Interface), StrategoApp a jednotlivých modulů hráčů. StrategoCLI bude zajišťovat veškerou komunikaci mezi uživatelem a zbytkem aplikace pomocí příkazové řádky. Uživatel bude spouštět aplikaci s parametry, které budou určovat typy hráčů, kteří budou proti sobě hrát a způsob rozestavění figurek v první fázi hry, k dispozici bude manuální rozestavění kdy jednu po jedné figurce je bude hráč pokládat na desku nebo rozestavění automatické. StrategoApp bude obsahovat implementaci hry Stratego, kontrolovat validitu tahů, rozhodovat výsledek útoků a kontrolovat zda někdo neukončil svým tahem hru. Moduly algoritmů pak budou obsahovat jednotlivé implementace herní algoritmů, které zastupují hráče. StrategoApp bude řídit předávání tahu a požádá vždy daného hráče nebo algoritmus o zadaní souřadnic pro provedení tahu. Aplikace nebude neuchovává žádná perzistetní data, nebude tedy poskytovat možnosti jako na příklad pozastavení hry a dohrání později (savefile), ale pokud potřeba, může tato funkcionalita být v budoucnosti jednoduše doimplementována. Součástí implementace bude také zaznamenávání všech provedených tahů úkolů. Bude probíhat automaticky a bude hodnotné jak pro hráče tak pro hledání chyb v algoritmech během samotné implementace.

5.1 Volba algoritmu

Jak bylo v dřívějších kapitolách zmíněno, Stratego je poměrně rozsáhlým tématem, existuje řada různých přístupů ke hraní této hry. Od klasických herních algoritmů jako minimax až po konvoluční neuronové sítě. V této práci se dále budeme zabývat implementací pomocí



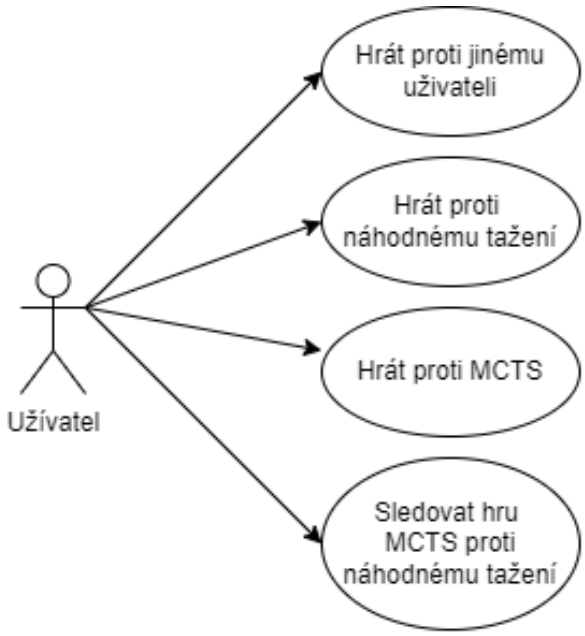
Obrázek 5.1: Návrh struktury aplikace

tří různých algoritmů, Monte Carlo Tree Search, Alfa-Beta ořezávání a Expectimax. Cílem práce je implementovat námi zvolené metody a otestovat ji vůči reálným hráčům, navzájem mezi sebou a ostatním počítačovým implementacím pokud takové existují. Bohužel většina implementací různých algoritmů z akademických publikací není zveřejněna a tedy není pro mě možné reprodukovat jejich výsledky a porovnat funkcionalitu vytvořených algoritmů, to je jeden z důvodů proč jsem se rozhodl implementovat algoritmů více abych mohl porovnat schpnost jednotlivých algoritmů mezi sebou. Na internetu již existují zhotovené algoritmy či umělé inteligence pro hraní hry Stratego, ale bohužel tyto programy jsou webové aplikace a tedy není možné programově napojit tyto aplikace do mé vytvořené Stratego aplikace aby bylo možné je automaticky testovat, možnost ručního testovaní bohužel není realistická kvůli průměrné délce jedné hry, která se pohybuje okolo 45 minut. Rozhodl jsem se tedy že programová část této bakalářské práce bude obsahovat zejména.

- Konzolová aplikace s možností budoucího rozšíření
- Možnost hry dvou lidských hráčů proti sobě
- Hra člověka proti libovolnému algoritmu
- Hra dvou libovolných algoritmů proti sobě

Use Case Diagram

V následujícím diagramu, obrázek 5.2, je výčet případu užití navržené aplikace Jen pro dodání možnost hry lidský hráč proti lidskému hráči bude realizována na stejném stroji nikoli on-line.



Obrázek 5.2: Návrh případů užití aplikace

Kapitola 6

Implementace

Tato kapitola se zaměřuje na programovou implementaci, na problémy nalezené během ní, jejich řešení a dalších poznatků. Aplikace byla vyvina v jazyce Python 3.8. Tak jak bylo v uvedeno v návrhu aplikace se dělí na dvě části, na uživatelské prostředí v příkazové řádce, které zajišťuje komunikaci uživatele s aplikací formou dotaz - odpověď a na veškerou logiku hry ve zbytku programu. Tato modularita umožní navázat na tento projekt pokud by se na něm někdo v budoucnu rozhodl stavět po stránce grafického uživatelského rozhraní, vytvoření výukové aplikace pro učení se Stratega nebo vylepšení stávajících nebo přidání nových algoritmů.

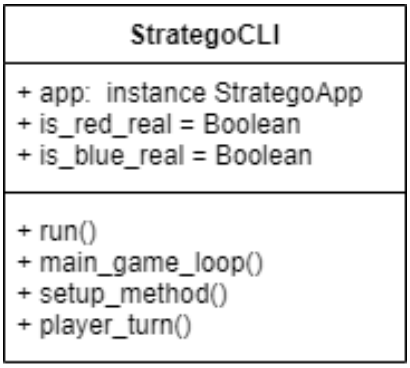
6.1 Implementace základní hry

Dříve než mohla hlavní část této práce, implementace algoritmu Monte Carlo Tree Search, být uskutečněna bylo potřeba implementovat samotnou hru a její pravidla. Po zvážení možnosti omezit velikost herní plochy jsem se rozhodl nezvolit tuto možnost. Ve Strategu herní deska obsahuje sto polí, osm z nich jsou voda a každý hráč má 40 figurek, to znamená, že na začátku hry máme pouze dvanáct prázdných polí. Zmenšení herní plochy by znamenalo, že by byl potřeba omezit počet figurek a přepracovat vodní plocha. Musel by být i zachován poměr hodností figurek což by u některých nebylo možné protože mají jen jeden kus. Celkově by se vyvrátil balanc hry, který autoři této hry zamýšleli, proto bude hra implementována v neredukované formě i za cenu náročnějšího testovaní či implementace algoritmu MCTS.

Jako první věc bylo potřeba vymyslet s jakými parametry se bude konzolová aplikace spouštět. Argumenty pro spuštění vyplývají z diagramu případu užití v předchozí kapitole o návrhu. Aplikace se spouští se třemi parametry:

- Parametr pro typ červeného hráče `-r real` nebo `-r random`
- Parametr pro typ modrého hráče `-b real` nebo `-b random` nebo `-b mcts`
- Název textového souboru pro zápis akcí, které se udály během hry (pokud soubor s tímto názvem neexistuje vytvoří se nový `-l log.txt`

Parametr "real"odpovídá volbě hry člověka, "random"hře stroje pomocí náhodných tahů a "mcts"odpovídá hře stroje pomocí Monte Carlo Tree Search, zde si lze povšimnout, že mcts může hrát pouze za modrou stranu, jedná se o úmyslné rozhodnutí nikoli nedostatek ve fázi návrhu. Jelikož Stratego nabízí symetrickou hru, tj. obě strany mají přístupe ke stejným

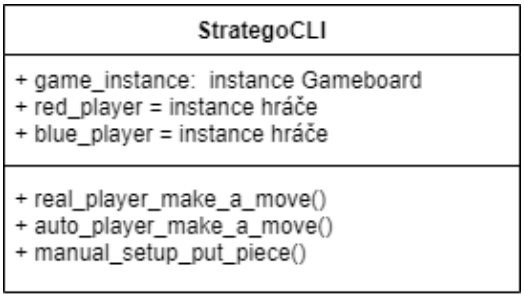


Obrázek 6.1: Třídní diagram StartegoCLI

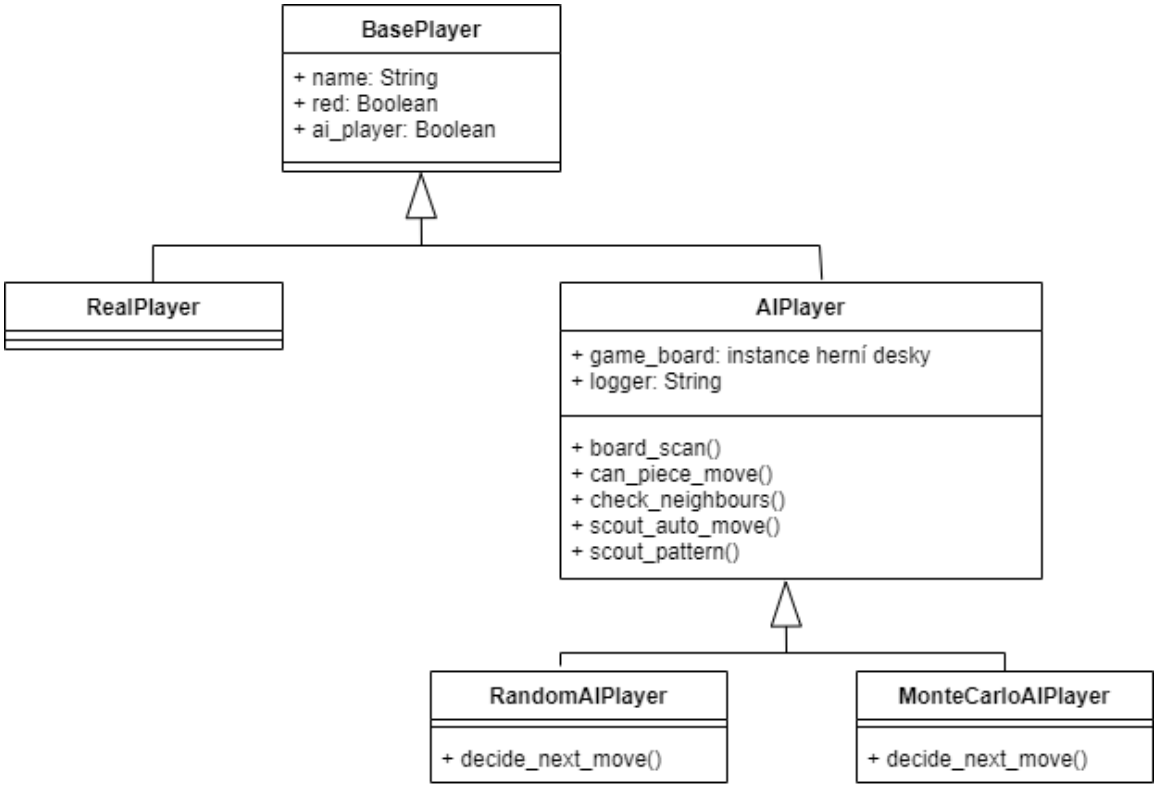
figurkám a herní pole je také symetrické, tím pádem by nemělo smysl nechat proti sobě hrát dvě stejné implementace MCTS, dopředu lze říci, že by výhernost byla 50/50 procent.

Nyní je vhodný moment ukázat jak vypadají některé třídy pro lepší pochopení chodu aplikace. První třídou je StrategoCLI, viz 6.1, které jak již řečeno zajišťuje ovládání v prostředí příkazové řádky. V aplikaci se vyskytuje pouze jedna instance této třídy, python ale neposkytuje možnost tvorby jedináčrků (singletonů) proto je implementována jako klasická třída. Atribut `app` uchovává instanci třídy StrategoApp o které bude více později v této sekci. Atributy `is_red_real` a `is_blue_real` uchovávají informaci o tom zda červený a modrý hráč jsou člověk nebo ne, což se později používá v jedné z metod. V diagramu třídy výše nejsou uvedeny všechny metody této třídy pouze ty které jsou nějakým způsobem zajímavé, například není potřeba podrobně rozebírat jak funguje načítání vstupu z klávesnice. Metoda `run()` je první metodou, která se provede, zavolá pro každého hráče hráče nejdříve metodu `setup_method()` a poté `main_game_loop()` jedná se tedy de facto o startovní bod. Další metodou je `setup_method()` tato metoda má za práci zjistit jakým způsobem budou pokládány herní figurky na herní desku. Byly implementovány dva způsoby manuální, kdy hráč je vždy informován jakou figurku pokládá a odpovídá dvojicí souřadnic. Druhý způsob je automatický, funkce k tomu to je v souboru `PieceSetup.py`, využívá se jedné ze základních taktik a to té, že vlajka je umístěna do poslední řady, sloupec je vybrán náhodně a vlajka je automaticky obklopena bombami. Více automatických taktik nebylo implementováno, i když mají impakt na hru která následuje nelze na tuto fázi nijak použít herní algoritmy, bylo by třeba je použít předem nastavené herní desky a vybírat z nich nebo použít pro rozestavení neuronové sítě a to není tématem této práce. Hlavním středem zájmu je fáze hraní.

Další důležitou třídou je `StrategoApp`, viz 6.2 opět jako třída předchozí máme pouze jednu instanci během chodu aplikace. Atribut `game_instance` představuje instanci třídy herní desky, o té bude více informací níže v této sekci. Atribut `red_player` a `blue_player` jsou objekty představující oba hráče. Metodami v této třídě jsou, `real_player_make_a_move`, která zpracovává tah lidského hráče, převezme zadané souřadnice a předá je dále ke kontrole zda tah je platný a provedení tahu. Metoda `auto_player_make_a_move` je stejná s tím rozdílem, že zpracovává tahy stroje. Nakonec metoda `manual_setup_put_piece`, která obsluhuje manuální pokládání figurek na herní plochu, nedochází zde ale k žadným kontrolám. Celkově třída `StrategoApp` funguje jako mediátor, pouze přijímá informace ze `StrategoCLI` a posílá je dále ke zpracování, toto je vidno na diagramu aplikace v kapitole o návrhu.

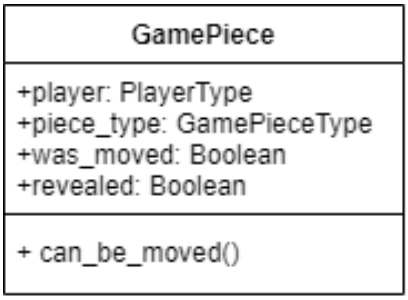


Obrázek 6.2: Třídní diagram StrategoApp

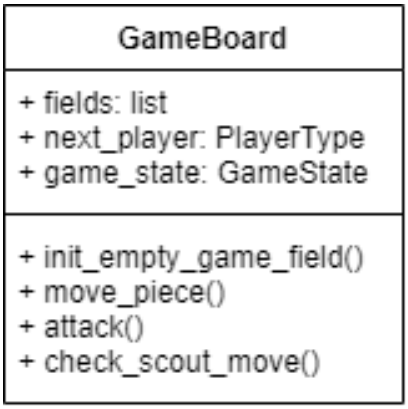


Obrázek 6.3: Třídní diagram hráčů

Další na řadě jsou skupiny tříd reprezentující hráče, viz 6.3, je jich celkem pět přestože jsou implementováni pouze tři různí hráči, to je z důvodu dědičnosti viz diagram tříd. Nejvyšší třída je `BasePlayer`, jejími atributy je jméno hráče, které se skládá z barvy a typu hráče (člověk, mcts nebo náhodný hráč), druhým atributem je Boolean hodnota vyjadřující zda je hráč červený nebo ne, tato hodnota se často využívá v programu pro kontrolu vlastníka figurky v různých funkcích. `RealPlayer` nemá žádné metody, protože se jedná o lidského hráče, jehož vstup je načten ve `StrategoCLI` a následně předán přes `StrategoApp` do `Gameboard`. Od `BasePlayer` dědí třída `AIplayer`, která zastřešuje všechny strojové implementace. Jejími atributy jsou `game_board` ve kterém je uložena instance herní desky v aktuálním stavu a druhým atributem je odkaz na `logger`, zápis všech akcí ve hře. Dále je z diagramu patrné, že tato třída obsahuje metody, které jsou určené pro hraní hry pomocí náhodných tahů, namísto toho aby byly v `RandomAIPlayer` toto umístění je úmyslné, tyto

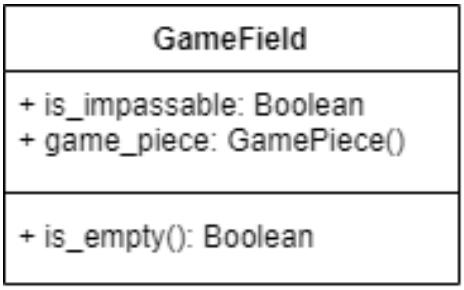


Obrázek 6.4: Třídní diagram herní figurky



Obrázek 6.5: Třídní diagram herního pole

metody jsou potřeba i v implementaci Monte Carlo Tree Search proto jsou zde. Nakonec posledními dvěma třídami v této sekci jsou `RandomAIplayer` a `MonteCarloAIPlayer`, obě třídy obsahují metodu `decide_next_move` která provede tah odpovídajícím algoritmem. Implementace tažení pomocí MCTS i náhodným tažením bude vysvětlena po popisu implementace herní desky a figurek, bez tohoto kontextu by vysvětlení nemuselo být srozumitelné. Další v pořadí je třída figurky, figurek už je více nejedná se tedy o singleton. Figurka je definována svojí barvou a hodnotou, navíc obsahuje pomocné atributy, které budou potřevné během implementace MCTS. Třídní diagram herní figurky vypadá takto. Atribut `player` představuje hráče kterému figurka náleží, `piecetype` představuje hodnotu figurky, hodnosti figurek ve Strategu byly již zmíněny v dřívější kapitole, viz 2.1, `GamePiece`, viz 6.4 může tedy nabývat jedné z dvanácti hodnot, hodnoty jsou uloženy v Enumerate v souboru `GamePieceType.py`. Poslední dva atributy `was_moved` a `revealed`, uchovávají hodnotu zda bylo figurkou již během hry taženo a pokud byla figurka odhalena, tyto dvě informace budou kritické při implementaci algoritmu MCTS. Jedinou metodou v této třídě je `can_be_moved`, metoda vrací True pokud se jedná o pohyblivou figurku, to znamená, nejedná se o bombu ani vlajku. Instance figurek jsou vytvářeny při manuálním pokládání figurek nebo automatickém rozmístění. Dále bude tedy popsána třída herní desky, viz 6.5, opět jako u prvních dvou tříd se jedná o singletona a digram této třídy vypadá následovně. Atribut `fields` je datová struktura 2D pole, která reprezentuje 10 x 10 herní plochu, atribut `next_player` uchovává informaci o tom který hráč je dále na řadě. Poslední atribut `game_state` uchovává informaci o aktuálním stavu hry, nabývá tří různých hodnot typu `GameState` buď hra právě probíhá `IN_PROGRESS`, nebo ji vyhrál jeden z hráčů zabráním vlajky nebo eliminací



Obrázek 6.6: Třídní diagram GameField

všech oponentových pohyblivých figurek, v tom případě je hodnota atribut nastavena na `BLUE_WON` nebo `RED_WON` podle toho kdo zvítězil. Metody v této třídě slouží k manipulaci figurek tj, tažení, útoků, kontrol validity tahu a zaznamenávání akcí do `log.txt`. První metodou této třídy je `init_empty_game_field()`, tato metoda se volá z konstruktoru při tvorbě instance herní desky a inicializuje prázdné herní pole, správně umístí jezera a připraví jej tak pro sázení figurek. Metoda `move_piece()` má na starosti vykonání tahu figurkou, vstupními hodnotami pro tuto metodu jsou dvě dvojice souřadnic, první pro výběr figurky a druhá pro její destinaci. Metoda také ošetřuje neplatné vstupy jako hráč snažící se táhnout oponentovou figurkou, pokus tahu na již okupované pole nebo výběr pole na kterém se žádná figurka nenachází. Pokud jsou všechny podmínky pro platný tah splněny, tah se provede a zaznamená do `log.txt`. V případě, že taženou figurkou je průzkumník je použita metoda `Check_scout_move()`, která kontroluje zda se nesnaží přeskočit figurku což je proti pravidlům. Poslední metodou v této třídě je `attack()`, tato metoda je volána během metody `move_piece()` nastane-li situace, že v destinaci je nepřátelská figurka a zahajuje se útok. V této metodě se porovnají hodnosti a rozhoduje se kdo útok vyhrál a kdo prohrál. Výsledek je poté zaznamenán do `log.txt`

Poslední třídou, která v této sekci bude popsána je `GameField` 6.6. Instance této třídy jsou uloženy v buňkách 2D pole v instanci `GameBoard`, viz 6.5. Třídní diagram potom vypadá následovně. Atributy této třídy jsou `is_impassable`, který značí zda se jedná o pole vody/jezera, na která nelze vstoupit ani je nijak přeskočit. Druhým atributem je `GamePiece`, který obsahuje instanci herní figurky, která je popsána v výše v této sekci. Jedinou metodou v této třídě je `is_empty()`, pokud herní pole není voda a nenachází se na něm žádná figurka vrací metoda `True`.

Na závěr sekce krátké shrnutí. Aplikace obsahuje jednu instanci `StrategoAPP`, dvě instance hráčů, jednu instanci herní desky `Gameboard`, herní deska obsahuje instance políček `GameField` a v nich jsou uloženy instance herních figurek `GamePiece`.

6.2 Úprava algoritmu Monte Carlo Tree Search

Nemodifikovaný algoritmus Monte Carlo Tree Search přesto, že pracuje úspěšně u her jako jsou Go, šachy nebo i některé hry s neúplnou informací jako Scotland Yard, ve Strategu při jeho použití narazíme na první problém poměrně rychle a to v druhé fázi, při expanzi. Expanze generuje synovské uzly způsobem, že provede všechny možné tahy ze stavu herní desky která se nachází v rodičovském uzlu. Kvůli neúplné informaci a tomu, že některé figurky jsou nepohyblivé a jiné naopak se mohou pohybovat o více polí najednou (průzkumník) nelze jednoznačně určit synovské uzly v případě, že expandujeme tah oponenta.

Jedno z možných řešení tohoto problému je získat seznam všech oponentových zbývajících figurek, čehož lze dosáhnout jednoduše stačí odečíst figurky vyřazené ze hry, jejichž hodnoty známe, protože se musí odhalit během útoku, od celkového počtu figurek ve hře. Tento seznam zbývajících figurek je poté náhodně distribuován na neodhalené figurky oponenta. Takto upravená herní deska se poté expanduje běžným způsobem. Nevýhodou tohoto řešení je, že nám vzniknou synovské uzly, které budou obsahovat neplatné herní stavy, kterých hra jediným tahem nemůže dosáhnout. Těchto stavů ale bude minimum, jelikož nepohyblivých figurek je ve Strategu pouze sedm z celkových čtyřiceti, a jejich dopad na celkový běh algoritmu bude pouze snížení jeho přesnosti nikoli jeho nefunkčnost. Proto jsem se rozhodl pro tento přístup.

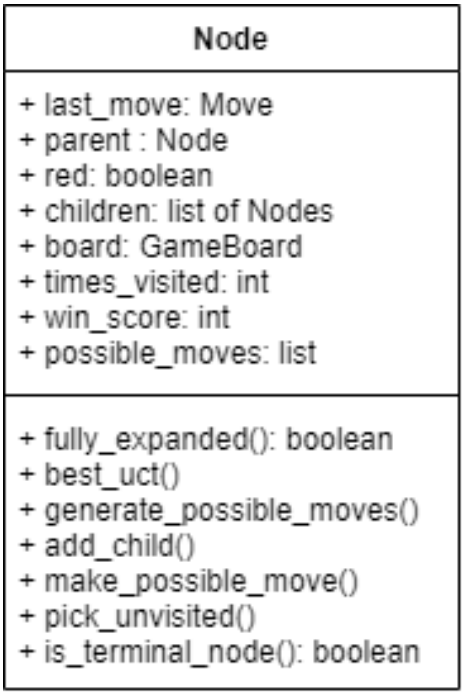
Druhým problém nastává ve fázi simulace. V Monte Carlo Tree Search se simulace zahajuje ze synovského uzlu a probíhá náhodným tažením obou hráčů dokud se neukončí hra. Problém je, že nelze jednoznačně říci, zda hra skončila nebo ne kvůli, neodhaleným figurkám. V simulaci je tedy problém přesně simulovat výsledky útoků a správně odhadnou kdy došlo zabrání vlajky, může zde dojít kde stavům, kdy hráč již vyhrál, ale neví o tom a proto simulace neskončí. Tento problém jsem se rozhodl řešit zavedením heuristik. Klasické Monte Carlo Tree Search žádné heuristické funkce neobsahuje, během žádné fáze algoritmu neohodnocuje stav herní desky. Zde bude potřeba je ale zavést ke správnému fungování algoritmu, heuristické funkce budou probrány níže v této kapitole. Ve zkratce, simulace nepoběží do konce hry, ale do parametrem stanovené hloubku např. 50 tahů a tam se hra zastaví. Heuristické funkce vyhodnotí stav herní desky, výhru nebo prohru bude pro Monte Carlo Tree Search představovat zhoršení nebo zlepšení stavu hru, nikoli prohra nebo výhra. Tento způsob zajistí, že fáze Zpětné propagace a selekce mohou zůstat nepozměněny a algoritmus bude stále plně funkční.

6.3 Implementace algoritmu Monte Carlo Tree Search

Tato sekce se bude zabývat implementací algoritmu Monte Carlo Tree Search (MCTS), jejím vysvětlením, důležitými metodami a postupy. Jak již probráno v kapitole 3.4, MCTS se skládá ze čtyř částí, selekce, expanze, simulace a zpětná propagace tato struktura se reflektuje i ve struktuře zdrojového kódu. Každá z těchto fází je implementována minimálně jednou metodou.

Vyhledávací stromová struktura

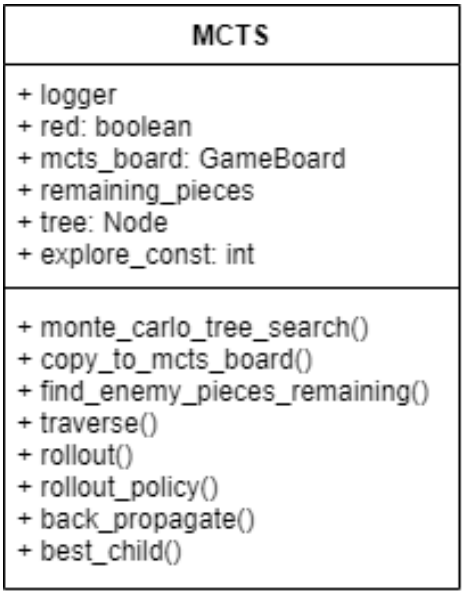
Pro začátek bylo třeba vytvořit stromovou strukturu a příslušící metody nad ní k manipulaci dat v uzlech. Herní strom u hry Stratego je obecný n-nární strom, v každé vrstvě mohou rodičovské uzly mít různý počet uzlů synovských v závislosti na počtu možných tahů. Třídní diagram pro herní strom, obrázek 6.7. Atributy v této třídě jsou, `last_move` je objekt s informací o posledním provedeném tahu, to znamená tah díky kterému tento uzel vznikl, jsou to pouhé dvě dvojice souřadnic, první pro selekci figurky a druhá pro tah. Další atribut je `parent` tento atribut obsahuje odkaz na rodičovský uzel, tento atribut se zužitkuje při zpětné propagaci. V prvotním řešení byly použity unikátní třídní identifikátory a pro nalezení rodičovského uzlu a funkce pro prohledávání do šířky, toto řešení nebylo efektivní. Atribut `red` tak jako v předchozích třídách značí zda uzel náleží červenému nebo modrému hráči, obsahuje hodnotu `True` nebo `False`. Atribut `children` je list který obsahuje instance všech synovských uzlů, herní strom je implementován tímto rekurzivním způsobem, každý uzel má svůj seznam. Atribut `board` je nejdůležitějším atributem, obsahuje stav



Obrázek 6.7: Třídní diagram uzlu

herní desky a definuje tak uzel. Tento atribut není nutně unikátní, může se stát, že dva uzly mají stejnou herní desku například poté co oba hráči provedou tah vpřed a na dalším tahu tou stejnou figurkou tah vzad. Atribut `possible_moves` je list co obsahuje všechny bezprostředně možné tahy ze stavu uloženém z uzlu. Tento list je naplněn při inicializaci třídy metodou `generate_possible_moves`. Jedná se o způsob šetření zdroji, kdy místo toho aby se při tvorbě nového uzlu ihned tvořili bezprostřední synovské uzly, pouze se vygenerují tahy vedoucí k těmto uzlům. Poslední dva atributy `Win_score` a `times_visited`, jsou hodnotami využívanými při selekci pomocí UCT metody probrané v 3.4. `Win_score` je proměnnou w_i a `times_visited` je proměnná n_i.

První metodou v této třídě je `fully_expanded()`, návratovou hodnotou této metody je typ boolean a říká zda je uzel plně expandovaný = všechny jeho možné synovské uzly byly přidány do stromové struktury. Metoda `best_uct()` využívaná během selekce, ohodnotí všechny synovské uzly a uzel s nejvyšší hodnotou je poté předán k expanzi. Metoda `generate_possible_moves()` vygeneruje všechny možné tahy ze stavu ve kterém se uzel nachází v závislosti, který hráč je dále na tahu a přidá je do listu `possible_moves`. Metoda `add_child()` generuje nové synovské uzly a přidává je do seznamu synovských uzlů. Metody `pick_unvisited()` a `make_possible_moves()` tyto dvě metody, vyberou uzel, který nebyl dosud plně expandován, vyberou náhodně jeden z jeho synovských uzlů a procesem expanze jej přidají do vyhledávacího stromu MCTS. Poslední metoda je `is_terminal_node()`, která zkontroluje zda je uzel terminální, v této implementaci se takovýto uzel vyznačuje prázdnými seznamy `children` a `possible_moves`.



Obrázek 6.8: Třídní diagram MCTS

Monte Carlo Tree Search

Třída `MCTS` implementuje logiku jednoho cyklu Monte Carlo Tree Search (MCTS) tak jak lze vidět na obrázku 3.3. Implementace v této třídě až na modifikace ze sekce 6.2 je standardní MCTS. Třídní diagram vypadá následovně.

Atribut `logger` je odkaz na stejnojmennou třídu, která zaznamenává akce provedené ve hře. Atribut `red` je opět boolean hodnota pro identifikaci barvy hráče. Atribut `mcts_board` je kopií herní desky ve stavu kdy Monte Carlo Tree Search je na tahu. Tato kopie neobsahuje hodnosti nepřátelských figurek, aby MCTS nemohlo podvádět, jedná se o interní kopii s kterou MCTS dále pracuje. Atribut `remaining_pieces` je seznam všech hodností nepřátelských figurek, které jsou stále ve hře. Atribut `tree` je odkaz na první instanci uzlu vyhledávacího stromu, jedná se o jeho kořen. A posledním atributem je `explore_const`, je to konstanta c z metody UCT v sekci 3.4, hodnota konstanty udává nakolik bude MCTS prohledávat do šířky / prozkoumávat, čím vyšší hodnota tím více prozkoumávání. Metodami v třídě MCTS jsou `copy_to_mcts_board()` a `find_enemy_pieces_remaining()` tyto dvě metody jsou volány ihned po zavolání MCTS. Za úkol mají vytvořit interní kopii hlavní herní desky s kterou MCTS může manipulovat. První z těchto metod překopíruje již odhalené nepřátelské figurky, pokud je figurka neodhalená vytvoří dočasnou figurky s neznámou hodností, metoda zachovává informaci o pohybu figurek a informaci zda se nějaká figurka pohnula o více jak jedno pole, což znamená, že se musí jednat o průzkumníka. Monte Carlo Tree Search má tedy v této implementaci jistou paměť což ve standardní implementaci není potřeba. Druhá metoda nám zjistí, které nepřátelské figurky byly již vyřazeny ze hry a které ne. S takto rozpracovanou herní deskou a seznamem nepřátelských figurek je deska doplněna do finální podoby ve funkci `distribute_remaining_pieces()`. v této funkci jsou přiřazeni známí průzkumníci, figurkám které se pohnuli náhodná pohyblivá figurka a figurkám o kterých není zjištěna žádná informace se přiřadí náhodná zbývající figurka. V prvotní implementaci měli nepřátelské figurky neznámou hodnost až do momentu prvního útoku a obsahovaly vážený seznam s pravděpodobnostmi o kterou hodnost se jedná, která se aktualizovala podle zbývajících figurek na herní desce. Tato implementace byla ale zby-

tečně složitá, informace o tom, které figurky opravdu zbývají ve hře je aktualizována pouze po provedení tahu, nikoli v simulaci. Hodnota figurky při útoku by byla přiřazena podle toho která hodnost má největší procentuální pravděpodobnost. Stejného efektu lze tedy dosáhnout okamžitou distribucí všech hodností náhodně protože pokud nějaká figurka má největší pravděpodobnost být například minér znamená to, že v seznamu zbývajících hodností je nejvíce minérů. Pokud teda náhodně vybere z tohoto seznamu máme největší šanci vytáhnout minéra.

Metoda `monte_carlo_tree_search()` je metodou ve které probíhá hlavní smyčka tohoto algoritmu, zde se opakovaně volají čtyři fáze MCTS algoritmu. Délka této smyčky lze omezit časově nebo počtem iterací, v této implementaci bylo zvoleno omezení časové a to 2 sekundy. Metoda `traverse()` kombinuje logiku selekce a expanze metoda ohodnocuje plně expandované uzly pomocí funkce UCT a vybírá nejlépe ohodnocený synovský uzel. Jakmile narazí na uzel který není plně expandovaný, vyberou náhodně jeden synovský uzel a přidají jej do vyhledávacího stromu a vrátí jej jako listový uzel. Pokud uzel je terminální a nemá již žádné další synovské uzly vrátí terminální uzel. Tento listový uzel získaný předchozí metodou je dále zpracován metodou `rollout()`, tato metoda provádí simulaci náhodným tažením nad herní deskou v listovém uzlu u obou hráčů do předem stanovené hloubky, v našem případě se jedná o 50 tahů. Po simulaci je takto rozehraná herní deska poslána k evaluaci pomocí heuristických funkcí o které jsou popsány v další podsekci. Po evaluaci dostaneme konečný výsledek simulace. Výhru nebo prohru, v našem případě zlepšení nebo zhoršení stavu hry.

Tuto hodnotu je potřeba zpětně propagovat po vyhledávacím stromě. To má za úkol metoda `back_propagate()`, tato metoda začne od listového uzlu a postupuje až ke koření vyhledávacího stromu. Při návštěvě každého uzlu aktualizuje atributy `win_score` a `times_visited`, první atribut je aktualizován pouze v uzlech náležícím výherci, hráči se střídají podle hloubky zanoření tak jako v algoritmu Minimax. Druhý atribut je aktualizován v každém uzlu.

Poté co algoritmu Monte Carlo Tree Search dojde výpočetní čas navrátí metodou `best_child()` bezprostřední synovský uzel kořenového uzlu, který byl navštíven nejvíce krát. Tento uzel přestavuje nejlepší možný tah který Monte Carlo Tree Search našlo za uplynulou dobu.

Heuristiky

Jelikož naše implementace Monte Carlo Tree Search nemůže simulovat do konce hry kvůli neúplné informaci ve Strategu, bylo potřeba použít heuristické funkce pro vyhodnocení stavu hry. Tato implementace obsahuje dvě funkce. První z nich jednoduše spočítá figurky obou hráčů a porovná jejich počet, hráč s vyšším počtem figurek vyhrává. Druhá funkce vezme tento koncept a rozšiřuje jej. Opět počítá počet figurek, ale bere v potaz i jejich hodnosti, tyto hodnoty sečte vytvoří tak hodnotu vyjadřující celkovou síla hráčovi armády. Tyto heuristiky poté vrátí hodnotu, která předstírá být výhrou nebo prohrou pro zbytek algoritmu MCTS a propaguje tento výsledek.

6.4 Implementace Alfa-Beta ořezávání

Teorii k tomuto algoritmu lze naleznou v kapitole 3.2, zde se budeme zabývat pouze implementací a modifikacemi. Stejně jako v případě Monte Carlo Tree Search 6.2 je potřeba u alfa-bety provést modifikace, aby se tento algoritmus dal aplikovat na Stratego. Kvůli

vysokému větvícímu faktoru není prakticky možné simulovat celý herní strom. Pro účely algoritmu alfa-beta jsem se rozhodl algoritmus hloubkově omezit na 3 úrovně prohledávání. Druhým problémem je neúplná informace, neznalost hodností oponentových figurek. Což způsobí, že při generovaní uzlů potomků mohou vzniknou neplatné stavy. Například, útočím na neznámou figurku, po aplikaci tohoto tahu vygeneruji novou herní desku, kvůli neúplné informaci, ale nevím zda jsem vyhrál nebo prohrál. Podobný problém nastane i u figurek, které se nehýbou nebo skautů, kteří se mohou pohybovat o více polí.

Tento problém jsem se rozhodl omezit vynecháním generování pohybů pro neznámé figurky, tímto přístupem se do simulace nezavedou neplatné stavy, ale jako protiklad se nevygenerují všechny možné uzly, tedy stavy, které na desce mohou nastat.

Druhou modifikací je adice heuristické funkce. U klasické alfa-bety není heuristická funkce potřeba, hra se simuluje dokud někdo nevyhraje a výhra nebo prohra se propaguje herním stromem ke kořenu, kde se poté vybere potomek směřující tím směrem, kde se ve stromu nachází nejvíce výher. Jelikož je generování potomků hloubkově omezeno, je potřeba zavést heuristiku, která ohodnotí vygenerované listové uzly. Podle heuristické hodnoty poznáme, zda je stav desky po aplikaci tahu pro nás výhodný nebo ne.

Heuristická funkce

Heuristická funkce použitá u tohoto algoritmu je podobná té použité u MCTS. Navíc jsou přidány podmínky, které mění hodnotu některých figurek.

- Pokud je přítomen nepřátelský špión, hodnota maršála je poloviční
- Pokud mám méně jak 3 mináry, jejich hodnota se zvýší, protože jsou kritičtí pro pozdější fáze hry
- Pokud mám méně jak 3 skauty, jejich hodnota se zvýší, jsou důležití pro zabraní vlajky nebo zneškodnění špióna
- Bomba poloviční hodnotu nejsilnější nepřátelské figurky

Hodnoty samotných figurek jsou převzány z práce[1] a v tabulce 6.1.

6.5 Implementace expectimax

Expectimax je variace minimax algoritmu, která se používá v oblasti umělé inteligence pro hru dvou hráčů s nulovým součtem, například vrhcáby. Liší se od minimaxu v tom, že navíc obsahuje rozhodovací uzly (chance nodes), které simulují náhodný jev ve hře, jako například hod kostkou. Minimalizující a maximalizující hráč poté nevybírají minimalní

Mashal 400	Sergeant 15
General 200	Miner 25
Colonel 100	Scout 30
Major 75	Spy 200
Lieutenant 50	Bomb 20
Captain 25	Flag 10000

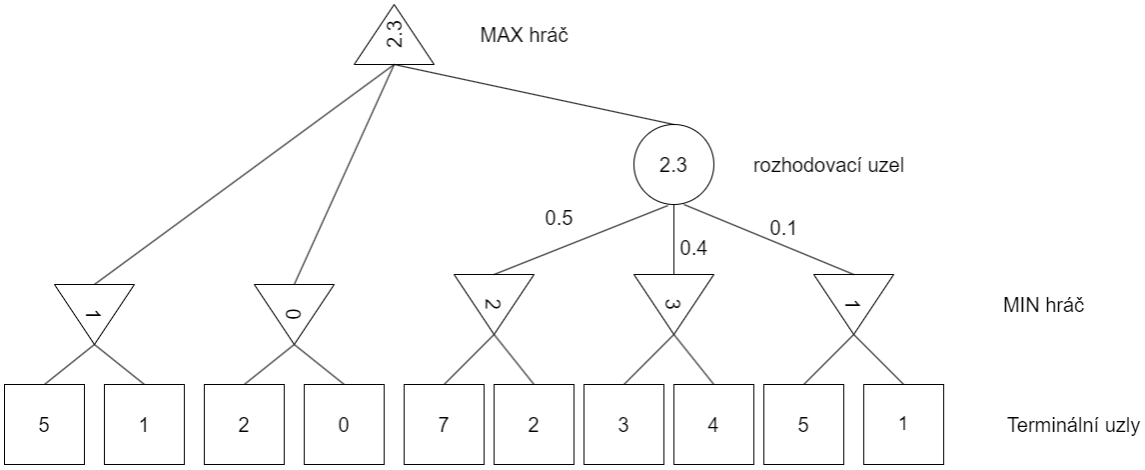
Tabulka 6.1: Hodnoty figurek pro heuristickou funkci

nebo maximální hodnotu z potomka ale vážený průměr z rozhodovacího uzlu. Pseudokód pro tento algoritmus poté vypadá takto 1. Opět je potřeba provést pád modifikací aby

Algorithm 1 Expectiminimax Algoritmus

```
1:  function EXPECTIMINIMAX(node, depth)
2:      if node is a terminal node or depth == 0 then
3:          return the heuristic value of node
4:      end if
5:      if the adversary is to play at node then
6:          α = +∞
7:          for each child of node do
8:              α = min(α, Expectiminimax(child, depth − 1))
9:          end for
10:     else if we are to play at node then
11:         α = −∞
12:         for each child of node do
13:             α = max(α, Expectiminimax(child, depth − 1))
14:         end for
15:     else if random event at node then
16:         α = 0
17:         for each child of node do
18:             α = α + (Probability[child] × Expectiminimax(child, depth − 1))
19:         end for
20:     end if
21:     return α
22: end function
```

tento algoritmus šel aplikovat na hru Stratego. Expectimax je hloubkově omezen také na 3 úrovně. Dále tento algoritmus používá stejnou heuristickou funkci jako alfa-beta v sekci 6.4. Jediná překážka, která zbývá je identifikovat co ve Strategu kde dochází k náhodnému jevu a správně implementovat rozhodovací uzly. K náhodnému jevu dochází pokud útočíme na figurku neznáme hodnosti anebo figurka neznáme hodnosti útočí na naši figurku. V tomto se Stratego liší od ostatních her, rozhodovací uzel se generuje pouze někdy. Ve zdrojovém kódu se tedy zavolá funkce expectimax, příslušný hráč vygeneruje všechny své možné tahy, a tahy, které způsobí útok na neznámou figurku nebo útok neznáme figurky, rekurzivně volají volají expectimax, ale jako `CHANCER_PLAYER` a generuje se rozhodovací uzel. Jelikož šance na výhru nebo prohru jsou různé používá se vážený průměr uzlů potomků pro výpočet hodnoty rozhodovacího uzlu. Tato šance se v algoritmu určuje podle zbývajících neodhalených figurek oponenta jejich hodností v porovnaní s hodností figurky naší. Výsledkem je vygenerování tří nových stavů/uzlů, jeden pro výhru, jeden pro prohru a jeden pro remízu 6.9.



Obrázek 6.9: Diagram Expectimax

Kapitola 7

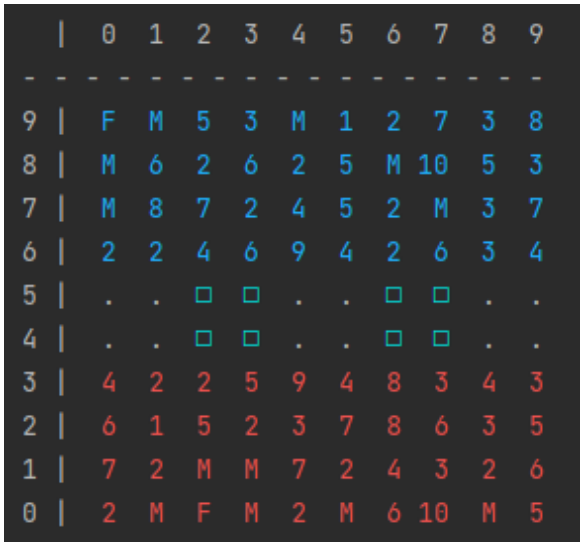
Testování

Tato kapitola je zaměřena na ověření funkčnosti implementace aplikace pro hraní Stratega. Testování proběhlo formou experimentů na implementaci s různými parametry a jejich následné vyhodnocení. Hlavním zájmem bude správnost a úspěšnost algoritmu Monte Carlo Tree Search, ve hře proti ostatním algoritmům a proti lidskému hráči.

7.1 Experimenty

MCTS Experiment 1

Náplní prvního experimentu bylo ověřit zda námi implementovaný algoritmus Monte Carlo Tree Search správně funguje, zkontrolovat zda správně staví herní strom a vybírá pomocí UCT správný synovský uzel. Experiment byl proveden při hře MCTS proti algoritmu s náhodným tažení figurek. Parametry algoritmu byly následující, čas na tah byl dvě sekundy, hloubka simulace byla 50 tahů a explorační konstanta c byla rovna 0.3. Oba hráči využívají automatického rozestavení figurek. Po zahrání každého tahu je herní deska překreslena, v příkazové řádce pak vypadá takto. 7.1 Voda jezera jsou vyznačena světle modrými čtverci



Obrázek 7.1: Herní deska Stratego

a prázdná po tečkami. Pro obarvení výstupu byla použita knihovna Colorama. Na první

pohled je jasné, že algoritmus pracuje jak má, vyhledávací strom 7.2 se staví správně a synovské uzly jsou korektně vybírány ty s nejvyšším počtem návštěv. Bohužel efektivita této implementace v počátečních fázích hry není vysoká. Monte Carlo Tree Search hraje v podstatě náhodnými tahy dokud nenasbírá dostatečné množství informací o nepřátelských figurkách, čím déle hra běží tím silněji MCTS hraje. Což v některých hrách vede k prohře protože MCTS algoritmus přijde v rané fázi hry o silné figurky a nedokáže tuto ztrátu již dohnat. Test je potřeba provést na větším vzorku. V této konfiguraci tedy bylo provedeno

```
Tree node:  visited:66 wins: 24

tah: [6, 0] [5, 0] wins: 2 visits: 6
tah: [6, 5] [4, 5] wins: 0 visits: 2
tah: [6, 5] [3, 5] wins: 0 visits: 2
tah: [6, 9] [4, 9] wins: 1 visits: 4
tah: [6, 1] [5, 1] wins: 11 visits: 24
tah: [6, 8] [5, 8] wins: 0 visits: 2
tah: [6, 4] [5, 4] wins: 3 visits: 7
tah: [6, 9] [3, 9] wins: 5 visits: 12
tah: [6, 9] [5, 9] wins: 0 visits: 1
tah: [6, 5] [5, 5] wins: 2 visits: 6
best child: [6, 1] [5, 1] wins: 11 visits: 24
```

Obrázek 7.2: Výběr nejsilnějšího tahu

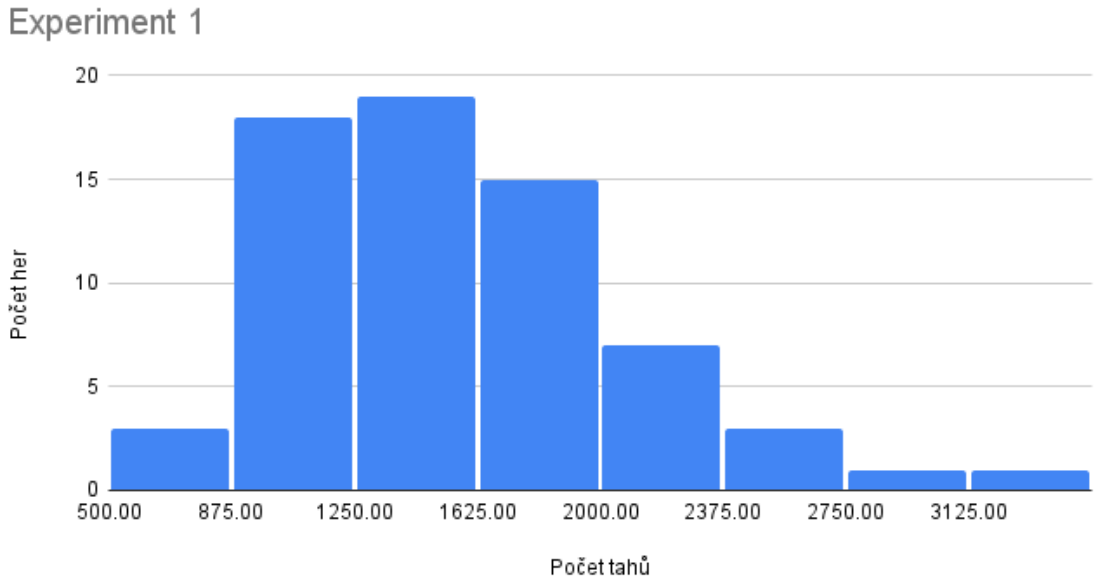
271 běhů aplikace z nichž Monte Carlo Tree Search vyhrálo 202 her. To MCTS dává výhernost 74,5 %. Pro zajímavost průměrná hra byla dohrána během 1596 tahů a trvala 53 minut. 7.3

MCTS Experiment 2

Cílem druhého experimentu je zvýšení úspěšnosti Monte Carlo Tree Search pomocí modifikace parametrů tohoto algoritmu. Čas je u MCTS důležitým faktorem čím déle algoritmus běží tím silnější tah je schopno najít. Proto v tomto experimentu byl zvýšen čas tahu MCTS na 10 sekund místo původních 2 sekund, všechny ostatní parametry byly zachovány. V této konfiguraci bylo provedeno 26 běhů hry, z nich MCTS vyhrálo 21. To dává výslednou úspěšnost 80,8 %. Jedná o více jak 5 procentní zlepšení oproti předešlému experimentu. Avšak časová náročnost takové implementace se zvýšila drasticky. Průměrná hra trvala 260 minut, to je jeden z důvodů proč je testovací vzorek malý. U tohoto experimentu nelze vyloučit, že pozorované zlepšení je statistická odchylka. Tento experiment by bylo potřeba ověřit na větším vzorku, výsledky z něj nejsou velmi konkluzivní.

MCTS Experiment 3

V třetím a posledním experimentu se opět pokusíme zvýšit úspěšnost algoritmu Monte Carlo Tree Search ve hře proti stroji hrajícím náhodnými tahy. V tomto experimentu byl změněn parametr simulační hloubky, z běžných 50 tahů na 100 tahů. Simulační fáze bude tedy dvojnásobně delší. Tato změna by měla mít pozitivní dopad na přesnost algoritmu hlavně v pozdějších fázích hry, kdy již máme téměř úplnou informaci. V této konfiguraci



Obrázek 7.3: Délka her

bylo provedeno 113 her z nichž MCTS vyhrálo 80. Výsledná úspěšnost v tomto experimentu je tedy 70,8 %. Jedná se o 5 procentní zhoršení oproti základní konfiguraci, ale opět jako u experimentu předchozího potenciálně se jedná o statistickou odchylku a tato modifikace neměla na algoritmus vliv.

Alfa-beta versus náhodný hráč

Cílem tohoto testování bylo ověřit, zda algoritmus alfa-beta správně funguje a otestovat jeho schopnost proti náhodnému tažení. Na rozdíl od MCTS nebyl tento algoritmus nějak časově omezen a při každém tahu běžel dokud neprohledal svůj hloubkově omezený stavový prostor. Celkem proběhlo 51 her, z kterých alfa-beta algoritmus vyhrál 28 her. Měl tedy výhernost 55 %.

Alfa-beta versus MCTS

Tento experiment proběhl mezi druhou modifikací MCTS, která měla v druhém experimentu 7.1 nejvyšší výhernost a algoritmem alfa-beta. Celkem proběhlo 60 her, z toho MCTS vyhrálo 39 her. Výsledná výhernost MCTS v tomto experimentu tedy je 65 %.

Expectimax versus náhodný hráč

V posledním experimentu jsem otestoval algoritmus expectimax. Celkem bylo odehráno 12 her, z kterých expectimax vyhrál 8 her. Výsledná výhernost expectimax je tedy 66.7 %. Ukázalo se, že expectimax je časově velmi náročný, mohou zato hlavně právě rozhodovací uzly. Při útoku nebo obraně s neznámou figurkou se generují 3 potomci místo jednoho, což nadále zvyšuje již tak velký větvící faktor a stavový prostor, i když je omezený do hloubky je i tak obrovský. v průměru simulace jedné této hry trvala 29 hodin.

Testování proti lidskému hráči

Proti implementovaným algoritmům jsem osobně odehrál několik her. Jiné hráče, kteří by hru znali a byli ochotni otestovat jsem bohužel nezajistil. Považuji se za začátečníka až mírně pokročilého, hru jsem v minulosti hrál, znám pravidla hry a základní taktiky. Rozestavení figurek jsem používal vlastní nikoli automatické. Celkem jsem odehrál 5 her, z toho jednu proti algoritmu alfa-beta, jednu proti expectimax a tři hry proti MCTS. Ve čtyřech případech jsem byl schopen zvítězit jednu hru proti MCTS jsem prohrál. Všechny tři algoritmy hráli pocitově podobně, až na délku tahu oponenta, ta byla v případě expectimax nejdelší, kdy jsem čekal až 15 sekund na tah oponenta. V počátečních fázích hry dokud moje figurky byly nepříteli neznámé, hráli algoritmy více méně náhodně, až po nasbírání informace se jevily alespoň trochu inteligentně, ale i tak nebyl problém je jako začátečník porazit. Jedna instance, kde jsem prohrál byla spíše zaviněna mým bezhlavým herním stylem, kde jsem zkoušel hrát agresivně, v raných fázích hry přišel o vysoce hodnotné figurky a v pozdější fázi jsem už nebyl schopný se zotavit.

Zhodnocení experimentů

Cílem bylo implementovat a vyhodnotit schopnost navržených algoritmů. Experimenty provedené na implementaci úspěšně ověřily její funkčnost a ukázaly, že nejužitečnějším z těchto tří algoritmů bylo Monte Carlo tree search. Během experimentů působila potíže hlavně velký stavový prostor a s tím spojená časová náročnost hry Stratego. Také implementace neredukované verze má za následek menší množství běhů jak lze vidět v obrázku 7.3. Experimenty ukázaly, že změny parametrů algoritmu MCTS mají dopad na jeho výkon až o 5 %.

7.2 Možné nedostatky v implementaci

Návrh aplikace se zdá být správný, výsledky experimentu 1 jsou dostatečně přesvědčivé, vyhledávací strom se staví správně, uzly s nejvyšším ohodnocením jsou vybírány. Výhernost algoritmu MCTS není ale tak vysoká, jak očekáváno. Nedostatky budou nejspíše v jedné z modifikací algoritmu navržených v sekci 6.2. Je pravděpodobné, že heuristické funkce nejsou dostatečné, aby dokázaly nahradit simulaci hry do hlubších vrstev stavového prostoru.

Lze těžko odhadnout, zda by jinak navržené heuristiky nějak pomohli tyto klasické algoritmy vylepšit, nebo zda by lepším přístupem bylo zvolit neuronové sítě. Realizace této práce za použití komplexních metod z oblasti umělé inteligence by mohla být vhodným pokračováním v navazujícím studiu v diplomové práci.

Kapitola 8

Závěr

Cílem této práce bylo navrhnout a implementovat algoritmus pro hrání deskové hry Stratego a následně jej otestovat vůči ostatním řešením a vyhodnotit schopnost tohoto systému.

Tento cíl byl splněn na základě úspěšných experimentů na tomto systému a validaci, že algoritmus pracuje dle očekávání.

Na začátku práce jsem se seznámil s pravidly hry Stratego a s metodami pro počítačové hraní her. Nahlédl jsem také na již existující řešení problematiky této hry, i her podobných, pro inspiraci. Pro řešení problematiky hraní hry Stratego počítačem jsem navrhl použití modifikovaného algoritmu Monte Carlo Tree Search s heuristickými funkcemi a dvou dalších algoritmů. Tyto metody jsem následně implementoval jako konzolovou aplikaci, která umožňuje hru hráče proti těmto metodám nebo hru stroje hrajícího náhodnými tahy proti těmto metodám. Funkčnost toho systému jsem validoval experimenty a pozorným sledováním hry. V poslední řade jsem vyhodnotil schopnost tohoto systému jako uspokojující avšak ponechává prostor pro zlepšení.

Tato práce byla zajímavou výzvou, nejedná se o velmi populární téma a chtěl jsem přispět do této oblasti vlastní prací. Pří jejím řešení bylo potřeba zapojit i kreativní část myšlení při tvorbě nových vlastních postupů a implementaci hypotéz. Dozvěděl jsem se nové poznatky z oblasti umělé inteligence a teorie her.

Již v první fázi návrhu jsem měl v plánu vytvořit aplikaci tak, aby byla v budoucnu rozšiřitelná. Ať se bude jednat o grafické uživatelské rozhraní nebo nové algoritmy, modulární návrh této práce to umožňuje. Chtěl bych v budoucnu na této práci pokračovat a najít nové lepší způsoby tvorby umělé inteligence. Chtěl jsem umožnit i komukoli, kdo bude z této práce čerpat, tuto možnost.

Literatura

[1] Arts, S. *COMPETITIVE PLAY IN STRATEGO*. 2010. Dostupné z: https://project.dke.maastrichtuniversity.nl/games/files/msc/Arts_thesis.pdf.

[2] Association, B. G. *How to Play*. 2017. Dostupné z: https://www.britgo.org/intro/intro2.html.

[3] Boer, V., Rothkrantz, L. a Wiggers, P. Invincible - a Stratego bot. *International Journal of Intelligent Games and Simulation*. 2008, sv. 5, č. 1, s. 1–10. ISSN 1477-2043.

[4] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I. et al. A Survey of Monte Carlo Tree Search Methods. *IEEE transactions on computational intelligence and AI in games*. PISCATAWAY: IEEE. 2012, sv. 4, č. 1, s. 1–43. ISSN 1943-068X.

[5] Coulom, R. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In: Herik, H. J. van den, Ciancarini, P. a Donkers, H. H. L. M. J., ed. *Computers and Games*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, s. 72–83. ISBN 978-3-540-75538-8.

[6] Ismail, M. *Multi-agent stratego*. Rotterdam University, Aug 2004. Dostupné z: http://www.kbs.twi.tudelft.nl/Publications/BSc/2004-Ismail-BSc.html.

[7] Liu, M. *General Game-Playing With Monte Carlo Tree Search*. 2017. Dostupné z: https://medium.com/@quasimik/monte-carlo-tree-search-applied-to-letterpress-34f41c86e238.

[8] Perolat, J., Vylder, B. D., Hennes, D., Tarassov, E., Strub, F. et al. Mastering the game of Stratego with model-free multiagent reinforcement learning. *Science*. American Association for the Advancement of Science (AAAS). dec 2022, sv. 378, č. 6623, s. 990–996. DOI: 10.1126/science.add4679. Dostupné z: https://doi.org/10.1126%2Fscience.add4679.

[9] Roy, R. *ML: Monte Carlo Tree Search (MCTS)*. Jul 2022. Dostupné z: https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/.

[10] Russell, S. J. S. J. *Artificial intelligence : a modern approach*. Third edition; Authorized adaptation from the United States edition 2010. Boston ; London: Pearson, 2016. Prentice Hall series in artificial intelligence. ISBN 978-1-292-15396-4.

[11] Smith, S. a Stanford. Learning to Play Stratego with Convolutional Neural Networks. In:. 2015. Dostupné z: https://api.semanticscholar.org/CorpusID:198164119.

[12] Team, T. U. *Best Tips to win Stratego*. 2020. Dostupné z: https://www.ultraboardgames.com/stratego/tips.php.

[13] Team, T. U. *The history of Stratego*. 2020. Dostupné z: https://www.ultraboardgames.com/stratego/history.php.

[14] Team, T. U. *Scotland Yard Game Rules*. 2020. Dostupné z: https://www.ultraboardgames.com/scotland-yard/game-rules.php.

[15] Team, T. U. *Stratego game rules*. 2020. Dostupné z: https://www.ultraboardgames.com/stratego/game-rules.php.

[16] Team, T. U. *Winning Stratego tactics*. 2020. Dostupné z: https://www.ultraboardgames.com/stratego/tactics.php.

[17] Vonášek, J. *Překvapivá účinnost algoritmu Monte Carlo tree search*. Prague, CZ, 2022. Master's thesis. České vysoké učení technické v Praze. Výpočetní a informační centrum. Dostupné z: http://hdl.handle.net/10467/99113.

[18] Whitehouse, D. Monte Carlo Tree Search for games with hidden information and uncertainty. In:. 2014.

[19] Wolf, H. *Stratego programming*. 2017. Dostupné z: https://home.hccnet.nl/jabcwolf/stratego/links.html.

[20] Zbořil, F. V. *Základy umělé inteligence – Studijní text*. 2022. Dostupné z: https://www.fit.vutbr.cz/study/courses/IZU/private/2022-opora-IZU.pdf.

Příloha A

Obsah přiloženého paměťového média

Přiložené médium obsahuje následující soubory a složky:

```
24779.pdf
StrategoAI
├── StrategoAI
│   ├── alfabeta
│   │   ├── ABEval.py
│   │   ├── ABPeiceMove.py
│   │   └── alfabeta.py
│   ├── expectimax
│   │   ├── BoardScanExp.py
│   │   ├── expectimax.py
│   │   ├── MakeMoveWithChance.py
│   │   └── PlayerMaxMin.py
│   ├── GameInfo
│   │   ├── GameBoard.py
│   │   ├── GameField.py
│   │   ├── GamePiece.py
│   │   ├── GamePieceType.py
│   │   ├── GameState.py
│   │   └── MctsGamePiece.py
│   ├── GameSetup
│   │   └── PieceSetup.py
│   ├── Logger.py
│   ├── MCTS
│   │   ├── DistributeRemainPieces.py
│   │   ├── Heuristics.py
│   │   ├── MCTS.py
│   │   ├── MctsPieceMove.py
│   │   ├── MoveList.py
│   │   ├── PrintBestChild.py
│   │   └── Tree.py
│   ├── Players
```

```
│   │   ├── AIPLayer.py
│   │   ├── AlfaBetaAIPlayer.py
│   │   ├── BasePlayer.py
│   │   ├── ExpectimaxPlayer.py
│   │   ├── MonteCarloAIPlayer.py
│   │   ├── PlayerType.py
│   │   ├── RandomAIPlayer.py
│   │   └── RealPlayer.py
│   ├── README.md
│   ├── StrategoApp.py
│   └── StrategoException.py
├── StrategoCLI
│   ├── DrawBoard.py
│   ├── StrategoCLI.py
│   └── __pycache__
│       ├── DrawBoard.cpython-37.pyc
│       └── StrategoCLI.cpython-37.pyc
├── log.txt
├── main.py
├── README.txt
├── results.txt
└── test.py
```