Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Umělá inteligence pro karetní hry s neúplnou informací

2020

Bc. Jan Molčík

Vedoucí práce: Mgr. Petr Osička, Ph.D.

Studijní obor: Aplikovaná informatika, prezenční forma

Bibliografické údaje

Autor:	Bc. Jan Molčík
Název práce:	Umělá inteligence pro karetní hry s neúplnou informací
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2020
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Petr Osička, Ph.D.
Počet stran:	44
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Bc. Jan Molčík
Title:	Artificial intelligence for card games with incomplete information
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2020
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Petr Osička, Ph.D.
Page count:	44
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Diplomová práce zkoumá moderní přístupy v doméně her s neúplnou informací. Shrnuje základní rysy a herní mechaniky sběratelských karetních her. Popisuje základní přístupy při tvorbě algoritmů pro doménu her s úplnou i neúplnou informací. Představuje zvolený framework pro simulaci Hearthstonu a jednotlivé strategie agentů v prostředí simulace. Sestavuje testy k experimentálnímu ověření navrhnutého agenta, jenž využívá MCTS s UCT algoritmem. Popisuje ovládání jednoduché konzolové aplikace k testování jednotlivých agentů. Diskuze poskytuje náhled na limity práce a pojednává o možných vylepšeních.

Synopsis

This thesis aims to examine state-of-the-art principles in the domain of games with incomplete information. Describes collectible card games. Summarizes main aproaches in the development of algorithms in this domain. Presents the framework for hearthstone simulation and strategies for agents in this particular domain. Tests our MCTS agent with UCT algorithm against other agents. Discuss limitations of this thesis and presents possible enhancements.

Klíčová slova: umělá inteligence; sběratelské karetní hry; neúplná informace; monte carlo tree search; hearthstone

Keywords: artificial intelligence; collectible card games; incomplete information; monte carlo tree search; hearthstone

Na tomto místě bych rád poděkoval Mgr. Petru Osičkovi, Ph.D. za odborné vedení diplomové práce, cenné rady a konzultace. Dále bych rád poděkoval Alexanderu Dockhornovi, M.Sc. za poskytnuté odborné materiály. V neposlední řadě děkuji také své rodině a přítelkyni za podporu při psaní této práce.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době je již obor umělé inteligence na tak vysoké úrovni, že tahové hry dvou hráčů s úplnou informací, jako jsou například šachy nebo Go, jsou dokonale prozkoumány a umělá inteligence v této doméně převyšuje lidskou inteligenci. Počínaje prvním vítězstvím šachového počítače Deep Blue nad mistrem světa v šachu Garri Kasparovem v roce 1997 se spustil bouřlivý vývoj i v dalších herních doménách. Dlouhou dobu se však zdálo, že například hra Go je natolik komplexní, že algoritmy přejaté z řešení šachových partií nelze jednoduše použít v této odlišné hře.

Téměř dalších dvacet let trvalo vynalézt dokonalejší algoritmy a vylepšení pro první vítězství nad člověkem. V roce 2016 vyhrál počítačový program AlphaGo nad profesionálním hráčem Lee Sedolem ve hře Go bez jakéhokoliv omezení a na standardní hrací desce. Dosáhl toho kombinací nejmodernějších přístupů umělé inteligence jako jsou například *Deep supervised learning*, *Deep reinforcement learning* a *Monte Carlo tree search* v kombinaci s paralelními výpočty.

Za další výzvu tak můžeme považovat hry s neúplnou informací, kdy musíme kromě všech možných tahů také zohlednit danou míru neurčitosti. Zejména v posledním desetiletí se hry typu Poker nebo sběratelské karetní hry začínají hojně prozkoumávat a vznikla celá řada různých vědeckých studií, které si kladou za cíl využívat algoritmy z domény her s úplnou informací a přizpůsobovat je pro doménu s neúplnou informací. Tato diplomová práce si také dala za cíl prozkoumat nejmodernější způsoby řešení her.

Diplomová práce se skládá celkem z šesti kapitol. První kapitola popisuje sběratelské karetní hry, jejich společné rysy a názvosloví. Dále jsou představeny dvě konkrétní karetní hry a jsou vysvětleny jejich odlišnosti a specifické mechanismy. V další kapitole jsou představeny základní přístupy tvorby algoritmů pro hry s úplnou informací a také moderní algoritmus pro obecnou doménu her s úplnou i neúplnou informací. Ve třetí kapitole je popsán výběr prostředí pro simulaci, jednotlivé strategie agentů v tomto prostředí a také způsob řešení nahrazení neúplné informace. Další kapitolu tvoří experimentální ověřování vytvořeného agenta. Agent je testován proti ostatním agentům a také sám proti sobě s různým nastavením parametrů a následně jsou vyhodnoceny výsledky všech testů. V páté kapitole je popsána konzolová aplikace, která slouží pro spouštění testů dvou agentů. V závěru práce následuje diskuze, která se věnuje vymezení limitů práce a možných vylepšení.

1 Sběratelské karetní hry

Sběratelské karetní hry označované jako CCG (z anglického „collectible card games“) nebo také TCG (z anglického „trading card games“) případně DCCG (z anglického „digital collectible card games“) jsou hry, ve kterých si hráč sestavuje hratelný balíček z kolekce herních karet různých druhů. Tuto kolekci si hráč postupně rozšiřuje zakupováním nebo výměnou karet či zakupováním takzvaných *boosterů*, což jsou balíčky určitého počtu předem neznámých karet. V případě digitálních her se ke koupi těchto *boosterů* využívá herní měna, kterou lze získat pravidelným hraním nebo lze jednoduše koupit za reálné peníze. Hlavní rozdíl mezi digitální a papírovou verzí těchto her je především v tom, že v digitálních CCG zpravidla nelze směňovat karty s ostatními hráči, ale existuje způsob, jakým je možno *rozložit* nehodící se karty na nějaký druh *šrotu* a z něj následně složit jiné karty.

Všechny karty mají také určitou *vzácnost* nebo raritu. Rarity jsou pro každou hru různé, ale většinou se jedná o rarity typu (vzestupně) *common*, *uncommon/rare*, *epic* a *legendary*. Se stoupající raritou také stoupá cena karty a především její herní hodnota. Je pravidlem, že karta vyšší rarity mívá lepší herní hodnoty, a tím také větší vliv na průběh hry. Proto se často omezuje počet nejvzácnějších karet pouze na jednu v hratelném balíčku.

Na rozdíl od deskových her nebo běžných karetních her tak nestačí pouze hrací set nebo univerzální karetní sada, ale každý hráč hraje se svými vlastními kartami. Balíček je většinou omezen maximálním a minimálním počtem karet, přičemž je limitován i maximální počet jednotlivých karet v balíčku (například v Hearthstonu je to balíček o právě 30 kartách, přičemž jedna karta se může vyskytovat maximálně dvakrát a takzvané legendární karty pouze jedenkrát). Při sestavování balíčku se hráč většinou drží předem promyšlené strategie, která velmi ovlivňuje průběh hry a způsob, jakým plánuje dosáhnout vítězství. Takto sestavený hratelný balíček je připraven pro hru dvou hráčů.

Nové karty jsou většinou představovány v takzvaných *expanzích*, kdy je vydána nová sada herních karet, obvykle s nějakou tématikou a příběhem, přinášející nové herní mechaniky, nové synergie s předchozími kartami a nová klíčová slova. Aby hráč nemusel disponovat všemi kartami ze všech historických expanzí, a aby mohl být schopný poskládat obstojný balíček, existují různé druhy formátů hry. Nejznámějším je *Standard*, ve kterém lze hrát pouze s kartami z expanzí, které vyšly například v posledních dvou letech. Počet expanzí ve standardu se různí, jelikož většinou se určí nová expanze, po které proběhne *rotace* starých, aby se standard neměnil po každé nové expanzi, ale třeba jen jednou ročně. Zejména pro nováčky je tento formát nejvhodnější, jelikož je omezen počet karet, které se hráč musí naučit a které se snaží sbírat do své kolekce. Dalším příkladem formátu je *Wild* v Hearthstonu, ve kterém může hráč využít karty ze všech expanzí, čili skládá balíček ze všech expanzí.

Herní balíčky se skládají takovým způsobem, aby se karty navzájem doplňovaly a využívaly synergii k dosažení výhry danou strategií. Je nespočet možností,

jakým se dá poskládat kvalitní fungující balíček, ale obecně lze rozlišovat 3 druhy archetypů:

- Aggro - agresivní archetyp balíčku, který využívá velkého počtu levných jednotek s vysokým poškozením, spelly s přímým damage do oponenta a snaží se ukončit hru co nejrychleji, dříve než je oponent schopen získat *board control* a stabilizovat se.
- Midrange - pomalejší archetyp balíčku, který se od rané fáze snaží získat *board control* a každým tahem postupně vylepšovat své jednotky. V *mid game* fázi má již připravené silné jednotky. Udržuje agresivní tempo, využívá připravené převahy na boardu a cílí na vítězství v průběhu *prostřední fáze* hry, než začne postupně ztrácet prostředky pro *late game*.
- Control - pomalý archetyp balíčku, využívající *removal* karty a jednotky s velkým počtem životů, jež slouží pro ničení oponentových jednotek a blokování útoků k přečkání *early game* a *mid game*. Po dosažení pozdní fáze hry hraje silné drahé karty jednotek nebo kouzel a kontroluje hru. Snaží se oponenta přehrát vyčerpáním jeho zdrojů.

Samotná hra probíhá podle pravidel. Ta se samozřejmě v mnoha ohledech různí podle dané hry, ale zůstávají základní rysy, které mají všechny CCG společné. Proto definujeme základní názvosloví, které popisuje fáze hry a které slouží k pojmenování herních objektů.

1.1 Názvosloví

- Deck - Herní balíček karet.
- Hand - Aktuální karty na ruce hráče.
- Board - Herní plocha.
- Graveyard - Odhazovací balíček pro použité nebo vyřazené karty. Každý hráč má svůj vlastní.
- Turn - Fáze tahu/kola hráče.
- Mulligan - Fáze výběru počátečních karet. Možnost znovu rozdat určitý počet karet.
- Draw - Fáze táhnutí karty z balíčku.
- Combat - Fáze souboje jednotek.
- End Turn - Fáze ukončení kola/tahu.
- Early game - Raná fáze hry

- Mid game - Pokročilá fáze hry
- Late game - Pozdní fáze hry
- Mana - Zdroj pro hraní jednotlivých karet. Každé kolo má hráč omezené množství.
- Mana Cost - Cena many pro zahrání karty.
- Minion/Creature - Karta jednotky, která zůstává na herní ploše.
- Attack - Hodnota poškození daného miniona.
- Health - Hodnota zdraví daného miniona.
- Damage - Hodnota poškození například ze spellu nebo z jiného zdroje (creatury/miniona, zbraně, atd.)
- Spell - Karta kouzla, která má jednorázový efekt a ihned po zahrání končí v graveyardu.

1.2 Základní rysy a průběh hry

Základní hra probíhá jako tahová strategie dvou hráčů. Cílem hry je snížit počet oponentových *životů* na 0. Každý hráč má k dispozici svůj zvolený deck, graveyard a board. Náhodně se zvolí pořadí hráčů. Druhý hráč získává nějakou kompenzaci, například více karet na výběr nebo speciální kartu navíc. Hra začíná fází *mulligan*, kdy oba hráči táhnou určitý počet karet z balíčku a určí, jestli si je chtějí nechat, nebo některé zamíchat a táhnout znovu, nebo zamíchat a táhnout znovu všechny karty.

Následuje *počáteční* fáze tahu hráče. Ta nejčastěji začíná táhnutím karty z balíčku, hráči se také například obnoví mana a mohou být také spouštěny nějaké speciální efekty karet zahraných v minulých kolech nebo zahraných oponentem.

Další je *hlavní* fáze. Zde je možné hrát libovolné karty. K jejich zahrání musí hráč zaplatit danou cenu many. Mana se získává různým způsobem. Někdy se každému hráči na začátku tahu alokuje určitý počet many, který se zvyšuje každé kolo o jedna (například v HS). Jindy je potřeba zahrát speciální kartu, která generuje manu každé kolo (MTG). V případě MTG patří k častým problémům například nízký počet těchto karet, čímž dojde k *vyhladovění* hráče, jelikož nemá dostatek prostředků k hraní karet. V opačném případě je *přesycen*, a přestože má prostředky k utrácení, nemá je do čeho investovat.

Další fází je *combat* fáze. Jak bylo stručně popsáno v názvosloví, jistý druh karet jsou takzvaní *minioni*. Je to karta jednotky, která po zahrání zůstává na boardu do té doby, dokud neklesne její počet životů na 0 nebo dokud není odstraněna nějakým spellem. V této fázi tedy probíhá souboj těchto jednotek. Průběh samotného souboje se opět různí, ale princip je takový, že jednotky bojující mezi

sebou si udělí poškození a jednotky s 0 nebo méně životy jsou odstraněny do *graveyardu*.

Poslední fáze kola se nazývá *koncová* fáze. Zde končí určité speciální efekty, které mají platnost jedno kolo, spouští se efekty na konci kola a další. Končí zde tah hráče a přechází se do počáteční fáze oponentova tahu.

Některé CCG mají striktně oddělené jednotlivé fáze (MTG), kdežto jiné mají jednu fázi, ve které lze hrát všechny druhy karet v libovolném pořadí, ale i provádět útok, který funguje jiným způsobem, než typická *combat* fáze (HS).

Následně jsou představeny dvě nejpopulárnější a nejrozšířenější sběratelské karetní hry. Jsou zběžně popsány rysy a mechanismy.

1.3 Magic: The Gathering

Magic: The Gathering (MTG) je první CCG vydaná v roce 1993. Postupem času začaly vznikat i její digitální verze, například Magic: The Gathering Online (MTGO) vydaná v roce 2002 nebo Magic: The Gathering Arena v beta verzi od roku 2017 a oficiálně v plné verzi v roce 2019.

V MTG existuje velké množství herních karet. Tyto karty jsou rozděleny do pěti tříd nebo frakcí podle barev, z nichž každá má svůj specifický herní styl, vyniká v určité oblasti a naopak je slabší v jiné. Níže je výčet těchto frakcí a jejich barva:

- Černá (swamp)
- Bíla (plains)
- Červená (mountain)
- Zelená (forest)
- Blue (island)

Hráč může při skládání balíčku využít karty všech barev, ale nejběžnější jsou balíčky dvoubarevné, které kombinují silné stránky obou barev.

Karty jednotek kromě hodnoty poškození a životů mohou mít další vlastnost. Vlastnosti jsou definovány jako *keywords*, tedy klíčovými slovy, které popisují funkcionalitu dané vlastnosti. Například klíčové slovo *lifelink* znamená, že jakékoliv poškození, které jednotka udělí oponentovi nebo jeho jednotkám, se přičte k aktuálnímu zdraví hráče. Karty mohou mít navíc ještě vlastnost *aury*, které po dobu pobytu na boardu ovlivňuje ostatní jednotky, například: „Ostatní jednotky mají +1/+1“ znamená, že všechny ostatní jednotky mají +1 k poškození a +1 ke zdraví.

Hlavní charakteristika MTG je nekonzistentní získávání many. Mana je generována specialními kartami zemí, takzvanými *landy*. Každé kolo může hráč vyložit jednu kartu landu. Nevýhoda je tedy v tom, že hráč musí v balíku hrát dostatečný počet těchto landů, aby se mu nestalo, že nebude mít dostatek many pro hraní ostatních karet.

Pro zaplacení many k zahrání karty je potřeba mít dostatek landů. Land se poté *tapne*, což znamená otočení o 90 stupnů, a hráči je přičtena jedna mana. Funkce tapování se aplikuje i na ostatní karty. Například jednotka, která úspěšně provedla útok na hráče, se tapne a v následujícím oponentově tahu tak nemůže blokovat. Stejně tak některé karty mají speciální vlastnosti, které k zahrání také vyžadují tapnutí. Creatura vyvolaná v aktuálním tahu hráče má *ressurection sickness* a může útočit až v dalším kole.

MTG má přesně dané herní fáze (obrázek 1) určující průběh hry. V hlavní fázi lze hrát všechny druhy karet. Hráč tedy může seslat kouzlo, vyvolat creaturu, zahrát artefakt nebo kartu s aura efektem, která má vliv na ostatní karty. *Combat* fázi zahajuje hráč na tahu deklarací útočících jednotek. Obránce vybírá své creatury, které budou blokovat útok na hráče. Má výhodu v tom, že může vybrat jednotlivé blokery a pořadí, ve kterém budou bránit. Udělení poškození se počítá tak, že obě creatury útočí zároveň a udělí si *poškození* rovné hodnotě jejich útoku. Pokud počet životů creatury klesne pod 0, je vyřazena do graveyardu. To je strategická výhoda pro bránícího hráče, jelikož může vybrat blokující creatury tak, aby zneškodnil protihráčovy creatury a zároveň zachoval své. Na konci kola se totiž počet životů creatur obnoví na počáteční hodnotu.

V MTG není nijak omezen počet jednotek na *boardu*. Maximální počet karet na ruce je sedm a pokud má hráč na konci kola více, diskartuje dle své volby přebývající počet karet.

Hra končí ve chvíli, kdy hodnota zdraví jednoho z hráčů klesne na 0 nebo když jednomu z hráčů dojdou karty v decku a již nemůže táhnout další kartu. Druhou možnost výhry také využívají určité druhy strategií, které se snaží kontrolovat hru a konstantně diskartovat oponentovy karty v decku.

Obrázek 1: Fáze hry v MTG

Mulligan je kritickou součástí strategie hráče. V MTG každý hráč na začátku

hry dostane 7 karet. Pokud je s výběrem spokojen, ponechá si všechny karty a začíná počáteční hráč. V opačném případě může zamíchat karty zpět do balíčku a vytáhnout si o jednu kartu méně. Toto lze opakovat do té doby, dokud není hráč spokojen, nebo dokud již nemá pouze jedinou kartu. Strategie mulliganu se různí podle archetypu balíčku, ale dobrá počáteční *handa* by měla mít 3-4 landy a zároveň hratelné karty do *early game*.

K vyvážení výhody začínajícího hráče se využívá mechanismu, kdy první hráč ve svém prvním tahu přeskočí *draw* fázi, kdežto druhý hráč ve svém prvním kole táhne kartu jako v každém dalším tahu.

Další charakteristikou MTG je možnost hrát karty v oponentově tahu. Po zahrání libovolné karty má tedy protihráč možnost reagovat na tuto kartou určitými spelly, které je možno zahrát mimo svůj tah. Pokud využije možnost a kartu zahraje, iniciativa se opět předá zpět původnímu hráči na tahu, který opět může reagovat na oponentův tah. Tímto způsobem vzniká *zásobník akcí*, který se následně vyhodnocuje jako klasický LIFO zásobník (poslední zahraná karta se vyhodnotí jako první atd.). Díky tomuto mechanismu jsou ve hře přítomny například karty typu *counter spell*, které lze zahrát jako reakce na oponentovu kartu a vyrušit její efekt (odstranit ze zásobníku akcí).

Obrázek 2: Ukázka boardu v MTG

1.4 Hearthstone: Heroes of Warcraft

Hearthstone (HS) je DCCG od firmy Blizzard Entertainment, která vyšla v roce 2014. Přejímá základní strukturu CCG od MTG (kolekce karet různých rarit, herní frakce a jejich rozdílný styl strategií, herní princip tahové strategie dvou hráčů).

Karty jsou, podobně jako v MTG, rozděleny do herních tříd. Každou herní třídu charakterizuje postava ze hry Warcraft. Jedná se o těchto 10 tříd:

- Demon Hunter
- Druid
- Paladin
- Priest
- Warrior
- Hunter
- Warlock
- Shaman
- Mage
- Rogue

Každá postava má zase specifické karty a hodí se více pro určitý druh strategie. Toto navíc ještě podtrhuje herní mechanika unikátní pro HS, kdy každá postava má svou *hero power*. Jedná se o schopnost, kterou lze použít jednou za kolo. Není potřeba hrát žádnou kartu, pouze zaplatit požadovanou cenu many.

Oproti MTG využívá konzistentní systém získávání many, kdy v prvním kole začínají oba hráči s jedním mana krystalem a na začátku tahu jsou všechny krystaly obnoveny a jejich počet je navýšen o 1. Tento herní mechanismus eliminuje tolik kritizovaný nedostatek MTG, kdy hráč v rané fázi hry není schopen hrát, jelikož má nedostatek landů, nebo mu naopak dojdou zdroje v pozdní fázi hry, protože si několikrát po sobě vytáhl land.

V HS nejsou rozděleny jednotlivé fáze hry. Na začátku kola si hráč táhne kartu a následně může provést jakoukoliv akci; útočit, vyvolávat miniona, hrát spell. *Combat* také probíhá odlišně. Namísto deklarace útočících jednotek může hráč přesně určit, jestli s danou jednotkou útočit přímo na oponenta nebo na jeho jednotky. To je obrovský rozdíl oproti MTG a výhoda je tedy na straně útočníka, jelikož může přesně kontrolovat distribuci poškození oponentových jednotek. Na rozdíl od MTG se ale jednotkám neobnovuje zdraví a poškození tak zůstává do dalších kol. Existuje však *keyword* vlastnost jednotek zvaná *taunt*, kterou musí oponent upřednostnit při útoku.

Board má v HS omezený maximalní počet vyvolaných jednotek na sedm. Maximální počet karet na ruce je omezen na deset a každá další následující táhnutá karta je spálena.

Další specifickou skupinou karet pro HS jsou *weapons* čili zbraně. Ty mají hodnotu poškození a *durability*, popisující kolik útoků s nimi lze provést.

Obrázek 3: Ukázka boardu v HS

Průběh kola je také přímočařejší, jelikož až na výjimky nelze hrát v oponentově tahu. Jediným způsobem vyvolání akce v oponentově tahu jsou karty typu *secret*, které jsou pro oponenta neznámé a spustí se po určité akci v protihráčově tahu (například: při zahrání spellu vyruší daný spell).

Mulligan v HS funguje tak, že první hráč dostane na výběr ze tří karet. Může si zvolit libovolný počet, který se zamíchá zpět do balíčku a náhodně si znovu vytáhne stejný počet z balíčku. Pro druhého hráče platí stejný princip, ale vybírá si ze čtyř karet.

Výhoda prvního hráče se v HS řeší tak, že druhý hráč obdrží speciální kartu zvanou *The Coin*, která slouží jako jednorázový mana krystal. Vzhledem ke konzistentní křivce získávání many dostane tento hráč možnost zahrát v jenom kole silnější kartu než první hráč. Také začíná ještě s jednou kartou navíc.

2 Umělá inteligence

V této části se budeme věnovat umělé inteligenci ve hrách. Nejdříve představíme terminologii a základní principy návrhů algoritmů pro umělou inteligenci ve hrách s úplnou informací. Následně popíšeme *state-of-the-art* neboli nejaktuálnější či nejmodernější přístup k řešení her. Dále nastíníme možnosti odstranění neúplně informace, aby bylo možné využít stávající přístupy i pro herní doménu, ve které hraje neurčitost značnou roli.

2.1 Základní terminologie a přístupy řešení her

Tato kapitola představí základní terminologii přístupu řešení her. Při definicích uvažujeme hru dvou hráčů s nulovým součtem a úplnou informací. Popíšeme sestavování herního stromu pro libovolnou herní doménu a základní způsoby vyhledávání optimální strategie v tomto stromu a optimalizace vyhledávacích algoritmů.

Vzhledem k problému exponenciálního růstu možností v herním stromu vznikaly různé optimalizace vyhledávání. Algoritmy můžeme rozdělit podle přístupu do dvou skupin:

- Depth-first
- Best-first

Depth-first prohledává strom *do šířky*. Tento přístup má nižší nároky na paměť, ale jeho časová složitost rychle roste kvůli obrovskému množství možných kombinací s přibývající hloubkou herního stromu. Proto se hloubka vyhledávání omezuje podle počtu přibývajících možností. Z toho také název *depth-first*.

Naproti tomu *best-first* prohledává herní strom *do hloubky*. Tyto algoritmy jsou schopny na úkor úzkého herního prostoru prozkoumávat herní strom až k terminálním uzlům. Problémem je však v tomto případě vysoká paměťová náročnost, jelikož musí udržovat informace o celém herním stromu.

2.2 AND/OR Tree a Minimax Tree

Předpokládejme, že hráč p se snaží dokázat, že z daného stavu dosáhne vítězství. Potom p musí dokázat, že existuje *alespoň jedna* akce, která zaručí vítězství. Pokud je však na tahu oponent, p musí být schopen vyhrát proti *všem* oponentovým akcím. Rekurzivním opakováním se pak dostáváme ke konceptu AND/OR stromového vyhledávání. AND/OR strom je tvořen dvěma druhy uzlů: AND uzlem a OR uzlem. OR uzly přísluší stavu, ve kterém se rozhoduje první hráč a AND uzly stavu, ve kterém je na tahu oponent. Změna stavu po akci a ze stavu uzlu m na stav uzlu n je značena jako orientovaná hrana z uzlu m do uzlu n. Všechny uzly kromě kořenového mají rodičovský uzel. Pokud předpokládáme střídání kol po jednom tahu, tak platí, že každý OR uzel je potomek AND uzlu a každý AND uzel je potomek OR uzlu.

Každý uzel může nabývat tří hodnot: *výhra*, *prohra* nebo *neznámo*. Uzel s hodnotou *výhra* nebo *prohra* označuje jistou výhru nebo prohru. Pro uzel s hodnotou *neznámo* ještě nebyl dokázán výsledek hry. Tento uzel musí být ještě dále prozkoumán. Toho lze docílit *expandováním* uzlu, kterým vygenerujeme všechny potomky, kteří reprezentují všechny možné tahy, spojené s rodičovským uzlem již zmíněnou orientovanou hranou.

Uzel bez potomka se nazývá *terminální* a má hodnotu *výhra* nebo *prohra*. Uzel s alespoň jedním potomkem se nazává *interní*. *Listový* uzel je dosud neexpandovaný uzel bez potomků s neznámou hodnotou, který musí být ještě expandován a určeno, zda se jedná o interní nebo terminální uzel.

AND/OR vyhledávání se snaží vyřešit, zda je daný kořen výhra nebo prohra. Hodnota interního uzlu se vypočítá z hodnot jeho potomků. OR uzel má hodnotu výhra, pokud alespoň jeden z potomků má také hodnotu výhra. V případě hodnoty prohra u všech potomků má OR uzel hodnotu taktéž prohra. V případě výhry AND uzlu je nutné, aby všichni potomci měli také hodnotu výhra. Jakmile má jeden z potomků hodnotu prohra, znamená to prohru i pro samotný AND uzel. Má-li alespoň jeden potomek neznámou hodnotu, pak má i rodičovský uzel neznámou hodnotu a k určení hodnoty je potřeba všechny tyto potomky expandovat.

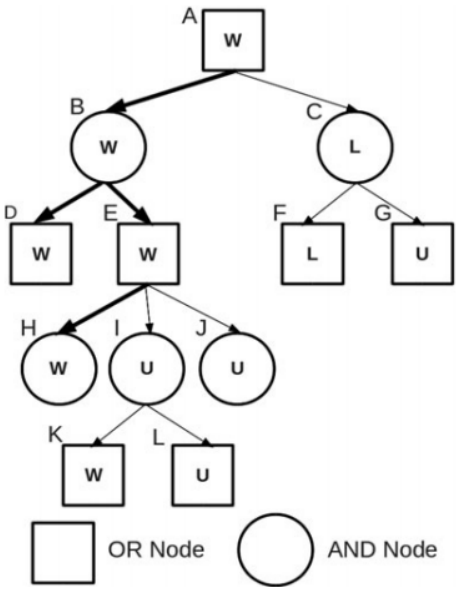
Uzel s hodnotou výhra se také nazývá *dokázaný*, přičemž uzlu, který má hodnotu prohra, se také říká *vyvrácený*. V případě, že je daný uzel dokázaný, tak podstrom původního AND/OR stromu obsahje výherní strategii. Tento se nazývá *dokázaný* strom, označíme jej T s kořenem r, a lze jej sestrojit následovně:

1. T obsahuje r
2. Pro každý interní uzel OR, T obsahuje alespoň jednoho potomka.
3. Pro každý interní AND uzel, T obsahuje všechny potomky.
4. Všechny terminální uzly jsou výhry.

Dále definujeme *vyvrácený* strom, který obsahuje výherní strategii pro protihráče. Ten lze sestrojit podobným způsobem, jako v definici výše, stačí prohodit AND a OR a v posledním bodě vyžadovat, aby všechny terminální uzly byly prohry.

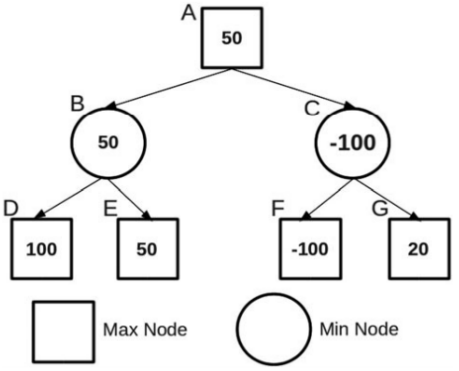
V AND/OR stromu se může vyskytovat mnoho dokázaných stromů, ale k vyřešení stačí najít pouze jeden takový strom. Například na obrázku 4 je ukázka AND/OR stromu, ve kterém byl nalezen dokázaný strom (označený tučnými hranami). Nyní by bylo bezpředmětné prohledávat zbytek listových uzlů, jelikož již známe výherní strategii.

Minimaxové stromy jsou zobecněním AND/OR stromů. Místo Boolovských hodnot se listovým a terminálním uzlům vypočítává číselné skóre na základě aktuálního stavu. OR uzel se v minimaxovém stromu nazývá *maximální* uzel a AND uzel se nazývá *minimální* uzel. Skóre se vypočítává voláním *ohodnocovací*



Obrázek 4: Ukázka AND/OR stromu

funkce, která stanovuje přibližnou šanci hráče na vítězství. Vyšší skóre představuje výhodnější pozici pro prvního hráče. Stejně jako v AND/OR stromu, skóre každého interního uzlu se získává výpočtem od listových uzlů směrem ke kořenu. Interní maximální uzel vybírá maximální skóre z potomků a snaží se tam dosáhnout co nejlepší herní pozice. Naproti tomu v minimálním uzlu se snaží protihráč minimalizovat výhodu prvního hráče vybráním uzlu s minimálním skóre (obrázek 5).



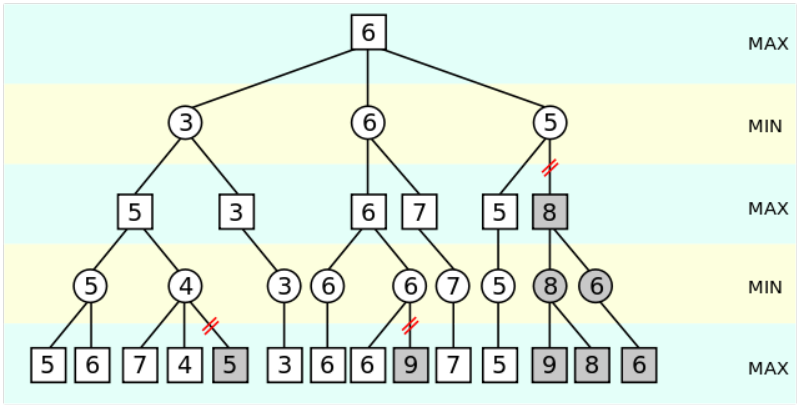
Obrázek 5: Ukázka Minimaxového stromu

2.3 $\alpha\beta$ pruning

$\alpha\beta$ *pruning* je optimalizace vyhledávání v minimaxovém stromu [1]. Patří do skupiny *depth-first* prohledávající herní strom do šířky. Hlavní princip tohoto algoritmu spočívá v omezení počtu procházených uzlů v případě, že v daném stavu nemohou dosáhnout lepšího výsledku.

$\alpha\beta$ uchovává nejnižší hranici α a nejvyšší hranici β pro minimaxový herní strom. Obecné $\alpha\beta$ vyhledávání prozkoumává strom do určité pevně dané hloubky a v průběhu aktualizuje $\alpha\beta$ hodnoty, které se používají k *ořezávání* podstromů, které jsou již pro výpočet zanedbatelné. Algoritmy založené na této optimalizaci tedy vrací stejný výsledek jako běžné minimaxové vyhledávání, ale ořeže spoustu zbytečných větví stromu, čímž šetří čas, který lze využít například pro rozšíření vyhledávání do větší hloubky stromu.

Na obrázku 6 je ukázka herního stromu. Vyšedlé části stromu jsou ořezané větve, které není potřeba procházet, jelikož již nemohou dosáhnout lepšího výsledku. Například u nejpravějšího podstromu kořenového uzlu po prohledání levé části podstromu víme, že hodnota tohoto uzlu bude ≤ 5, jelikož hledáme minimum. Avšak z prostředního podstromu již víme, že hodnota kořenového uzlu bude ≥ 6, jelikož hledáme maximum. Můžeme tedy ořezat celou část podstromu, protože nikdy nezískáme větší hodnotu než 5.



Obrázek 6: Ukázka $\alpha\beta$ pruning

2.4 Proof-Number Search

Proof-Number search (PNS) patří do skupiny algoritmů *best-first*, tedy prohledávající herní strom do hloubky [2]. Základní myšlenka algoritmu je počítání *čísla dokazatelnosti* nebo *čísla vyvratitelnosti* pro ocenění jednotlivých uzlů a na jejich základě prozkoumává strom od nejsnadněji dokazatelných nebo vyvratitelných uzlů. Díky tomuto přístupu dokáže efektivně prohledat úzkou část stromu do hloubky.

Formálně definujeme číslo dokazatelnosti uzlu jako minimální počet listových uzlů v podstromu, které musí být dokázány, aby se potvrdilo, že jsou výhra,

zatímco číslo vyvratitelnosti je minimální počet listových uzlů v podstromu, které musí být vyvráceny, aby se potvrdilo, že jsou prohra. Čím menší je číslo dokazatelnosti/vyvratitelnosti, tím jednodušeji lze odhadnout výhru/prohru.

Nechť u je uzel s potomky $u_1, u_2, ..., u_k$. Jeden dokázaný potomek OR uzlu je dostačující pro důkaz výhry, kdežto u AND uzlu musí být všichni potomci dokázaní (a obdobně pro vyvrácený uzel). Číslo dokazatelnosti d a číslo vyvratitelnosti v uzlu u se vypočítá následovně:

1. Pro dokázaný terminální uzel u platí $d(u) = 0$ a $v(u) = \infty$.
2. Pro vyvrácený terminální uzel u platí $d(u) = \infty$ a $v(u) = 0$.
3. Pro nenavštívený listový uzel u platí $d(u) = v(u) = 1$.
4. Pro vnitřní OR uzel u s potomky $p_1, p_2, ..., p_k$ platí $d(u) = min(d(p_1), ..., d(p_k))$ a $v(u) = v(p_1) + ... + v(p_k)$.
5. Pro vnitřní AND uzel u s potomky $p_1, p_2, ..., p_k$ platí $d(u) = d(p_1)+...+d(p_k)$ a $v(u) = min(v(p_1), ..., v(p_k))$.

Na obrázku 7 je uveden příklad PNS. Čislo dokazatelnosti je uvedeno nad číslem vyvratitelnosti uvnitř každého uzlu. Uzel I je v tomto případě terminální uzel značící prohru. Uzly D, F, G, H, J, K jsou listové uzly s výchozí hodnotou 1. Pro výpočet čísel interních hodnot je potřeba využít výše popsaných pravidel. Například hodnota čísel uzlu E se spočítá jako $d(E) = min(d(I), d(J), d(K)) = min(\infty, 1, 1) = 1$ a $v(E) = v(I) + v(J) + v(K) = 0 + 1 + 1 = 2$.

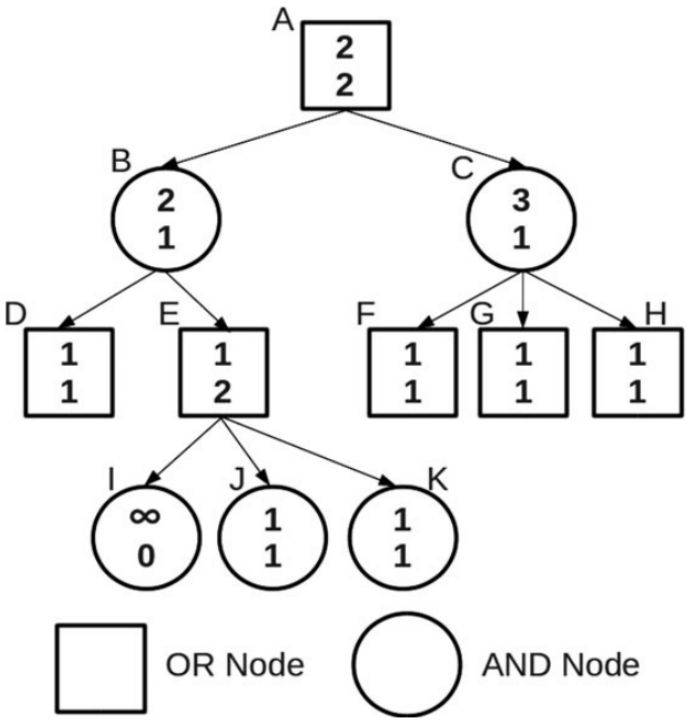
2.5 Monte Carlo Tree Search

Základní myšlenka Monte Carlo Tree Search (MCTS) vychází z Monte Carlo metod využívající náhodnost ke statistickému řešení deterministických problémů. Obecný koncept MCTS lze shrnout na aproximaci hodnot jednotlivých akcí ve stromu pomocí náhodného prohledávání stromu do hloubky a úpravou statistiky podle výsledků těchto náhodných her.

Postup tohoto prohledávání zahrnuje iterativní sestavování herního stromu v rámci daného výpočetního rozpočtu. Po vyčerpání tohoto rozpočtu se vyhledávání zastaví a vybere se uzel s nejlepším výsledkem. Každý uzel představuje možnou akci v rámci dané domény a orientovaná hrana z rodičovského uzlu do potomka značí přechod z původního stavu do stavu, který je výsledkem aplikace dané akce.

Samotné vyhledávání probíhá iterací následujících čtyř kroků:

- Selekce - Počínaje kořenovým uzlem se iterativně prohledává strom a vybírá se nejnaléhavější uzel k expandování. Expandovat lze neterminální uzel, který má nějaké nenavštívené potomky.



Obrázek 7: Ukázka PNS

- Expanze - Expanze uzlu znamená vybrat jednu z neprozkoumaných akcí a přidat korespondujícího potomka do herního stromu.
- Simulace - Simulace se spouští u nových uzlů a získává se hodnota daného uzlu podle *default policy*.
- Zpětná propagace - Výsledek simulace je propagován nahoru herním stromem přes vybrané uzly a aktualizuje jejich statistiky.

Tyto kroky lze dále rozdělit do dvou skupin:

- Tree policy - Vybere nebo vytvoří listový uzel z uzlů, které jsou již zahrnuty v herním stromu (selekce a expanze).
- Default policy - Z daného neterminálního uzlu provádí simulaci až k terminálnímu uzlu, aby určil výslednou hodnotu uzlu (simulace)

Zpětná propagace výsledku nespadá do žádně skupiny, jelikož se jedná pouze o aktualizaci statistických výsledků jednotlivých uzlů herního stromu.

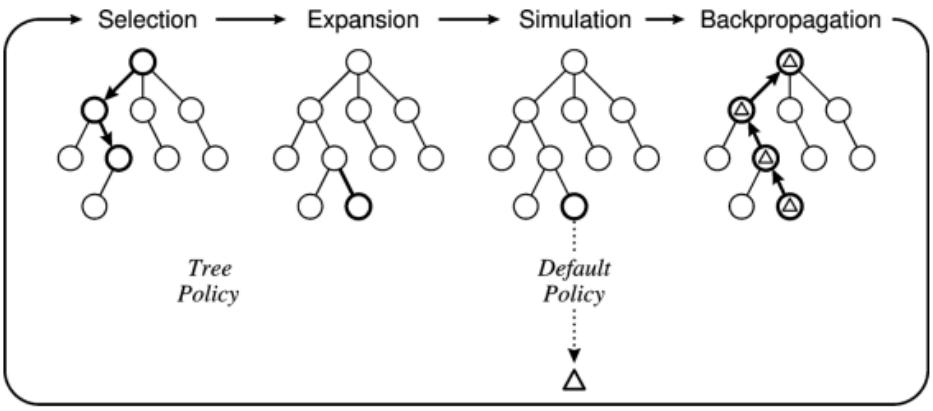
Pseudokód obecného přístupu řešení MCTS popisuje algoritmus 1, kde v_0 je kořenový uzel se stavem s_0, v_l je poslední dosažený uzel vybraný během *tree policy* a k němu patřící stav s_l a Δ je výsledek terminálního stavu dosaženího v *default policy* ze stavu s_l. Výsledek celého vyhledávání $a(BestChild(v_0))$ je pak

function `MCTSSearch(`s_0`)`
  vytvoř kořenový uzel v_0 se stavem s_0
  while *v rámci časového rozpočtu* **do**
    $v_l \leftarrow$ `TreePolicy(`v_0`)`
    $\Delta \leftarrow$ `DefaultPolicy(`$s(v_l)$`)`
    `Backup(`v_l, Δ`)`
  end
  return a(`BestChild(`v_0`)`)
end

Algorithm 1: Ukázka obecného MCTS algoritmu.

akce patřící uzlu, který je vyhodnocen jako nejlepší. Výběr nejlepšího uzlu záleží na implementaci *BestChild*.

Grafickou ukázku jedné iterace obecného algoritmu MCTS lze vidět na obrázku 8. Od kořenového uzlu t_0 jsou rekurzivně vybírány uzly podle dané vyhodnocovací funkce, dokud se nedosáhne uzlu t_n, který je buď terminální nebo který ještě není plně expandován. Následně se vybere nenavštívená akce a ze stavu s a do herního stromu je přidán uzel se stavem s', který je výsledkem přechodu ze stavu s při akci a. Tímto končí *tree policy* v aktuální iteraci.



Obrázek 8: Ukázka jednotlivých kroků MCTS

Následuje simulace pro tento nově přidaný uzel t_l. Simulací se získá hodnota Δ, která je následně zpětně propagována přes vybrané uzly a jsou aktualizovány statistické údaje jednotlivých uzlů. Inkrementuje se počet navštívení uzlu a jeho odměna Q na základě hodnoty Δ. Hodnota Δ může být jednoduše diskrétní hodnota výhra/prohra/remíza, nebo vektor hodnot pro jednotlivé agenty v případě komplexní hry více hráčů.

Jakmile je dosažen terminální uzel nebo vyprší čas určený simulací, vyhledávání se ukončí a je vybrána akce a z kořenového uzlu t_0. Výběr této akce záleží na zvoleném způsobu. V literatuře se nejčastěji objevují různé varianty těchto čtyř metod [3]:

- *Max-child*: vrací potomka v s nějvětší hodnotu odměny, Q

$$v_{vic} = \underset{v \in potomci\ v_0}{\arg\max}\ Q(v).$$

- *Robust-child*: vrací potomka v s nejvyšším počtem navštívení, N

$$v_{vis} = \underset{v \in potomci\ v_0}{\arg\max}\ N(v).$$

- *Max-robust-child*: vrací potomka v s největším součtem odměny Q a navštívení, N

$$v_{rob} = \underset{v \in potomci\ v_0}{\arg\max}\ (Q(v) + N(v)).$$

- *Secure-child*: vrací potomka v, který maximalizuje *lower confidence bound*,

$$v_{lcb} = \underset{v \in potomci\ v_0}{\arg\max}\ \frac{Q(v)}{N(v)} - c\sqrt{\frac{2\ln N(v_0)}{N(v)}}$$

2.5.1 Multi-Armed Bandits

Multi-armed bandit je problém z teorie pravděpodobnosti. Jedná se o problém sekvenčního rozhodování, ve kterém se opakovaně vybírá jedna z K akcí tak, aby se maximalizovala kumulativní odměna [4]. Výběr možnosti bývá složitý, jelikož hodnoty odměn jsou často neznámé a lze je získat až zpětnou indukcí. To vede k problému *exploitation-exploration*, ve kterém se musí správně vyvážit volba akce, která je dobře prozkoumaná a zdá se jako nejlepší, a na druhé straně prohledávání ostatních v daném okamžiku horších akcí, které se ale nakonec mohou ukázat v pozdější fázi jako výhodnější.

Pro řešení problémů banditů se definuje *Upper Confidece Bound* (UCB), jež představuje maximální možnou míru přesvědčení, že daná *paže* bandity bude optimální. Nejběžnější politika pro UCB je UCB1 [4]

$$UCB1 = \bar{X}_j + \sqrt{\frac{2\ln n}{n_j}}$$

kde $\bar{X}_j$ je průměrná odměna paže j, n_j je počet zahrání paže j a n je celkový počet tahů. Levá část vzorce $\bar{X}_j$ je odpovědná za *exploitation* neboli vybírání nejslibnější paže a druhá část $\sqrt{2\ln n/n_j}$ za *exploration*, která vyvažuje výběr nejslibnější a nabádá k prohledávání méně navštívených paží.

2.5.2 Upper Confidence Bound for Trees (UCT)

UCT patří mezi nejpopulárnější algoritmy v MCTS. Důležitou součástí při vytváření vyhledávacího stromu je výběr potomka. UCB1 je jednoduchý a efektivní

způsob jak rychle určit přibližnou hodnotu jednotlivých akcí. UCT přejíma řešení *exploitation-exploration* problému z UCB1:

$$UCT = \bar{X}_j + C_p\sqrt{\frac{2\ln n}{n_j}}$$

kde n je počet navštívení aktuálního rodičovského uzlu n, n_j je celkový počet navštívení potomka j a $C_p > 0$ je konstanta. Za nejslibnějšího potomka je vybrán ten, který má největší hodnotu UCT, přičemž remíza se většinou rozhoduje náhodným výběrem.

Pro hodnotu $\bar{X}_j$ se používá číslo z intervalu $\langle 0, 1\rangle$. V případě, že n_j je 0 (potomek j ještě nebyl navštíven), pak UCT vrací ∞, a potomek tak má nejvyšší prioritu pro vybrání, aby všichni potomci byly prozkoumáni alespoň jednou. Tímto se dosáhne účinného iterativního lokálního vyhledávání.

Vyvažování *exploration* a *exploitation* funguje tak, že každou návštěvou uzlu se zvětšuje jmenovatel v druhé části vzorce, čímž se zmenšuje celý výraz pro prioritu prozkoumávání. Je-li však navštíven jiný potomek rodičovského uzlu, zvětší se čitatel u všech potomků a tím se zvětšuje priorita pro prozkoumávání ostatních potomků. Výhodou tak je i to, že u všech potomků je nenulová pravděpodobnost výběru, takže při dostatečném časovém rozpočtu je zaručeno, že se každá možnost prozkoumá alespoň jednou a tím může najít výherní sekvenci akcí, která by jinak nebyla prozkoumána, jelikož nedosahuje tak dobrých výsledků.

Exploration konstanta C_p se využívá pro snížení nebo zvýšení prohledávání méně navštívených uzlů. Pro hodnoty X_j v intervalu $\langle 0, 1\rangle$ se používá $C_p = 1/\sqrt{2}$ [4].

Každý uzel u si udržuje počet navštívení $N(u)$ a hodnotu odměny všech odehraných akcí $Q(u)$. Při každé odehrané hře se aktualizují tyto statistické hodnoty směrem ke kořenu. Po uplynutí časového rozpočtu se vyhledávání zastaví a vrátí se nejlepší akce.

Algoritmus 2 popisuje pseudokód UCT.

function `UCTSearch(`s_0`)`
  vytvoř kořenový uzel v_0 se stavem s_0
  while *v rámci časového rozpočtu* **do**
    $v_l \leftarrow$ `TreePolicy(`v_0`)`
    $\Delta \leftarrow$ `DefaultPolicy(`$s(v_l)$`)`
    `Backup(`v_l, Δ`)`
  end
  return $a($`BestChild(`v_0, 0`)`$)$
end
function `TreePolicy(`v`)`
  while v *není terminální* **do**
    if v *není zcela expandován* **then**
      return `Expand(`v`)`
    else
      $v \leftarrow$ `BestChild(`v, C_p`)`
    end
  end
  return v
end
function `Expand(`v`)`
  vyber $a \in$ nevyzkoušené akce z $A(s(v))$
  přidej nového potomka v' do v
  se stavem $s(v') = f(s(v), a)$
  a akcí $a(v') = a$
  return v'
end
function `BestChild(`v, c`)`
  return $\underset{v' \in \textit{potomci } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2lnN(v)}{N(v')}}$;
end
function `DefaultPolicy(`s`)`
  while s *je neterminální* **do**
    zvol $a \in A(s)$ náhodně
    $s \leftarrow f(s, a)$
  end
  return *výsledek pro stav s*
end
function `Backup(`v, Δ`)`
  while $v \neq$ *null* **do**
    $N(v) \leftarrow N(v) + 1$
    Qé$(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ *rodič* v
  end
end

Algorithm 2: UCT algoritmus

3 Druhy strategií agentů

V této části se budeme věnovat výběru strategií pro agenta. Ve zvoleném frameworku jsou pro ukázku uživateli připraveni 4 základní agenti, kteří mají různé druhy strategií. Tyto agenty a jejich strategie zde popíšeme. Dále představíme strategii využívající Monte Carlo Tree Search a popíšeme samotnou implementaci.

3.1 Framework pro simulaci

K simulaci byl využit Hearthstone AI Competetion [5], který je postaven na frameworku SabberStone spadající do kolekce frameworků pro simulaci HearthStonu skupiny vývojářů HearthSim. Tento framework pro simulaci HS byl vytvořen za účelem pořádání soutěží o nejlepšího agenta hrajícího HS ve dvou různých kategoriích. V první kategorii je cílem naprogramovat agenta hrajícího s předem sestaveným balíčkem 30 karet proti agentům s 9 různými balíky karet, z toho 6 balíků je dopředu známo a 3 jsou před zahájením soutěže pro řešitele neznámé. V druhé kategorii je agentům povoleno měnit karty ve svém balíčku v průběhu soutěže tak, aby byl schopen porazit co nejvíce soupeřů. Cílem tohoto agenta je tedy reagovat na změny strategií ostatních agentů modifikací své vlastní strategie a karet. Tato soutěž je součástí IEEE Conference on Games 2020.

3.2 Random Agent

Agent bez jakéhokoliv druhu strategie. Z množiny všech možných akcí vybere náhodně a nikdy nevyhodnocuje, který stav by po simulaci tohoto tahu byl nejvýhodnější. Slouží jako ukázka funkcionality agenta. Primárně určena pro demonstrativní účely pro nového uživatele frameworku, který má možnost se seznámit se základní kostrou herního agenta.

3.3 Greedy Agent

Strategie vybírající akci, která v aktuální konfiguraci zaručí nejvyšší možnou výplatu vyhodnocením otevřené části stavu hry. Jedná se o efektivní strategii, ale její limitace je pouze výběr jedné akce v aktuální situaci s tím, že nezjišťuje následující stavy a neprovádí jejich simulaci. Síla této strategie je tedy přímo úměrná kvalitě funkce vyhodnocující aktuální stav hry.

3.4 Beam Search Agent

Beam Search je heuristická strategie využívající iterativní vyhodnocování stavu a prozkoumává pouze určitý počet nejlepších akcí v aktuálním stavu. Jedná se o optimalizační algoritmus *best-first* přístupu vyhledávání, který snižuje potřebnou operační paměť. Omezuje totiž maximální šířku herního stromu pouze na nejslibnější akce, a tak nemusí uchovávat všechny ostatní možnosti. Tímto způsobem

je agent schopen rychle odehrát hru do pokročilejší fáze a zjistit, která akce se vyplatí hrát v rané fázi pro lepší výsledek v dlouhodobějším horizontu.

3.5 Dynamic Lookahead Agent

Dynamic Lookahead strategie využívá prohledávání herního stromu do určité omezené hloubky. Název *Dynamic* tedy znamená, že hloubka prohledávaného stromu se dynamicky mění na základě počtu možností v daném tahu. V případě, že je jen málo možností, zvětší hloubku vyhledávacího stromu. Pokud je velký počet možností, pak algoritmus degraduje na *greedy* strategii, kdy vyhledává pouze v aktuální úrovni. Jedná se tedy o *depth-first* heuristický algoritmus prohledávající herní strom do šířky.

DynamicLookaheadAgent připravený ve frameworku je ve skutečnosti vítězný agent loňského turnaje The HearthStone-AI Competetion 2019. Kromě této strategie má agent evolučně optimalizované hodnoty jednotlivých atributů v ohodnocovací *Score* funkci.

3.6 Monte Carlo Tree Search Agent

Monte Carlo Tree Search agent byl implementován na základě UCT algoritmu popsaného v kapitole 2.5. Původní záměr *default policy* odehrávání her až do konce a propagace výhry/prohry/remízy byl však téměř nerealizovatelný kvůli vzrůstající míře neurčitosti s každým dalším odehraným tahem. I po implementaci odhadování oponentových karet stále proběhlo pouze malé množství her, a tak byl příliš malý vzorek pro rozhodování a agent prohrával i proti obyčejnému greedy agentovi.

Simulace byla převedena z hraní do samotného konce na odehrání určitého počtu kol zadaného parametrem a heuristickým vyhodnocením aktuálního stavu hry. Místo výhry/prohry/remízi se tedy propaguje přibližná hodnota aktuálního stavu hry, která popisuje odměnu po zahrání jednotlivé akce.

Heuristická funkce pro hodnocení stavu vychází ze všech *score* funkcí připravených ve frameworku. Nicméně jsme ještě přidali více parametrů pro detailnější vyhodnocení aktuálního stavu hry.

3.7 Nahrazení neúplné informace

Pro implementaci a použití MCTS bylo velkou výzvou také odstranění neúplné informace. Jelikož MCTS ve své podstatě funguje na statistických údajích o odehraných kolech, bylo nutné nějakým způsobem omezit či úplně nahradit neurčitost stavů. Vzhledem k danému frameworku, ve kterém je agent implementován, se jedná především o předvídání oponentových karet a pak také simulace táhnutí vlastních karet. Když totiž v simulaci přejdeme do dalšího tahu agenta, ve fázi táhnutí karty je tato karta typu "No Way!", jež slouží jako štítek označující neznámou kartu. V případě oponenta jsou takto označeny všechny jeho karty

(ze zřejmých důvodů, a to aby nebylo možné podvádět a dívat se oponentovi do karet nebo předvídat svůj card draw).

Pro eliminaci neurčitosti vlastního card draw byl implementován algoritmus, který prochází historii odehraných karet a karet na ruce a jednoduše zjistí, které ještě zbývají v balíčku a nahradí tuto neurčitou kartu náhodnou kartou z balíčku.

Pro nahrazení oponentových karet byl využit přístup odhadování oponentova balíčku podle historie hraných karet. Agent má tedy seznam nejběžnějších decků a jejich karet. Po každém oponentově tahu si aktualizuje podle jím zahraných karet pravděpodobnost, s jakou by se mohlo jednat o jeden z uložených balíčků. Po překročení určité míry přesvědčení, že by se mohlo jednat o daný deck, začne agent nahrazovat oponentovy neznámé karty kartami z balíčku, které ještě oponent nehrál. Zkouší simulovat výsledek různého počtu náhodně vybraných karet a následně vybere ty, které jsou pro oponenta nejvýhodnější.

4 Experimentální ověřování

V této kapitole se budeme věnovat testování různých konfigurací agentů a analýze výsledků simulace mezi jednotlivými agenty a zhodnotíme jednotlivé přístupy vzhledem k prostředí daného frameworku, ve kterém jsou hry simulovány.

4.1 Prostředí simulace

K simulaci byl vybrán fork Hearthstone-AI Competetion frameworku SabberStone, ve kterém je naprogramováno jádro DCCG Hearthstone: Heroes of Warcraft a podporuje většinu herních karet. Tento fork je navíc připraven pro implementaci vlastních agentů, kteří se pak v turnaji utkají mezi sebou. Tímto bylo možné vybrat dobře otestovanou doménu simulace a zabývat se pouze implementací strategií agenta.

Framework umožňuje simulaci hry pouze mezi jednotlivými agenty, proto není možné vyzkoušet si zahrát proti libovolnému agentovi. Proto jsme také k experimentům vybrali schéma, ve kterém jsme zvolili různé druhy balíčků a různé strategie agentů a následně jsme je nechali odehrát určitý počet her mezi sebou, abychom měli dostatečný vzorek pro následnou analýzu výsledků.

4.2 Výběr herních decků

Vybrali jsme 3 druhy decků od každého archetypu:

- Aggro Pirate Warrior
- Midrange Buff Paladin
- Control Kazakus Mage

Popíšeme design struktury balíčků a strategii každého z nich. Jednotlivé balíčky jsou buď navrhnuté proti určitému archetypu nebo dokonce cíleny právě proti určitému decku, nebo je poskládán tak, aby byl schopen čelit široké škále decků a byl obecně stejně silný proti všem ostatním.

4.2.1 Aggro Pirate Warrior

Rychlý agresivní balíček *aggro* archetypu. Jeho strategie je nátlak na oponenta v rané fázi hry velkým počtem malých jednotek, které preferují útok přímo do oponenta. Využívá k tomu synergii pirátů a zbraní, které piráti vylepšují nebo díky kterým získávájí výhodu. Účelem zbraně je efektivní ničení oponentových jednotek, přes které nelze útočit přímo na protihráčovo zdraví. Jelikož se snaží ukončit hru co nejrychleji, využívá i své zdaví jako zdroj a snaží se tak udržet své jednotky na boardu za cenu ztráty svého zdraví.

4.2.2 Midrange Buff Paladin

Pomalejší balíček který využívá vylepšování vlastních jednotek a snaží se od začátku hry o board control. Udržuje agresivnější tempo a snaží se neustále vyvíjet nátlak na oponenta a vyhrát v *mid game* fázi. Strategie balíčku spočívá ve vylepšování ještě nezahraných jednotek na ruce a výhodné zneškodňování oponentových jednotek. Tuto strategii velmi dobře doplňuje klíčová vlastnost jednotek *divine shield* specifická pro herní třídu *paladina*. Toto klíčové slovo u jednotky znamená, že první příchozí poškození jednotky je ignorováno. Jednotky s touto klíčovou vlsatností jsou většinou dražší a mají sníženy *staty*, aby se vyvážila tato silná vlastnost. Díky hromadnému vylepšování jednotek však odpadá tato nevýhoda a jednotky se na boardu stávají velmi dominantními.

4.2.3 Control Kazakus Mage

Pomalý balíček *control* archetypu. Snaží se především přečkat počáteční fázi ničením oponentových jednotek a zpomalováním hry. Jakmile se dostane do pokročilejší fáze hry, začne vyvolávat silné jednotky a drahá silná kouzla. Tento balíček je ještě specifický v tom, že nehraje žádnou kartu dvakrát. Tímto se sice ochuzuje o určité silné karty, které je vždy lepší hrát vícekrát, ovšem obsahuje karty, které poskytují obrovskou výhodu v případě, že balíček neobsahuje duplicitní karty. Příkladem může být karta, která vyléčí hráče do plného zdraví, ale pouze pokud balíček neobsahuje duplikáty.

4.3 Průběh experimentů

V první fázi proběhlo testování MCTS agenta proti ostatním agentům. Konfigurace byla nastavena následovně:

- MCTS agent hrál proti každému agentovi
- S každým agentem se hrály všechny variace (s opakováním) decků
- pro každou variaci bylo odehráno alespoň 100 her

MCTS agentu byly nastaveny následující parametry:

- Hloubka prohledávání: 1 (pouze tahy v aktuálním kole).
- Časový rozpočet pro jeden tah: 2 sec.
- Metoda výběru nejlepšího potomka: UCT.
- Metoda hodnotící *score* funkce: Greedy.
- Exploration koeficient pro UCT: $1/\sqrt{2}$.

Do experimentálního ověření našeho MCTSAgenta jsme vybrali čtveřici agentů:

- Random
- Greedy
- BeamSearch
- DynamicLookahead

Každý z nich byl již detailně popsán v kapitole 3.

Postupně bylo odehráno nejméně 100 her proti každému agentu s každým archetypem balíčku. Hlavním cílem této fáze bylo ověřit, zda agent obstojí v konkurenci ostatních agentů. Vybraní agenti a jejich strategie jsou popsány v předchozí kapitole.

Ve druhé fázi jsme zkoušeli různé nastavení parametrů MCTS agenta. Porovnávali jsme například jaký vliv na winrate bude mít různě dlouhý čas pro každý tah. Zkoušeli jsme různé druhy výběru nejlepšího potomka, metody pro rozhodování remíz a různou hodnotu *exploration* konstanty.

V této části jsme nechali agenty hrát pouze *mirror match*, jelikož jsme chtěli mít stejné podmínky pro oba agenty, aby výsledky reflektovaly pouze změnu konfigurace agenta a nikoliv výběr balíčku (jelikož některé balíčky jsou silnější proti jiným atp.).

Jako hlavní testovací agent byl zvolen MCTSAgent se stejnou konfigurací jako v první fázi. Opět se odehrálo alespoň 100 her proti každému agentovi, ale pouze stejné decky proti sobě.

V poslední třetí fázi jsme zkoušeli nastavovat různé hodnoty pro *exploration* konstantu UCT. Jelikož původní hodnota $C_p = 1/\sqrt{2}$ je otestována pro *exploitation* hodnoty v intervalu $\langle 0, 1 \rangle$ a naše heuristická metoda dosahuje hodnot mimo tento interval, zkoušeli jsme různé nastavení této konstanty. Nastavení pro výchozího agenta jsme opět ponechali stejné jako v první fázi.

Agenti opět hráli minimálně 100 her a také jsme zkoumali pouze *mirror match*, stejně jako ve druhé fázi.

4.4 Vyhodnocení výsledků

Výsledky první fáze experimentů jsou zobrazeny v tabulkách 1 (a) (b) (c). Tabulka 1(a) zobrazuje procentuální úspešnost MCTS agenta hrající Aggro balíček proti všem ostatním agentům. Tabulka 1(b) zobrazuje procentuální úspešnost MCTS agenta hrající Midrange balíček a poslední tabulka v první fázi 1(c) obsahuje winrate MCTS agenta hrající Control balíček.

Winrate proti Random agentu je pokaždé podle předpokladu 100%. Zajímavější je však úspěšnost proti Greedy agentovi. Při hraní aggro balíčku dosahuje náš agent winrate přes 90 %. Při hraní control je tento winrate okolo 80 % a u midrange se výsledky různí podle oponentova balíčku. Dle našeho názoru je vysoká úspěšnost u aggro balíčku způsobena tím, že u tohoto archetypu je často velmi důležité pořadí provedených útoků a také maximální využití dostupné

Tabulka 1: První fáze testování proti ostatním agentům

(a) Flat MCTS hrající Aggro

Winrate (%)	Aggro	MidRange	Control
Random	100	100	100
Greedy	95	91	93
BeamSearch	62	69	55
DynamicLookahead	15	48	10

(b) Flat MCTS hrající MidRange

Winrate (%)	Aggro	MidRange	Control
Random	100	100	100
Greedy	51	74	89
BeamSearch	24	25	43
DynamicLookahead	3	13	30

(c) Flat MCTS hrající Control

Winrate (%)	Aggro	MidRange	Control
Random	100	100	100
Greedy	83	83	80
BeamSearch	60	73	46
DynamicLookahead	61	47	41

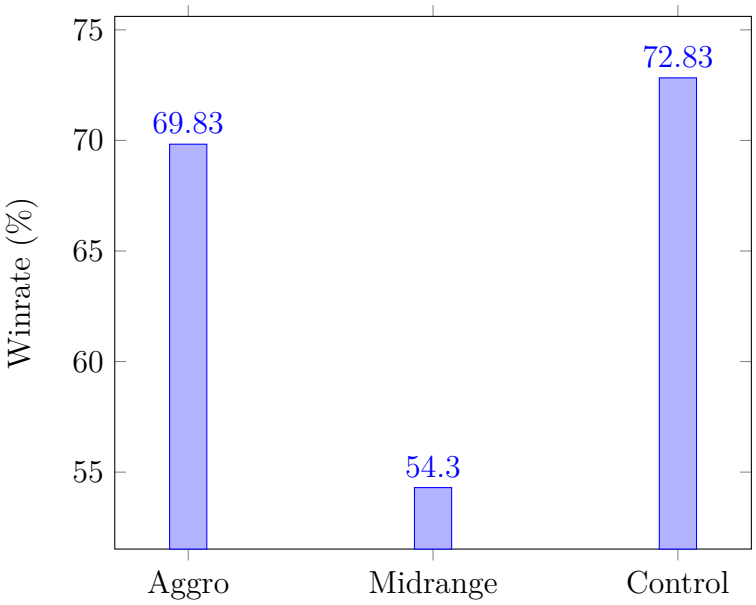
many. Zejména první kritérium MCTS agent skvěle ovládá, jelikož je schopen prozkoumat všechny kombinace útoků v různém pořadí a vybrat to nejlepší. Navíc obsahuje propracovanější ohodnocovací funkci, která dokáže detailněji hodnotit i klíčové vlastnosti jednotek, a upřednostnit tak útok do jednotky, která by mohla v následujících kolech zničit všechny ostatní jednotky, což je pro aggro nejhorší možný scénář.

Obecně nejlépe se MCTS agentovi dařilo s control balíčkem. Jeho průměrný winrate s tímto balíčkem je 72.83 %, jak lze vidět na grafu (obrázek 9). V posledním sloupci tabulky 1(a) jde vidět, že aggro je obecně slabší proti control balíčkům, protože dosahují zpravidla nejhorších výsledků. To stejné lze pozorovat v prvním sloupci tabulky 1(c), kde zase control dosahuje nejlepších výsledků proti aggru.

Obecně slabý byl náš agent při hraní midrange balíčku. Například proti DynamicLookahead agentovi byl v nejlepším případě schopen vyhrát pouze 30 % her. Naproti tomu při hraní control balíčku byl schopen vyhrát v 61 % proti aggru stejného agenta. U ostatních her s control balíčkem se stejným agentem však už byl wirate pod 50 %.

Ve druhé fázi byly zkoumány různé metody výběru nejlepších potomků. Jednotlivé metody byly detailně popsány v kapitole 2.5. Kromě těchto metod byl ještě do testu zařazen agent MCTS4000, který měl stejné nastavení, jen časový rozpočet měl nastaven na 4000 ms. Dále jsme přidali predMCTS, který měl hloubku prohledávání rozšířenou na oponentův následující tah.

Obrázek 9: Graf průměrného wr MCTS agenta



Tabulka 2: Druhá fáze testování různého výběru nejlepšího potomka MCTS

Winrate (%)	Aggro	MidRange	Control
vicMCTS	49	59	48
visMCTS	100	99	98
robMCTS	56	62	54
lcbMCTS	60	52	56
MCTS4000	49	45	46
predMCTS	47	50	53

V tabulce 2 jsou výsledky jednotlivých zápasů. Zarážející je především winrate u visMCTS, jelikož se příliš vzdaluje od výsledků ostatních metod. Může se jednat o chybu při implementaci této metody, nebo může naznačovat neoptimální volbu expandování listového uzlu, která může úzce souviset s touto metodou.

Kromě visMCTS výsledky korespondují s naším odhadem, že UCT metoda by měla být nejvýhodnější pro volbu nejlepšího potomka. Při navýšení časového rozpočtu v předposledním řádku je vidět mírné zlepšení, jelikož máme větší vzorek odehraných simulací pro statistické vyhodnocení odhadované hodnoty.

Také lze vidět, že predMCTS agent, který se snaží předpovídat oponentovy tahy, má celkově přesně 50 % proti flatMCTS, a tak nepřináší žádné zlepšení ani zhoršení. Tady jsme očekávali alespoň mírně vyšší winrate proti flatMCTS. Toto signalizuje, že naše metoda pro předpovídání oponentových karet není účinná a bylo by potřeba ji vylepšit.

V poslední fázi byly otestovány různé *exploration* konstanty. Vzhledem k poměrně vysokým hodnotám odměn, které dostáváme z heuristické ohodnocovací funkce, byl náš předpoklad takový, že vyšší hodnota této konstanty by mohla

Tabulka 3: Třetí fáze testování různých *exploration* konstant UCT

Winrate (%)	Aggro	MidRange	Control
$C_p = 1$	48	49	50
$C_p = 10$	48	50	50
$C_p = 100$	43	45	46

lépe vyvažovat nabádání k prohledávání méně slibných potomků. To se také potvrdilo a v posledním řádku tabulky 3 je vidět mírné zlepšení při použití hodnoty $C_p = 100$.

5 Uživatelská příručka

Framework SabberStone pro simulaci HS je naprogramován na jádře .NET Core ve verzi 2.1. Tato verze však nepodporuje tvorbu grafického uživatelského rozhraní. Proto jsme se rozhodli vytvořit jednoduché konzolové uživatelské rozhraní, které pomocí zadávání příkazů umožní uživateli nastavit parametry dvou agentů a spustí simulaci zadaného počtu her. Průběh her je ve formě počtu odehraných a zbývajících her, stejně jako aktuální skóre obou agentů, vypisován do konzole. Po skončení simulace je možno uložit si herní log pro případnou analýzu jednotlivých tahů agentů.

5.1 Hlavní nabídka

Po spuštění aplikace se uživatel nachází v hlavní nabídce a má na výběr ze čtyř příkazů:

- player1 - přejde do nastavení konfigurace prvního hráče
- player2 - přejde do nastavení konfigurace druhého hráče
- play n - postupně spustí n simulací her mezi player1 a player2
- exit - ukončí aplikaci

Výchozí nastavení hráčů je GreedyAgent s balíčkem AggroPirateWarrior.

5.2 Konfigurace hráče

Po zadání příkazu ke konfiguraci hráče se uživatel dostane do nabídky nastavení parametrů agenta. Výběr herních balíčků je následovný:

1. AggroPirateWarrior - Rychlý agresivní balíček
2. MidrangeBuffPaladin - Pomalejší balíček silný pro board control
3. MidrangeJadeShaman - Pomalejší balíček, který každou zahranou kartou vylepšuje určitý druh jednotek
4. MidrangeSecretHunter - Balíček, který vygeneruje velké množství malých jednotek a hromadně je vylepší
5. MiraclePirateRogue - Balíček, který hraje levné spelly a rychle prochází svůj balík karet
6. RenoKazakusDragonPriest - Control balíček, lečí své jednotky a sebe
7. RenoKazakusMage - Control balíček, hraje velké jednotky a silné kouzla

K dispozici jsou následující agenti:

- RandomAgent
- GreedyAgent
- BeamSearchAgent
- DynamicLookaheadAgent
- MCTSAgent

Popisu jednotlivých agentů jsme se věnovali v kapitole 3. Například příkaz pro zvolení BeamSearch agenta hrající balíček MidrangeBuffPaladin je ukázán na obrázku 10. Při zvolení MCTSAgenta se ještě navíc přejde do části pro detailní nastavení všech doplňujících parametrů.

```
Duels of Agents
Use command 'player1' to setup first agent.
Use command 'player2' to setup second agent.
Use command 'play (count)' to start (count) number of simulations.
Use command 'exit' to quit.
> player1
<---------- Player Setup ---------->
Choose player deck and agent
Available decks: AggroPirateWarrior, MidrangeBuffPaladin, MidrangeJadeShaman, MidrangeSecretHunter, MiraclePirateRogue,
RenoKazakusDragonPries, RenoKazakusMage
Available agents: RandomAgent, GreedyAgent, BeamSearchAgent, DynamicLookaheadAgent, MCTSAgent
Example: AggroPirateWarrior BeamSearchAgent
> MidrangeBuffPaladin BeamSearchAgent
```

Obrázek 10: Ukázka konzolové aplikace

Po navolení parametrů hráče je uživatel přesměrován do hlavní nabídky, ve které může spustit simulaci nakonfigurovaných hráčů.

5.3 Pokročilé nastavení parametrů MCTS

U MCTS agenta lze nastavit ještě jednotlivé parametry pro MCTS vyhledávání. Jmenovitě se jedná o tyto možnosti:

1. Turn Depth
2. Time Budget
3. Selection strategy
4. State rate strategy

Prvním nastavitelným parametrem je hloubka vyhledávání. Jelikož v HS většinou hráč může hrát více než jednu akci, je tato hloubka nazvaná *turn depth*, což znamená prohledávání akcí v daném tahu. Lze tedy nastavit, kolik tahů dopředu uvažovat ve vyhledávání. Při hloubce 1 tedy agent prohledává pouze možnosti v aktuálním kole. Při hodnotě větší nebo rovno dvěma se pro simulaci oponentových akcí použije algoritmus, který se snaží předpovídat akce na základě odehraných karet.

Dalším parametrem je časový rozpočet v milisekundách. Tento čas slouží pro výpočet jedné akce v tahu. Jedno kolo v HS trvá maximálně 75 sekund. Po vypršení časového limitu je kolo automaticky ukončeno. Agent nezná dopředu přesný počet akcí, jelikož ten se může v průběhu kola měnit podle akcí, které volí. Proto je potřeba zvolit časový rozpočet tak, aby i v pozdější fázi hry, kdy může mít k dispozici třeba 7 jednotek pro útok a 6 karet k zahrání, byl schopen odehrát všechny možné akce. Výchozí hodnota je 2000 ms. Jak jsme již ukázali v předchozí kapitole, při dvojnásobném časovém rozpočtu byl rozdíl vyhraných her minimální, nicméně rychlost výpočtu se mnohonásobně zpomalila.

Další je parametr pro strategii výběru potomka. Lze zvolit z následujících možností:

- MaxChild
- RobustChild
- MaxRobustChild
- MaxRatioChild
- SecureChild

Jedná se o metody popsané v kapitole 2.5. Jediná metoda navíc je MaxRatioChild, která vybírá potomka na základě nejvyšší hodnoty $Q(v)/N(v)$ (hodnota odměny uzlu vůči počtu navštívení).

Posledním nastavitelným parametrem je metoda ohodnocení stavu hry. Jedná se o tyto metody:

- Aggro - usiluje o rychlou výhru zahlcením oponenta levnými jednotkami
- Control - snaží se přečkat *early game* a hrát silné karty v pozdější fázi hry
- Ramp - snaží se co nejrychleji získat mana krystaly a hrát silné karty
- Ejnar - jednoduché a rychlé hodnocení pro libovolný archetyp
- Greedy - námi sestrojená metoda pro detailní hodnocení stavu
- Fatigue - diskartuje oponentův balíček a snaží se vyhrát vyčerpáním jeho zdrojů

6 Diskuze

Hlavním cílem diplomové práce bylo vytvořit počítačového hráče, který bude obstojně hrát proti ostatním počítačovým hráčům, případně proti skutečným hráčům. Volbou frameworku pro simulaci Hearthstonu jsme přišli o možnost testování agenta proti skutečnému hráči, nicméně jsme díky této volbě mohli pracovat v průběžně aktualizovaném a poměrně robusním prostředí pro simulaci hry.

V tomto frameworku byl vytvořen MCTS agent, který implementuje algoritmus UCT popsaný v kapitole 2.5. Při implementaci bylo potřeba upravit části algoritmu pro dané prostředí simulace a také naprogramovat několik optimalizací pro lepší výsledky.

Hlavní změna byla především nahrazení *default policy*. Místo odehrání hry až do konce a následné propagaci přesného výsledku hry byl zvolen heuristický algoritmus, který po určité hloubce ohodnotí aktuální stav a vrací tuto hodnotu. Tímto bylo možné opakovaně procházet menší množství stavů a získat tím větší vzorek pro následné statistické vyhodnocování výběru nejlepšího potomka.

Dále bylo v *default policy* také ještě upravena volba náhodné akce na výběr té nejlepší akce. Touto optimalizací bylo možné provádět menší počet iterací k dosažení rozumného odhadu odměny pro danou simulaci.

Při vyhodnocování výsledků testů mezi jednotlivými agenty výrazně prohrával pouze proti vítěznému agentovi z minulých let soutěže Hearthstone-AI Competition. Také se ukázalo, že jednoduché předvídání oponentových karet v balíčku nemá téměř žádný vliv na úspěšnost agenta. To ale neznamená, že se nevyplatí prohledávat herní strom do větší hloubky. Například u BeamSearch agenta ale i u DynamicLookahead agenta se oponentovy tahy zanedbávají, ale i tak je možné uvažovat následující tahy a plánovat budoucí útoky nebo plánovat, které karty je lepší upřednostnit a které si ponechat na pozdější fázi hry.

Zajímavým přístupem odhadování oponentových karet byla například analýza velkého množství odehraných her a vytvoření databáze bigramů karet, které byly většinou zahrány ve stejném tahu nebo v následujících kolech. Bylo ukázáno, že tento postup dokáže vylepšit úspěšnost agenta [6].

Dalším možným vylepšením by mohla být samotná metoda pro ohodnocování stavu simulace. Ta obsahuje omezené množství informací, jako je například aktuální počet jednotek na boardu, počty zdraví, klíčové vlastnosti jednotek atp., pomocí kterých se nakonec vyhodnocuje aktuální odměna pro daný stav simulace z pohledu agenta. Myslíme si, že tato metoda je vhodným kandidátem pro vylepšení pomocí genetického algoritmu, který by dokázal škálovat jednotlivé hodnoty pomocí opakování her proti sobě a upravování hodnot na základě jejich úspěšnosti. Ostatně například v DynamicLookahead agentovi byl podobný princip využit.

Závěr

Výzkum herní domény s neúplnou informací se po dominanci v hrách s úplnou informací stal v dnešní době další výzvou pro vývoj umělé inteligence. Hry jako Hearthstone nebo Poker se staly základním stavebním kamenem pro práci na nových přístupech pro tuto doménu.

Hlavním cílem diplomové práce bylo prozkoumat moderní algoritmy pro tuto doménu a vytvořit agenta, který bude úspěšným počítačovým hráčem v konkurenci ostatních agentů.

V první části jsme obecně popsali sběratelské karetní hry, jejich společné rysy a také dvě konkrétní nejrozšířenější hry tohoto typu. Druhá kapitola pojednávala o základních principech tvorby algoritmů pro hry s úplnou informací a také moderním algoritmu MCTS. Další kapitola představila framework pro prostředí simulace hearthstonu a popsala jednotlivé agenty a jejich strategie, včetně námi implementovaného MCTS agenta. Následovaly experimenty pro ověření úspěšnosti našeho agenta, včetně testování různých nastavení parametrů. Další kapitola se zabývala popisem konzolové aplikace pro testování libovolných agentů proti sobě. Poslední část diplomové práce tvoří diskuze, ve které jsme se věnovali limitům práce a také případným možným vylepšením.

Conclusions

Since the game domain with a complete information is well researched the next big challenge for artificial intelligence development lies in the domain of games with incomplete information. Games such as Hearthstone or Poker served as a sandbox for an early stage algorithm development in this domain.

The main aim of this thesis was to examine state-of-the-art algorithms in this domain and to design an agent that would be able to compete with other agents.

In the first section we introduced the basics of collectible card games and two most common games of this kind. The second section was about major design principles of algorithms for games with complete information and about state-of-the-art MCTS algorithm. Next section described selected framework for hearthstone simulation and agent strategies. Next we set up experiments whose aim was to compare our agent againts each other agent including different agent's parameters configuration. In the next section we described console aplication for different agents testing purpose. Last part of the thesis is the discussion where we highlighted limits of the thesis and possible enhancements.

A Obsah přiloženého CD/DVD

Na samotném konci textu práce je uveden stručný popis obsahu přiloženého CD/DVD, tj. jeho závazné adresářové struktury, důležitých souborů apod.

bin/

win10-x64/ - Obsahuje spustitelný soubor (.exe) aplikace pro operační systém Windows 10 s 64bitovou architekturou.

win10-x86/ - Obsahuje spustitelný soubor (.exe) aplikace pro operační systém Windows 10 s 32bitovou architekturou.

doc/

Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

src/

Kompletní zdrojové kódy programu SabberStoneAICompetition (v ZIP archivu).

readme.txt

Instrukce pro spuštění programu SabberStoneAICompetition, včetně všech požadavků pro jeho bezproblémový provoz.

U veškerých cizích převzatých materiálů obsažených na CD/DVD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro všechny použité (a citované) materiály, u kterých toto není splněno a nejsou tak obsaženy na CD/DVD, je uveden jejich zdroj (např. webová adresa) v bibliografii nebo textu práce nebo v souboru `readme.txt`.

Literatura

[1] KNUTH, Donald E.; MOORE, Ronald W. An analysis of alpha-beta pruning. *Artificial Intelligence*. 1975, roč. 6, č. 4, s. 293–326. ISSN 0004-3702.

[2] ALLIS, L.Victor; VAN DER MEULEN, Maarten; VAN DEN HERIK, H.Jaap. Proof-number search. *Artificial Intelligence*. 1994, roč. 66, č. 1, s. 91–124. Dostupný také z: ⟨http://www.sciencedirect.com/science/article/pii/0004370294900043⟩. ISSN 0004-3702.

[3] SANTOS, A.; SANTOS, P. A.; MELO, F. S. Monte Carlo tree search experiments in hearthstone. In. *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. 2017, s. 272–279.

[4] BROWNE, Cameron; POWLEY, Edward; WHITEHOUSE, Daniel aj. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*. 2012, roč. 4:1, s. 1–43. Dostupný také z: ⟨http://dx.doi.org/10.1109/TCIAIG.2012.2186810⟩.

[5] DOCKHORN, Alexander; MOSTAGHIM, Sanaz. Introducing the Hearthstone-AI Competition. 2019, s. 1–4. Dostupný také z: ⟨http://arxiv.org/abs/1906.04238⟩.

[6] DOCKHORN, Alexander; FRICK, Max; AKKAYA, Ünal; KRUSE, Rudolf, 2018. Predicting Opponent Moves for Improving Hearthstone AI, s. 621–632. Dostupný také z: ⟨http://dx.doi.org/10.1007/978-3-319-91476-3_51⟩. ISBN 978-3-319-91475-6.

[7] CHOE, J. S. B.; KIM, J. Enhancing Monte Carlo Tree Search for Playing Hearthstone. In. *2019 IEEE Conference on Games (CoG)*. 2019, s. 1–7.

[8] RUSSELL, Stuart; NORVIG, Peter. *Artificial Intelligence: A Modern Approach*. 3rd. USA: Prentice Hall Press, 2009. ISBN 0136042597.

[9] ZHANG, S.; BURO, M. Improving hearthstone AI by learning high-level rollout policies and bucketing chance node events. In. *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. 2017, s. 309–316.

[10] ŚWIECHOWSKI, M.; TAJMAJER, T.; JANUSZ, A. Improving Hearthstone AI by Combining MCTS and Supervised Learning Algorithms. In. *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. 2018, s. 1–8.

[11] NAKATSU, Ryohei; RAUTERBERG, Matthias; CIANCARINI, Paolo (ed.). *Handbook of Digital Games and Entertainment Technologies*. 2017. Dostupný také z: ⟨https://doi.org/10.1007/978-981-4560-50-4⟩.