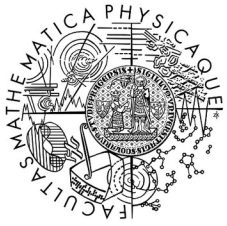


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Jiří Filek

Umělá inteligence pro Sushi Go!

Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Holan, Ph.D.

Studijní program: Informatika

Studijní obor: Obecná informatika

Praha 2020

Chtěl bych poděkovat RNDr. Tomáši Holanovi, Ph.D. za cenné rady a podněty při psaní práce.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V …..... dne..... podpis

Název práce: Umělá inteligence pro Sushi Go!

Autor: Jiří Filek

Katedra: Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Holan, Ph.D., Katedra softwaru a výuky informatiky

Abstrakt: Práce se věnuje vývoji umělé inteligence pro karetní hru Sushi Go!. Jedná se o hru dvou až pěti hráčů, ve které všichni hráči hrají najednou. Práce prezentuje různé přístupy pro tvorbu umělé inteligence. Největší důraz je kladen na metody MCTS, konkrétně na metody DUCT a EXP3. Také bude vyzkoušena hra na základě vážených pravidel, kde váhy přiřadí genetický algoritmus. V první části je obsažena analýza hry a popis vybraných metod umělé inteligence. Následuje určení parametrů agentů a také srovnání všech zkoumaných umělých inteligencí při hře dvou, ale i více hráčů. To je provedeno na základě mnoha sehraných her. Poslední část práce se zabývá implementací.

K účelu vývoje umělé inteligence vznikla grafická a konzolová aplikace. Konzolová aplikace slouží k nalezení parametrů a také ke srovnání umělých inteligencí. Grafická aplikace slouží pro hru člověka proti umělé inteligenci nebo jiným lidem na jednom počítači.

Pro všechny experimenty dopadla nejlépe metoda DUCT i přes teoretické nevýhody proti EXP3.

Klíčová slova: Sushi Go!, karetní hra, umělá inteligence

Title: Artificial intelligence for Sushi Go!

Author: Jiří Filek

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Tomáš Holan, Ph.D., Department of Software and Computer Science Education

Abstract: The thesis deals with an artificial intelligence for a Sushi Go! card game. It is a game with simultaneous moves for two to five players. The thesis presents multiple approaches for development of an artificial intelligence. The main focus is on methods based on the MCTS algorithm, namely DUCT and EXP3. An artificial intelligence using weighted rules is tried as well. The weights are assigned by a genetic algorithm. The first part of the thesis is about game analysis and description of chosen methods. The second part of the thesis is about parameter tuning and comparison of different agents for two or more players. The comparison is based on a large number of games played between agents. The last part of the thesis deals with an implementation.

A graphical and a console applications were created for the purpose of developing an artificial intelligence. The console application is used for parameter tuning and also to compare artificial intelligences. The graphical application is used for a game of human against artificial intelligence or other humans on a single computer.

Overall, DUCT performs best in every experiment despite its theoretical disadvantages.

Keywords: Sushi Go!, card game, artificial intelligence

Obsah

Úvod....1
1 Sushi Go....2
1.1 Pravidla hry....2
1.2 Bodování....2
1.3 Průběh hry....3
2 Sushi Go a umělá inteligence....3
2.1 Základní definice....3
2.2 Prostředí....4
2.3 Současné tahy....5
2.4 Větvící faktor....6
2.5 Agenti....7
3 Monte Carlo tree search....7
3.1 Selekce....8
3.2 Expanze....8
3.3 Simulace....9
3.4 Zpětná propagace....9
3.5 UCT....9
3.6 Výběr výsledného tahu....9
4 Rozšíření MCTS pro Sushi Go....10
4.1 Determinace....10
4.2 Současné tahy....11
4.2.1 DUCT....12
4.2.2 EXP3....14
4.3 Hra více hráčů....14
4.4 Bodový rozdíl....15
4.5 Generování akcí....16
4.5.1 Chopsticks....17
5 Evoluční algoritmy....18
5.1 Genetický algoritmus....18
5.2 Fitness....19
5.3 Selekce....19
5.4 Křížení a mutace....20
6 Výsledné parametry....20
6.1 DUCT....21
6.2 EXP3....26
6.3 Genetický agent....28
7 Výsledky....31
7.1 Hra dvou hráčů....31
7.2 Hra více hráčů se slabým soupeřem....33
7.3 Hra více hráčů....34
8 Uživatelská dokumentace....36
8.1 Obsah CD....36
8.2 Grafická aplikace....37
8.2.1 Výběr hráčů....37
8.2.2 Herní obrazovka....38
8.2.3 Obrazovky s výsledky....38

8.3 Konzolová aplikace....39
8.3.1 Genetický algoritmus....39
8.3.2 Hra mezi agenty....40
9 Implementační dokumentace....41
9.1 Qt....42
9.2 Správa Paměti....42
9.3 Třída Card....42
9.4 Třída Deck....42
9.5 Třída Player....43
9.6 Třída Game....43
9.7 Třída Genetic_algorithm....44
9.8 Třída MCTS....44
Závěr....47
Seznam použité literatury....48
Seznam tabulek....49
Seznam obrázků....50
Seznam použitých zkratek....51
Přílohy....52

Úvod

Práce se zabývá hrou Sushi Go a vývojem umělé inteligence pro tuto hru. Sushi Go je rychlá karetní hra pro dva až pět hráčů, ve které všichni hrají současně. Hry jsou dobrou příležitostí pro vývoj a výzkum umělé inteligence, protože mají jednoduchá a přesná pravidla, která umožňují jednoznačnou implementaci. Druhou klíčovou vlastností her je, že i přes ona jednoduchá pravidla bývají velmi složité na optimální zahrání.

Práce má hned několik cílů. Prvním cílem je implementace aplikace, která umožní hru člověka proti člověku a také proti umělé inteligenci. Aplikace bude fungovat pro dva až pět hráčů pomocí standardních pravidel hry. S tím souvisí i implementace rychlé aplikace sloužící pouze pro umělé inteligence, která bude sloužit k jejich vývoji.

Druhým a hlavním cílem je vývoj samotné umělé inteligence. V průběhu práce se zaměříme na několik způsobů. Důraz je kladen především na metody MCTS, které jsou v oblasti herních umělých inteligenci velmi úspěšné. Také chceme zkusit hru na základě ohodnocených pravidel, kde váhy přiřadíme genetickým algoritmem.

Celkově chceme najít umělou inteligenci, která hraje co nejlépe. Tudíž je pro nás klíčové srovnání jednotlivých přístupů ve vzájemné hře dvou i více hráčů.

Nyní stručně zmíníme strukturu práce. Nejprve popíšeme samotnou hru Sushi Go a její pravidla v kapitole 1. Následně rozebereme hru z pohledu umělé inteligence v kapitole 2. Zde nejdříve definujeme základní pojmy, které budeme v průběhu práce používat. Následně se zaměříme především na současné tahy, větvící faktor, ale i na samotné vlastnosti prostředí hry.

V kapitole 3 probereme základní verzi metody MCTS a v kapitole 4 ji budeme postupně rozšiřovat pro hru Sushi Go. Nakonec se zaměříme na hru pomocí vážených pravidel v kapitole 5. K nalezení příslušných vah budeme používat genetický algoritmus.

V kapitole 6 budeme určovat parametry pro naše umělé inteligence pomocí mnoha odehraných her. Následně necháme hrát jednotlivé umělé inteligence proti ostatním v kapitole 7. Zde zkusíme hru dvou, ale také i více hráčů.

Kapitola 8 obsahuje uživatelskou dokumentaci. Jedná se o popis, jak hru spustit a také jak ji ovládat. Poslední kapitola 9 se zabývá implementačními záležitostmi.

Umělou inteligencí se pro hru Sushi Go zatím nejspíše nikdo pořádně nezabýval, nepodařilo se nám najít žádné takové práce. Ovšem některé studie se zabývaly hrami s podobnými vlastnosti. Nejdůležitější vlastností Sushi Go jsou současné tahy. Asi nejvíce zkoumanou je hra *Goofspiel* [5]. Dále existují práce, které se zabývají hrou *Lord of the Rings: The Confrontation* [6], *Tron* [10], ale také hraním obecných her (General Game playing) [11]. Všechny tyto práce používaly různé metody MCTS. Většina se zabývá pouze hrou dvou hráčů.

1 Sushi Go

V této kapitole se zaměříme na samotnou hru Sushi Go. Stručně popíšeme její pravidla a bodování.

1.1 Pravidla hry

Sushi Go je rychlá karetní hra pro dva až pět hráčů. Hra obsahuje dvanáct typů karet, které podrobně rozebereme v části 1.2. Při hře dvou hráčů dostane každý hráč deset karet, při hře tří hráčů devět, čtyř hráčů osm a pěti hráčů sedm. Jedno kolo probíhá tak, že všichni hráči najednou vyloží jednu kartu z ruky a pak předají své karty v ruce hráči napravo. Tato kola se opakují, dokud mají hráči alespoň jednu kartu v ruce. Poté nastává fáze bodování, kde hráč dostane body za své vyložené karty.

1.2 Bodování

Postupně rozebereme všechny karty a body, které za ně hráč dostane. Hra má více různých pravidel, budeme se řídit pravidly, která jsou dostupná online na stránkách autorů [2]. Rozdíl je u bodování karet **Maki rolls** a **Pudding**, které používáme z pravidel oficiální karetní verze hry a podrobně je popíšeme později.

Hráč dostane pět bodů za každý pár karet **Tempura**. Za liché karty **Tempura** nedostane žádné body. Pokud má hráč pět karet **Tempura**, tak má dvě dvojice a jednu navíc, tedy získá deset bodů.

Popsaným způsobem funguje i karta **Sashimi**. U karty **Sashimi** získá hráč deset bodů za každou dokončenou trojici zahraných karet.

Za kartu **Dumpling** dostává hráč více bodů s jejich rostoucím počtem. Pokud má hráč jednu kartu **Dumpling**, dostane jeden bod, za dvě karty tři body, za tři karty šest bodů, za čtyři karty deset bodů, za pět a více patnáct bodů.

Karta **Nigiri 1** přidá hráči jeden bod, karta **Nigiri 2** dva body a karta **Nigiri 3** tři body.

Zahraná karta **Wasabi** říká, že následující jedna hráčem zahraná karta **Nigiri** musí být položena na zahranou kartu **Wasabi**. Za tuto kartu **Nigiri** dostane hráč trojnásobek bodů, za **Nigiri 1** tři body, za **Nigiri 2** šest bodů a za **Nigiri 3** devět bodů. Hráč může mít najednou více vyložených karet **Wasabi** a zahraním **Nigiri** je použita právě jedna karta **Wasabi**. Za nepoužitou kartu **Wasabi** nedostane hráč žádné body.

Zahraná karta **Chopsticks** umožní hráči během tahu zahrát kartu navíc a vyměnit ji za tuto kartu **Chopsticks**. Tedy hráč zahraje dvě karty a vrátí kartu **Chopsticks** ze zahraných karet zpět do ruky. Hráč může mít zahraných více karet **Chopsticks**, ale při tahu může použít vždy jen jednu. Za kartu **Chopsticks** nedostane hráč žádné body.

Karty **Maki roll 1**, **Maki roll 2** a **Maki roll 3** přidávají hráči příslušný počet bodů Maki, tedy 1, 2 a 3 v tomto pořadí. **Všichni** hráči s nejvyšším počtem bodů Maki dostanou šest bodů a **všichni** hráči s druhým nejvyšším počtem bodů Maki dostanou tři body. Podmínkou je získání alespoň jednoho bodu Maki Toto je změna proti [2], kde pokud má více hráčů stejný počet Maki bodů, pak se přiřazené body dělí počtem hráčů. Pro hru dvou hráčů to nemá vliv, ale při hře tří hráčů by mohl hráč s nižším počet bodů Maki dostat více bodů, než hráč s vyšším počtem bodů Maki.

1.3 Průběh hry

Hra má tři série a každá se skládá z rozdání karet, odehrání hry a přiřazení bodů. Po odehrání série se dají všechny karty stranou a nevrací se do balíčku. Jedinou výjimkou jsou karty **Pudding**, které si hráč nechá u sebe. Na konci série za ně hráč nedostane žádné body, ale body za karty **Pudding** se přiřadí až na konci celé hry. Všichni hráči s nejvyšším počtem zahraných karet **Pudding** dostanou šest bodů a všichni hráči s nejnižším počtem karet **Pudding** ztratí šest bodů. Při hře dvou hráčů žádný hráč body neztrácí. Toto je stejná změna, v pravidlech [2] se počet bodů opět dělí počtem hráčů s nejvyšším a nejnižším počet karet **Pudding**.

Hru vyhrává hráč s nejvyšším počtem bodů. V případě stejného počtu bodů vyhrává hráč s vyšším počtem karet **Pudding**. Počty jednotlivých karet v balíčku ukazuje Tabulka 1.

Karta	Počet
Tempura	14
Sashimi	14
Dumpling	14
Nigiri 1	5
Nigiri 2	10
Nigiri 3	5
Wasabi	6
Maki roll 1	6
Maki roll 2	12
Maki roll 3	8
Chopsticks	4
Pudding	10

Tabulka 1: Počty jednotlivých karet v balíčku

2 Sushi Go a umělá inteligence

V následující kapitole budeme rozebírat Sushi Go z pohledu umělé inteligence.

2.1 Základní definice

Nejprve definujeme základní pojmy, které budeme dále používat. Definice vychází z knihy *Artificial Intelligence: A Modern Approach* [1], kde je vše popsáno podrobněji. První definujeme pojmy stav, akce a agent, které mají velmi obecné definice.

Definice 1 stav: Stav je jednoznačné nastavení atributů, které jsou pro daný problém důležité.

Tyto atributy se pro různé problémy liší. V případě Sushi Go nám stav popisují karty, které mají jednotliví hráči v ruce a také karty, které již hráči zahráli.

Definice 2 akce: Akce je každá možnost, kterou můžeme v daném stavu vykonat.

V případě Sushi Go se jedná o zahrání karty, kterou máme v ruce.

Definice 3 agent: Agent je cokoliv, co něco dělá. To můžeme popsat, že agent dostane stav a vrátí jednu z akcí. Zpravidla chceme, aby agent maximalizoval očekávaný zisk.

Agenty budeme uvažovat všechny umělé inteligence pro Sushi Go a budeme se jimi více zabývat v části 2.5.

Nyní definujeme základní prostředí, která rozeznáváme.

Definice 4 deterministické prostředí: Prostředí je deterministické, pokud můžeme výsledný stav plně odvodit z aktuální stavu a z akce, kterou agent provedl.

Definice 5 stochastické prostředí: Prostředí je stochastické, pokud nemůžeme výsledný stav plně odvodit z aktuální stavu a z akce, kterou agent provedl. U stochastického prostředí uvažujeme pravděpodobností rozdělení nad množinou výsledných stavů.

Definice 6 nedeterministické prostředí: V nedeterministickém prostředí jsou akce charakterizovány množinou výsledných stavů. Na rozdíl od stochastického prostředí nemáme k výsledným akcím přiřazené pravděpodobnosti.

Pro hry rozlišujeme dva typy informací, úplnou a částečnou.

Definice 7 úplná informace: Pokud máme úplnou informaci, tak plně známe stav a akce, které mohou agenti provést.

Definice 8 částečná informace: V prostředí s částečnou informací nám zpravidla chybí část informace, abychom mohli jednoznačně popsat stav. Alternativně nemusíme znát všechny dostupné akce.

Dále rozlišujeme, pokud v prostředí máme jednoho, nebo více agentů. Prostředí s více agenty dále dělíme na kooperativní a konkurenční.

Definice 9 kooperativní prostředí: Prostředí je kooperativní, pokud agenti musí spolupracovat, aby dosáhli svého cíle.

Definice 10 konkurenční prostředí: V konkurenčním prostředí se agent snaží maximalizovat svůj očekávaný zisk, který většinou minimalizuje očekávaný zisk ostatních hráčů.

Rozlišujeme mnoho dalších typů prostředí, ale ty už pro nás nejsou důležité, takže je nebudeme uvádět.

2.2 Prostředí

Prvním krokem pro vytvoření umělé inteligence je daný problém správně zařadit. Tím se budeme zabývat v této kapitole.

Sushi Go je zřejmě hra s více agenty. Stejně jako ve většině her je v Sushi Go konkurenční prostředí. Každý hráč hraje proti ostatním hráčům, snaží se maximalizovat svůj bodový zisk a minimalizovat bodový zisk ostatních.

Obecně se dá uvažovat, že karetní hry mají stochastické prostředí. Ale mezi karetními hrami a stochastickými hrami, jako například Begamon, je zásadní rozdíl. Ve stochastických hrách máme většinou prvek náhody, který je u každého tahu. Takovým prvkem náhody může být například hod kostkou. Naopak v karetních hrách je jen jeden prvek náhody, kterým je zamíchání balíčku. To platí i ve hře Sushi Go. Lepší pohled na karetní hry je jako na deterministické hry s částečnou informací. V deterministickém prostředí lze plně určit následující stav z aktuálního stavu a z akce, kterou agent vykoná. To je velmi užitečné, protože deterministické prostředí se obvykle poměrně dobře prohledává. Hry s částečnou informací jsou pro umělé inteligence složité, protože neznají stav hry a dostupné akce soupeřů. Nemůžeme v nich tedy provádět prohledávání. Možným řešením je převést hru na instanci hry s úplnou informací, čímž se budeme zabývat později.

Ve hře Sushi Go mohou hrát i více než dva hráči. Pro tyto hry je téměř nemožné najít optimální řešení. V těchto hrách nelze předpokládat, že se každý hráč bude snažit maximalizovat svůj bodový zisk. Pokud je jeden hráč ve výrazně lepší pozici než ostatní, tak dává smysl, aby se protihráči spojili a vytvořili alianci. Taková aliance ztrácí smysl, když vedoucího hráče doženou a poté se zpravidla aliance rozpadne.

Jedná se o karetní hru, ve které je vliv náhody. Proto má většinou nějaký hráč výhodu a nějaký nevýhodu. Ovšem na rozdíl od většiny karetních her při hře dvou hráčů nemusíme brát v potaz, který hráč začíná. Hráči hrají současně, tedy hra je z tohoto pohledu symetrická. U více hráčů si musíme dávat pozor na jejich pořadí, protože hrát po špatném hráči (tento hráč nám předává karty) by mohlo být výhodné. [1]

2.3 Současné tahy

Nejdůležitější vlastností v Sushi Go z pohledu umělé inteligence je to, že všechny tahy jsou zahrané současně. Z důvodu současných tahů je prostředí spíše nedeterministické, protože akce má množinu možných výsledků (stavy hry po akcích ostatních hráčů) a chceme, aby byl agent úspěšný a měl dobré skóre se všemi možnými výsledky. Ovšem výsledný stav není náhodný, ale založený na akcích ostatních hráčů. Takže můžeme brát v potaz pouze podmnožinu výsledných stavů, které dostaneme za použití rozumných akcí od ostatních hráčů.

V prostředí se současnými tahy můžeme hrát dvě různé strategie. Pokud ve stavu s vždy zahrajeme fixní akci a_i, tak tuto strategii nazýváme čistou. Naopak, pokud máme ve stavu s pro své akce pravděpodobnostní rozdělení, tak se tato strategie nazývá smíšená.

Čistá strategie je lepší v situacích, kdy existuje dominantní akce. Dominantní akce dává nejlepší výsledek pro každou možnou akci soupeře. Uvažme hru se současnými tahy dvou hráčů, kde každý hráč má karty A a B a každý hráč musí zahrát právě jednu kartu. Hra končí po tahu obou hráčů. Výsledek hry můžeme reprezentovat pomocí matice. První číslo reprezentuje zisk hráče v řádku při dané akci a druhé číslo reprezentuje zisk hráče ve sloupci při dané akci.

	A	B
A	10 / 5	10 / 2
B	5 / 5	5 / 2

Tabulka 2: Hra s dominantní strategií

Můžeme nahlédnout, že oba hráči mají dominantní strategii a tou je zahrát kartu A, protože při zahrání karty B dopadnou při každé akci soupeře hůř. Pokud ovšem výsledek hry upravíme podle Tabulky 3, tak získáme hru, která nemá dominantní strategii pro žádného hráče.

	A	B
A	1 / 0	0 / 1
B	0 / 1	1 / 0

Tabulka 3: Hra bez dominantní strategie

V této hře chce hráč určený řádkem, aby oba zahráli stejnou kartu. Naopak hráč určený sloupcem se snaží, aby zahráli různé karty. Pro tuto hru není žádná čistá strategie optimální, protože soupeř se znalostí naší strategie může vždy vyhrát. V tomto případě je důležité hrát smíšenou strategii. Pro oba hráče je optimální hrát obě karty s pravděpodobností 0.5 a mají očekávaný zisk 0.5 bodů. V této situaci si žádný hráč nemůže pomoct změnou strategie. Navíc je to optimální strategie i pokud ji náš soupeř zná. Tomuto stavu se říká Nashova rovnováha. Každá hra se současnými tahy má Nashovu rovnováhu a dá se najít pomocí metod lineárního programování. To je ovšem možné dopočítat pouze u velmi jednoduchých her. Pokud máme více než dva hráče, tak se z toho dokonce stává problém nelineárního programování, který je ještě mnohem složitější. [1]

2.4 Větvící faktor

Větvící faktor je důležitou vlastností prostředí, pokud chceme dělat libovolné prohledávání. Když je větvící faktor příliš velký, tak je zpravidla těžké prohledat stavový prostor nebo i jen jeho netriviální část.

Hra Sushi Go má velmi malou hloubku stromu, pokud uvažujeme každou sérii nezávislou a budeme chtít jen optimálně zahrát jednu sérii. V případě dvou hráčů hra končí po deseti tazích, v případě pěti hráčů dokonce už po sedmi. Na druhou stranu současné tahy způsobují velký větvící faktor. Ten je definován jako maximální počet přímých potomků některého uzlu ve stromě. Často se uvádí průměrný větvící faktor (průměrný počet přímých potomků), protože může existovat jen málo uzlů, které mají obrovský počet možných akcí. Pak může být maximální větvící faktor zavádějící. Jako akci ve stromě musíme uvažovat kartézský součin akcí všech hráčů, jelikož akce jednoho hráče nedává ve hře se současnými tahy smysl, protože nevytvoří platný uzel ve stromě.

Nyní se podíváme na konkrétní hodnoty počtu potomků ve hře Sushi Go. V případě dvou hráčů je na začátku hry počet potomků $b = 10^2 = 100$.

V případě, že oba hráči zahrají v prvním kole kartu **Chopsticks**, pak bude větvící faktor v druhém kole $b = \left(9 + \binom{9}{2}\right)^2 = 45^2 = 2025$. A to je maximální větvící faktor pro hru dvou hráčů. V průběhu hry počet akcí, které mohou hráči udělat, klesá.

Pří hře pěti hráčů je počet potomků v prvním kole $b = 7^5 = 16807$. V případě, že čtyři hráči zahrají v prvním kole kartu **Chopsticks** (pět ji zahrát nemůže, protože v balíčku jsou jen čtyři karty **Chopsticks**), je větvící faktor dokonce $b = \left(6 + \binom{6}{2}\right)^4 \cdot 6 = 21^4 \cdot 6 = 1166886$.

A to je i nejvyšší větvící faktor, který může ve hře Sushi Go nastat s danými pravidly. Tato situace je ovšem velmi málo pravděpodobná. Přesto je i průměrný větvící faktor pro hru pěti hráčů obrovský. Navíc máme úplnou informaci o stavu hry až po zahrání čtyř karet. K tomu musíme ještě připočítat aliance a neoptimální tahy ostatních hráčů, které dělají ze hry pěti hráčů velmi složitý problém.

Na druhou stranu může mít hráč maximálně $\max_{i \in \{1,2\}, s \in S} |A_i(s)| = 9 + \binom{9}{2} = 45$ možných akcí, při hře pěti hráčů pouze $\max_{i \in \{1,2,\ldots 5\}, s \in S} |A_i(s)| = 6 + \binom{6}{2} = 21$ možných akcí. To nám dává možnost, abychom při prohledávání uvažovali u každého hráče pouze jeho akce a neuvažovali kartézský součin.

2.5 Agenti

Snažíme se vytvořit algoritmus, který bude hrát Sushi Go co nejlépe. Vyzkoušíme proto hned několik různých přístupů pro vytvoření agentů. Prvním je náhodný agent. Tento agent si v každém stavu vygeneruje seznam dostupných akcí a vybere z nich rovnoměrně náhodně. Toto je velmi jednoduchý agent, který by neměl ostatní agenty konsistentně porážet. Ovšem je pro nás užitečný, protože na něm můžeme testovat vlastnosti hry a bude to také výchozí bod pro srovnání s ostatními agenty.

Druhým agentem je hladový agent. Tento agent se snaží v každém stavu zahrát co nejlepší tah pomocí fixních pravidel, která ohodnocují dostupné akce. Hladový agent je plně deterministický a ve stavu s vždy zahraje stejnou akci a_i. Tento agent by měl být první poměrně dobře hrající agent a je důležitý pro srovnání se složitějšími metodami.

Následně vyzkoušíme složitější agenty. Nejprve se budeme zabývat metodami MCTS v kapitolách 3 a 4. Nakonec v kapitole 5 vyzkoušíme agenta na základě genetického algoritmu.

3 Monte Carlo tree search

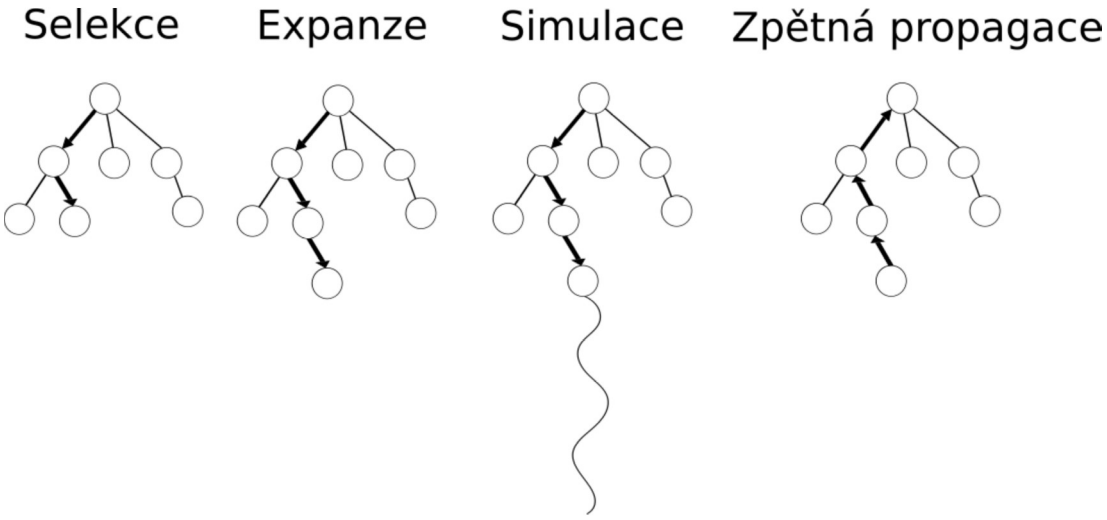
V této kapitole se budeme zabývat základní variantou algoritmu Monte Carlo tree search [4] a v následující kapitole 4 se zaměříme na možná rozšíření, aby se dal použít pro hru Sushi Go.

MCTS (Monte Carlo tree seach) je algoritmus, který popularizoval program AlphaGo od společnosti DeepMind, když za použití neuronových sítí a algoritmu MCTS porazil mistra světa ve hře Go Lee Sedola výsledkem 4-1. To byl obrovský úspěch vzhledem k tomu, že hra Go má velmi velký větvící faktor a není známá žádná efektivní heuristika, podle které by se dala ohodnotit herní pozice. Toto byly hlavní problémy, kvůli kterým standardní přístup použití minimaxu s $\alpha - \beta$ prořezáváním neměl ve hře Go úspěch.

MCTS se snaží ze seznamu dostupných akcí v daném stavu hry vybrat tu nejlepší. Nejlepším tahem můžeme uvažovat ten, po kterém máme nejvyšší pravděpodobnost vyhrát hru. Kdybychom tyto pravděpodobnosti znali, tak jednoduše vybereme tah s nejvyšší pravděpodobností na výhru. MCTS vychází z toho, že pokud bu-

deme mnohokrát simulovat náhodné hry, tak pravděpodobnosti na výhru z těchto simulací budou konvergovat ke skutečným pravděpodobnostem na výhru. Kritickým parametrem pro MCTS je pak zjevně počet simulovaných her a s tím související čas, který má algoritmus na daný tah.

Hlavní část v základní verzi algoritmu MCTS [4] je odsimulování jedné hry. Ta se skládá ze čtyř fází. Tyto fáze jsou Selekce, Expanze, Simulace a Zpětná propagace. Algoritmus tímto způsobem simuluje mnoho her a postupně si staví herní strom, který používá pro výběr tahů v následujících simulacích. Uzel v MCTS obsahuje stav hry, seznam dostupných akcí pro hráče, který je zrovna na tahu, počet navštívení uzlu a odměnu za výsledky simulací, které používaly daný uzel. Dále potřebuje ukazatel na rodiče pro zpětnou propagaci. Hrany mezi uzly odpovídají zahraným akcím.



Obrázek 1: Fáze v Monte Carlo tree search [10]

Základní verze MCTS funguje pro deterministickou hru dvou hráčů s úplnou informací, kde se hráči ve svých tazích střídají. Příkladem takových her jsou šachy, dáma nebo i Go. Další velmi zásadní vlastností MCTS je, že nemusíme mít žádnou znalost o problému, který chceme řešit. Stačí nám umět generovat možné akce a poznat koncové stavy hry, a kdo ji vyhrál. Různé heuristiky, které využívají vlastnosti hry, nám ovšem mohou pomoct s rychlejším nalezením nejlepší akce. [4]

3.1 Selekce

Selekce je první fází MCTS. Vybírá akci, kterou hráč v daném uzlu použije. Selekce je zodpovědná za průchod již postaveným herním stromem k uzlu, který ještě není plně expandovaný, tedy existuje v něm akce, kterou jsme ještě nevyzkoušeli. Poté nastává expanze. Selekce potřebuje způsob, jak vybrat akci, kterou hráč použije. Nejčastějším způsobem je UCT, ke kterému se v této kapitole ještě dostaneme. Tato fáze je důležitá, protože vždy vybírá větev, kterou budeme prohledávat. Z tohoto důvodu můžeme určité části herního stromu prohledat více, než jiné. Toto je jedna z hlavních myšlenek MCTS, protože výsledný strom bude asymetrický. Strom je menší u akcí, které se zdají být horší a je naopak větší u akcí, které se zdají být lepší.

3.2 Expanze

Expanze je zodpovědná za přidávání nových uzlů do stromu. Z důvodu nedostatku paměti se při jedné odsimulované hře přidá do stromu jen jeden uzel. Expanze nastává právě tehdy, když selekce narazí na akci, která se v daném stavu ještě nezkoušela a přidá uzel, který odpovídá výsledku této akce. Poté nastává simulace.

3.3 Simulace

Simulace dohraje hru až do konce, tedy do koncového stavu, kde můžeme jednoduše ohodnotit výsledek hry. Hráč, který vyhraje, dostane jeden bod a hráč, který prohraje, dostane nula bodů. V případě remízy dostanou oba hráči půl bodu. V základní verzi MCTS pro dva hráče stačí přiřadit body pouze jednomu hráči a dívat se na body druhého hráče jako doplněk do jedné.

$$hodnota\ \ druhého\ \ hráče = 1 - hodnota\ \ prvního\ \ hráče$$

Hra se zpravidla dohrává do konce pomocí náhodných tahů obou hráčů, ale dají se použít i heuristiky, aby hráči používali rozumné tahy. Náhodné tahy mají problém, pokud existuje jen jeden dobrý tah a hodně špatných, tak většinou vybereme špatný tah a pozice, která je vyhrávající, vypadá jako prohrávající. V těchto případech je potřeba velké množství simulací, aby je daná větev pořádně prozkoumala. Větev vygenerovaná simulací se následně zahazuje, abychom přidali jen jeden uzel.

3.4 Zpětná propagace

Po konci simulace nastává zpětná propagace, která aktualizuje informace o použitých uzlech v dané simulaci. Všem uzlům v prozkoumané větvi zvýší počet navštívení o jedna a zvýší odměnu v daném uzlu o výsledek simulace. Tento krok je potřeba pro selekci v následujících simulacích, která dostává informaci o kvalitě tahů.

3.5 UCT

UCT (UCB1 applied to trees) [4] je zobecněním metody UCB1 (Upper confidence bound 1) [4] tak, aby se dala používat pro stromy. Jedná se o nejčastější způsob, který se používá pro výběr uzlů v selekci. UCB1 řeší dilema explorace a exploatace, tedy jestli je lepší prozkoumávat tahy, o kterých máme málo informací a mohou být potencionálně optimální, nebo jestli prozkoumávat tahy, o kterých už víme, že jsou dobré, a chceme z nich vybrat ten nejlepší. Toto je klíčový krok v MCTS, protože pomocí toho staví algoritmus asymetrický strom a může mnohem podrobněji prozkoumat důležité tahy. UCB1 se řeší pomocí maximalizace následující funkce.

$$UCB1 = \bar{X}_i + c\sqrt{\frac{\ln(N_i)}{n_i}}$$, kde i označuje i-tou akci ve stavu s. Pak n_i je počet simulací přes uzel určený akcí i, N_i je počet simulací přes aktuální uzel, $\bar{X}_i$ je průměrná odměna v i-té akci, tedy $\frac{r_i}{n_i}$, kde r_i označuje sumu všech odměn v i-té akci, a c je konstanta, která upravuje poměr explorace a exploatace. První člen je velký pro dobré akce, tedy odpovídá exploataci. Druhý člen je velký pro málo navštívené akce, tedy odpovídá exploraci. Jejich poměr, který je určen konstantou c, se většinou určuje empiricky. Pokud existuje akce, která je zatím nenavštívená, tak interpretujeme její UCT hodnotu jako $+\infty$, tedy pak pro expanzi vybereme vždy jeden z nenavštívených uzlů.

3.6 Výběr výsledného tahu

Po skončení všech simulací z daného herního uzlu musíme vybrat výslednou akci. Zde můžeme vybrat potomka našeho uzlu s nejvyšším počtem navštívení

$\max_{a_i \in A(s)} n_i$. Alternativně můžeme vybrat uzel s nejvyšší celkovou odměnou tedy $\max_{a_i \in A(s)} r_i$. Obě hodnoty budou silně korelované. Pokud má uzel vysokou celkovou odměnu, tak bude často navštěvován a naopak bude často navštěvován, jen pokud bude mít velkou celkovou odměnu.

4 Rozšíření MCTS pro Sushi Go

Základní verze MCTS funguje pouze pro deterministické hry dvou hráčů s úplnou informací. Z tohoto důvodu je nutné algoritmus MCTS rozšířit, aby zvládal i hru Sushi Go. Je tedy potřeba vyřešit částečnou informaci, současné tahy a hru více než dvou hráčů.

4.1 Determinace

Determinace je způsob přímočarého převodu hry s částečnou informací na hru s úplnou informací. Pokud vezmeme v potaz všechna možná rozdání karet, tak máme *N* her s úplnou informací a můžeme každou zvlášť vyřešit. Jako výsledek vezmeme akci, která je v průměru nejlepší. To je zjevně i optimální tah. Tento tah ale nemusí být optimální vzhledem k aktuálním kartám soupeře. Tah, který by byl optimální vzhledem k aktuálním kartám soupeře, nejde určit bez toho, abychom se podívali na jeho karty. Proto najdeme tah, který je optimální v průměrném případě. To má ale jeden velký problém, protože vyřešit i jen jednu hru bývá dost náročné a počet všech možných rozdání karet je obrovský. Naštěstí se ukázalo, že i když vezmeme jen malou podmnožinu všech možných rozdání, tak tato metoda dává poměrně dobré výsledky. [1] Vybereme tedy akci podle následujícího vzorce.

$$\underset{a \in A}{argmax} \frac{1}{N} \sum_{1}^{N} MCTS(RESULT(s,a))$$, kde *s* je aktuální stav, *A* je množina akcí v aktuálním stavu, *RESULT(s, a)* je stav, který dostaneme použitím akce *a* ve stavu *s*. MCTS nám tento stav ohodnotí pomocí simulací.

Determinace je obecná metoda a můžeme místo MCTS použít například i MINIMAX. Na druhou stranu tato metoda předpokládá, že po prvním tahu mají oba hráči úplnou informaci o stavu hry. To je ve skutečnosti pravda pro Sushi Go při hře dvou hráčů. Obecně to může být problém. Předpokladem, že máme úplnou informaci, ignorujeme fakt, že hráč může hrát horší tahy, ale nechávat si taktickou výhodu tím, že nám bude dávat málo informací o své pozici a svých kartách. Tento problém řeší metody IS-MCTS (Information set Monte Carlo tree search) [6], které uvažují, že se nám soupeř snaží zatajit nějaké informace. IS-MCTS je ovšem výrazně složitější metoda než běžné MCTS. Z tohoto důvodu trvají simulace déle a nemůžeme jich provést tolik. Pro hru Sushi Go tento problém nemusíme řešit, protože velmi brzo, přesněji po *počet hráčů – 1* tazích, máme úplnou informaci o stavu hry. To víme, jelikož jsme už postupně viděli ruce všech hráčů a můžeme si pamatovat karty, které zahráli. Navíc se po každém kole ruka hráče přesune dalšímu hráči, takže nám hráči v Sushi Go nemohou úmyslně zatajit žádné informace.

Pro determinaci je důležité, aby byla konzistentní s herním stavem. Na začátku hry můžeme libovolně zamíchat balíček a rozdat ho ostatním hráčům. Před prvním tahem už známe své karty a víme, že je nemohou je mít protihráči. Z tohoto důvodu můžeme naše karty odebrat z balíčku a rozdat ostatním hráčům zbytek balíčku. Tato myšlenka je důležitá pro Sushi Go, kde se zahrané karty z předchozích sérií nevrací do balíčku. Pokud si budeme pamatovat všechny zahrané karty za celou hru, tak na

začátku třetí série jsme při hře dvou hráčů viděli 40 karet a dalších 10 karet vidíme v naší aktuální ruce. Pro hru pěti hráčů známe celkem 77 karet. Proto jsou pozdější determinace mnohem přesnější a nemusíme jich dělat tolik.

Výhodou determinace je, že se dá jednoduše paralelizovat. Každou vygenerovanou herní pozici můžeme řešit na jednom vláknu, takže rozumný počet determinací není až tak výpočetně náročný.

4.2 Současné tahy

U her se současnými tahy metoda UCT počítá s tím, že by jeden hráč hrál první a po něm by hrál druhý hráč. První hráč by byl příliš pesimistický a druhý hráč by byl příliš optimistický. Nejjednodušším příkladem hry se současnými tahy je asi hra kámen, nůžky, papír. Pokud budeme uvažovat fixní pořadí hráčů, tak první hráč ze svého pohledu vždy prohraje a druhý vždy vyhraje, zná-li tah prvního hráče.

Pro MCTS existují dvě hlavní metody, jak řešit současné tahy. První metodou je DUCT (Decoupled UCT), kterou se budeme zabývat v části 4.2.1. Druhou je EXP3 (Exponential-weight algorithm for Exploration and Exploitation), kterou rozebereme v část 4.2.2. Obecně se těmto algoritmům říká SM-MCTS (Simultaneous move Monte Carlo tree search). Jedna odsimulovaná hra se dá popsat pseudokódem z Algoritmu 1. Metody fungují velmi podobně, jako běžné MCTS. Důležitou funkcí jsou *SELECT* na řádku 12, která vybírá zvolenou k-tici akcí a odpovídá selekci. Další důležitou funkcí je *UPDATE* na řádcích 10 a 16, která je zodpovědná za zpětnou propagaci výsledků pro další simulace. V těchto funkcích se liší algoritmy DUCT a EXP3.

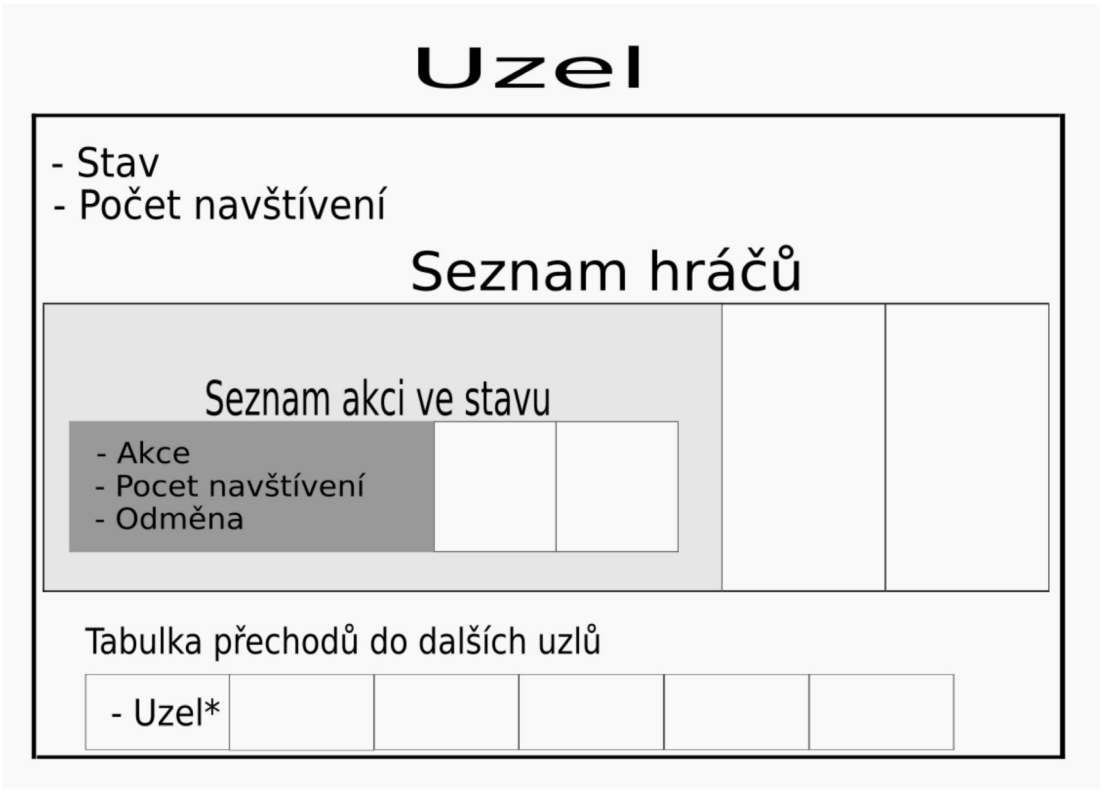
Alternativně se pro hry se současnými tahy dá použít SUCT (sekvenční UCT) [3], které funguje stejně, jako běžné MCTS. Jednoduše určíme pořadí, ve kterém budou hrát hráči současné tahy. Zde ovšem nastává problém. Pokud budeme hrát jako první, budeme hrát přehnaně obranně, protože vždy budeme předpokládat nejlepší možnou odpověď soupeře. Naopak, pokud budeme hrát až jako druhý, budeme hrát přehnaně agresivně, protože víme jaký tah soupeř zahrál. DUCT má běžně lepší výsledky, než SUCT, protože netrpí těmito problémy. [3] Z tohoto důvodu se SUCT nebudeme dále zabývat. Dalším algoritmem je Regret Matching [3]. Jedná se o stochastický algoritmus a je podobný EXP3. Jeho výsledkem je smíšená strategie. Regret Matching funguje asi nejlépe pro hry, kde je nutné hrát smíšenou strategii. Ovšem pro hry, kde to není potřeba, dopadá Regret Matching velmi špatně. Proto se jím nebudeme zabývat a ze stochastických variant vyzkoušíme pouze EXP3. [5]

1 $SM-MCTS(\text{node } s)$
2 **if** s is a terminal state $(s \in Z)$ **then**
3 **return** $u_1(s)$
4 **else if** $\exists(a_1, a_2, \ldots, a_k)$ unexplored **then**
5 $s_{new} \leftarrow T(s, a_1, a_2, \ldots, a_k)$
6 Add s_{new} to T
7 $u_1 \leftarrow PLAYOUT(s_{new})$
8 $X_{s_{new}} \leftarrow X_{s_{new}} + u_1$
9 $n_{s_{new}} \leftarrow n_{s_{new}} + 1$
10 $\underline{UPDATE}(s, a_1, a_2, \ldots, a_k)$
11 **return** u_1
12 $(a_1, a_2, \ldots, a_k) \leftarrow \underline{SELECT}(s)$
13 $s_{new} \leftarrow T(s, a_1, a_2, \ldots, a_k)$
15 $u_1 \leftarrow T(s, a_1, a_2, \ldots, a_k)$
16 $\underline{UPDATE}(s, a_1, a_2, \ldots, a_k)$
17 **return** u_1

Algoritmus 1: SM-MCTS

4.2.1 DUCT

V DUCT (Decoupled UCT) jsou všichni hráči a jejich akce v jednom uzlu. Uzel si pamatuje počet navštívení a každý hráč si pro každou svou akci pamatuje navíc počet navštívení a celkovou odměnu po výběru této akce. Schéma uzlu ukazuje Obrázek 2.



Obrázek 2: Uzel v DUCT

Důležitá je také funkce pro přechody do dalších uzlů. V běžném MCTS nám každá akce určuje jeden uzel. Zde nám uzel určuje k-tice vybraných akcí.

Když se vybírají akce ve fázi selekce MCTS (funkce *SELECT* v algoritmu), tak každý hráč vybere svou akci nezávisle na ostatních z vlastní hodnoty UCT. Zde existuje více způsobů, jak se zachovat k ještě nevyzkoušeným akcím. Prvním způsobem je, že pokud $\exists(a_1, a_2, ...a_n): a_1 \in A_1, a_2 \in A_2, ...a_n \in A_n$ taková, že tato akce ve stromě ještě nebyla použita, tak vybereme jednu z těchto akcí. Intuitivně to znamená, že pokud existuje nějaký potomek daného uzlu, kterého jsme ještě nenavštívili, tak vybereme jeden z nich. Druhým přístupem je vybírat akce úplně nezávisle na soupeřích. Tedy $\exists a_i \in A_j$ taková, že nebyla použita, pak hráč *j* musí použít jednu z těchto akcí. To znamená, že každý hráč musí v daném uzlu vyzkoušet všechny své akce, ale nemusí vyzkoušet kartézský součin akcí všech hráčů. Pro Sushi Go jsme vybrali druhou variantu, protože ta první je spíše určena pro hru dvou hráčů, kde není tak velký větvící faktor. Sushi Go může mít pro hru pěti hráčů obrovský větvící faktor (řádově 10^6) a za čas, který má algoritmus na hledání tahu by nejspíše téměř nikdy nepřešel do fáze exploatace, tedy prohledávání dobrých tahů.

Funkce *UPDATE* funguje stejně jako u běžného MCTS s UCT, tedy k odměnám za zvolenou akci přičte výsledek simulace a počet navštívení uzlů zvýší o jedna.

$$Pro\ hráče\ i: X^i_{s,a_i} \leftarrow X^i_{s,a_i} + u_i, kde(u_1, u_2, ..., u_k)\ je\ výsledek\ dané\ silumace$$

Po skončení simulace hráč vybere tah, který má nejvyšší odměnu $\max_{a\, is\, A_i(s)} X^i_{s,a}$.

DUCT obecně nekonverguje k Nashově rovnováze a existuje i elementární protipříklad. Jedná se o deterministický algoritmus a jeho výsledkem je čistá strategie. [3][3][5]

4.2.2 EXP3

EXP3 (Exponential weight algorithm for Exploration and Exploitation) [12] je asi nejčastěji používaný algoritmus pro řešení současných tahů. Stejně jako v DUCT se tahy všech hráčů vybírají v jednom uzlu a každý hráč je vybírá nezávisle na ostatních. Hlavním rozdílem je, že DUCT je deterministický algoritmus. DUCT vybírá akci, která maximalizuje hodnotu UCT. Naopak EXP3 je stochastický algoritmus, takže akcím přiřadíme váhy na základě předchozích zkušeností a pak z nich vybereme náhodnou. To má mnoho důležitých vlastností.

Výsledkem EXP3 je smíšená strategie, tedy v daném stavu hry zahrajeme s určitou pravděpodobností různé akce. Tato vlastnost je klíčová pro hry, kde má soupeř pro každou naši akci ideální odpověď v podobě více různých akcí. Pro hru dvou hráčů s nulovým součtem a úplnou informací dokonce EXP3 konverguje k Nashově rovnováze. [7] Tudíž z teoretického hlediska má EXP3 velkou výhodu nad DUCT, ale ve skutečnosti EXP3 zpravidla nemá tak dobré výsledky ve hrách, kde není potřeba hrát smíšenou strategii, ale existuje nejlepší tah.

Pravděpodobnost ve funkci *SELECT* pro hráče *i* a akci *j* je:

$$\sigma_j^i(s,a_j)=\frac{(1-\lambda)e^{\eta w^i_{s,a_j}}}{\sum_{a_k\in A_i(s)} e^{\eta w^i_{s,a_j}}}+\eta \quad \text{, kde} \quad \eta=\frac{\lambda}{|A_i(s)|}, w^i_{s,a_j}=x^i_{s,a_j}-\max_{a_k\in A_i(s)} x^i_{s,a_k}$$

Tady je λ pravděpodobnost pro exploraci, tedy výběr náhodného tahu. Ve funkci *UPDATE* odměnu vydělíme pravděpodobností, že bude vybrán daný tah a počet navštívení standardně zvýšíme o jedna stejně jako v předchozích případech. Formální zápis je následující $n^i_{s,a_j} \leftarrow n^i_{s,a_j}+1$, $x^i_{s,a_j} \leftarrow x^i_{s,a_j}+\frac{u_i}{\sigma^i_j(s.a_j)}$. To udělá x^i_{s,a_i} odhadem součtu odměn přes všechny iterace a ne jen přes ty, kde byla akce a_j vybraná, protože akce vybíráme na základě pravděpodobnosti. Tudíž u akce, která má malou pravděpodobnost výběru, bude velké bodové ohodnocení.

Závěrečnou smíšenou strategii po skončení simulací vybíráme na základě počtu navštívení. Opět ji samozřejmě vybíráme stochasticky. Nejprve ovšem znormalizujeme počty navštívení

$$n^{final}_{s,a_i} \leftarrow max\left(0, n_{s,a_i}-\frac{\lambda}{|A_i(s)|}\sum_{a_j\in A_i(s)} n_{s,a_j}\right) \quad .$$

Tímto způsobem nastavíme nulové pravděpodobnosti pro tahy, které byly navštíveny pouze v rámci explorace. Výsledný tah vybereme z rozdělení

$$\sigma^{final}_i(s,a_i)=\frac{n^{final}_{s,a_i}}{\sum_{a_j\in Ai(s)} n^{final}_{s,a_j}} \quad .\text{ [3][5]}$$

4.3 Hra více hráčů

MCTS se dá přímočaře rozšířit pro hru více než dvou hráčů. Do stromu se jednoduše kromě uzlů prvního a druhého hráče přidají vždy i uzly dalších hráčů. Podobně se dá rozšířit i SM-MCTS, kde se do uzlu přidají další hráči. Při výběru tahu u DUCT a EXP3 budeme vybírat tahy i pro tyto hráče. Dále je potřeba upravit přechodové funkce do dalších uzlů tak, aby respektovaly i jejich tahy.

Vetším problémem je ohodnocení hry, kterou dohraje simulace. MCTS předpokládá výsledky v intervalu $[0,1]$. Tady ovšem záleží na hráči. Pokud hráč bere první místo jako výhru a všechna ostatní jako prohru, tak můžeme použít následující funkci.

$$hodnota = \begin{cases} 1 \ldots výhra \\ 0 \ldots jinak \end{cases}$$

Pokud ale hráč bere v potaz pozici a chce se umístit co nejlépe, tak můžeme použít funkci tohoto typu.

$$hodnota = 1 - \frac{pozice\,hráče - 1}{počet\,hráčů}$$

4.4 Bodový rozdíl

Téměř vždy se MCTS snaží maximalizovat šanci na výhru. Algoritmus nebere v potaz, jestli vyhraje o jeden bod nebo o dvacet bodů. Tato vlastnost může být problematická. Pokud algoritmus vidí, že vždy vyhrává, tak dělá chyby. Stejný případ nastává, pokud si algoritmus myslí, že nemůže vyhrát. To je ale špatné, protože i soupeř může udělat chybu a pak je důležité mít co nejlepší pozici.

Jedná se o velký problém hlavně ve hře jako je Sushi Go, pokud chceme simulovat vždy jen jednu sérii. Ve hře Sushi Go je praktické simulovat vždy jen jednu sérii, protože velmi rychle získáme úplnou informaci o stavu hry v dané sérii. Úplnou informaci o celé hře máme až v poslední sérii. Kdybychom chtěli dohrát hru až do úplného konce, tak bychom potřebovali mnohem více determinací a v podstatě to ani nedává smysl, protože chceme dohrát každou sérii co možná nejlépe. Zde ovšem nastává problém, jak ohodnotit konec série. Problematická je hlavně karta **Pudding**, protože se za ni připočítávají body až na konci celé hry. Máme několik možností, jak tuto kartu řešit.

První možností je počítat body za karty **Pudding** pouze v poslední sérii. To má ale problém, že za karty **Pudding** se přiděluje poměrně velké množství bodů a agent by neměl důvod tyto karty hrát v první a druhé sérii.

Druhou možností je normálně přidělovat body za karty **Pudding** na konci každé série. Tady ovšem bude agent hrát až moc karet **Pudding** na úkor lepších akcí.

Poslední možností je kompromis mezi oběma způsoby. Tím je přidávat v první a druhé sérii za karty **Pudding** méně bodů. Zde můžeme například v první sérii přiřadit hráči s nejvyšším počtem karet **Pudding** čtyři body, v druhé sérii pět bodů a v poslední třetí sérii standardně šest bodů.

Při ohodnocení dané série nemůžeme použít klasické ohodnocení, kdy se snažíme maximalizovat procento vyhraných her. Sushi Go je karetní hra a často má jeden hráč výhodu proti ostatním. Zde je důležité tuto výhodu využít pro velký bodový zisk. Klasické ohodnocení se pouze snaží vyhrát a nezáleží na bodovém rozdílu. Pokud by měl hráč výhodu, tak ji pořádně nevyužije, a pravděpodobně vyhraje

sérii jen o jeden nebo dva body. To samé se stane, pokud má hráč nevýhodu, kde při optimální hře soupeře musí prohrát. Zde je možné, že agent bude dělat špatné tahy, protože stejně nemůže vyhrát, ať dělá cokoliv.

Musíme přijít s funkcí $f:\mathbb{R}\rightarrow[0,1]$, která bude brát jako parametr bodový rozdíl, ideálně monotónní rostoucí nebo monotónní neklesající. Jako bodový rozdíl bereme následující hodnotu, která předpokládá, že každý hráč se snaží mít co největší náskok na prvním místě, případně co nejmenší ztrátu na prvního.

$$br=\begin{cases}body\ prvního hráče-body druhého hráče \ldots první hráč\\ body\ aktuálního hráče-body prvního hráče \ldots\ldots jinak\end{cases}$$

Přímočarým řešením je funkce sigmoid $f(br)=\frac{1}{1+e^{-br}}$ nebo její zobecnění, $f(br)=\frac{1}{1+c^{-br}}:c\in\mathbb{R}^{+}$. Alternativně můžeme uvažovat funkci typu $f(x)=c_1+c_2\cdot g(br):c_1\in[0,1],c_2\in\mathbb{R}^{+}$, kde *g* je funkce, ideálně monotónní rostoucí.

4.5 Generování akcí

Chytrým generováním akcí můžeme výrazně snížit větvící faktor. V Sushi Go můžeme jako akci uvažovat zahrání karty na libovolném indexu. Ovšem na výsledek nemá vliv, jestli zahrajeme kartu **Tempura**, která je na první pozici nebo na třetí. Proto je lepší jako akci brát „zahrát kartu **Tempura**“ a neuvažovat její pozici.

Ve hře Sushi Go mohou existovat akce, které nemá cenu hrát nikdy. Pokud máme v ruce karty **Maki roll 3** a **Maki roll 2**, pak neexistuje stav, ve kterém by bylo lepší zahrát kartu **Maki roll 2**. Tedy **Maki roll 3** ji nutně dominuje a nemá cenu ani generovat akci, která bude hrát **Maki roll 2**. Z tohoto důvodu můžeme vždy najít pouze jednu kartu **Maki roll** s nejvyšší hodnotou. Podobná logika funguje i pro karty **Nigiri**. Zde je ovšem problematické, že **Nigiri 3** nemusí dominovat **Nigiri 2**. Mějme hru dvou hráčů, kde první hráč má karty **Nigiri 2**, **Nigiri 3**, **Nigiri 3** a druhý hráč má karty **Wasabi**, **Wasabi**, **Tempura**. V této situaci je optimální pro prvního hráče zahrát **Nigiri 2**, protože v příštím kole může zahrát kartu **Wasabi** a následně kartu **Nigiri 3** na kartu **Wasabi**.

Hráč 1	Hráč 2
Nigiri 3	**Wasabi**
Wasabi	**Nigiri 3**
Nigiri 2	**Tempura**
9	9

*Tabulka 4: Průběh hry po zahráním karty **Nigiri 3***

Tabulka 4 popisuje optimální strategie pro oba hráče po zahrání karty **Nigiri 3**. Poslední řádek udává počet získaných bodů.

Hráč 1	Hráč 2
Nigiri 2	**Wasabi**
Wasabi	**Nigiri 3**
Nigiri 3	**Tempura**
11	9

*Tabulka 5: Průběh hry po zahraní karty **Nigiri 2***

Zahráním karty **Nigiri 2** dopadne hra pro hráče lépe, což je způsobené kartou **Wasabi**. Ovšem tato situace nastává pouze tehdy, pokud máme dvě karty **Nigiri 3** a navíc náš soupeř musí mít dvě karty **Wasabi**. Vzhledem k tomu, že se jedná o velmi důležité karty, které se hrají brzy, tak nemá smysl spoléhat na to, že má soupeř dvě karty **Wasabi**. Pro takto složité prostředí stejně nejspíše nenajdeme optimální strategii. Tudíž podobně jako v případě **Maki roll** se vyplatí generovat jako akci jen karty **Nigiri** s nejvyšší hodnotou.

Vyřazení těchto akcí sníží větvící faktor. Navíc odebere i některé akce, o kterých víme, že jsou špatné. To je důležité pro náhodné simulace v MCTS. Špatné akce během simulací mají špatný vliv na výsledek a je potřeba více simulací na důkladné prozkoumání dané větve.

4.5.1 Chopsticks

Pokud máme zahranou kartu **Chopsticks**, tak je generování akcí mnohem složitější. Prvním pozorováním je, že při použití karty **Chopsticks** nemá smysl zahrát z ruky další kartu **Chopsticks**, protože se pouze vyruší. Tudíž můžeme karty **Chopsticks** v ruce ignorovat. U karet **Maki roll** a **Nigiri** musíme najít jen nejlepší dvojici, protože dominuje všechny ostatní dvojice těchto karet. Podobně u všech ostatních karet musíme najít dvojici, máme-li je alespoň dvakrát. Navíc musíme najít všechny dvojice různých karet. Zde budeme pro **Maki roll** a **Nigiri** opět uvažovat jen tu nejvyšší. Celkově můžeme na **Maki roll** a **Nigiri** nahlížet jako na ostatní karty. Jen to vyžaduje práci navíc.

Karty v ruce	**Akce**	**Akce s Chopsticks**
Maki roll 3	Maki roll 3	Maki roll 3 & Maki roll 2
Maki roll 2	Nigiri 3	Nigiri 3 & Nigiri 1
Maki roll 1	Tempura	Tempura & Tempura
Nigiri 3	Chopsticks	Maki roll 3 & Nigiri 3
Nigiri 1	Pudding	Maki roll 3 & Tempura
Tempura		Maki roll 3 & Pudding
Tempura		Nigiri 3 & Tempura
Chopsticks		Nigiri 3 & Pudding
Pudding		Tempura & Pudding

Tabulka 6: Vygenerované akce s chytrým generátorem

Tabulka 6 ukazuje akce vygenerované pomocí chytrého generátoru. Předpokládáme, že jsme v prvním kole zahráli kartu **Chopsticks**. Pomocí běžného generátoru

bychom měli $9+\binom{9}{2}=45$ akcí. Zde máme pouze $5+9=14$ akcí. To je obrovská redukce průměrného větvícího faktoru. Nejvyšší možný počet akcí dostaneme, pokud máme v ruce karty: **Maki roll**, **Maki roll**, **Nigiri**, **Tempura**, **Tempura**, **Sashimi**, **Dumplings**, **Pudding** a **Wasabi**. Zde nezáleží na tom, které karty máme dvakrát. Pak máme $7+2+\binom{7}{2}=30$ akcí. To je pořád dost, ale je to velké zlepšení. Navíc toto je velmi specifický případ. Průměrně budeme mít akcí mnohem méně.

5 Evoluční algoritmy

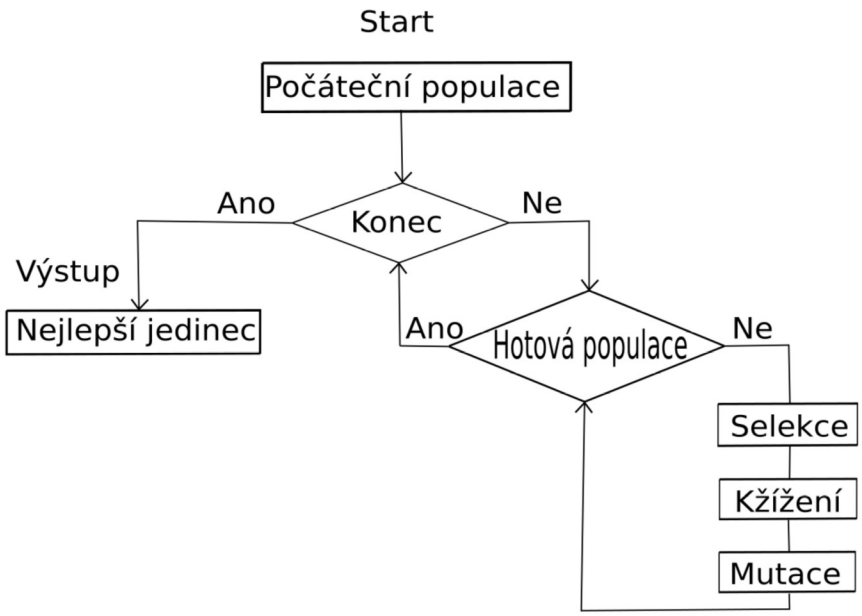
Alternativní možností pro umělou inteligence do her je použití jednoduchého pravidlového agenta. Tento agent má množinu pravidel, pomocí kterých vybírá tah. Takové pravidlo může vypadat například, *máš-li v ruce kartu **Pudding** a máš zahráno stejný počet karet **Pudding**, jako soupeř, tak zahraj kartu **Pudding***. Zde je velmi obtížné najít taková pravidla, aby agent hrál dobře.

Můžeme ovšem použít větší množinu pravidel a přiřadit jim váhy pomocí algoritmu pro hledání lokálních extrémů. To nám eliminuje pravidla, která mají špatný vliv na hru agenta. Při tahu agenta najdeme všechna splněná pravidla a vybereme výsledný tah z distribuce vah těchto pravidel. Výsledkem je pak smíšená strategie. Ta může fungovat velmi dobře především při hře více než dvou hráčů, kde je obrovský stavový prostor.

Příkladem algoritmu pro hledání lokálních extrémů je genetický algoritmus, který se snaží simulovat přirozený výběr přírody. Jinak můžeme použít například simulované žíhání, ale genetické algoritmy jsou v této oblasti používány nejčastěji.

5.1 Genetický algoritmus

V genetickém algoritmu [1] máme populace o N jedincích. Každý jedinec reprezentuje jeden stav. Jedinec je reprezentován pomocí řetězce. V našem případě jeden znak v řetězci odpovídá váze, kterou přiřadíme odpovídajícímu pravidlu. Tudíž délka řetězce je stejně dlouhá, jako je počet pravidel, která chceme ohodnotit. Náš genom se skládá M reálných čísel na intervalu $[0,100]$, kde M označuje počet pravidel. V genetickém algoritmu přecházíme z jedné populace do druhé tím, že nahradíme všechny jedince novými, kteří vytvoří novou populaci. Nového jedince vytvoříme vždy kombinací dvou jedinců z předchozí populace. Tomuto procesu se říká křížení. Některé verze genetických algoritmů křížení zobecňují a berou více než dva jedince. Zde je důležitá i Selekce, tedy způsob jakým budeme vybírat jedince pro křížení. Pro selekci potřebujeme Fitness, což je ohodnocení jedinců. Po křížení může nastat mutace, která může upravit některou hodnotu v řetězci jedince. Schéma algoritmu ukazuje Obrázek 3. Ukončovací podmínka bývá maximální počet generací, časový limit nebo existence jedince, který je již dostatečně dobrý.



Obrázek 3: Schéma genetického algoritmu

U genetických algoritmů je důležitě, že dobrá část řetězce jedince se může celá přenést na potomka. [1]

5.2 Fitness

U jedinců v genetickém algoritmu je potřeba určit fitness. Fitness nám popisuje kvalitu jedince a je důležitá pro selekci. V Sushi Go neumíme jednoduše určit kvalitu jedince. Proto ho necháme odehrát určitý počet her proti jiným agentům a jako jeho fitness vezmeme výsledek těchto simulací.

Zde je ovšem důležité, proti kterému agentovi ho necháme hrát. Můžeme použít hladového agenta. Tam nám ovšem hrozí, že se náš genetický algoritmus přeučí a nebude hrát tahy, které jsou dobré. Místo toho bude hrát tahy, které jsou dobré proti hladovému agentovi. Můžeme použít DUCT agenta. Ten je kvůli prohledávání velmi flexibilní, tudíž nám nehrozí přeučení. Ale u DUCT máme jiný problém, protože odehrát hru trvá mnohem déle a simulace budou výrazně pomalejší. Proto je asi nejlepší použít nejlepšího agenta z minulé generace genetického algoritmu. Zde se budou agenti každou generaci měnit, takže nám nehrozí přeučení. Navíc jsou simulace proti nim rychlé.

Zvolit genetického agenta z předchozí generace má jeden malý problém. Nebude nám růst fitness. To se nemusí zdá jako problém, ale častá ukončovací podmínka v genetickém algoritmu je, že nějaký jedinec má už dostatečnou fitness. To se nám nemůže stát, takže budeme muset simulovat fixní počet generací a zvolit tento počet dostatečně velký.

5.3 Selekce

V selekci vybíráme jedince ke křížení. Většinou chceme vybrat jedince s vysokou fitness, ale někdy budeme chtít i jedince s menší fitness. Tyto jedince vybíráme z důvodu zachování vysoké variability populace. V momentě, když jsou všichni jedinci stejní, tak už nemá cenu pokračovat.

Fitness v dané generaci počítáme pomocí mnoha odehraných her proti stejnému soupeři. V Sushi Go má velký vliv náhoda. Z těchto důvodů můžeme předpokládat, že určitě procento her vyhrají všichni jedinci a podobně žádný jedinec nevyhraje všechny hry. Výsledné fitness se tedy bude pohybovat na určitém intervalu, který bude výrazně menší, než $[0,100]$. Můžeme předpokládat, že výsledná fitness bude na intervalu $[30,70]$, spíše dokonce pouze na intervalu $[40,60]$. To nám přinejmenším nastane po několika málo generacích, protože se budou hodnoty fitness postupně stabilizovat.

Představme si, že máme dva jedince, kde první má hodnotu fitness 60 a druhý má hodnotu fitness 40. V selekci je můžeme vybrat rovnoměrně náhodně s tím, že budeme uvažovat hodnoty fitness, jako váhy. Pak pravděpodobnost výběru prvního jedince bude 0.6 a pravděpodobnost výběru druhého bude 0.4. Tímto způsobem běžně probíhá selekce. [1] Zde by ovšem měl být první jedinec mnohem lepší, než ten druhý, proto můžeme pro selekci zavést penalty. Tedy pokud od obou jedinců odečteme stejnou konstantu, tak se nám změní poměr jejich výběru. Můžeme od všech jedinců odečíst například $\frac{\textit{fitness nejlepšího jedince}}{2}$. V našem případě by byly nové hodnoty fitness pro selekci 0.3 a 0.1. Poté pravděpodobnost výběru prvního jedince bude 0.75 a pravděpodobnost výběru druhého bude pouze 0.25. Tento způsob použijeme v našich experimentech, protože odečíst část hodnoty fitness nejlepšího jedince funguje i pro hru více hráčů.

Obecně bychom mohli chtít hodnotu fitness znormovat například na interval $(0,1]$. Zde je ovšem problém, že se bude hodnota fitness jednotlivých jedinců v různých generacích lišit. Navíc není zřejmé, jak hodnotu fitness normovat na cílový interval. Naše řešení se snaží vyřešit podobný problém a zároveň nám zvětšit rozdíly v hodnotě fitness pro lepší a horší jedince. Může existovat jedinec, který bude mít hodnotu fitness pro selekci rovnu nule. Toto nechceme, aby se nám stávalo. Na druhou stranu, těchto jedinců bude jen velmi málo (pokud vůbec alespoň jeden) a navíc budou velmi špatní.

5.4 Křížení a mutace

Budeme používat běžné způsoby křížení a mutace popsané v [1]. Pro křížení vždy vybereme dva jedince a uděláme z nich dva nové. Vybereme náhodný index v genetické informaci jedince. Pak jeden z nových jedinců bude mít část od začátku po tento index od prvního jedince a zbytek genetické informace od druhého jedince. Pro druhého z nových jedinců necháme stejný index, jen vezmeme první část genomu od druhého jedince a druhou část od prvního.

Pro mutaci pak vezmeme náhodný index v novém genomu a přičteme k ní náhodnou hodnotu na malém intervalu. Tento interval může být v našem případě například $[-5,5]$.

6 Výsledné parametry

V této kapitole budeme ladit parametry pro naše agenty a v následující kapitole budeme zjišťovat, který agent je nejlepší. Všechny výsledky získáme z mnoha odehraných her v konzolové aplikaci 8.3, o které budeme mluvit později. Experimenty budeme provádět distribuovaným výpočtem na deseti počítačích na platformě Linux za použití překladače GCC 9.3.0. Všechny počítače mají stejné specifikace, konkrétně procesor *AMD Ryzen 5 PRO 2400G with Radeon Vega Graphics* o frekvenci 3.6 GHz. Výpočty budeme provádět na CPU.

Všechny experimenty jsou na základě mnoha sehraných her proti referenčním agentům. Sushi Go je karetní hra a má v ní vliv náhoda. Nejprve musíme zjistit, kolik her je potřeba odehrát, aby naše výsledky příliš neovlivnil vliv náhody. Pro tento experiment necháme hrát dva agenty, kteří hrají vždy náhodné tahy. Každý počet her jsme odsimulovali desetkrát.

Počet her	Střední hodnota	Rozptyl	CI – spodní	CI – horní	Velikost CI
100	51.25	24.12	47.73	54.77	7.04
200	50.40	4.84	48.83	51.97	3.15
500	49.55	7.69	47.57	51.53	3.97
1000	51.31	3.66	49.95	52.68	2.74
2000	50.95	4.51	49.52	52.47	3.04
5000	49.72	2.83	48.52	50.92	2.41
10000	49.49	0.39	49.05	49.95	0.90
20000	50.06	0.27	49.69	50.43	0.74

Tabulka 7: Výsledky hry náhodných agentů, hodnoty normalizovány na 100 her

Získaná hodnota odpovídá průměrnému počtu získaných bodů ze 100 her. Hráč získá 1 bod za výhru, 0.5 bodu za remízu a 0 bodů za prohru. Všechny hodnoty jsou z pohledu prvního hráče. To si můžeme dovolit, protože hra dvou hráčů je symetrická. Očekávaná střední hodnota je 50. Rozptyl chceme minimalizovat. Poslední tři sloupce udávají 95% interval spolehlivosti. Chceme, aby interval spolehlivosti obsahoval 50 a byl co možná nejmenší. Zjevně máme nejlepší výsledky při 20000 odehraných hrách. Ovšem i 10000 odehraných her má velmi malý rozptyl, což je hlavní parametr, který chceme minimalizovat počtem odehraných her. Zde interval spolehlivosti neobsahuje 50, ale je jí velmi blízko, 99% interval spolehlivosti už 50 obsahuje. Ale už 5000 her nám dává dobré výsledky. Proto pro vybírání parametrů agentů budeme používat 5000 her a pro závěrečné porovnání agentů 10000 her.

Vhodné parametry potřebujeme najít pro DUCT, EXP3 a genetické agenty. Všechny parametry budeme hledat pomocí hraní agenta proti sobě s jiným nastavením parametrů. To uděláme, aby neměl některý agent výhodu proti jinému. Vždy budeme měnit pouze jeden parametr a necháme všechny ostatní fixní.

6.1 DUCT

DUCT má následující parametry, pro které budeme hledat optimální hodnoty. Prvním je jako u všech metod MCTS počet simulací. Zde je vyšší počet vždy lepší, ale nechceme, aby agent hledal tah příliš dlouho. Místo počtu simulací můžeme uvažovat čas na tah. Jelikož děláme experimenty na více různých počítačích, tak zvolíme počet simulací. Tam máme jistotu, že se program bude chovat vždy stejně. Druhým parametrem je konstanta c, která určuje poměr explorace a exploatace při počítání hodnoty UCT. Třetím parametrem je počet determinací. Podobně, jako u počtu simulací je větší počet vždy lepší, ale prodlužuje dobu hledání tahu. Zbývajícími parametry jsou funkce, která hodnotí stav na základě bodového rozdílu, hodnotí karty **Pudding** a také druh simulace.

Parametr	Výchozí hodnota	Obor hodnot
Počet simulací	5000	$\mathbb{N}$
c (pro UCT)	0.7	$\mathbb{R}^+$
Počet determinací	1	$\mathbb{N}$
Ohodnocení hry	Sigmoid (se základem e)	Sigmoid (se základem $\mathbb{R}^+$)
Body za Pudding	6/6/6	{ 6/6/6, 4/5/6, 4.5/5.5/5.5 }
Druh simulace	Náhodná	{ Náhodná, Vážená }

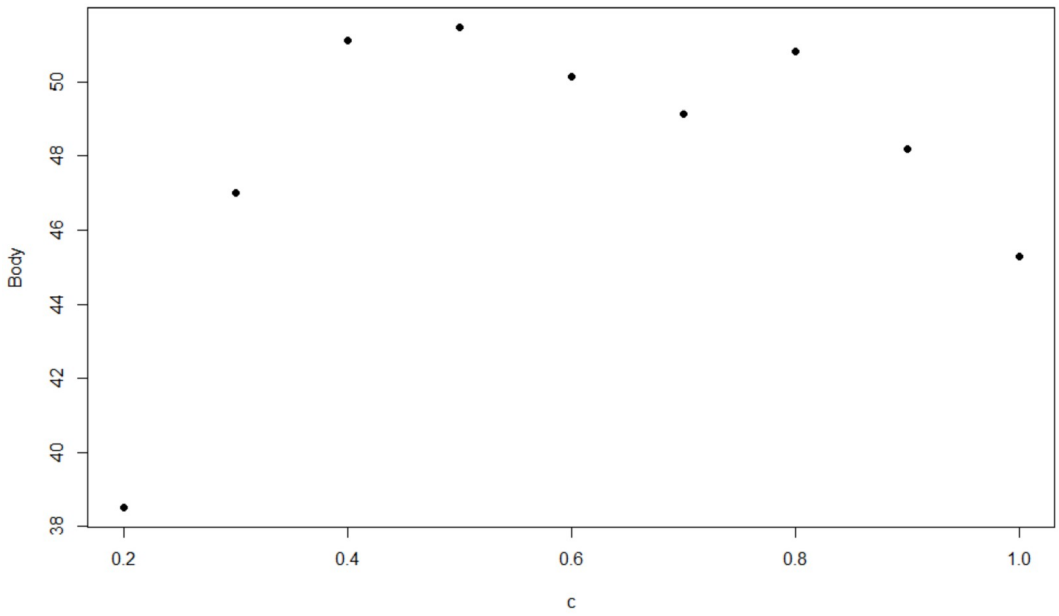
Tabulka 8: Výchozí parametry pro DUCT

Tabulka 8 ukazuje výchozí parametry pro DUCT a jejich příslušné obory hodnot. Tyto parametry budeme používat pro následující experimenty.

Nejprve určíme konstantu *c*. Všechny agenty necháme hrát proti agentovi s c = 0.7. Odehrajeme vždy 5000 her a budeme uvádět průměrné výsledky na 100 her, tedy procentuální úspěšnost. Výsledky vždy uvádíme z pohledu agenta, jehož parametry měníme.

c	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Výhry	33.06	41.12	45.60	45.96	44.52	44.02	45.42	42.62	39.82
Prohry	56.04	47.12	43.36	43.00	44.26	45.76	43.80	46.24	49.22
Remízy	10.90	11.76	11.04	11.04	11.22	10.22	10.78	11.14	10.96
Body	38.51	47.00	51.12	51.48	50.13	49.13	50.81	48.19	45.30

Tabulka 9: Výsledky pro c v DUCT



Obrázek 4: Vliv hodnoty c v DUCT

Z Tabulky 9 a Obrázku 4 vidíme, že nejlepší výsledky proti soupeři s $c = 0.7$ má $c = 0.5$ a těsně za ním je $c = 0.4$. Překvapivě dobře dopadlo i $c = 0.8$, kde je

úspěch nejspíše způsobem tím, že ve hře je velký vliv náhody. Obzvlášť když děláme jen jednu determinaci a ta je špatná. Pro další experimenty budeme používat hodnotu *c* rovnu 0.5.

Nyní se podíváme, jaký má vliv počet simulací na hru DUCT. Všechny parametry použijeme z Tabulky 8. Jen $c = 0.5$. Hrajeme vždy proti DUCT s 5000 simulacemi.

Počet her	**500**	**1000**	**2500**	**5000**	**7500**	**10000**	**15000**	**20000**
Výhry	25.70	32.52	40.34	44.12	47.24	46.58	50.10	50.68
Prohry	65.58	57.60	48.48	43.60	40.82	40.86	37.70	37.70
Remízy	8.72	9.88	11.18	12.28	11.94	12.36	12.20	11.62
Body	30.06	37.46	45.93	50.26	53.21	52.76	56.20	56.49

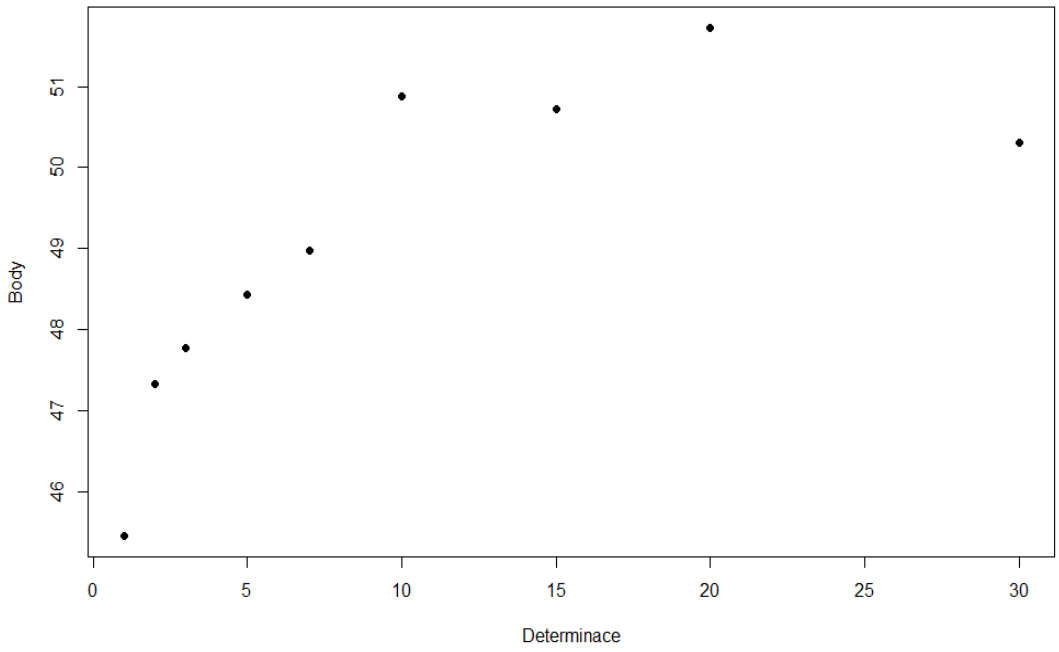
Tabulka 10: Počet simulací v DUCT

Vidíme, že bodový zisk postupně roste. Hlavně u malého počtu simulací má jejich navýšení velký vliv na výsledky agenta. Ale i při větším počtu her máme výrazné zlepšení. Při 15000 simulací máme bodový zisk 56.2. To je očekávané chování pro MCTS a potvrzuje, že počet simulací je opravdu klíčový parametr. Jediný pokles je mezi 7500 a 10000 her. Ten není moc velký a podobně jako u parametru *c* je nejspíše způsoben vlivem náhody. Počet her je těžký parametr na správné nastavení, protože chceme, aby byl co možná nejvyšší a zároveň tah agenta netrval příliš dlouho. Výslednou hodnotou pro počet simulací je 5000, což budeme používat pro další experimenty. Pro srovnání s jinými přístupy vyzkoušíme různé počty her. Také potřebujeme zjistit, jaký vliv má počet determinací na hru agenta. Ta by se měla s rostoucím počtem determinací zlepšovat, podobně jako u počtu simulací. Počet determinací chceme ale co nejmenší, protože nám rapidně prodlužuje čas, který je potřebný na tah. Ten můžeme odhadnout jako $počet\,simulací \cdot počet\,determinací$, kde počet determinací bude malý. Všechny hry jsou hrány proti agentovi s 10 determinacemi.

Determinace	**1**	**2**	**3**	**5**	**7**	**10**	**15**	**20**	**30**
Výhry	39.74	40.72	41.8	41.8	42.56	44.58	44.12	45.56	44.02
Prohry	48.84	46.06	46.26	44.94	44.62	42.82	42.66	42.10	43.40
Remízy	11.42	13.22	11.94	13.26	12.82	12.6	13.22	12.34	12.58
Body	45.45	47.33	47.77	48.43	48.97	50.88	50.73	51.73	50.31

Tabulka 11: Počet determinací v DUCT

V Tabulce 11 a v Obrázku 5 vidíme, že získané body opravdu rostou a to hlavně při malém počtu determinací. Od 10 determinací nemá příliš velký vliv přidávání dalších. Determinace výrazně prodlužují čas, který trvá najít výsledný tah a na hru agenta nemají takový vliv jako počet simulací. Proto zvolíme jen 3 nebo 5 determinací a raději budeme simulovat více her. Proti lidskému protihráči chceme minimalizovat maximální čas, který nám trvá najít tah. Proto by dávalo smysl po skončení determinací (stav kdy už máme úplnou informaci o hře) simulovat více her.



Obrázek 5: Získané body pro hodnoty počty determinací v DUCT

Dále potřebujeme najít vhodnou funkci, pomocí které budeme ohodnocovat stav po dohrání hry. Výchozí je funkce sigmoid.

$$br=\begin{cases} body\ prvního\ hráče - body\ druhého\ hráče \ldots první\ hráč \\ body\ prvního\ hráče - body\ aktuálního\ hráče \ldots\ldots jinak \end{cases}$$ a

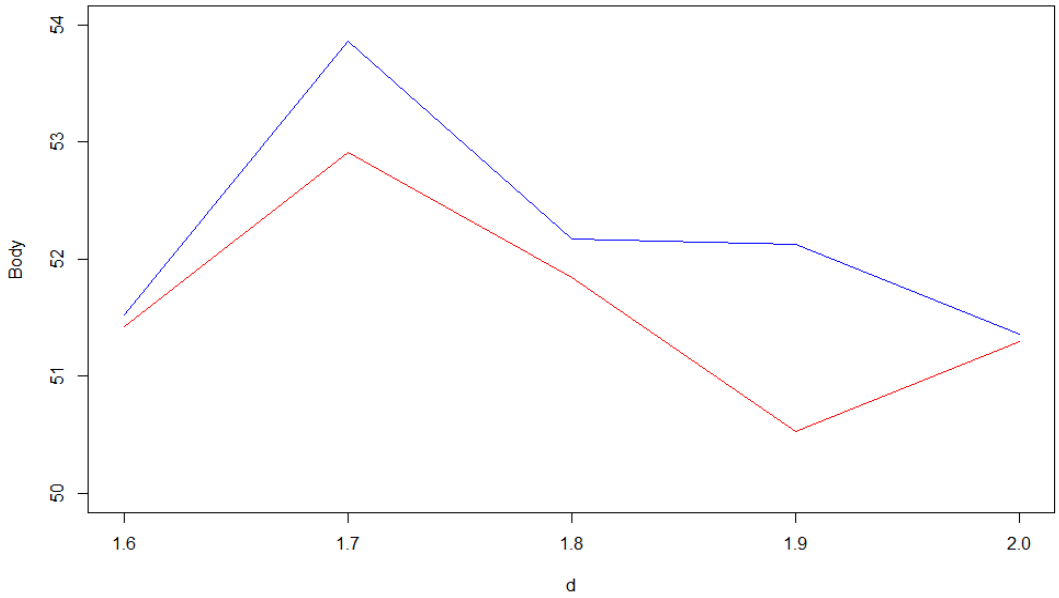
$f(br)=\frac{1}{1+d^{-br}} : d \in \mathbb{R}^{+}$, kde výchozí hodnota $d=e$. Nyní můžeme vyzkoušet různé hodnoty pro *d*. Zde možná budeme muset změnit konstantu *c*, která určuje poměr mezi explorací a exploatací. Pokud vedeme o 3 body, tak $\frac{1}{1+e^{-3}}=0.95$ a pokud $d=2$ pak $\frac{1}{1+2^{-3}}=0.89$. Z tohoto důvodu možná budeme muset snížit společně se základem *d* i konstantu *c*. Příliš velké *d* způsobuje, že od určité hodnoty bodového rozdílu nemá vyšší bodový rozdíl téměř žádný vliv. Naopak pokud je *d* příliš malé, tak je funkce málo citlivá pro hodnoty blízko jedné, které jsou nejběžnější. Jako soupeře budeme opět používat agenta podle Tabulky 8, jen *c* = 0.5 a budeme používat 3 determinace.

d	**2.0**	**1.9**	**1.8**	**1.7**	**1.6**
Výhry	45.78	44.54	46.38	47.40	46.00
Prohry	43.18	43.48	42.70	41.58	43.16
Remízy	11.04	11.98	10.92	11.02	10.84
Body	51.30	50.53	51.84	52.91	51.42

Tabulka 12: Základ funkce sigmoid v DUCT, c = 0.5

d	2.0	1.9	1.8	1.7	1.6
Výhry	45.50	46.34	46.52	47.98	46.28
Prohry	42.78	42.08	42.18	40.26	43.24
Remízy	11.72	11.58	11.3	11.76	10.48
Body	51.36	52.13	52.17	53.86	51.52

Tabulka 13: Základ funkce sigmoid v DUCT, c = 0.4



Obrázek 6: Bodový zisk v závislosti na základu funkce sigmoid pro ohodnocení, modrá odpovídá c = 0.4, červená je c = 0.5

Vidíme, že všechny hodnoty bodového zisku jsou větší než 50. To znamená, že snížení základu funkce sigmoid výrazně pomohlo. Nejvyšší bodový zisk je 53.86, což je velké zlepšení. Také vidíme, že c rovno 0.4 je pro všechny hodnoty lepší, než $c = 0.5$. To potvrzuje náš předpoklad o chování MCTS s funkcí sigmoid. Nejlepší hodnoty jsou $d = 1.7$ a $c = 0.4$, které budeme používat pro další experimenty. Tento základ je citlivý pokud je $|\textit{bodový rozdíl}| \leqslant 5$. Jestliže je bodový rozdíl větší, tak už se tento rozdíl v naší funkci téměř neprojeví.

Nakonec vyzkoušíme jiné bodování karty **Pudding**. Navíc ještě zkusíme při fázi simulace u MCTS přiřadit akcím různé váhy. Pomocí nich bychom mohli najít nejlepší tah rychleji. Pro všechny váhy platí, že $v \in \{1,2,3,4\}$. Váhy jsme přiřadili podle intuice o důležitosti karet po zahraní přibližně 50 her proti lidem.

	Pudding 4/5/6	**Pudding** 4.5/5.5/6.5	Vážené simulace
Výhry	49.80	49.88	43.74
Prohry	38.86	48.08	44.64
Remízy	11.34	2.04	11.62
Body	55.47	50.90	49.55

Tabulka 14: Zbývající parametry v DUCT

V Tabulce 14 vidíme, že změna hodnocení karty **Pudding** má obrovský vliv na hru agenta. Agent s hodnotami 6/6/6 dával na začátku hry zbytečně vysokou prioritu kartě **Pudding**. Hodnocení **Pudding** 4.5/5.5/6.5 je testováno proti hodnotám karty **Pudding** 4/5/6. Vidíme, že hodnocení 4.5/5.5/6.5 dopadlo ještě lépe a má velký vliv na snížení počtu remíz. Proto jej budeme používat jako výslednou hodnotu. Naopak vážené simulace nemají v podstatě žádný vliv a budeme dále používat náhodné.

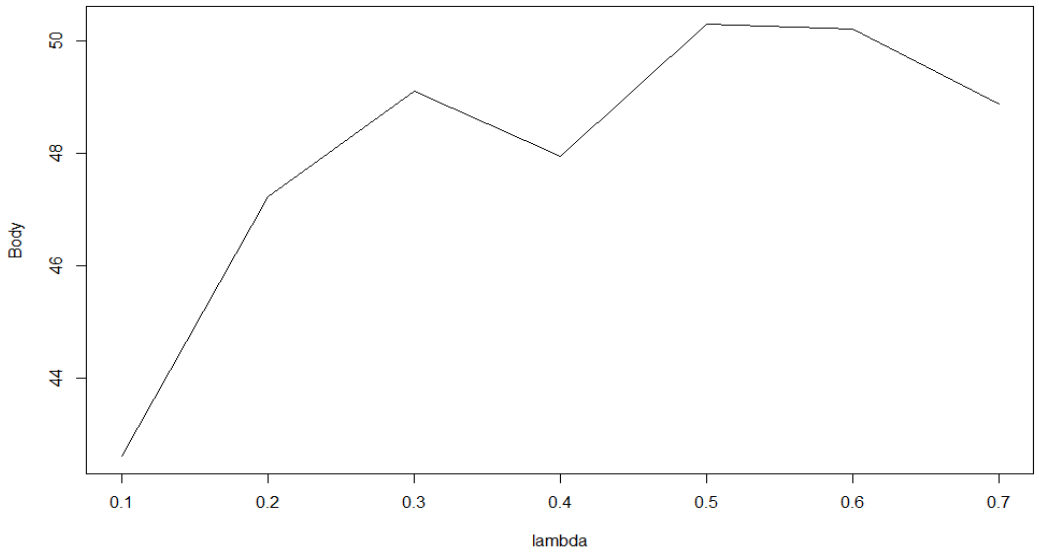
Tabulka 15 shrnuje výsledné parametry pro DUCT, které jsme určili v této kapitole. Počty simulací a determinací můžeme zvýšit, pokud bychom chtěli dát agentovi delší čas na tah.

Parametr	**Výsledná hodnota**
Počet simulací	5000
c (pro UCT)	0.4
Počet determinací	3
Ohodnocení hry	Sigmoid (se základem 1.7)
Body za **Pudding**	4.5/5.5/6.5
Druh simulace	Náhodná

Tabulka 15: Výsledné parametry pro DUCT

6.2 EXP3

Pro EXP3 budeme ladit stejné parametry, jako pro DUCT. Zajímavý parametr je λ, který funguje podobně, jako *c* v DUCT. Vzhledem k tomu, že EXP3 je stochastický algoritmus, tak λ určuje pravděpodobnost na prozkoumání náhodného tahu. Všechny ostatní parametry se týkají vlastností hry, tudíž můžeme vzít výsledné hodnoty jako pro DUCT. Parametr $\lambda \in (0,1]$ a jako výchozí hodnotu soupeře zvolíme $\lambda = 0.4$.



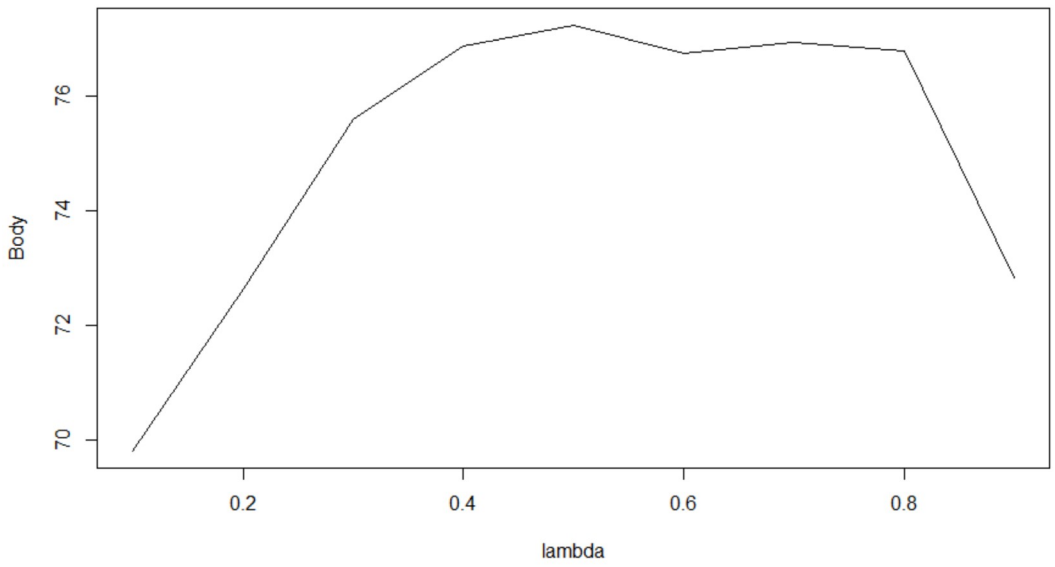
Obrázek 7: Výsledky pro λ v EXP3

Vidíme, že bodový zisk je dost nestabilní. To může být způsobeno tím, že některé konkrétní hodnoty λ jsou dobré proti jiným. Zajímavý je také malý bodový zisk pro λ = 0.4, kde jsme získali pouze 48 bodů. To je nejspíše způsobeno stochastickými vlastnostmi algoritmu.

Nyní provedeme stejný experiment proti hladovému agentovi. Tento experiment by nám měl dát stabilnější výsledky. Výsledky těchto experimentů popisují Tabulka 16 a Obrázek 8.

Lambda	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
Výhry	68.04	70.52	73.62	75.06	75.50	74.90	75.18	75.12	70.80
Prohry	28.40	25.24	22.40	21.28	21.02	21.38	21.30	21.52	25.14
Remízy	3.56	4.24	3.98	3.66	3.48	3.72	3.52	3.36	4.06
Body	69.82	72.64	75.61	76.89	77.24	76.76	76.94	76.8	72.83

Tabulka 16: Výsledky pro λ proti hladového agentovi

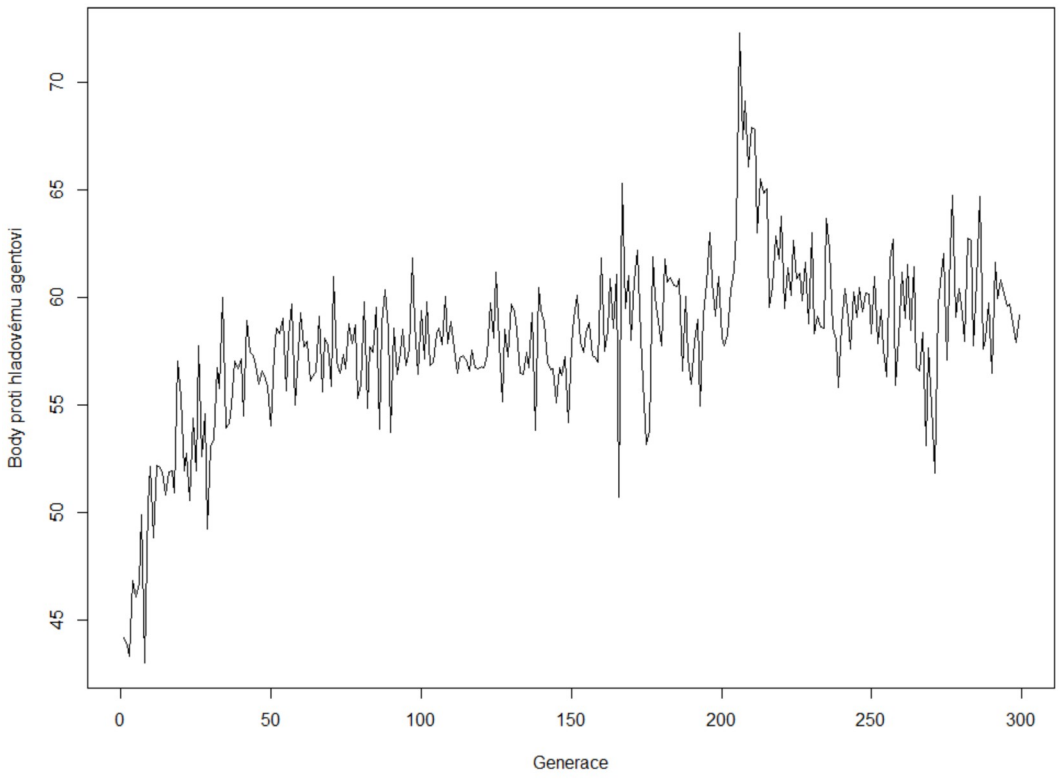


Obrázek 8: Výsledky pro λ proti hladového agentovi

Z Obrázku 8 je patrné, že proti hladovému agentovi jsou výsledky stabilnější. Také vidíme, že EXP3 má velikou úspěšnost proti hladovému agentovi. Opět nám vyšlo nejlépe λ rovno 0.5, takže tuto hodnotu budeme dále používat.

6.3 Genetický agent

Nyní se budeme zabývat parametry genetického algoritmu. Jako soupeře budeme používat nejlepší genetické agenty z předchozích generací. Tyto agenty budeme měnit každých pět generací. Pomocí měnění agentů se vyhneme přeučení na chybách soupeře. Tím, že je neměníme každou generaci, tak se vyhneme případům, typu kámen, nůžky, papír, kde máme tři různé agenty, kteří se vzájemně porážejí. Jelikož bychom neviděli růst fitness, tak v každé generaci necháme hrát nejlepšího agenta proti hladovému agentovi. Tento graf bude mít velké výchylky, protože agenty neučíme proti hladovému algoritmu. Zde spíše uvidíme, jak se agent pomalu postupně zlepšuje proti neznámému soupeři. Toto srovnání ukazuje Obrázek 9. Provedli jsme evoluci o 300 generacích, kde v každé generaci máme 128 jedinců. Pro každého jedince jsme počítali fitness pomocí 1000 odehraných her. To se může zdát jako málo, ale zvýšení počtu simulovaných her výrazně zpomalí celou simulaci. Navíc měníme v jednotlivých generacích soupeře, takže daný jedinec bude mít různé hodnoty fitness v různých generacích. Dohromady jsme pouze pro tento experiment sehráli téměř 40 milionů her.



Obrázek 9: Výsledky nejlepšího agenta z generace proti hladovému agentovi

Nyní se podíváme na přiřazené váhy jednotlivým pravidlům. V předpokladech pravidel navíc vždy uvažujeme, že máme v ruce danou kartu. Váhy jsou reálná čísla z intervalu $[0,100]$.

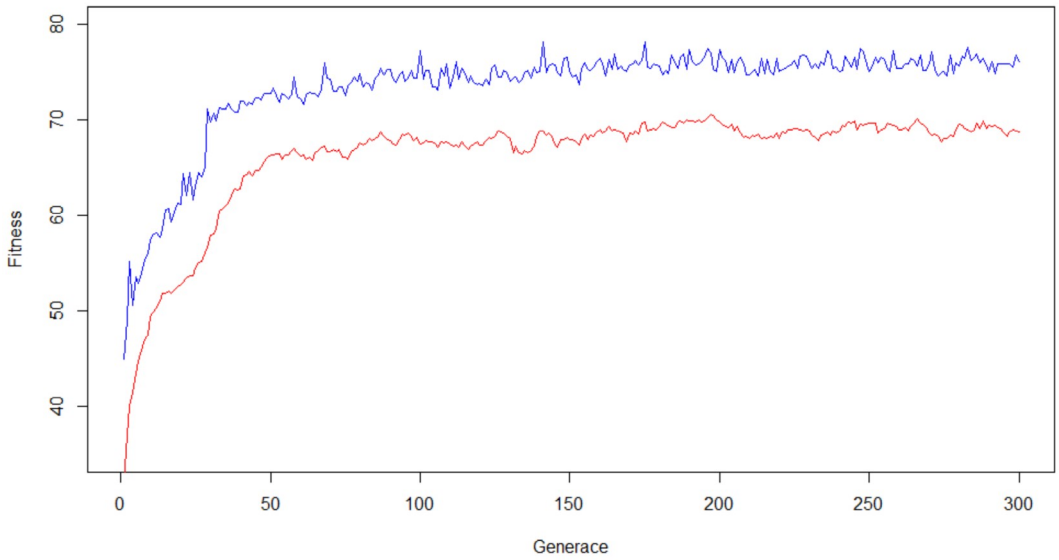
Předpoklad	**Akce**	**Váha**
Nezahráno **Wasabi**, 7+ karet v ruce	Wasabi	100.00
---	Nigiri 3	94.24
Nemáme **Nigiri 3**	Nigiri 2	6.89
Zahrané **Wasabi**	Nigiri 3	48.14
Zahrané **Wasabi**, nemáme **Nigiri 3**	Nigiri 2	48.11
Zahrané **Wasabi**, nemáme **Nigiri 3** a **Nigiri 2**	Nigiri 1	0.00
Zahráno sudý počet **Tempura**	Tempura	69.48
Zahráno lichý počet **Tempura**	Tempura	0.25
3+ **Tempura** v ruce	Tempura	63.00
součet **Dumplings** v ruce a zahrané >= 3	Dumplings	91.42
součet **Sashimi** v ruce a zahrané >= 3	Sashimi	82.98
| max Pudding dif | <= 1	Pudding	14.83
| min Pudding dif | <= 1	Pudding	0.00
| max Maki diff | < 3	Maki 3	0.31
| max Maki diff | < 3, nemáme **Maki 3**	Maki 2	0.00
| max Maki diff | < 3, nemáme **Maki 3** a **Maki 2**	Maki 1	81.97

Tabulka 17: Výsledné ohodnocení pravidel

Tabulka 17 je setříděná podle výsledné akce, pouze u karet **Nigiri** nejprve zkoušíme varianty bez zahrané karty **Wasabi** a následně s ní. Navíc toto pořadí pravidel používáme v případě genetického agenta s vlastními vahami, o kterém bude řeč později v části 8.3.2 Nyní pořádně vysvětlíme předpoklady pravidel. *Nemáme* znamená, že příslušnou kartu nemáme v ruce. Toto pravidlo používáme, abychom nehráli například **Maki 2**, pokud máme **Maki 3**. *Zahráno* označuje karty, které jsme zahráli v této sérii. *Maki diff* definujeme následovně: *max Maki diff := Max Maki body ostatních hráčů – naše Maki body*. Stejným způsobem definujeme i *Pudding diff*.

Z výsledků vidíme, že pravidla pro karty **Wasabi** a **Nigiri** 3 mají vysoké váhy. Podobně i **Nigiri 2**, pokud máme zahrané **Wasabi**. Naopak **Nigiri 2** bez **Wasabi** a **Nigiri 1** jsou špatné. Velmi důležitá je také karta **Tempura**. Překvapivě dobré jsou karty **Dumplings** a **Sashimi**, pokud jich máme hodně. Naopak velmi špatně dopadla karta **Pudding** a karty **Maki roll**. Překvapivě velkou váhu má i karta **Maki roll 1**, která se nejspíše hraje, když už nejsou lepší akce. Výsledného agenta budeme označovat GAG (genetický agent trénovaný proti genetickým agentům).

Zkusíme ještě tento experiment zopakovat s tím, že budeme našeho agenta učit proti hladovému agentovi. Tohoto agenta budeme nazývat GAH (genetický agent trénovaný proti hladovému agentovi).



Obrázek 10: Fitness proti hladovému agentovi, modrá křivka je maximální fitness v dané generaci a červená je průměrná fitness

Zde už máme typický růst fitness. Graf je mnohem stabilnější a z počátku roste rychleji. Vidíme, že maximální ale také průměrná fitness jsou výrazně vyšší, než v Obrázku 9. Máme tedy výrazně lepší výsledky proti hladovému agentovi, jak bychom předpokládali. V následující kapitole agenty porovnáme a uvidíme, zda budeme mít lepší výsledky i proti ostatním agentům. Přiřazené váhy ukazuje Tabulka 24, která je v přílohách a nebudeme ji více rozebírat.

Nakonec vytvoříme ještě agenta GA5 (genetický agent trénovaný pro hru pěti hráčů). Fitness tohoto agenta budeme počítat ze hry pěti hráčů proti čtyřem různým genetickým agentům. V každé generaci z nich nejstaršího vyměníme za nejlepšího agenta z minulé generace. Opět budeme simulovat 300 generací se 128 jedinci v kaž-

dé generaci. Fitness budeme opět počítat jako průměrný bodový zisk ze 100 her, kde pro každého agenta odehrajeme 1000 her. Tohoto agenta budeme používat pro hru více než dvou hráčů. Podobně jako u GAH ukazuje přiřazené váhy Tabulka 25, která je opět uvedena v přílohách.

7 Výsledky

V minulé kapitole jsme získali všechny potřebné parametry pro naše agenty. Pokud nebude řečeno jinak, tak vždy budeme používat tyto získané parametry. Nyní můžeme nechat hrát různé agenty vzájemně proti sobě a zjistit, který je nejlepší. Vždy sehrajeme 10000 her, abychom měli jistotu, že výsledky neovlivní faktor náhody, který je ve hře Sushi Go.

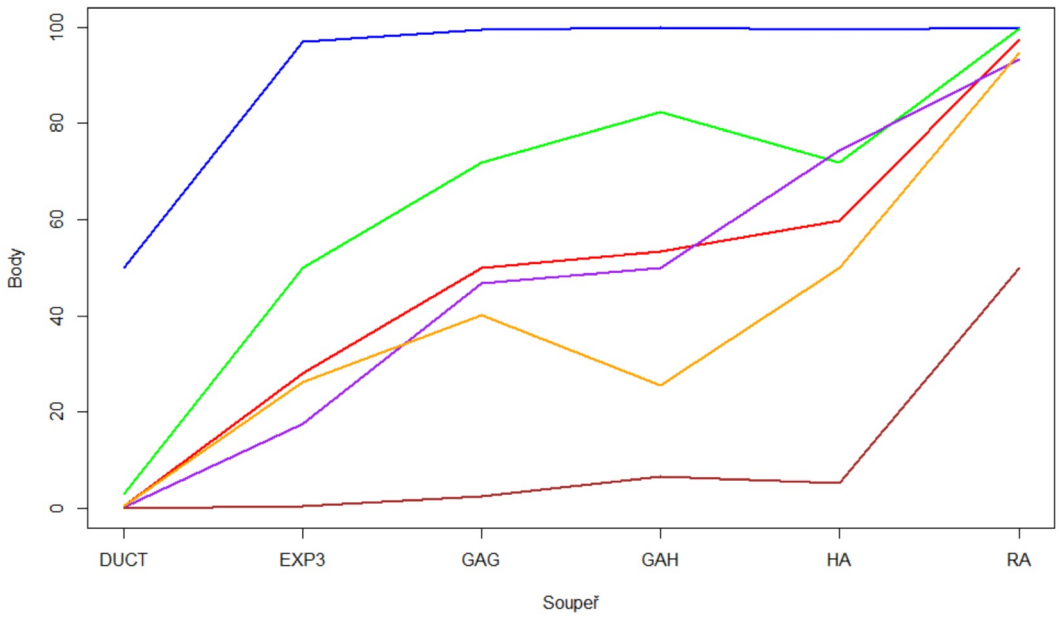
Máme zde jeden zásadní faktor. Agenti na bázi MCTS (DUCT a EXP3) dělají prohledávání, takže jim nějakou dobu trvá najít výslednou akci. Naopak ostatní agenti žádné prohledávání nedělají a akci zvolí velmi rychle. Zde se tedy musíme rozhodnout, jak dlouhou dobu budeme chtít dát agentům k nalezení výsledného tahu.

7.1 Hra dvou hráčů

V této části porovnáme agenty pro hru dvou hráčů. Necháme je hrát každého s každým. Výsledky těchto her jsou uvedeny v Tabulce 18. Hodnoty v tabulce určují průměrný bodový zisk ze 100 her z pohledu agenta na daném řádku.

	DUCT	**EXP3**	**GAG**	**GAH**	**Hladový**	**Náhodný**
DUCT		96.97	99.46	99.83	99.43	99.99
EXP3	3.03		71.98	82.35	71.81	99.64
GAG	0.54	28.02		53.32	59.80	97.52
GAH	0.17	17.65	46.68		74.41	93.40
Hladový	0.57	26.19	40.20	25.59		94.72
Náhodný	0.01	0.36	2.48	6.60	5.29	

Tabulka 18: Vzájemné výsledky mezi agenty



Obrázek 11: Vzájemné výsledky mezi agenty, modrá křivka je DUCT, zelená je EXP3, červená je GAG, fialová je GAH, oranžová je HA a hnědá je RA

Výsledky shrnuje Obrázek 11. HA označuje hladového agenta a RA je náhodný agent. Hodnota v grafu vždy ukazuje bodový zisk agenta označeného křivkou proti soupeři na ose *y*, takže nejlepší agent má nejvyšší plochu pod křivkou. Vidíme, že DUCT agent je s přehledem nejlepším agentem pro hru dvou hráčů. Druhý je EXP3 agent, ale proti DUCT má před ostatními mnohem menší náskok. GAG má lepší výsledky než GAH proti všem agentům kromě hladového. Hladový agent skončil hned za nimi. Náhodný agent je zdaleka nejhorší a neměl šanci proti žádnému jinému agentovi.

DUCT agent dopadl výrazně lépe než EXP3 agent. To nám ukazuje, že ve hře Sushi Go nejspíše stačí hrát čistou strategii.

DUCT agent dopadl zdaleka nejlépe. Nyní vyzkoušíme, co se stane, pokud budeme v DUCT simulovat méně her. To je důležité, pokud budeme chtít, aby agent hrál velmi rychle a nemuseli jsme čekat, než najde nejlepší tah. Zde nemá cenu zkoušet EXP3, protože jsme zjistili, že DUCT funguje výrazně lépe a u EXP3 bychom také museli snížit počet simulovaných her.

Počet her	**1000**	**500**	**100**	**50**
GAG	99.50	99.38	96.84	92.36
GAH	99.83	99.73	98.64	96.53
Hladový	99.42	99.15	97.33	93.55

Tabulka 19: Úspěšnost DUCT proti ostatním agentům

Vidíme, že i s malým počtem simulací je DUCT velmi dominantní. Jedná se nejspíše o ideální přístup pro hru dvou hráčů. Takto velká dominance bude nejspíše způsobena tím, že jsme nad metodami MCTS strávili více času. Genetický algoritmus by mohl dopadnout lépe s lepšími pravidly. Ovšem prohledávání, které dělá

MCTS je pro hru Sushi Go klíčové. Především při hře dvou hráčů, kde máme po prvním tahu úplnou informaci o stavu hry a výsledek hry nemohou ovlivnit ostatní hráči.

Genetický agent se zdá být velkým zklamáním. Musíme si ale uvědomit, že tento agent má pouze ohodnocená pravidla a nedělá prohledávání. Navíc se ukázalo, že je prohledávání pro hru Sushi Go velmi důležité. Můžeme to připodobnit ke hře, jako jsou šachy. Zde by nejspíš agent tohoto typu také neměl šanci proti prohledávání typu MCTS.

Výsledek, že MCTS bude nejlepší agent pro hru dvou hráčů, je očekávaný, protože se jedná o hlavní metodu, která se používá pro hry se současnými tahy. [3][5] [6]

Už víme, že DUCT je nejlepší z našich agentů. Nyní zkusíme nechat hrát DUCT proti Cheating DUCT. Cheating DUCT je agent, který funguje stejně jako běžné DUCT. Pouze se na začátku série podívá na karty ostatních hráčů. Z tohoto důvodu má úplnou informaci a nemusí dělat determinace. Cheating DUCT jsme nezařadili do turnaje, protože na rozdíl od ostatních agentů podvádí. Porovnáváme je spíše ze zajímavosti. Pro oba agenty použijeme parametry uvedené v Tabulce 15. Pro DUCT budeme zkoušet různé hodnoty pro počet determinací. Odehrajeme 5000 her a pro oba agenty budeme uvádět průměrný bodový zisk ze 100 her.

Počet determinací	**1**	**3**	**5**	**10**
DUCT	27,28	31,72	33,42	33,61
Cheating DUCT	72,72	68,28	66,58	66,39

Tabulka 20: Výsledky DUCT proti Cheating DUCT

Vidíme, že Cheating DUCT dopadl výrazně lépe. Simulovat alespoň tři determinace je důležité, protože jen s jednou determinací má agent výrazně horší výsledky.

Musíme si ovšem uvědomit jednu věc. I kdybychom dělali nekonečno determinací, tak pořád bude Cheating DUCT lepší. Rostoucí počet determinací nám pomáhá, abychom průměrně dopadli lépe. Naopak, Cheating DUCT zná aktuální rozdání karet, což běžné DUCT nikdy nezná.

7.2 Hra více hráčů se slabým soupeřem

Už víme, že hra dvou hráčů je symetrická a pořadí hráčů nezáleží. Oba hrají současně a vzájemně si předají své karty. Tato situace už neplatí při hře alespoň tří hráčů. Zde se mohou dva hráči spojit proti třetímu. To ovšem nemáme jak efektivně měřit. Můžeme pouze měřit, co se stane, když s dvěma dobrými agenty necháme hrát jednoho špatného. Agent, který dostává karty od horšího soupeře by mohl mít výhodu, protože horší soupeř nám může nechat dobrou kartu.

Podíváme se na hru tří hráčů, kde vždy budeme mít dva identické silné agenty a jednoho slabého agenta. Stejně jako u všech experimentů odehrajeme 10000 her a budeme uvádět průměrný bodový zisk ze 100 her. V každé hře budeme rozdělovat jeden bod. Když agent vyhraje sám, tak dostane celý bod. Pokud bude mít více hráčů maximální počet bodů a stejný počet karet **Pudding**, tak si tento bod mezi sebe rovnoměrně rozdělí.

Agenti	DUCT / RA	DUCT / HA	GAG / RA
Silný agent 1	78.88	68.67	62.08
Silný agent 2	21.07	29.39	36.41
Slabý agent	0.05	1.94	15.1

Tabulka 21: Výsledky hry tři hráčů s jedním slabým agentem. Popisek agentů značí silný agent / slabý agent.

Pořadí agentů je *silný agent 1, silný agent 2, slabý agent*. Tedy slabý agent agent předává karty *silnému agentovi 1*. Z Tabulky 21 vidíme, že slabý agent má obrovský vliv na výsledek hry. Toto nám dává zajímavý poznatek o hře, že je lepší hrát za horším hráčem a před lepším hráčem. Z tohoto důvodu si u hry více hráčů musíme dávat velký pozor na pořadí agentů.

Nyní vyzkoušíme, zda bude výhodné být za nejhorším agentem i při hře čtyř hráčů. Zde budeme pro DUCT agenta simulovat pouze 3000 her na tah, protože hru více hráčů trvá simulovat déle. Pořadí agentů je DUCT 1, Hladový agent, DUCT 2 a Náhodný agent.

Agent	Bodový zisk
DUCT 1	55.53
Hladový agent	5.88
DUCT 2	38.54
Náhodný agent	0.05

Tabulka 22: Hra čtyř hráčů se slabými soupeří

V Tabulce 22 opět vidíme, že na pořadí hráčů opravdu záleží. Není zde až tak obrovský rozdíl, jako při hře tří hráčů. Tato vlastnost hry nám zásadně komplikuje turnaj více hráčů.

7.3 Hra více hráčů

Už jsme prozkoumali hru dvou hráčů. V této kapitole se zaměříme na hru více hráčů. Uděláme tedy turnaj pro pět různých agentů. Na hru tří a čtyř hráčů se nebudeme zaměřovat, protože hra pěti hráčů je z pohledu umělé inteligence nejsložitější.

V minulé části jsme zjistili, že si musíme dát velký pozor na pořadí hráčů. Navíc slabý agent má špatný vliv na výsledky. Proto v tomto turnaji nenecháme hrát náhodného agenta. Ideálním způsobem by bylo vyzkoušet všechny permutace agentů. Těch je ovšem $5! = 120$. To je poměrně velké číslo. Na druhou stranu si můžeme uvědomit, že permutace $1-2-3-4-5$ a $2-3-4-5-1$ jsou z pohledu Sushi Go identické. V Sushi Go záleží pouze na tom, komu předáváme karty a od koho karty dostáváme, což je zde pro všechny hráče stejné. Pro každou permutaci máme tedy další čtyři identické permutace. To odpovídá všem permutacím, když na prvním místě zafixujeme 1. Takových permutaci pak je $4! = 24$. Pro 24 permutací už můžeme takový problém vyřešit.

Teď se zaměříme na agenty, které necháme hrát v turnaji. Samozřejmě musíme nechat hrát DUCT a EXP3. Z genetických agentů necháme hrát GAG a GA5, kterého jsme speciálně trénovali pro hru pěti hráčů. Posledním agentem bude Hladový agent. Tyto agenty očíslujeme následovně DUCT = 1, EXP3 = 2, HA = 3, GAG = 4

a GA5 = 5.Permutace $1-2-3-4-5$ znamená posloupnost agentů DUCT, EXP3, HA, GAG, GA5. U DUCT a EXP3 budeme pro každý tah simulovat 2000 her. Pro každou permutaci odehrajeme také 2000 her, dohromady tedy odehrajeme 48000 her.

Výsledky jsou uvedeny v Tabulce 22. Pro hru pěti hráčů je očekávaný průměrný bodový zisk 20 bodů. I u hry pěti hráčů mají nejlepší výsledky DUCT a EXP3. DUCT má průměrný bodový zisk dokonce přes 49. GA5 dopadl podle očekávání lépe, než GAG. Velkým překvapením ale je, že HA dopadl lépe, než oba genetické algoritmy. To nejspíš bude způsobeno velkými rozdíly, mezi úrovní hry jednotlivých agentů. Opět velmi záleží na pořadí agentů. EXP3 má vždy bodový zisk menší než 20, pokud hrál bezprostředně po DUCT, Naopak všechny hry, kde nehrál EXP3 za DUCT, tak má vždy bodový zisk alespoň 20 bodů a často má dokonce i přes 30 bodů. HA, GAG a GA5 společně průměrně získaly téměř 25 bodů. Je tedy zjevné, že při hře pěti hráčů mohou konzistentně vyhrávat i trochu slabší agenti.

Pořád musíme brát v potaz, že se někteří agenti mohou spojit proti jiným. Nemáme žádný způsob, jak toto měřit. Obecně se nám ale potvrdilo, že je ideální hrát za nejslabším soupeřem. Toto by mělo platit i pro hru lidí.

Permutace	DUCT	EXP3	HA	GAG	GA5
1-2-3-4-5	51.43	16.68	11.45	7.55	12.85
1-2-3-5-4	54.55	14.60	9.95	8.45	12.40
1-2-4-3-5	54.82	17.75	9.39	5.94	12.09
1-2-4-5-3	56.30	14.13	16.05	5.33	8.20
1-2-5-3-4	54.05	13.43	17.38	9.25	5.90
1-2-5-4-3	60.38	13.29	11.42	8.59	6.33
1-3-2-4-5	48.10	27.94	7.31	6.39	10.26
1-3-2-5-4	49.84	28.69	5.45	9.53	6.49
1-3-4-2-5	47.38	33.58	7.33	4.70	7.03
1-3-4-5-2	42.49	33.05	7.54	5.47	11.45
1-3-5-2-4	45.75	32.00	5.13	7.23	9.90
1-3-5-4-2	41.78	33.83	5.79	8.64	9.97
1-4-2-3-5	51.10	25.25	9.40	3.63	10.63
1-4-2-5-3	53.78	24.36	13.76	2.53	5.58
1-4-3-2-5	48.54	29.59	10.63	3.83	7.42
1-4-3-5-2	43.60	31.28	10.13	3.83	11.18
1-4-5-2-3	51.38	26.95	12.00	2.38	7.30
1-4-5-3-2	39.83	33.66	16.65	4.18	6.47
1-5-2-3-4	49.68	26.41	12.13	6.95	4.83
1-5-2-4-3	55.49	24.59	9.29	7.13	3.50
1-5-3-2-4	47.78	30.70	12.90	5.43	3.23
1-5-3-4-2	40.72	35.94	12.63	7.68	3.04
1-5-4-2-3	51.49	27.35	9.50	7.60	4.10
1-5-4-3-2	40.35	34.74	12.58	9.47	2.87
Průměr	**49.19**	**26.24**	**10.66**	**6.32**	**7.62**

Tabulka 23: Výsledky pro hru pěti hráčů

8 Uživatelská dokumentace

Tato kapitola obsahuje informace k použití programů. Práce obsahuje dvě verze, jedna je grafická pro hru člověka proti člověku nebo proti umělé inteligenci. Druhá verze je efektivnější konzolová pro vývoj a testování umělých inteligenci.

Konzolová verze je psaná v C++ (C++17) s použitím pouze standardních knihoven. Grafická verze je psaná v C++ v prostředí Qt [8].

8.1 Obsah CD

Spustitelné soubory jsou umístěny ve složce *bin*. V této složce se nachází soubor *Sushi_Go_evaluation.exe*, což je konzolová aplikace sloužící k vývoji umělých inteligencí (Windows 32-bit). Dále se ve složce *bin* nachází složka *SushiGo*. V této složce je soubor *SushiGo.exe*, který slouží ke spuštění grafické aplikace (Windows 32-bit). Dále složka obsahuje několik *.dll* souborů, které obsahují knihovny potřebné k spuštění aplikace.

Všechny zdrojové kódy jsou umístěny ve složce *src*. Zde se nachází složka *Sushi_go_evaluation*, ve které jsou umístěny soubory pro konzolovou aplikaci. Jedná se především o soubor *Sushi_go_evaluation.sln*, který slouží k načtení projektu do Visual Studio 2019. Dále se zde nachází zdrojové soubory projektu ve formátu *.h* a *.cpp*.

Ve složce *src* je také složka *SushiGo*, která obsahuje zdrojové soubory ke grafické aplikaci. Hlavním souborem je *SushiGo.pro*, který souží k načtení projektu do Qt. Dále složka opět obsahuje soubory formátu *.h* a *.cpp* se zdrojovými kódy. Grafická aplikace má navíc soubory formátu *.ui*, které popisují jednotlivé obrazovky. V této složce jsou ještě umístěny složky s obrázky jednotlivých karet a to ve formátu *.png*.

Poslední částí je složka *výsledky*. V této složce jsou soubory s výsledky experimentů. Všechny soubory jsou ve formátu *.csv*.

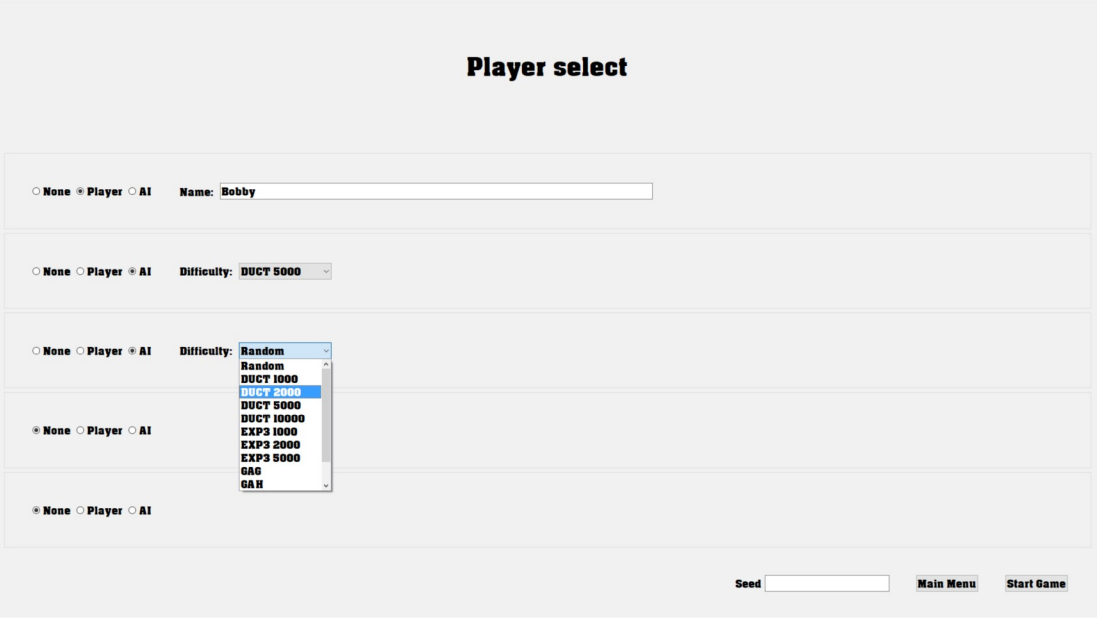
8.2 Grafická aplikace

V této části se podíváme na grafickou aplikaci a v následující části se podíváme na konzolovou aplikaci. Spuštění grafické aplikace nás dostane do hlavního menu. V hlavním menu můžeme ukončit aplikaci pomocí tlačítka *Quit*, zobrazit pravidla hry pomocí tlačítka *Rules* nebo se přepnout na obrazovku s výběrem hráčů pomocí tlačítka *New Game*.

8.2.1 Výběr hráčů

Obrazovka pro výběr hráčů slouží k výběru počtu a typů jednotlivých hráčů. Nejprve vždy navolíme typ hráče *Player*. *AI* nebo *None*. *Player* je lidský hráč a *AI* je umělá inteligence. Pokud zvolíme *AI*, tak se nám objeví seznam možných agentů. Zde si pak vybereme příslušné protihráče.

Pro začátek hry musíme vždy zvolit alespoň dva hráče typu *AI* nebo *Player*. Z nich alespoň jeden hráč musí být typu *Player*. Pro hru pouze mezi počítačovými agenty slouží konzolová aplikace.



Obrázek 12: Obrazovka pro výběr hráčů

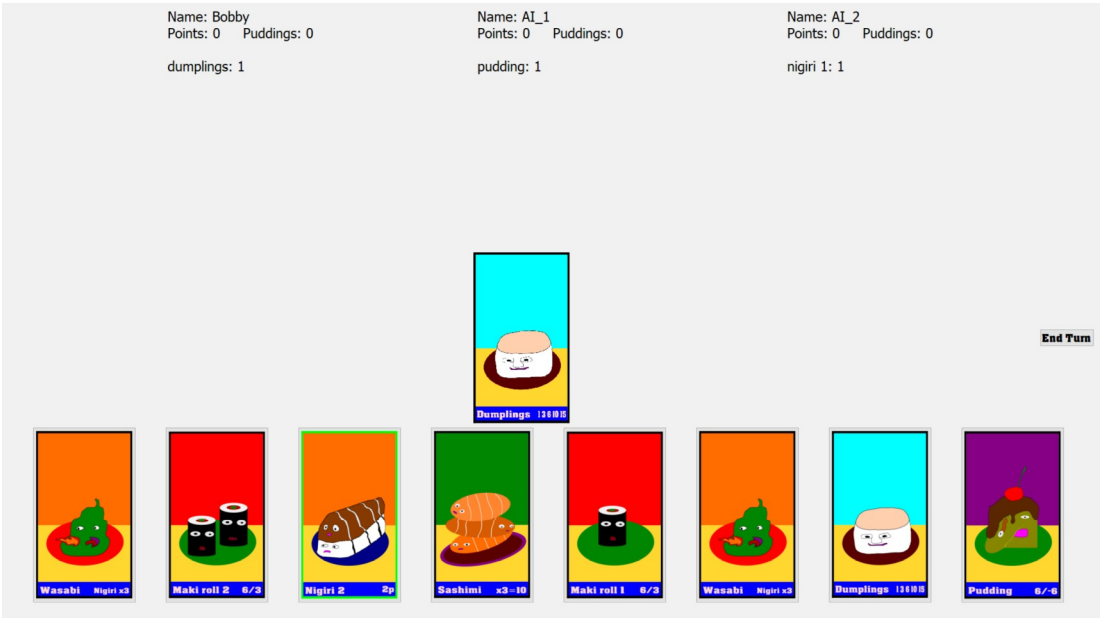
Stiskem tlačítka *Start Game* se přeneseme na herní obrazovku, kde začíná samotná hra s aktuálním nastavením hráčů. Po stisku tohoto tlačítka se uloží aktuální nastavení hráčů a při dalším spuštění aplikace se obrazovka pro výběr hráčů automatický načte do stejné situace z minulé hry.

Můžeme si nastavit vlastní seed, kdybychom chtěli hrát stejnou hru znovu. Seed ovlivně pouze zamíchání a rozdání karet, ale neovlivňuje hru soupeřů. Platnou hodnotou jsou pouze přirozená čísla. Pokud seed nezadáme nebo zadaný seed nebude platný, tak se spustí náhodná hra.

8.2.2 Herní obrazovka

Jedná se o nejdůležitější obrazovku v celé hře. Herní obrazovka je rozdělena na tři části. V horní části hráč vidí informace o sobě a o ostatních hráčích. Tyto informace se vždy skládají z jména hráče, počtu zahraných karet **Pudding** během celé hry a z karet, které hráč již zahrál v této sérii. Informace o hráčích jsou vždy seřazeny tak, že aktuální hráč je první, druhý je hráč, kterému budeme předávat karty a poslední je hráč, od kterého dostáváme karty.

V prostřední části obrazovky se nachází karty, které jsme již zahráli v této sérii. Ve spodní části obrazovky jsou karty, které máme aktuálně v ruce. S těmito kartami můžeme interagovat. Pokud na nějakou z těchto karet klikneme, karta se vybere, a objeví se kolem ní zelený okraj. Pokud na ni klikneme znovu, výběr se zruší. Můžeme mít vybraných několik karet najednou, ale když klikneme na tlačítko *End Turn*, tak musíme mít vybranou právě jednu kartu. Jinak dostaneme upozornění, že máme vybraný špatný počet karet a kolo se neukončí. Příklad herní obrazovky je na Obrázku 13. V případě, že máme zahranou kartu **Chopsticks** a máme v ruce alespoň dvě karty, objeví se tlačítko s nápisem *Use Chopsticks*. Použití tohoto tlačítka nastaví, že použije kartu **Chopsticks** a pak musíme vybrat právě dvě karty. Pokud znovu klikneme na toto tlačítko, tak se použití karty **Chopsticks** zruší a opět musíme mít vybranou právě jednu kartu. Pokud máme vybraný správný počet karet a klikneme na tlačítko *End Turn*, tak se ukončí náš tah a zahrají se vybrané karty.



Obrázek 13: Herní obrazovka

Po úspěšném ukončení tahu se na řadu dostává další hráč. Pokud máme alespoň dva lidské hráče, tak se mezi jejich tahy objeví čekající obrazovka s jménem hráče, který je na tahu. Tato obrazovka slouží k výměně hráčů, aby si vzájemně neviděli své karty. Herní obrazovka se týká pouze hráčů typu *Player*. Hráči typu *AI* hrají automaticky bez vykreslování této obrazovky.

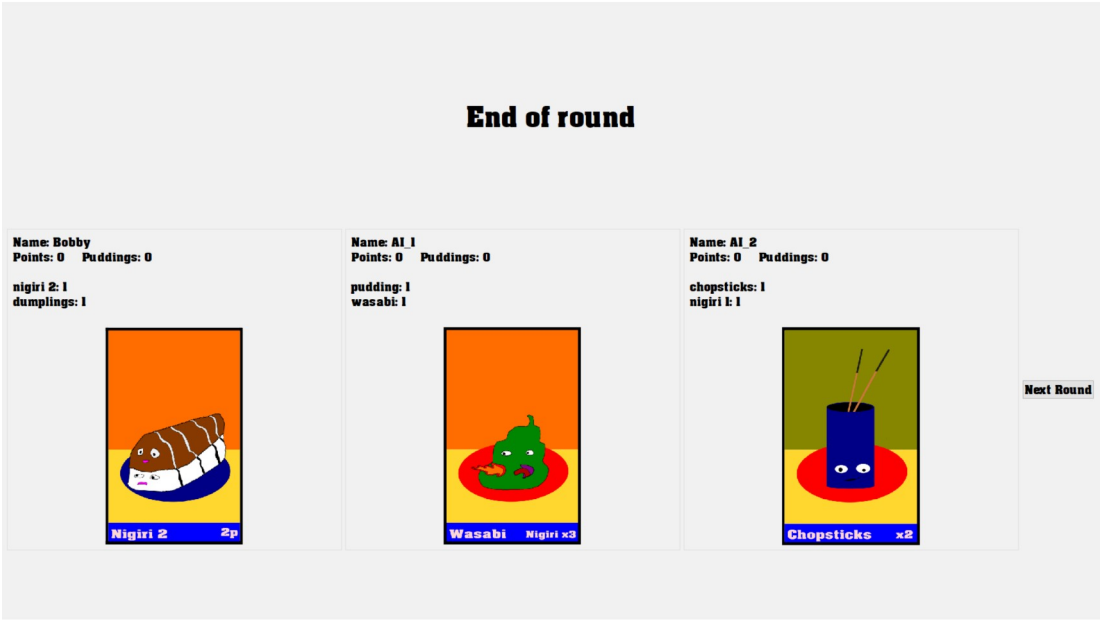
Po dokončení celého kola se nám objeví obrazovka s výsledky.

8.2.3 Obrazovky s výsledky

V aplikaci máme hned tři různé obrazovky s výsledky. První je obrazovka s výsledky kola, druhá je obrazovka s výsledky série a třetí je obrazovka s výsledky celé hry. Se žádnou z těchto obrazovek nejde interagovat a jsou pouze informativní. Všechny pouze obsahují tlačítko pro pokračování.

Obrazovka pro konec kola ukazuje, které karty zahráli jednotliví hráči. U každého hráče máme zobrazenu jednu kartu, kterou zahrál. Pokud máme u některého hráče zobrazeny dvě karty, tak to znamená, že použil kartu **Chopsticks**. Obrazovka je znázorněna na Obrázku 14. Nebudeme uvádět příklady ostatních obrazovek s výsledky, protože jsou velmi podobné.

Obrazovka pro konec série ukazuje výsledky série. U každého hráče vidíme, jaké karty během série zahrál a kolik za ně dostal bodů.



Obrázek 14: Obrazovka pro konec kola

Obrazovka pro konec hry nám ukazuje celkové výsledky hry. U každého hráče vidíme, kolik bodů dostal v jednotlivých sériích a kolik bodů dostal nebo ztratil za karty **Pudding**. Hráči zde jsou seřazeni podle výsledků hry, kde vítěz je první zleva. Z této obrazovky se následně vrátíme zpět do hlavního menu, kde můžeme zapnout novou hru nebo vypnout aplikaci.

8.3 Konzolová aplikace

Konzolová aplikace slouží pro ladění parametrů a testování agentů. Aplikace má dvě nastavení. Pokud ji spustíme s přepínačem *-ga*, tak se aplikace spustí v módu pro genetický algoritmus. Pokud ji spustíme bez tohoto přepínače, tak se nám otevře verze pro vzájemnou hru agentů.

8.3.1 Genetický algoritmus

Tento mód slouží k nalezení vah pro naše genetické agenty. V tomto módu se nás nejprve program zeptá, proti kolika hráčům chceme hrát. Validní možnosti jsou přirozená čísla mezi 2 a 5 včetně. Následně se aplikace zeptá, kolik chceme simulovat generací. Zde musíme zadat přirozené číslo. Poté se program zeptá na počet jedinců v každé generaci. Zde opět musíme zadat přirozené číslo. Posledním parametrem je typ protihráče. Zde můžeme zvolit buď 0 pro genetické agenty nebo 1 pro hladové agenty.

Výstupem tohoto programu je vždy čtveřice: číslo generace, maximální fitness, průměrná fitness, výsledek nejlepšího agenta proti hladovému agentovi. Po výsledku poslední generace program ještě vypíše výsledné váhy.

```
Number of players (2-5)
2
Number of generations (Natural numbers)
10
Individuals per generation (Natural numbers)
50
Type of opponents:
0 - Genetic agent
1 - Hungry agent
1

1 46.25 34.014 48.1
2 48.2 40.666 47.45
3 50.9 43.285 47.15
4 49.8 43.962 50.3
5 52.95 45.149 57.6
6 55.85 47.006 54.1
7 56.55 48.888 52.75
8 60.3 49.209 57.2
9 57.3 49.452 55.15
10 58.55 50.268 57.7
50.9626 82.4297 89.2055 3.26952 64.5026 5.41316 77.1945 5.06585 58.7976 73.6735 60.8924 2.19065 5.50729 1.72448 19.0805 95.7298
```

Obrázek 15: Příklad genetického algoritmu

Na Obrázku 15 vidíme příklad použití genetického algoritmu. V první části tohoto příkladu máme nastavení pro hru dvou hráčů proti hladovému agentovi, kde simulujeme 10 generací a v každé generaci máme 50 jedinců.

8.3.2 Hra mezi agenty

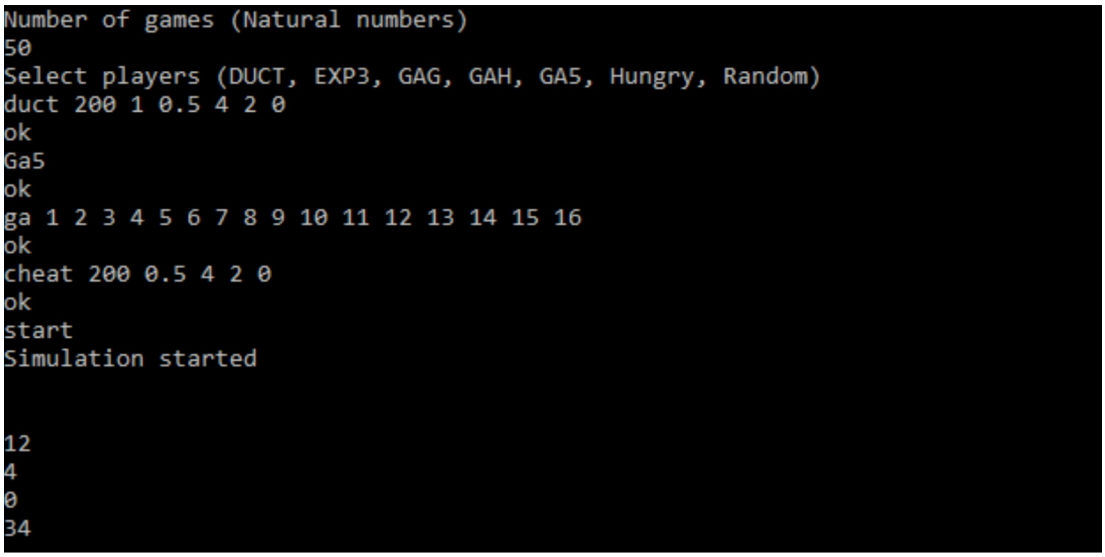
Druhou a asi důležitější částí konzolové aplikace je hra mezi různými agenty. Pomocí této aplikace můžeme ladit parametry pro MCTS a také vzájemně porovnávat různé agenty. Tato verze se spustí automaticky se spuštěním aplikace. Program se nás nejprve zeptá, kolik chceme odehrát her. Zde se očekává přirozené číslo. Následně musíme nastavit agenty, kteří budou hrát. Můžeme zvolit jednoho z přednastavených agentů s vyladěnými parametry z kapitoly 6. Možnosti pro agenty jsou *DUCT, EXP3, Hungry, Random, Cheating, GAG, GAH* a *GA5*. Po vložení platného agenta nám aplikace vždy odpoví *ok*. V momentě, kdy budeme mít pět zvolených agentů nebo napíšeme *Start*, se aplikace spustí.

Kromě přednastavených agentů můžeme zvolit i agenty s vlastními parametry. Toto se týká DUCT, EXP3 a GA. Pro genetického agenta s vlastními vahami zvolíme možnost *GA* a 16 čísel na intervalu $[0{,}100]$, která odpovídají vahám jednotlivých pravidel. Tato pravidla jsou popsána v Tabulce 17 a jsou ve stejném pořadí. Vstup musí být na jednom řádku a každý parametr musí být oddělen mezerou.

Pro DUCT a EXP3 opět zvolíme typ agenta a následně jeho parametry. Těmito parametry je počet simulací, počet determinací, konstanta pro exploraci (c nebo λ), způsob hodnocení výsledku hry, způsob hodnocení karty **Pudding** a nakonec typ simulace. Počet simulací a počet determinací jsou přirozená čísla. Pro DUCT je *c* kladné reálné číslo a pro EXP3 je λ reálné číslo na intervalu $(0{,}1]$. Pro hodnocení výsledků hry máme několik možností: 0 - pouze výhry, 1 - bodový rozdíl, 2 - sigmoid se základem e, 3 - sigmoid se základem 2 a 4 - sigmoid ze základem 1.7.

Pro karty **Pudding** máme tři možnosti. Tyto možnosti popisují přidělené body v jednotlivých sériích: 0 - 6/6/6, 1 - 4/5/6 a 2 - 4.5/5.5/6.5. Máme dva typy simulací: 0 - náhodná, 1 - vážená.

Stejným způsobem, jako DUCT můžeme vytvořit i Cheating agenta pomocí vstupu *cheat*, kde pouze vynecháme parametr pro počet determinací. Všechny ostatní parametry jsou stejné jako pro DUCT.



Obrázek 16: Příklad použití konzolové aplikace pro hru agentů

Obrázek 16 ukazuje příklad použití konzolové aplikace. V tomto příkladu simulujeme 50 her hry čtyř hráčů. Prvním agentem je DUCT s vlastními parametry. Druhým agentem je GA5. Třetím agentem je genetický agent s vlastními parametry a posledním čtvrtým agentem je cheating agent také s vlastními parametry. Vstup uživatele vypadá následovně:

```
50
duct 200 1 0.5 4 2 0
Ga5
ga 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
cheat 200 0.5 4 2 0
start
```

Zbytek textu je výstup programu. U vstupu nemusíme rozlišovat velká a malá písmena. Výstupem programu je pak počet získaných bodů pro každého hráče v odpovídajícím pořadí. V případě, že hráč nechá odsimulovat alespoň 500 her, tak se navíc po každých deseti procentech odehraných her vypíše informace, kolik procent her máme aktuálně odehráno, protože celé simulace mohou trvat velmi dlouho.

9 Implementační dokumentace

V této kapitole stručně popíšeme implementaci aplikace. Hra je napsaná v programovacím jazyku C++ s normou C++17. Pro grafickou aplikaci jsme použili prostředí Qt [8] open source verze 4.10.0. Grafická aplikace je přeložena pomocí kompilátoru MingW 7.3.0 32-bit a běží pod systémem Windows. Měla by běžet i pod systémem Linux, ale v tomto systému nebyla testována.

Konzolová aplikace je psaná pouze za pomocí standardních knihoven C++. Pro její překlad jsme použili překladač MSVC 2019 32-bit. Většinu výpočtů jsme prováděli pomocí distribuovaných výpočtů na platformě Linux. Zde jsme pro překlad použili GCC verze 9.3.0.

Dále se zaměříme na objektový návrh, hlavní třídy a logiku celé hry. Obecně budeme mluvit o grafické i konzolové verzi, protože mají jen poměrně malé rozdíly.

9.1 Qt

Qt je cross-platform vývojové prostředí pro GUI pomocí widgetů. Qt je asi nejpoužívanější vývojové prostředí pro grafiku v C++. Používá ho například Adobe Photoshop nebo Google Earth. My používáme open-source verzi Qt, která je licencována pod LGPLv3. [8]

Alternativně bychom mohli místo Qt použít nějaký herní engine, jako je například Unreal Engine nebo Unity. Zvolili jsme Qt, protože nám stačí jednoduché GUI, zmíněné engine jsou pro náš zbytečně složité. Navíc jejich použití pro nás nemá žádné výhody.

9.2 Správa Paměti

Jelikož C++ nemá Garbage Collector, tak musíme řešit správu paměti. Z tohoto důvodu budeme vždy používat *std::unique_ptr* [9], který při svém zániku smaže alokovaný objekt. Zároveň budeme používat přístup RAII (Resoure acquisitoin is initializaton) [9].

Všechny naše abstraktní třídy budou navíc obsahovat virtuální destruktor. Virtuální destruktor má na starost smazání poděděného objektu i pokud máme ukazatel na základní třídu. Jinak nám nastane Slicing a smaže se pouze část poděděného objektu.

9.3 Třída Card

Nyní se zaměříme na hlavní třídy. Nejprve se podíváme na třídy, které řeší logiku hry a následně se ještě podíváme na třídy pro agenty.

Třída *base_card* je abstraktní třída, která reprezentuje kartu. Tato třída obsahuje pouze virtuální funkce. Hlavní funkcí je *play()* pro zahrání karty. Zahrání karty řešíme pomocí struktury *card_list*, která obsahuje počty jednotlivých typů zahraných karet. Funkce *play()* pouze přidá zahranou kartu do této struktury, kterou dostane jako parametr. V případě grafické aplikace obsahuje navíc funkce pro vykreslování karty na tlačítko a štítek za pomoci funkcí z Qt. Pro tlačítko máme navíc funkci, která vykreslí vybrané karty se zeleným rámečkem. Dále tato třída obsahuje ještě pomocné funkce, které nám vrátí například typ karty. Tyto funkce používají agenti o kterých budeme mluvit později. Navíc samozřejmě obsahuje virtuální destruktor.

Každý typ karty (Tempura, Wasabi…) má vlastní třídu, která je poděděná od třídy *base_card* a příslušně přepisuje její virtuální funkce.

9.4 Třída Deck

Tato třída reprezentuje balíček karet. V konstruktoru vyrobí balíček s odpovídajícím počtem karet, které jsou uvedeny v Tabulce 1. Následně tyto karty zamíchá pomocí funkce *std::shuffle()* za použití Mersenne Twister Engine *std::mt19937*. [9] Tento Engine rovněž používáme pro všechna pseudonáhodná čísla v celé aplikaci. Jako seed používáme *std::random_device*. V Qt je problém, že při každém spuštění dává *std::random_device* stejnou hodnotu, takže bychom měli vždy stejné rozdání karet. Proto v Qt používáme jako seed *std::time()*, tedy aktuální čas. Přesněji počet

sekund od 1. ledna 1900. V grafické aplikaci můžeme nastavit seed, který ovlivní pouze zamíchání karet.

Pro uložení karet používáme datovou strukturu *std::vector*, což je dynamické pole. Typ položky ve vektoru je *std::unique_ptr<base_card>*, který budeme dále nazývat *card_t*. Tento typ budeme nazývat stejně i ve zdrojovém kódu pomoci *typedef*. Jedná se běžný způsob v C++, abychom dostali různé třídy (typy karet) do jednoho vektoru.

Třída *deck* má pouze jednu důležitou funkci a tou je *draw()*, která nám vrátí *card_t* na vrchní kartu v balíčku a tuto kartu z balíčku odebere. Pomocí této funkce rozdáme karty jednotlivým hráčům.

9.5 Třída Player

Třída *base_player* popisuje hráče a jedná se opět o abstraktní třídu. Dále od ní budou dědit jednotliví agenti. Hlavní součástí třídy jsou dva *std::vector<card_t>*, kde jeden obsahuje karty v ruce a druhý již zahrané karty. Třída navíc obsahuje strukturu *card_list*, kterou používáme pro vypsání našich zahraných karet ostatním hráčům. Dále zde máme *std::vector<bool>,* který označuje indexy vybraných karet. Nakonec tato třída obsahuje proměnné pro počty bodů získaných v jednotlivých sériích a také pro počet zahraných karet **Pudding**.

Nyní se zaměříme na důležité funkce. Nejdůležitější je rozhodně virtuální funkce *play()*, která vybere karty, které zahrajeme. Následně použijeme funkci *play_cards()*, která provede efekt zahrání karet. Tato funkce není virtuální a funguje pro všechny hráče stejně. Dále třída obsahuje několik funkcí, které používají agenti pro aktualizování stavu hry. Grafická verze navíc obsahuje několik dalších pomocných funkcí pro vykreslení stavu hry na GUI.

Všichni agenti jsou třídy, které dědí od třídy *base_player*. Zde musí přepsat virtuální funkci *play()*. Někteří agenti navíc přepisují některé z funkcí pro aktualizaci stavu. Třída *Random_player*, která reprezentuje náhodného agenta přepisuje funkci *play()* tak, že vybere náhodnou akci z rovnoměrného rozdělení pomocí *std::uniform_int_distribution* [9]. Třída *Hungry_player*, která reprezentuje hladového agenta přepisuje také pouze funkci *play()*. Tato třída ohodnotí všechny karty v ruce a zahraje tu, která má nejvyšší hodnotu.

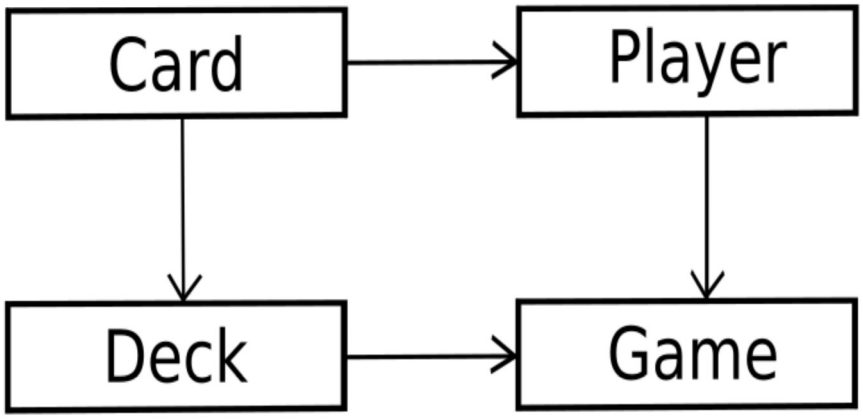
V případě grafické aplikace máme navíc třídu *AI_player*, která dědí od třídy *base_player*. Tato třída spravuje jména agentů a také přepisuje důležitou virtuální funkci *IsAI()*. Pomocí této funkce víme, zda máme vykreslit GUI pro hráče. V grafické aplikaci dědí jednotliví agenti až od třídy *AI_player*. To v konzolové verzi nemusíme řešit, protože v konzolové verzi nemůže hrát lidský hráč.

Často budeme používat *std::unique_ptr<base_player>*, což budeme dále označovat *player_t*. V konzolové aplikaci máme navíc funkci virtuální *clone()*, která vrací *player_t* na kopii aktuálního agenta. Tato funkce je důležitá, abychom mohli opakovaně spouštět stejné hry a nemuseli jsme vždy znovu parsovat vstup, protože instance třídy *std::unique_ptr* nelze kopírovat.

9.6 Třída Game

Třída *game* se stará o samotný průběh hry. Hlavní součástí této třídy je *std::vector<player_t>*, kde si držíme hráče. Dále třída obsahuje instanci třídy *deck*, pomocí které rozdáváme hráčům karty. Důležitá je také proměnná *player_index*, která nám říká, jaký hráč je aktuálně na tahu.

Jedinou veřejnou funkcí je v konzolové aplikaci funkce *play_game()*, která začne simulaci hry. Funkce *play_game()* nám vrátí *std::vector<double>* s bodovými zisky jednotlivých hráčů. Tato funkce na začátku rozdá karty a poté je to cyklus, kde vždy hráči postupně zahrají své karty a následně aktualizujeme stav hry. Po dohrání série použijeme velmi důležitou funkcí *add_points()*, která vyhodnotí konec série a přičte hráčům body podle pravidel hry. Tato funkce ještě používá pomocnou funkci *maki()*, která řeší body za karty **Maki**. Další důležitou funkcí je *swap_hands()*, která na konci kola předá karty dalším hráčům. Funkce *swap_hands()* je implementována pomocí move sémantiky C++ a funkce *std::move()*, což nám umožní pouze přesměrovat ukazatele a nemusíme celý objekt kopírovat. To bychom stejně nemohli, jelikož ruce hráčů obsahují *std::unique_ptr*. V grafické verzi opět potřebujeme další pomocné funkce pro vykreslení GUI.



Obrázek 17: Schéma závislosti hlavních tříd

Obrázek 17 ukazuje schéma závislosti hlavních tříd. Šipka ukazuje, že instance třídy na konci šipky v sobě obsahuje instanci třídy na začátku šipky. Konkrétně instance třídy *game* obsahuje instance tříd *deck* a *player*.

Grafická aplikace navíc obsahuje třídu pro každou obrazovku. Tyto třídy ovšem nejsou příliš zajímavé, takže je tu nebudeme rozepisovat. Většina z funkcionality těchto tříd je obsluha tlačítek na daných obrazovkách.

9.7 Třída Genetic_algorithm

Třída *Genetic_algortihm* obsahuje přímočarou implementaci genetického algoritmu. Třída pracuje se strukturou *individual*, která popisuje jedince. Tato struktura obsahuje *std::vector<double>* s váhami pravidel a *double* ve kterém je uložena fitness jedince. Fitness si jedinec spočítá ve svém konstruktoru pomocí funkce *evaluate()*, která odsimuluje 1000 her proti vybraným soupeřům.

Třída *Genetic_algortihm* si drží *std::vector<individual>*, který obsahuje aktuální populaci. Dále obsahuje číslo generace a několik dalších pomocných proměnných s počtem generací a způsobem počítání fitness.

Hlavní funkcí je *simulate()*, která odsimuluje celý průběh genetického algoritmu. Další důležité funkce jsou *select_individual()*, která provádí selekci, *reproduce()*, jenž provádí křížení a *mutate()*, která má na starost mutace. Funkce přímočaře implementují problematiku diskutovanou v kapitole 5 a nebudeme je více rozebírat.

9.8 Třída MCTS

Třída *MCTS* je nejrozsáhlejší a nejzajímavější třída v celém projektu. Schéma je znázorněno v Obrázku 18 Nejprve popíšeme menší pomocné třídy. Nejmenší strukturou je *MCTS_action_data*, která nám popisuje akci. Tato struktura obsahuje popis akce (zahrané karty) a také informace pro algoritmus MCTS, tedy kolikrát byla tato akce vyzkoušena a jakou má celkovou odměnu.

Další třídou je *MCTS_player*, která popisuje hráče. Tato třída obsahuje karty v ruce, zahrané karty, možné akce typu *std::vector<MCTS_action_data>* a také index vybrané akce ze seznamu. Tato třída má několik důležitých funkcí. Umí generovat možné akce pomocí funkce *smart_generate()*, která je implementována za základě kapitoly 4.5. Umíme také vybrat výslednou akci pomocí UCT (metoda se jmenuje DUCT, ovšem každý agent počítá hodnotu UCT) a EXP3. Tuto akci si následně uložíme, protože ji rovnou nevykonáváme, ale musíme nejprve najít příslušné akce pro ostatní hráče.

Třída *MCTS_node* nám popisuje uzel ve stromu, který staví algoritmus MCTS. Hlavní součástí této třídy je *std::vector<MCTS_player>*, který drží naše hráče. Navíc potřebujeme tabulku přechodů do dalších uzlů, kterou podmíníme vybranými akcemi jednotlivých hráčů. Velikost této tabulky je kartézský součin počtu akcí jednotlivých hráčů v tomto uzlu. Tato tabulka musí fungovat pro dva až pět hráčů. Mějme tři hráče, kteří mají N_1, N_2 a N_3 akcí a vyberou si akce na indexech a_1, a_2 a a_3, které indexujeme od nuly. Pak index v tabulce spočítáme následovně $index = a_1 + a_2 \cdot N_1 + a_3 \cdot N_1 \cdot N_2$. Můžeme nahlédnout, že takto spočítaná funkce f určuje bijekci $f : \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2} \times \mathbb{Z}_{N_3} \to \mathbb{Z}_{N_1 \cdot N_2 \cdot N_3}$, takže můžeme tuto tabulku reprezentovat jako jednodimenzionální vektor. Obě množiny mají zřejmě stejnou velikost. Navíc, pokud a_i není nula, pak je *i-tý* člen ostře větší, než součet všech předchozích členů. Z tohoto důvodu se nemohou dva různé výrazy zobrazit na stejné číslo. Tudíž máme prosté zobrazení mezi dvěma stejně velkými množinami, což je nutně bijekce. *MCTS_node* má několik důležitých funkcí. Funkce *table_index()* nám počítá index do tabulky zmíněné výše. Další důležitou funkcí je *evaluate()*, která ohodnotí herní stav na základě dodané heuristiky.

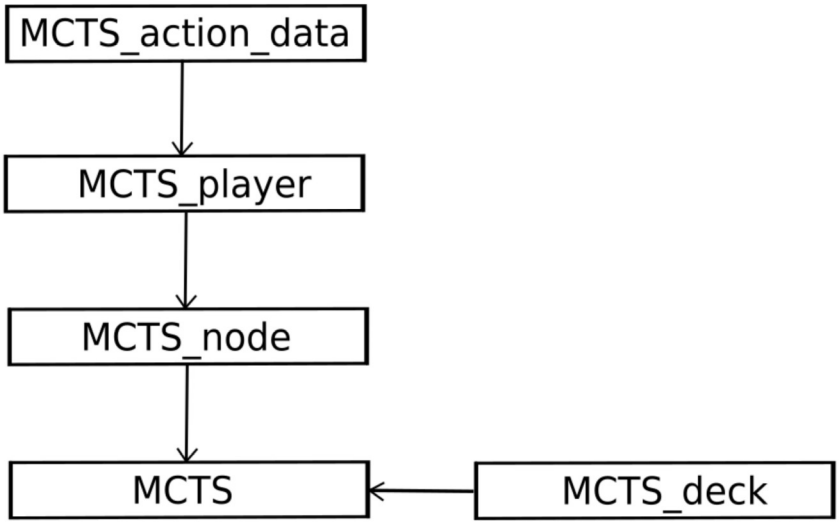
Poslední pomocnou třídou je *MCTS_deck*, která nám pomáhá s determinacemi. Ve třídě *MCTS* si ukládáme již zahrané karty a pak ze zbytku karet vytvoříme pomocný balíček. U této třídy je zajímavé, že funkce *draw()* neodebírá karty z balíčku a pouze mění index karty, kterou si máme líznout. U dalších determinací nám pak stačí balíček pouze znovu zamíchat a resetovat index. Nemusíme tedy vždy vytvářet nový balíček.

Konečně se dostáváme k hlavní třídě *MCTS*, která funguje pro DUCT i EXP3. Tato třída obsahuje *std::vector<MCTS_node>*, což jsou kořeny našich herních stromů. Dále třída obsahuje seznam hráčů, balíček, seznam zahraných karet a mnoho proměnných pro různá nastavení simulací. Obzvlášť důležitá je proměnná *round_index*, která nám určuje aktuální kolo. Tuto proměnnou používáme ve funkci *update()*, která se podívá na zahrané karty všech ostatních hráčů a také na karty v ruce všech hráčů, které jsme již viděli. Počet těchto hráčů je minimum z *round_index* a počtu hráčů. Tímto způsobem můžeme elegantně vytvořit Cheating agenta, který si pouze na začátku nastaví velkou hodnotu proměnné *round_index* a pak už funguje stejně jako běžný agent.

Tato třída obsahuje spoustu důležitých funkcí. První je funkce *determize()*. Pokud nemáme úplnou informaci o stavu hry, tak provede determinace za pomoci *MCTS_deck* a seznamu zahraných karet. Nejdůležitější funkce jsou

DUCT_best_move() a *EXP3_best_move()*. Tyto funkce zavolají paralelně funkce, které odsimulují *N* her algoritmu MCTS s tím, že se každá determinace počítá na jiném vlákně. Z tohoto důvodu je vhodné dělat jen málo determinací například tři nebo pět. Z každého kořene postavíme herní strom na základě *N* simulací. Nakonec funkce *DUCT_best_move()* a *EXP3_best_move()* ještě vyhodnotí výsledky simulací a vyberou výsledný tah. Další důležitou funkcí je *add_points()*, kterou voláme po konci série pro aktualizování bodů.

DUCT i EXP3 agenti tedy fungují velmi podobně a mají jednu společnou třídu. Liší se pouze funkcí, kterou zavoláme pro výběr tahu.



Obrázek 18: Schéma tříd MCTS

Obrázek 18 ukazuje schéma tříd v MCTS. Šipka stejně jako v Obrázku 17 značí, že instance jedné třídy v sobě obsahuje instance druhé. Tedy *MCTS* obsahuje instanci *MCTS_node* a *MCTS_deck*.

Závěr

V rámci práce jsme prozkoumali různé přístupy pro tvorbu umělé inteligence do karetní hry Sushi Go. Implementovali jsme grafickou aplikaci sloužící pro hru hráče proti jiným hráčům, ale také pro hru proti různým umělým inteligencím. Také jsme implementovali rychlou konzolovou aplikace, kterou jsme použiji k nalezení parametrů pro naše umělé inteligence a pro jejich vzájemné porovnání.

Věnovali jsme se především algoritmu MCTS, jeho rozšířením a různým zobecněním pro hru Sushi Go. Dále jsme vyzkoušeli hru na základě vážených pravidel, pro něž jsme našli váhy pomocí genetického algoritmu.

Prozkoumali jsme hru dvou hráčů. V té je hlavním úspěchem výsledek agentů na bázi MCTS a to především metody DUCT. Funkce sigmoid s různými základy se ukázala být velmi dobrým řešení bodového rozdílu a mohla by se více vyzkoušet i v jiných oblastech MCTS. Efektivitu algoritmu jsme zvýšili pomocí doménových informací o hře. Jedná se o heuristiku pro hodnocení konce série a také efektivní generátor akcí, který nám snížil větvící faktor. Zjistili jsme, že v Sushi Go není potřeba hrát smíšenou strategii a stačí hrát čistou strategii. Také jsme ukázali, že prohledávání je ve hře Sushi Go klíčové.

Vyzkoušeli jsme i hru více než dvou hráčů. Zde jsme zjistili, že pořadí hráčů je velmi důležité a bývá výhodné hrát po nejslabším soupeři. Nejlepší agent byl opět DUCT.

Naopak velkým zklamáním jsou genetičtí agenti. Při hře dvou hráčů dopadli proti všem soupeřům lépe než hladový agent, ale nemohli se rovnat s MCTS. Při hře pěti hráčů dopadl hladový agent dokonce lépe než oba genetičtí agenti.

Možnosti pokračování

V práci lze pokračovat především vývojem dalších umělých inteligencí. Vyzkoušeli jsme dvě hlavní metody MCTS pro hry se současnými tahy, ale nezkoušeli jsme SUCT a Regret Matching. Regret Matching nejspíše nebude nejlepší metodou, protože jsme zjistili, že bývá lepší hrát čistou strategii. SUCT má potenciál, ale DUCT většinou dopadá lépe. Přesto obě metody stojí za vyzkoušení. Za vyzkoušení může stát i metoda IS-MCTS, která řeší jiným způsobem částečnou informaci o hře.

Hlavní prostor je nejspíše v lepší sadě pravidel pro genetického agenta. Navíc můžeme pro přiřazení vah použít složitější metody, než je základní genetický algoritmus. Zajímavé mohou být například neuronové sítě. Prohledávání se ovšem ukázalo být velmi důležité.

Další možností pokračování je hra alespoň tří hráčů, pro kterou by se dalo udělat více experimentů. Zde mají pravidloví agenti rozhodně větší potenciál a s trochou prohledávání by se mohli přiblížit i DUCT.

Seznam použité literatury

[1] S. J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Third Edition, 2010, ISBN: 978-0-13-604259-4
[2] Spilleregler, RULES FOR SUSHI GO! [online], [cit. 20. 6. 2020], dostupné na https://www.gamesweb.dk/spilleregler/sushigo_eng.pdf
[3] Mandy J. W. Tak, Marc Lanctot, and Mark H. M. Winands, Monte Carlo Tree Search Variants for Simultaneous Move Games, 2014, doi: 10.1109/CIG.2014.6932889
[4] M. C. Fu, MONTE CARLO TREE SEARCH: A TUTORIAL, 2018, doi: 10.1109/WSC.2018.8632344
[5] M. Lanctot, V. Lisý, M. Winands, Monte Carlo Tree Search in Simultaneous Move Games with Applications to Goofspiel, 2014, doi: 10.1007/978-3-319-05428-5_3
[6] P. I. Cowling , E. J. Powley ; D. Whitehouse, Information Set Monte Carlo Tree Search, 2012, doi: 10.1109/TCIAIG.2012.2200894
[7] V. Lisý, V. Kovařík, M. Lanctot B. Bošanský, Convergence of Monte Carlo Tree Search in Simultaneous Move Games [online], [cit. 22. 6. 2020], dostupné na https://papers.nips.cc/paper/5145-convergence-of-monte-carlo-tree-search-in-simultaneous-move-games.pdf
[8] The Qt company, Qt [online], [cit. 28. 6. 2020], dostupné na https://www.qt.io/
[9] Cpp Reference, C++ reference [online], [cit. 28. 6. 2020], dostupné na https://en.cppreference.com/w/
[10] M. Lanctot, C. Wittlinger, M. Winands, N. Teuling, Monte Carlo Tree Search for Simultaneous Move Games: A Case Study in the Game of Tron. [online], [cit. 18. 7. 2020], dostupné na https://www.researchgate.net/publication/257926115_Monte_Carlo_Tree_Search_for_Simultaneous_Move_Games_A_Case_Study_in_the_Game_of_Tron
[11] Y Björnsson, H. Finnsson, CadiaPlayer: A Simulation-Based General Game Player, 2009, doi: 10.1109/TCIAIG.2009.2018702
[12] P. Auer, N. Cesa-Bianchi, Gambling in a rigged casino: The adversarial multi-armed bandit problem, 1998, doi: 10.1109/SFCS.1995.492488

Seznam tabulek

Tabulka 1: Počty jednotlivých karet v balíčku....3
Tabulka 2: Hra s dominantní strategií....6
Tabulka 3: Hra bez dominantní strategie....6
Tabulka 4: Průběh hry po zahráním karty Nigiri 3....16
Tabulka 5: Průběh hry po zahraní karty Nigiri 2....17
Tabulka 6: Vygenerované akce s chytrým generátorem....17
Tabulka 7: Výsledky hry náhodných agentů, hodnoty normalizovány na 100 her....21
Tabulka 8: Výchozí parametry pro DUCT....22
Tabulka 9: Výsledky pro c v DUCT....22
Tabulka 10: Počet simulací v DUCT....23
Tabulka 11: Počet determinací v DUCT....23
Tabulka 12: Základ funkce sigmoid v DUCT, c = 0.5....24
Tabulka 13: Základ funkce sigmoid v DUCT, c = 0.4....25
Tabulka 14: Zbývající parametry v DUCT....26
Tabulka 15: Výsledné parametry pro DUCT....26
Tabulka 16: Výsledky pro λ proti hladového agentovi....27
Tabulka 17: Výsledné ohodnocení pravidel....29
Tabulka 18: Vzájemné výsledky mezi agenty....31
Tabulka 19: Úspěšnost DUCT proti ostatním agentům....32
Tabulka 20: Výsledky DUCT proti Cheating DUCT....33
Tabulka 21: Výsledky hry tři hráčů s jedním slabým agentem. Popisek agentů značí silný agent / slabý agent....34
Tabulka 22: Hra čtyř hráčů se slabými soupeří....34
Tabulka 23: Výsledky pro hru pěti hráčů....36
Tabulka 24: Přiřazené váhy pro GAH....52
Tabulka 25: Přiřazené váhy pro GA5....52

Seznam obrázků

Obrázek 1: Fáze v Monte Carlo tree search [10].....8
Obrázek 2: Uzel v DUCT.....13
Obrázek 3: Schéma genetického algoritmu.....19
Obrázek 4: Vliv hodnoty c v DUCT.....22
Obrázek 5: Získané body pro hodnoty počty determinací v DUCT.....24
Obrázek 6: Bodový zisk v závislosti na základu funkce sigmoid pro ohodnocení, modrá odpovídá c = 0.4, červená je c = 0.5.....25
Obrázek 7: Výsledky pro λ v EXP3.....27
Obrázek 8: Výsledky pro λ proti hladového agentovi.....28
Obrázek 9: Výsledky nejlepšího agenta z generace proti hladovému agentovi.....29
Obrázek 10: Fitness proti hladovému *agentovi*, modrá křivka je maximální fitness v dané generaci a červená je průměrná fitness.....30
Obrázek 11: Vzájemné výsledky mezi agenty, modrá křivka je DUCT, zelená je EXP3, červená je GAG, fialová je GAH, oranžová je HA a hnědá je RA.....32
Obrázek 12: Obrazovka pro výběr hráčů.....37
Obrázek 13: Herní obrazovka.....38
Obrázek 14: Obrazovka pro konec kola.....39
Obrázek 15: Příklad genetického algoritmu.....40
Obrázek 16: Příklad použití konzolové aplikace pro hru agentů.....41
Obrázek 17: Schéma závislosti hlavních tříd.....44
Obrázek 18: Schéma tříd MCTS.....46

Seznam použitých zkratek

MCTS – Monte Carlo tree search, algoritmus pro hraní her 3
UCB1 – Upper confidence bound 1, řešení problému explorace a exploatace 3.5
UCT – Upper confidence bound 1 applied to trees, UCB 1 pro stromy 3.5
DUCT – Decoupled UCT, zobecnění UCT pro současné tahy 4.2.1
SUCT – Sekvenční UCT, UCT přímočaře aplikované na současné tahy [3]
EXP3 – Exponential weight algorithm for Exploration and Exploitation, stochastická metoda, která řeší současné tahy 4.2.2
SM-MCTS – Simultaneous move Monte Carlo tree search, oběcně všechny metody, které se zabývají MCTS a současnými tahy [3]
IS-MCTS – Information set Monte Carlo tree search, varianta MCTS ve které předpokládáme, že se nám soupeř snaží zatajit informace [6]
GA – genetický algoritmus 5.1

GAG – genetický agent trénovaný pro genetickým agentům, náš vlastní agent 6.3
GAH – genetický agent trénovaný pro hladovému agentovi, náš vlastní agent 6.3
GA5 – genetický agent trénovaný pro hru pěti hráčů, náš vlastní agent 6.3
HA – hladový agent, náš vlastní agent 2.5
RA – náhodný agent, náš vlastní agent 2.5

Přílohy

Předpoklad	Akce	Váha
Nezahráno **Wasabi**, 7+ karet v ruce	Wasabi	6,23
---	Nigiri 3	80,41
Nemáme **Nigiri 3**	Nigiri 2	88,24
Zahrané **Wasabi**	Nigiri 3	57,35
Zahrané **Wasabi**, nemáme **Nigiri 3**	Nigiri 2	71,75
Zahrané **Wasabi**, nemáme **Nigiri 3** a **Nigiri 2**	Nigiri 1	11,15
Zahráno sudý počet **Tempura**	Tempura	67.50
Zahráno lichý počet **Tempura**	Tempura	0,51
3+ **Tempura** v ruce	Tempura	24.04
součet **Dumplings** v ruce a zahrané >= 3	Dumplings	91.90
součet **Sashimi** v ruce a zahrané >= 3	Sashimi	71.53
| max Pudding dif | <= 1	Pudding	0.00
| min Pudding dif | <= 1	Pudding	0.00
| max Maki diff | < 3	Maki 3	0.65
| max Maki diff | < 3, nemáme **Maki 3**	Maki 2	0.00
| max Maki diff | < 3, nemáme **Maki 3** a **Maki 2**	Maki 1	28.06

Tabulka 24: Přiřazené váhy pro GAH

Předpoklad	Akce	Váha
Nezahráno **Wasabi**, 7+ karet v ruce	Wasabi	23.02
---	Nigiri 3	15.44
Nemáme **Nigiri 3**	Nigiri 2	0.33
Zahrané **Wasabi**	Nigiri 3	37.73
Zahrané **Wasabi**, nemáme **Nigiri 3**	Nigiri 2	44.49
Zahrané **Wasabi**, nemáme **Nigiri 3** a **Nigiri 2**	Nigiri 1	0.00
Zahráno sudý počet **Tempura**	Tempura	23.49
Zahráno lichý počet **Tempura**	Tempura	0.00
3+ **Tempura** v ruce	Tempura	57.02
součet **Dumplings** v ruce a zahrané >= 3	Dumplings	10.08
součet **Sashimi** v ruce a zahrané >= 3	Sashimi	95.46
| max Pudding dif | <= 1	Pudding	87.81
| min Pudding dif | <= 1	Pudding	85.26
| max Maki diff | < 3	Maki 3	11.98
| max Maki diff | < 3, nemáme **Maki 3**	Maki 2	0.58
| max Maki diff | < 3, nemáme **Maki 3** a **Maki 2**	Maki 1	50.25

Tabulka 25: Přiřazené váhy pro GA5