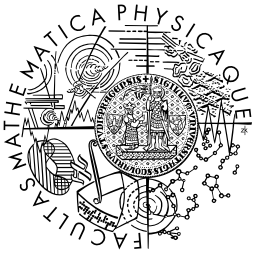


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Lucie Ambrožová

Umělá inteligence pro hru Hanabi

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Martin Pilát, Ph.D.

Studijní program: Informatika

Studijní obor: Informatika se specializací Umělá inteligence

Praha 2023

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

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 autora

V první řadě bych chtěla poděkovat vedoucímu mé práce Martinu Pilátovi za trpělivost, se kterou řešil všechny moje problémy a odpovídal na hloupé dotazy. Dále bych chtěla poděkovat mému snoubenci Jirkovi za podporu a podnětné připomínky k textu.

Název práce: Umělá inteligence pro hru Hanabi

Autor: Lucie Ambrožová

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Martin Pilát, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Hra Hanabi je v současnosti velice populární pro vývoj nejrůznějších umělých inteligencí. Tato popularita spočívá především v tom, že Hanabi je stochastické, možné hrát ve více hráčích, a především kooperativní. V literatuře je možné dohledat mnoho přístupů k vývoji umělých inteligencí pro tuto hru, které jsou založeny na především na ručně psaných algoritmech, bayesovském přístupu a hlubokém Q-učení.

V rámci této práce na tento vývoj navazujme zabýváme se vývojem ručně psaných inteligencí, ručně psaných umělých inteligencí s využitím evolučních algoritmů a umělé inteligence založené na hlubokém Q-učení.

Na výsledcích z pěti ručně psaných umělých inteligencí ukazujeme, jak přidávání jednotlivých vylepšení agentů má vliv na výsledné skóre. Dále nejlepší ručně psanou umělou inteligenci ještě vylepšujeme tím, že její parametry optimalizujeme pomocí evolučního algoritmu. Tato výsledná umělá inteligence dosahuje nejlepších výsledků prezentovaných v literatuře pro hru 5 hráčů pro ručně psané umělé inteligence bez využití hat guessing strategií. Umělou inteligenci založenou na hlubokém Q-učení jsme nakonec z výpočetních důvodů a kvůli problémům s memory leakem zkoušeli jen na výrazně zmenšené a zjednodušení verzi Hanabi, kde se nám ji však úspěšně během 5000 her povedlo natrénovat tak, že dosahovala nad polovinu možných bodů.

Klíčová slova: Hanabi Umělá inteligence Evoluční algoritmus Q-učení

Title: Artificial Intelligence for the Hanabi Game

Author: Lucie Ambrožová

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Martin Pilát, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: The Hanabi game is currently very popular for the development of various Artificial Intelligences (AI). This popularity is mainly based on the fact that Hanabi is stochastic, possible to play in multiplayer, and mainly cooperative. In the literature, many approaches on how to develop AI for this game can be found. These approaches are mainly based on hand-written algorithms, Bayesian approaches, and deep Q learning.

In this thesis, we continue this research looking at the development of handwritten AI, handwritten AI using evolutionary algorithms, and AI based on deep Q-learning.

Using results from five handwritten Artificial Intelligences, we show how adding individual agent enhancements affects the final score. We further improve the best handwritten AI by optimising its parameters using an evolutionary algorithm. This resulting AI achieves the best results presented in the literature for a 5-player game for hand-written AI without the use of hat guessing strategies. The deep-Q learning-based AI was tested only on a significantly scaled-down and simplified version of Hanabi due to computational power and memory leak issues, but on this simplified version we successfully trained it to score above half of the possible points over 5000 games.

Keywords: Hanabi Artificial Intelligence Evolutionary Algorithm Q-Learning

Obsah

Úvod

Druhá polovina 20. století zaznamenala prudký rozvoj počítačové techniky, která postupně umožňovala provádět náročnější a náročnější výpočty a začala se uplatňovat téměř ve všech odvětvích. Už od samého počátku sloužily nejrůznější hry k testování toho, jak mohou být počítače využity k složitým rozhodovacím úlohám. Nejdéle a nejvíce studovanou hrou byly šachy a vyvinout počítač, který dokáže porazit člověka, znamenalo značnou dávku prestiže.

Vývojem počítačů a algoritmů pro hodnocení šachů se tedy zabývaly i velké technologické firmy. První velký úspěch přišel v roce 1997, kdy šachový superpočítač Deep Blue od firmy IBM porazil v zápase na šest partií tehdejšího mistra světa v šachu Garriho Kasparova (Yao, 2022).

Nicméně v této době byl úspěch Deep Blue založen spíše na hrubé výpočetní síle než na sofistikovaných hodnotících algoritmech (Proai, 2020). V následujících letech rychlý rozvoj jak na poli techniky, tak hodnotících algoritmů zapříčinil, že lidé ztratili s počítači velice rychle krok. Mimo šachů byly zpočátku zkoumané i jiné hry pro dva hráče s kompletní informací, jako Go nebo Shogi (Granter a kol., 2017), (Matsubara a kol., 1996).

Časem se začaly zkoumat i stochastické hry, například Vrhcáby a Poker pro dva hráče (Tesauro a Sejnowski, 1989) (Brown a Sandholm, 2018).

Postupně se vývoj začal zabývat i hrami jako Diplomacie, která je specifická tím, že spolu hráči musí komunikovat, smlouvat a domlouvat spojenectví (FAIR a kol., 2022).

Další významný milník ve vývoji umělých inteligencí pro hry byl spojen rozvojem zpětnovazebního a hlubokého učení. První velký úspěch, který vešel v povědomí i široké veřejnosti byl tentokrát ve hře Go v roce 2016. Byl vyvinut program AlphaGo, který dokázal porazit jednoho z nejlepších hráčů Go. Až do této doby byla tato hra významná tím, že v ní nejlepší lidští hráči poráželi počítače, jelikož v ní může existovat tolik stavů, že to počítače nedokázaly upočítat. Později vyvinuté AlphaGo Zero předčilo i AlphaGo. Díky snaze vytvořit program, který se nebude specializovat pouze na jednu hru, bylo v roce 2017 z AlphaGo Zero vyvinuto AlphaZero, která umí hrát Go, šachy a shogi. AlphaZero v roce 2018 dokázalo například suverénním způsobem porazit nejlepší šachový program té doby (Pete, 2019).

V posledních letech začala být intenzivně zkoumána hra Hanabi. Hra je zajímavá tím, že je stochastická, možné hrát ve více hráčích, a především kooperativní. Hlavně kooperativnost přináší nové výzvy pro vývoj umělých inteligencí. Vznikají dva typy výzev, a to, aby umělá inteligence byla schopná hrát sama se sebou, a aby byla schopná hrát s jinými umělými inteligencemi, případně lidskými hráči (Bard a kol., 2020).

V teoretické části této práce zanalyzujeme hru Hanabi z pohledu teorie her. Rozebereme, jak hrají hru lidští hráči a které strategie používají. Dále se zaměříme na dosavadní vývoj umělých inteligencí pro hru Hanabi. V praktické části rozebereme implementaci hry Hanabi v jazyce C# a implementaci samotných umělých inteligencí založených na ručně psaném algoritmu, případně zpětnovazebním učení. Jednotlivé umělé inteligence budou navzájem porovnány a získané výsledky budou také diskutovány v souvislostech s ostatní odbornou literaturou.

Hlavním přínosem této práce je postupný vývoj ručně psaných umělých inteligencí od nejjednodušších po nejkomplexnější a analýza toho, jaký vliv mají jednotlivá vylepšení na jeho výsledky.

1. Popis hry

V následující kapitole je popsána hra Hanabi a její mechaniky. Oficiální pravidla ke hře v českém jazyce jsou k přečtení na oficiálních stránkách (Bauza, 2010).

Názvosloví

Níže jsou nadefinovány pojmy, které budeme v textu dále využívat.

- Pozitivní nápověda – Nápověda, která přímo o kartě říká její hodnotu nebo barvu
- Negativní nápověda – Nápověda, která se přímo netýká dané karty. Tedy víme, že karta nemá hodnotu nebo barvu, která byla napovězena
- Barevná nápověda – Nápověda napovídající na barvu
- Číselná nápověda – Nápověda napovídající na číslo
- Zahodit kartu – Odložit kartu do odkládacího balíčku
- Vyložit kartu – Pokusit se přidat kartu k nějaké z postupek na stole
- Zahrát kartu – Zahodit nebo vyložit kartu
- Vybuchnout – Třikrát nesprávně vyložit kartu
- Unikátní karta – Karta, která už se ve hře vyskytuje pouze jedenkrát
- Nová karta – Poslední karta, kterou si hráč dobral z dobíracího balíčku
- Zbytečná karta – Karta, která už nemůže být využita k doplnění jakékoliv postupky
- Tah – Zahraná akce jednoho hráče

Základ hry

Hanabi je kooperativní hra, tedy hra, při které hráči spolupracují a snaží se společně dosáhnout co nejlepšího výsledku. Hráči se snaží vytvořit postupky z karet, přičemž za každou kartu v postupce získají bod. Hlavní myšlenkou hry je, že hráč nevidí na své karty, pouze na karty svých spoluhráčů. Hráči si proto dávají nápovědy, aby si navzájem naznačili, kdy se které karty mají hrát.

Karty

Ve hře je celkem 50 karet, se kterými hráči hrají. Každá karta má barvu a číslo. Od každé z pěti barev jsou tři karty s hodnotou *jedna*, dvě karty s hodnotu *dvě*, dvě karty s hodnotou *tři*, dvě karty s hodnotou *čtyři* a jedna karta s hodnotou *pět*.

Tah hráče

Pokud je hráč na tahu, rozhoduje se ze tří možností, co udělá.

1. Vyložení karty: Hráč jednu ze svých karet vyloží na stůl. Pokud se karta hodí do jakékoli postupky na stole (nebo je to karta s hodnotou *jedna* od barvy, která ještě nebyla vyložena), hráč kartu úspěšně zahrál a doplní ji do postupky. Pokud karta nemůže být dle výše popsaných pravidel úspěšně vyložena, karta se odloží do odhazovacího balíčku a hráči jsou potrestáni otočením žetonu bomby, viz sekce žetony.

2. Zahození karty: Hráč jednu ze svých karet zahodí do odhazovacího balíčku. Za to se obnoví jeden žeton nápovědy, viz sekce žetony.

3. Napovězení karty: Hráč kterémukoli hráči může říct informaci o jeho kartách, viz následující sekce Nápovědy.

Nápovědy

Pokud je hráč na tahu a má k dispozici alespoň jeden s osmi žetonů nápovědy, může se rozhodnout napovědět jinému hráči. V takovém případě si vybere kteréhokoliv z hráčů, kterému bude napovídat. Hráč si může vybrat mezi dvěma typy nápověd, kterou chce dát. Nápověda týkající se barvy: Hráč si vybere jednu barvu, a poté označí všechny karty od této barvy, které má hráč na ruce. Pokud hráč žádnou kartu od některé barvy nemá, tato barva mu nesmí být napovězena. Nápověda týkající se hodnoty: Hráč si vybere jednu hodnotu, a poté označí všechny karty od této hodnoty, které má hráč na ruce. Pokud hráč žádnou kartu od některé hodnoty nemá, tato hodnota mu nesmí být napovězena. Tedy pokud má například hráč na ruce pět karet, z nichž všechny mají hodnotu jedna a pouze prostřední karta je žlutá, nápovědy by mohly být například: "Všechny karty na tvé ruce jsou jedničky", nebo "Prostřední karta je žlutá". Nemohlo by však být řečeno "Žádná z tvých karet není dvojka", neboť dvojku hráč nemá na ruce.

Žetony

Ve hře jsou dva typy žetonů: žetony nápověd a žetony bomb. Žetonů nápověd je dohromady osm, žetony bomb jsou tři. Jeden z žetonů nápovědy se použije pokaždé, kdy některý z hráčů dá nápovědu. Pokud hráči již žádné žetony nemají, nemohou napovídat. Na začátku hry mají hráči k dispozici všech osm nápověd. Žetony nápovědy se dají obnovit dvěma způsoby: zahozením karty a poskládáním celé postupky od barvy. Žeton bomby se naopak aktivuje pokaždé, když hráč vyložil nesprávnou kartu. Pokud jsou aktivované všechny tři žetony bomby, hráči okamžitě prohrávají a získají 0 bodů. Pokud se žeton bomby jednou aktivoval, nejde jej žádným způsobem deaktivovat.

Konec hry

Existují dva způsoby, jak ukončit hru. První způsob, tedy kdy hráči vybuchnou, byl popsán výše. Druhá možnost nastane v momentu, kdy se dobere poslední karta z dobíracího balíčku. V tu chvíli začíná poslední kolo hry. Tedy všichni, včetně hráče co si právě dobral poslední kartu, odehrají svůj poslední tah. Poté

hra končí a hráči si spočítají body. Pokud hráči vybuchnou v průběhu posledního kola, získávají stále 0 bodů. Pokud hráči nevybouchli, spočítá se počet úspěšně vyložených karet do postupek, který odpovídá výslednému bodovému zisku hráčů. Maximální počet bodů, které je možno získat, je tedy 25 (postupky od 1 do 5, pro všech pět barev).

Nicméně ve většině výzkumů umělé inteligence na hru Hanabi je počítání mírně odlišné. V případě výbuchu otočením třetího žetonu bomby hra sice končí, ale výsledné skóre není vyhodnoceno jako 0, ale dle počtu vyložených karet do postupek (Bard a kol., 2020).

Tento způsob počítání tedy bude využit i v této práci, jelikož nám usnadní porovnání našich výsledků s výsledky publikovanými v literatuře.

Rozšíření hry

V základní hře hráči hrají pouze s pěti barvami. Existuje rozšíření, které do balíčku přidává přidává ještě *duhovou* barvu. V závislosti na různých vydáních hry tato nová barva obsahuje buď standardní počet karet (3, 2, 2, 2, 1) nebo obsahuje od každé hodnoty pouze jednu kartu. V této podobě s pouze jednou kartou je i české vydání hry Hanabi. V rámci tohoto rozšíření mají hráči na výběr mezi dvěma variantami hry. V první variantě budou napovídat na duhové karty přímo (tedy "tvá prostřední karta je duhová"). Ve druhé variantě se počítá, že duhová barva zahrnuje i všechny ostatní barvy a nejde na ni napovídat samostatně. Tedy jakákoliv barevná nápověda zahrnuje i všechny duhové karty. V roce 2015 přišlo rozšíření Zvláštní cena poroty, díky kterému mohou hráči po vyložení kompletního ohňostroje získat i jiné odměny než žeton nápovědy. V roce 2020 vzniklo rozšíření Black powder, které přináší do balíčku další barvu, tentokrát *černou*, na kterou se nedávají žádné barevné nápovědy. Navíc je unikátní tím, že se staví *naopak*, tedy začíná se pětkou a končí jedničkou. Počty karet jsou také otočené, tedy pětky jsou tři a jednička jedna.

2. Analýza hry

2.1 Dělení her

V této sekci popíšeme dělení her podle nejrůznějších aspektů a jednotlivé kategorie her v krátkosti popíšeme. V závěru klasifikujeme hru Hanabi dle těchto kategorií.

Kooperativní a nekooperativní hry

V kooperativních hrách spolu agenti kooperují v rámci předem daných koalic, kdy se pomocí vzájemných interakcí snaží vytvořit a uchovat hodnotu. Tyto koalice pak mohou soupeřit s jinými koalicemi. V případech, kdy se agent snaží maximalizovat svůj zisk, jde o nekooperativní hry (Chatain, 2016).

Symetrické a asymetrické hry

Symetrické hry jsou takové, kde odměna za strategii nebo tah závisí pouze na dané strategii a ne na tom, který hráč ji zahrál. Všichni hráči mají stejnou množinu strategií, ze které mohou vybírat. V opačném případě je hra asymetrická (Cheng a kol., 2004).

Simultánní a sekvenční hry

Simultánní hry jsou hry, při kterých všichni agenti hrají své tahy naráz nebo kdy neví, co zahráli ostatní agenti. Tedy agent dělá rozhodnutí bez toho, aby znal strategie ostatních agentů.

V sekvenčních hrách se hráči v tazích střídají a vědí, jaké rozhodnutí udělali agenti hrající před nimi, tedy jim mohou svoji strategii přizpůsobit (Davis a Brams, 2023).

Hry s nulovým a nenulovým součtem

Ve hrách s nulovým součtem platí, že zisk hráče se rovná součtu ztrát ostatních hráčů. Hry s nenulovým součtem jsou naopak hry, kdy součet zisků a ztrát všech hráčů nemusí být nula (Davis a Brams, 2023).

Hry s perfektní a neperfektní informací

Říkáme, že hra má perfektní informaci, pokud každý hráč v momentu, kdy dělá jakékoliv rozhodnutí, přesně zná všechny události, které se do té doby staly. To zahrnuje i *inicializační události* (například jak je zamíchán balíček karet). Aby byla hra s perfektní informací, hráč nemusí mít informaci o funkcích odměn ostatních hráčů, případně nemusí znát celou strukturu hry.

Pokud hráč nezná kompletní stav hry, hra má neperfektní informaci (Russell a Norvig, 2010).

Hry s kompletní a nekompletní informací

Ve hře s kompletní informací jsou každému hráči známy funkce odměn ostatních hráčů a struktura hry. Hráč však nemusí znát všechny tahy ostatních hráčů (například počáteční rozestavení ve hře Lodě).

Ve hře s nekompletní informací hráči neznají všechny informace o ostatních hráčích (Russell a Norvig, 2010).

Stochastické hry

Ve stochastických hrách postupuje hra ze stavu do jiného náhodného stavu, jehož distribuce závisí na předchozím stavu hry a na akci, která se v něm stala (Russell a Norvig, 2010).

Konečné a nekonečné hry

Konečná hra je hra, která zaručeně skončí po konečném počtu tahů. Konečné hry mohou mít nekonečně možností (Davis a Brams, 2023).

2.2 Analýza Hanabi

Dle výše popsaných kategorií může být hra Hanabi charakterizována následovně:

- Kooperační hra, neboť hráči tvoří jednu koalici, jejíž cílem se společně dosáhnout co nejlepšího výsledku.
- Symetrická hra, všichni hráči můžou hrát stejné tahy (zahodit nebo vyložit kartu, napovědět). Odměna je stejná nehledě na to, který hráč tah zahraje.
- Sekvenční hra, tedy v jednu chvíli hraje pouze jeden hráč. Zároveň tento hráč ví, jaké akce udělali hráči před ním.
- Hra s nenulovým součtem, například pokud hráč vyloží správně kartu, zvýší tím zisk sobě i ostatním hráčům.
- Hra s neperfektní informací. Hráči například nemusí vědět, jaké karty mají v ruce, případně jak je zamíchán dobírací balíček.
- Hra s kompletní informací, jelikož je každému hráči známa struktura hry i cíle ostatních hráčů.
- Stochastická hra, jelikož dobírací balíček karet je náhodně zamíchán, tedy pokud si z něj hráč dobere kartu, je nový stav hry náhodný.
- Konečná hra, protože hra končí jedno kolo po dobrání balíčku. Hráč si dobírá kartu pokaždé, když zahraje některou kartu ze své ruky. Jelikož je ve hře pouze omezený počet nápověd, hráči po chvíli začnou být nuceni hrát karty z ruky. Tedy hra musí skončit.

Zároveň je zajímavé zmínit, že i v případě, kdy všichni hráči vědí, jaké mají karty na ruce, je hra NP-úplná. (Baffier a kol., 2016)

3. Strategie ve hře Hanabi

V následující kapitole popíšeme, jaké strategie se uplatňují při hře Hanabi. Základní popis strategií vychází převážně z rozsáhlého dokumentu Hanabi reference for bga (2023), který se strategiemi pro hru Hanabi zabývá. Informace obsažené v tomto dokumentu jsou navíc v dobré shodě s tím, co jsme zjistili během našich vlastních her.

3.1 Počet nápověd

Ve hře Hanabi jsou nápovědy a jejich efektivní využití hlavním prvkem, který rozhoduje o výsledném skóre. Před samotnou hrou tedy musíme vědět, kolik nápověd je možné za celou hru dát. Počítáme s nejlepším možným výsledkem, tedy že se nám podaří správně vyložit všech 25 karet. Na začátku hry máme k dispozici 8 nápověd. Dále získáme nápovědu za každý hotový ohňostroj. Jelikož však dostavením posledního ohňostroje končí hra, nápovědu z něj získanou nikdy nevyužijeme. Tedy počítejme pouze další 4 nápovědy za hotové ohňostroje. Nakonec ještě nápovědy získáme ze zahozených karet. Počet takto získaných nápověd se liší v závislosti na počtu hráčů ve hře. Počet nápověd pro různé počty hráčů je shrnut v Tabulce 3.1. Jinými slovy, tabulka také uvádí, jaké je maximální možné množství zahozených karet, pokud ve hře chceme dosáhnout maximálního skóre.

počet hráčů	2	3	4	5
karty v ruce hráče	5	5	4	4
nevyužité karty	8	12	12	15
zahozené karty	17	13	13	10
celkový počet nápověd	29	25	25	22
nápovědy na kartu	1,16	1	1	0,88

Tabulka 3.1: Maximální počet nápověd.

Jak je vidět z tabulky, nejvíc nápověd máme k dispozici při hře dvou hráčů, naopak nejméně při hře pěti hráčů. Obecně ale vidíme, že na každou vyloženou kartu můžeme spotřebovat zhruba jednu nápovědu. Je tedy zřejmé, že s nápovědami je potřeba šetřit a pokusit se předat co nejvíce informací pomocí jedné nápovědy. Také vidíme, že hlavně ve vyšším počtu hráčů jsme velmi limitováni maximálním množstvím karet, které můžeme zahodit.

3.2 Nástin strategií

Ve výše zmíněném dokumentu je rozebráno, že naprostá většina nápověd by měla mít jeden ze tří následujících cílů:

1. Napovědět na kartu, která se dá vyložit.
2. Napovědět na kartu, kterou je potřeba si ponechat a vyložit později.
3. Opravit chybný předpoklad.

Dávat například nápovědu pouze proto, abychom označili kartu vhodnou k zahození, není výhodné, jelikož jsme využili nápovědu pouze k získání další nápovědy a celkové množství daných nápověd je omezené.

Spolu s nápovědami se přímo pojí pořadí (stáří) karet na ruce. Jak je popsáno v sekci Nápovědy (1), nápověda musí zahrnovat všechny karty, které splňují napovězenou vlastnost. Je jasné, že dostaneme nápovědu na více karet zároveň, relevance pro právě dobranou kartu bude výrazně vyšší než pro kartu, kterou máme na ruce více kol. Pokud by tato karta byla relevantní, pravděpodobně by na ni přišla nápověda už dříve.

S tím souvisí i strategie ohledně zahazování karet. Jak bylo popsáno výše, nápovědy by měly směřovat na důležité karty. Tedy pokud o nějaké kartě, kterou už máme na ruce více kol, nemáme žádnou informaci, je velmi pravděpodobné, že je zbytečná a je vhodným kandidátem k zahození.

Jak bylo popsáno výše, pomocí každé nápovědy se snažíme předat co nejvíce informací najednou. Za tímto účelem bylo vytvořeno několik strategií, které se tento princip snaží optimalizovat například tím, že nápověda dává přímou informaci nejen hráči, kterému je určena, ale i některému dalšímu. V krátkosti nastiňme strategii finesse vysvětlenou v dokumentu Hanabi reference for bga (2023).

Na stole je vyložená *červená 1*. Hráč A má na ruce karty *x-x-x-červená 2* a tato karta *červená 2* je pro všechny hráče známá jako 2. Hráč B má na ruce karty *x-x-červená 3-x* a o kartě *červená 3* není nic známo. Hráč C napoví hráči B nápovědu *červená*. Hráč A ví, že hrát *červená 3* není možné, protože chybí *červená 2*, zároveň ale ví, že dle nápovědy by se hráč B měl domnívat, že tato karta je hratelná. Aby zabránil této chybě, dojde mu, že musí vyložit vlastní kartu, o které ví, že má hodnotu 2. Hráč B tedy úspěšně vyloží kartu *červená 2*, hráč C kartu *červená 3* a pomocí jedné nápovědy byly vyloženy dvě karty.

3.3 Jiné strategie

Vyvinuly se i strategie, které využívají toho, že nápovědy mohou nést i jinou informaci, pokud se na tom hráči předem domluví. Jedním z příkladů je strategie *hat guessing*, která je považována za nejsilnější způsob, jak hru Hanabi hrát. Při ní hráči kódují informace, které chtějí ostatním předat do nápověd za pomocí modulární aritmetiky (Cox a kol., 2015).

Pro ukázku uveďmě následující příklad ze hry čtyř hráčů volně přeložený od Zamiell (2023):

Alice hraje první. Bob má na první pozici v ruce modrou 1, Cathy má na první pozici v ruce červenou 1 a Donald má na první pozici v ruce zelenou 1. Alice tedy chce, aby všichni tři její spoluhráči hráli kartu z první pozice. Z předchozí domluvy víme, že "hrát kartu z první pozice"má hodnotu 1. Alice sečte všechny akce: 1 + 1 + 1 = 3. Alice tedy potřebuje týmu sdělit "3", a proto dle domluveného kódování napoví červenou barvou Bobovi. Na řadu přichází Bob a musí zjistit, který tah mu Alice naplánovala. Bob se podívá a zjistí, že Cathy bude přiřazena hodnota 1 a Donaldovi bude také přiřazena hodnota 1. Bob vypočítá: (Alicina nápověda) - 1 (budoucí akce Cathy) - 1 (budoucí akce Donalda) = 1 (zahrej kartu na první pozici). Bob tuto kartu zahraje. Cathy provede podobnou analýzu jako Bob, ale je to pro ni o něco jednodušší. Místo toho, aby musela zjistit akce dvou budoucích hráčů (Cathy + Donald), musí pro svůj výpočet zjistit pouze

akci jednoho hráče (Donalda). Cathy spočítá: (Alicina nápověda) - 1 (Bobova minulá akce) - 1 (Donaldova budoucí akce) = 1 (zahrej kartu na první pozici). Cathy zahraje kartu. Donald to má ze všech nejjednodušší, stačí, když si dosadí hodnoty toho, co udělali oba předchozí hráči, aby zjistil, že má též hrát kartu z první pozice. Donald zahraje kartu z této pozice.

4. Principy evoluce a zpětnovazebního učení

Je mnoho přístupů k tomu, jak vytvořit umělého hráče. Dříve hojně používané algoritmy byly takové, které se spoléhaly na co nejhlubší propočet variant. Tímto přístupem sice v některých hrách dokázaly být lepší jako člověk, například v šachách, ale ve větších hrách, například v Go, byl člověk stále lepší. Zároveň byl problém v tom, že způsob, jakým AI hodnotila pozice, byl napsán člověkem, proto špatně hledala nové přístupy ke hře. S rozvojem nových technik a zvětšováním výpočetního výkonu už stroje dokázali nejen porážet člověka ve většině her, ale i přinášet nové pohledy na hru, pro člověka často nepřirozené. Nyní se podíváme na to, jaké techniky lidé používají pro vývoj algoritmů pro hru Hanabi.

4.1 Evoluční algoritmy

Následující sekce vznikla pomocí textů Pilát (2023b) a Hansen a kol. (2015). Evoluční strategie (ES) jsou inspirovány principy biologické evoluce. Stejně jako v ní pracujeme se skupinou jedinců, tzv. populací P. Každý jedinec reprezentuje jedno možné řešení problému, který se snažíme pomocí ES řešit. Každému jedinci je zároveň přiřazena fitness hodnota podle toho, jak dobrý jedinec (řešení) je.

4.1.1 Průběh evoluce

Evoluční algoritmus běží v cyklech, tzv. generacích. Každý cyklus se skládá z několika částí, které si teď blíže popíšeme.

Selekce

Nejdřív z populace potřebujeme vybrat jedince, ze kterých budou vytvořeni potomci, kterým budeme říkat rodiče. Jedince vybíráme v závislosti na jejich hodnotě fitness. Jako příklad uveďme ruletovou selekci, kde šance p_i výběru jedince i se spočítá jako $p_i = \frac{f_t}{\sum_{j=1}^{N} f_j}$, kde f_j je fitness jedince j. Jeden jedinec může být vybrán jako rodič i vícekrát (nebo taky vůbec).

Křížení

V tomto kroku jsou z rodičů jejich rekombinací vytvořeni potomci. Jako příklad můžeme uvést jednobodové křížení: Náhodně se vybere jeden bod v jedinci a nový potomek vznikne tak, že se překopíruje část z jednoho rodiče před tímto bodem a z druhého od tohoto bodu dále.

Mutace

Potomci poté mají určitou šanci, že budou mutovat. To může například znamenat, že se některé bity v jednici s určitou pravděpodobností změní. Díky tomu je populace různorodější a tím pádem může lépe prohledávat možná řešení.

Přenos jedinců do další generace

Nakonec ještě musíme vybrat jedince, ze kterých vytvoříme další generaci. Například můžeme vzít pouze potomky, nebo nejlepší jedince z rodičů i potomků dohromady. Další možností může být vygenerovat více potomků, než je potřeba rodičů, a vybrat pouze nejlepší z nich.

4.2 Zpětnovazební učení

Zpětnovazební učení je technika, kdy se agent učí dosáhnout cíle pomocí interakcí s prostředím. Interagují spolu tím, že agent provede akci a prostředí na tuto akci reaguje tím, že agentovi řekne, jak vypadá stav, do kterého se dostal. Kromě toho mu ještě dá odměnu podle toho, jak dobrá byla jeho akce. Agentův cíl je naučit se chovat tak, aby maximalizoval odměnu, kterou získá. Tato sekce vznikla z textů Sutton a Barto (2018), Pilát (2023c), Pilát (2023a), Bhatt (2019) a Bajaj (2023).

Markovské rozhodovací procesy

Každé prostředí můžeme formálně popsat jako markovský rozhodovací proces (MDP). MDP je čtveřice (S, A, P, R), kde S a A jsou konečné množiny stavů a akcí, $P_a(s, s')$ je přechodová funkce a $R_a(s, s')$ je funkce odměn. Agentův výběr akce v závislosti na stavu se nazývá strategie π. Cílem je najít takovou strategii, která maximalizuje celkovou odměnu $\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1})$, kde $a_t = \pi(s_t)$ je akce provedená agentem v kroku t a $\gamma < 1$ je diskontní faktor, který zajišťuje, že suma je konečná.

Hodnota stavu a hodnota akce

Hodnota $V^\pi(s)$ stavu s při použití strategie π si zadefinujeme jako $V^\pi(s) = \mathrm{E}[R] = \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s]$, kde R značí celkovou získanou diskontovanou odměnu a r_t značí odměnu obdrženou v čase t. Kromě hodnoty stavu je často dobré uvažovat také hodnotu $Q^\pi(s, a)$ akce a provedené ve stavu s pokud budeme dále sledovat strategii π.

Strategie prohledávání

Agent má za úkol maximalizovat svůj celkový zisk R a musí se naučit vhodnou strategii, která ovlivňuje hodnoty stavů. Pokud agent vybírá vždy nejlepší akci, může se mu stát, že některé stavy nikdy nenavštíví, ačkoliv by mu mohly přinést větší zisk. Problém spočívá v tom, že agent nemá předem informace o hodnotě jednotlivých stavů a musí se je učit postupně. Proto je důležité vyvážit prohledávání prostoru a využívání naučených znalostí. ϵ-hladová strategie je jednou z populárních metod, která umožňuje kombinovat využívání znalostí a prohledávání nových stavů. S určitou pravděpodobností vybere nejlepší akci, ale také se občas rozhodne pro náhodnou akci, aby mohl objevovat nové stavy.

Temporal-Difference metody

Temporal-Difference metody (TD) jsou přístupem k zpětnovazebnímu učení, který je založen na Bellmanových rovnicích. Ty říkají, že hodnota funkce $V^{\pi}(s)$ v daném stavu s za předpokladu, že se používá strategie π, je rovna očekávané hodnotě celkové odměny, kterou agent získá, pokud zahájí hru v tomto stavu a bude dodržovat strategii π.

V TD metodách se hodnota funkce aktualizuje po každém kroku agenta, na základě získané odměny a informace o novém stavu. Konkrétně, aktualizace hodnotové funkce se provádí jako $V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$, kde $r + \gamma V(s')$ je nová cílová hodnota pro $V(s)$, α je parametr učení a γ je diskontní faktor. TD metody mají výhodu oproti Monte Carlo metodám v tom, že aktualizace hodnoty se týká více stavů najednou.

TD metody lze také chápat jako propagaci informace o odměnách z cílových stavů do předchozích stavů. Navíc řeší problém rozdělení odměny v čase, kdy agent dostává odměnu až při dosažení cíle. TD metody tuto odměnu postupně rozdělí mezi akce, které vedly k dosažení cíle.

Q-učení

Konkrétní algoritmus založený na TD metodách je Q-učení, který se přímo učí funkci $Q(s,a)$ hodnot, kde s je stav a a je akce. V tradičních případech je Q reprezentována jako matice, na začátku inicializovaná samými nulami. V každém kroku agent pozoruje aktuální stav, provede akci, dostane odměnu a pozoruje nový stav. Poté se aktualizuje hodnota funkce Q podle vzorce $Q^{new}(s_t,a_t) \leftarrow (1-\alpha) \cdot Q(s_t,a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a))$, kde α je parametr učení, r_t je odměna za aktuální akci a γ je diskontní faktor.

Tradiční implementace Q-učení pomocí matice má ten problém, že funguje jen v diskrétních prostorech a s diskrétními akcemi. To je možné do nějaké míry řešit tím, že stavy a akce zdiskretizujeme. Dalším problémem je, že stavové prostory mohou být hodně velké, to potom vede k tomu, že matice jsou také velké a algoritmus se učí pomalu, nebo vůbec.

Hluboké Q-učení

Hluboké Q-učení se zakládá na stejné myšlence jako tradiční Q-učení, tedy na odhadování diskontované odměny pro každý stav s a každou akci a. V tradičním Q-učení jsou tyto hodnoty reprezentovány jako matice Q, což může být problematické v případě, že je mnoho stavů nebo akcí. V Hlubokém Q-učení se proto namísto matice Q používá neuronová síť, která pro každý stav vrací ohodnocení všech akcí.

Při trénování neuronové sítě se používá Bellmanova rovnice pro Q, která porovnává aktuální odměnu od prostředí $R_a(s,s')$ s hodnotou spočítanou pomocí Bellmanových rovnic z Q. Cílem je minimalizovat rozdíl mezi $Q(s,a)$ a $R_a(s,s') + \gamma \max_{a'} Q_k(s',a')$, což znamená, že se v dalším stavu vybere nejlepší akce podle aktuálního odhadu Q. Chybová funkce tedy vypadá jako

$$\mathbb{E}_{s' \sim P_a(s,s')} \left[\left(R_a(s,s') + \gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a) \right)^2 \right]_{\|\theta=\theta_k},$$

kde Q_θ jsou parametry neuronové sítě reprezentující matici Q.

Při trénování však často vzniká problém se stabilitou, protože přímou změnou funkce, která odhaduje Q, můžeme měnit i chování agenta. Pro zajištění stability se používají dva triky: cílová síť a přehrávání zkušeností. Cílová síť odděluje síť pro výběr akce od sítě pro odhad hodnoty a při trénování se fixují parametry sítě pro výběr akce a mění se pouze parametry sítě, která odhaduje hodnotu. Po daném počtu iterací se parametry obou sítí nastaví na stejné hodnoty a pokračuje se s trénováním podle stejného postupu. Experience replay se pak snaží minimalizovat vliv náhodnosti při trénování a pamatuje si čtveřice (s, a, r, s') stavu, akce, odměny a následujícího stavu, ze kterých se náhodně vybírají přechody pro trénink.

5. Současný stav poznání

Výzkumem umělé inteligence pro hru Hanabi se lidé začali zabývat v roce 2015. Tyto první práce se zaměřovaly na vývoj ručně psaných umělých inteligencí (Osawa, 2015) a (Walton-Rivers a kol., 2017), případně zkoumaly teoretické aspekty hry (van den Bergh, 2015), (Baffier a kol., 2016), (Cox a kol., 2015).

5.1 Standardní Hanabi

V následující sekci rozebereme různé přístupy k psaní umělé inteligence pro standardní verzi Hanabi.

5.1.1 Ručně psané algoritmy

Při psaní ručně psaných algoritmů se autoři převážně zaměřovali na vytvoření co nejlepší sady akcí, které může AI provádět, a co nejlepší sadu pravidel, podle kterých se má rozhodnout, co bude hrát (Walton-Rivers a kol., 2017), (Osawa, 2015), (Eger a kol., 2021).

Autoři ručně psaných algoritmů se většinou snažili optimalizovat algoritmus pro určitý počet hráčů. Nejčastěji vidíme algoritmy optimalizované pro dva hráče (Osawa, 2015), (Eger a kol., 2021), případně i pro tři hráče (van den Bergh a kol., 2017).

Jedna z nejlepších z těchto umělých inteligencí je strategie ze článku Eger a kol. (2017) v režimu *Full*. V experimentu, kdy tato umělá inteligence hrála sama se sebou, získala ve hře dvou hráčů průměrně 17,1 bodu.

Pokud chce dát nápovědu, potom o každé kartě na spoluhráčově ruce rozhodne, co by se s ní mělo stát. Rozhoduje mezi čtyřmi cíli: vyložit, zahodit, možná zahodit a ponechat si. Potom pro každou možnou nápovědu provede simulaci, tedy odhadne, jak by se změnilo, co si jeho spoluhráč o svých kartách myslí. Tento odhad je poté porovnán s cíli. Pokud se všechny odhady shodují s cíli, je spočítáno skóre nápovědy. Z těchto nápověd je poté vybrána ta nejlepší.

Při zahazování karet se pouze snaží zahodit kartu s nejmenší střední hodnotou ztracených bodů, pokud bude karta zahozena. To je početně náročné určit přesně, proto využívá heuristiku. Karty, u kterých se očekává, že budou zahrány dříve, jsou více cenné než ty, které budou hrány později. Dále karty, které jsou unikátní jsou cennější než ty, které nejsou. Nápovědy mají hodnotu půl bodu, což je zahrnuto ve ztrátě ze zahozené pětky, a v zisku ze zahození zbytečné karty.

Pokud dostane nápovědu, na spočítání, co se mu snažil spoluhráč svou nápovědou říct, používá stejný algoritmus, jako když sám napovídá a odhaduje, jak si jeho spoluhráč nápovědu vyloží.

Autoři tuto umělou inteligenci vytvořili s úmyslem, aby s ní co nejlépe dokázal hrát člověk. Proto například pracuje s faktem, že pokud hráč dostane nápovědu, očekává, že je relevantní k jeho příštímu tahu. Ve hrách s člověkem byla úspěšnější strategie v režimu *Intentional*. Tyto dva režimy používají stejný algoritmus pro rozhodnutí, jakou nápovědu dají. Liší se v tom, že pokud obdrží nápovědu, *Intentional* na ni tento algoritmus neaplikuje. Hráči hrající s *Intentional* v první

hře (tedy neměli čas zjistit, jak reaguje) získali v průměru 14,99 bodů se směrodatnou odchylkou 4,17. Hráči s *Full* potom 12,88 se směrodatnou odchylkou 5,98. Pokud hráli více her, jejich výsledky s *Full* se zlepšily na průměr 13,16 se směrodatnou odchylkou 5,76. Jak se změnily výsledky po více hrách s *Intentional* autoři neuvádějí.

Stejní autoři později vytvořili ještě vylepšení této inteligence, *Pravděpodobnostního hráče* (Eger a kol., 2021). V něm se snaží u každé karty udržovat pravděpodobnost jednotlivých kombinací. Tyto pravděpodobnosti mění podle řečených nápověd i podle situace, kdy byly nápovědy řečeny. Tato strategie dokonce ještě předčila jejich minulého hráče, průměrně dosahovala 18,56 bodu.

Nejvíce dosažených bodů z ručně psaných strategií dosahuje pravděpodobně FireFlower (Wu, 2018). Samotní autoři o ní nenapsali žádný text, pouze ve článku Bard a kol. (2020) je krátký popis toho, jak funguje. Uvádíme jeho volný překlad.

Implementované konvence zahrnují následující:

- Nápovědy obecně označují hrací karty, zpravidla nejprve novější karty.
- Nápovědy *řetězí* k ostatním nápovědám. Tedy napovídá-li hráč A hráči B hratelnou červenou dvojku jako červenou, pak B může usoudit, že je to skutečně červená dvojka, a pak napoví A červenou trojku v ruce A jako červenou a očekává, že A uvěří, že je to červená trojka, aby ji zahrál poté, co B zahraje svou červenou dvojku.
- Při odhazování odhazujte prokazatelně zbytečné karty, jinak nejstarší *nechráněnou* kartu.
- Nápovědy na nejstarší nechráněnou kartu *chrání* tuto kartu, až na mnoho výjimek, kdy to místo toho znamená zahrát.
- Nápovědy o zbytečných kartách znamenají *chránit* karty starší než ona.
- Záměrné vyřazení známých hratelných karet signalizuje partnerovi, že má kopii této karty (s dostatečně konkrétním určením místa na základě konvencí, aby ji mohl zahrát zcela bez skutečných nápověd).
- V mnoha případech nápovědy o kartách, které již byly napovězeny, různým způsobem mění přesvědčení o těchto kartách tak, že partner pravděpodobně změní to, co dělá (v souladu s širší heuristikou, že člověk by měl obvykle dávat nápovědy tehdy a jen tehdy, pokud by chtěl změnit partnerovo budoucí chování).

Podle autora se agenti soustředí na co největší počet perfektních her, ne nutně na maximalizování skóre. Autor nepíše o výsledcích pro pět hráčů.

Ostatní ručně psané inteligence se od těchto dvou často liší tím, že o kartě musí být známa barva i číslo, aby byla zahrána. Což má za následek, že spotřebují daleko více nápověd.

van den Bergh a kol. (2017) se snažili problém jednoznačného určení karty řešit tím, že AI zahrála kartu, pokud si byla *dostatečně jistá*, že je to bezpečné. Autoři měnili mez pro to, co znamená *dostatečně jistá* pro zahození a vyložení karet.

Walton-Rivers a kol. (2017) vyvinuli několik ručně psaných umělých inteligencí. Nesnažili se je však optimalizovat pro určitý počet hráčů. Stejně jako výše zmiňované práce, autoři vyvinuli (případně převzali) několik akcí, které může agent zahrát. Následně každé umělé inteligenci přiřadili několik akcí, které má hrát a sledovali, jak se výsledky budou lišit. Některým inteligencím zároveň napsali algoritmus, podle kterého se má rozhodovat, jakou akci zahrát. Jiní agenti na rozhodování o akci používali Monte-Carlo Tree Search.

Canaan a kol. (2018) také vytvořili sadu akcí a pravidel, podle kterých se agenti mohou chovat. Pomocí evoluce se poté snažili seřadit akce podle toho, jak se je má agent pokoušet provést. O tom, jak fungují evoluční algoritmy, jsme psali v kapitole Principy evoluce a zpětnovazebního učení (4.1). Akce, které popsali, jsou například:

- Zahraj kartu, která je nejpravděpodobněji hratelná, pokud je pravděpodobnost větší než p.
- Zahoď nejstarší kompletně neznámou kartu.
- Řekne jakoukoli informaci o kartě, která nikdy nebude hratelná.

Mahajan (2022) se snažil o napsání agenta, který se bude co nejlépe přizpůsobovat novým spoluhráčům, hlavně lidským, a bude měnit svůj styl hry podle jejich. Proto dopředu vytvořil několik modelů intuitivních strategií, které lidé často používají. Při samotném hraní se snaží najít model, který hraje co nejvíc podobně jako jeho skutečný spoluhráč. Nesnažil se adaptovat na spoluhráčovo hraní během hry, neboť stav hry je natolik složitý, že by na to nestačilo malé množství her, které s lidským hráčem typicky sehrají. Aby vybral co nejlepší model, všímal si na hráči těchto věcí (přímý překlad):

- Jak hráč počítá karty, když interpretuje nápovědy.
- Preferuje vykládat/zahazovat starší nebo novější karty, případně vybírá náhodně.
- Jak velkou váhu dává předchozím nápovědám.
- Jak je ochotný riskovat, když vykládá/zahazuje kartu.
- Jak důležité je pro něj schovávat si karty, které se můžou hodit později ve hře.
- Jakou prioritu dává možným akcím.

5.1.2 Zpětnovazební učení

Dalším často využívaným přístupem je zpětnovazební učení.

Bard a kol. (2020) vytvořili agenta Rainbow, který využívá hluboké Q-sítě (jejich fungování viz kapitola Principy evoluce a zpětnovazebního učení (4.2)). V jejich kódu všichni hráči sdílejí stejné parametry a používají neuronovou síť se dvěma skrytými vrstvami, kde každá má 512 neuronů. Při každém updatu sítě používají dříve získané zkušenosti. Používají epsilon-greedy strategii, při níž

epsilon sníží na nulu během prvních 1000 kroků trénování. Rainbow agent se trénoval 100 milionů kroků a trénink zabral zhruba sedm dní.

Stejní autoři vytvořili také agenta založeného na přístupu Actor-Critic (Bard a kol., 2020).

V diplomové práci Grooten (2021) se autor mimo jiné zajímal o to, jak nastavit funkce odměn při zpětnovazebním učení. Popisuje, že Hanabi samotné má některé funkce samo zabudované, a to +1 bod za každou vyloženou kartu a $-skóre$ za třetí žeton bomby (pokud počítáme s variantou, kde v případě výbuchu hráči získávají 0 bodů). Vytvořil ještě další funkce odměn, které dále používal při experimentech. Například zjistil, že se agenti často učili napovídat a karty zahazovat, neboť tím omezili pravděpodobnost, že vybuchnou. To se snažil vyřešit tím, že tyto akce penalizoval, a naopak více odměňoval agenta za hraní karet.

5.1.3 Další způsoby

Mezi další použité způsoby patří například také bayesovský přístup (Hu a Foerster, 2021), (Foerster a kol., 2019), který dosahoval též výborných výsledků.

Úplně jiný přístup můžeme vidět v článku (Eger a Gruss, 2019), ve kterém autoři kódují nápovědy do časových intervalů mezi provedenými akcemi.

Vlastní kategorie ručně psaných algoritmů jsou ty, které implementovaly hat guessing strategie (Wu, 2) (Fpvandoorn, 2020) zmiňované v (Cox a kol., 2015).

5.2 Mini Hanabi

Většina výzkumů se věnuje standardní verzi Hanabi. Některé výzkumy spojené se zpětnovazebním učením používaly i verze Hanabi s menším počtem karet. Natrénovat agenta na tomto zmenšeném Hanabi bylo daleko snazší, jelikož velikost stavu hry byla výrazně omezena (Fei Tong, 2020).

5.3 Hráči, co vidí své karty

Od začátku zároveň byla zkoumána varianta hry, kdy všichni hráči vidí své karty. Už v prvním článku (Osawa, 2015) autoři vytvořili hráče, který své karty vidí a průměrně dosáhli téměř dokonalého skóre. Zároveň se různí autoři pomocí těchto hráčů snažili spočítat, kolik existuje uspořádání karet v balíčku takových, že lze získat 25 bodů. To je totiž velmi obtížné spočítat přímo a pro standardní verzi Hanabi se to nikomu nepodařilo. Z dosažených výsledků je vidět, že takových balíčků je pro dva hráče alespoň 91 % (Cox a kol., 2015).

6. Implementace umělých inteligencí

Vytvořili jsme několik ručně psaných umělých inteligencí, nejlepší z nich jsme se poté snažili ještě vylepšit pomocí evoluce. Dále jsme vytvořili agenta založeného na Q-učení. V následující kapitole si vznik a chování těchto umělých inteligencí blíže popíšeme.

6.1 Ručně psané umělé inteligence

V následující sekci popíšeme jednotlivé ručně psané hráče. Postupně jsme vytvořili pět různých implementací od nejjednodušší po nejsložitější s tím, že novější verze vždy přímo vychází ze starší verze.

6.1.1 Náhodné hraní

Algoritmus Náhodné hraní hraje naprosto náhodně, vůbec nebere v potaz nápovědy, které jí byly řečeny, ani žádným způsobem nevyužívá to, co vidí. Pokud je na tahu, náhodně vybere jednu ze tří akcí (pokud jí pravidla dovolují napovídat, jinak vybírá pouze ze dvou zbylých). Pokud se rozhodne pro zahrání karty, náhodně vybere jednu kartu ze svých karet a tu zahraje (tedy vyloží nebo zahodí, podle vybrané akce). Pokud se rozhodne pro napovídání, pak náhodně vybere hráče, kterému bude napovídat, a jednu kartu z jeho ruky. Nakonec s poloviční šancí napoví hodnotu vybrané karty a s poloviční šancí na barvu vybrané karty.

6.1.2 Téměř náhodné hraní

Dalším krokem je algoritmus Téměř náhodné hraní. Tato umělá inteligence již bere v potaz nápovědy, které jí byly řečeny. Zatím si pamatuje pouze pozitivní nápovědy. Zároveň se snaží nezahazovat karty, o kterých ví, že mají hodnotu pět. Tyto karty jsou totiž od začátku unikátní a po jejich zahození již nejde dostavět postupka dané barvy. Algoritmus, kterým rozhoduje o tahu, je velmi jednoduchý a vypadá následovně:

Nejdřív se podívá, jestli má na ruce nějakou kartu, kterou může stoprocentně úspěšně vyložit. Pokud má, potom ji vyloží. Pokud ji nemá, ale zároveň má k dispozici alespoň jeden žeton nápovědy, potom náhodně napoví jinému hráči. Pokud žeton nápovědy k dispozici nemá, zahodí kartu. O tom, jakou kartu zahodí, se rozhodne následovně: Pokud má v ruce samé pětky, potom jednu náhodně zahodí. Jinak náhodně zahodí jedu z karet, o kterých ví, že to není pětka.

Jelikož tato umělá inteligence zahraje kartu pouze v případě, kdy si je stoprocentně jistá, že je to bezpečné, nikdy nemůže vybuchnout. Většina nápověd, které dává, jsou však úplně zbytečné.

6.1.3 Napovídání

Proto jsme přidali jednoduché napovídání. Když se hráč rozhodne napovídat, podívá se, jestli má některý z hráčů na ruce kartu, kterou by mohl hrát, ale neví o ní plnou informaci. Pokud takovou kartu najde, doplní o ní chybějící informaci. Pokud o kartě zatím nebyla řečena žádná informace, napoví její hodnotu.

Jestli takovou kartu nenajde, potom přejde k akci zahození karty. Pokud zahození karty v tu chvíli nedovolují pravidla, dá náhodnou nápovědu. Zbytek algoritmu zůstává stejný.

6.1.4 Lepší počítání karet

Algoritmus do této doby stále nevyužíval všechny informace, které mu byly řečeny nápovědami. Od teď si tedy bude pamatovat i negativní nápovědy. Zároveň také nevyužívá informace, které získává díky tomu, že vidí karty ostatních hráčů. Mohl by si například dopočítat, že nějakou kartu nemůže mít, protože je mají v ruce jeho spoluhráči.

Hráč si proto udržuje tabulku o rozměrech *počet barev* × *počet hodnot*, v případě standardního Hanabi tedy 5×5. Číslo v poli znamená, kolik karet o této kombinaci ještě hráč neviděl (například pokud na začátku hry u žádného jiného hráče nevidí žádnou "bílou trojku", má u této kombinace číslo 2). Tedy jde o karty a jejich počet, které by teoreticky mohl mít hráč v ruce. Každou kartu si zároveň také udržuje jako pole booleanů o velikosti *počet barev* × *počet hodnot*. Dokud mu na kartu není řečena žádná nápověda, je celá tabulka vyplněna hodnotami `true`. Ve chvíli, kdy mu je řečena o této kartě nápověda, se kombinace, které s nápovědou nejsou slučitelné, nastaví na `false`. Tento přístup funguje i pro negativní nápovědy. K této tabulce mají přístup i ostatní hráči, aby viděli, jaké informace už hráči byly řečeny. Pro to se do této tabulky nesmí přenést informace, které si hráč dopočítal. Proto si hráč udržuje ještě jednu tabulku booleanů o každé kartě, ve které kombinuje dopočítané i napovězené informace. Díky tomu je pak hráč nejenom že lépe schopen pracovat se svými nápovědami, ale i lépe počítat s kartami, které vidí u ostatních hráčů nebo které byly zahrané. Pokud si totiž pomocí těchto karet spočítá, že některou kombinaci mít nemůže, jednoduše si zanese do této tabulky `false`.

Zároveň se mění způsob, jak počítá, jestli může kartu vyložit, popřípadě zahodit. Když se rozhoduje o tom, jestli může například vyložit kartu, porovná všechny kombinace, kterých tato karta může nabývat, s hratelnými kartami. Pokud by všechny kombinace byly zároveň hratelné, nevadí, že hráč neví, kterou kombinaci přesně hraje, i přesto ji může zahrát. To se typicky může stát na začátku hry, kdy ještě nebyla vyložena žádná karta, tedy hratelné karty jsou všechny jedničky. Pokud se hráč o některé ze svých karet v tuto chvíli dozví, že je to jednička, nevadí mu, že neví, jakou má barvu, jelikož hrát může všechny.

Stejným způsobem postupuje i u karty, kterou chce zahodit. Tedy porovná všechny její možné kombinace se všemi kartami, o kterých ví, že by bylo bezpečné je zahodit.

Díky tomuto přístupu by bylo možné vnést do hry nejistotu tím, že by hráč mohl zahrát kartu i v případě, že by $\frac{\textit{hratelné kombinace karty}}{\textit{všechny kombinace karty}}$ bylo méně, než 1. Tento hráč má však obě meze nastavené na 1 a neriskuje.

6.1.5 Stáří karet a nápověd

V kapitole Strategie ve hře Hanabi (3) jsme se věnovali tomu, jak důležité jsou nové karty a nové nápovědy. Naše umělá inteligence s tímto bohužel zatím neumí vůbec pracovat. Tedy nerozlišuje mezi nápovědou, kterou dostala na začátku hry a nápovědou, kterou dostala v aktuálním kole. Přitom novější nápověda je určitě daleko důležitější pro nynější situaci. Stejným způsobem nedokáže správně posoudit situaci, kdy má na ruce starou a novou kartu a dostane společnou nápovědu pro obě dvě. V tuto chvíli není schopna říct, že nápověda byla s největší pravděpodobností mířena na novou kartu.

Mějme situaci, kdy je vyložená pouze červená jednička a nejsou zahozené žádné červené karty. Pokud si v tuto chvíli hráč dobere novou kartu, a je mu na ni okamžitě napovězeno *červená*, tato karta musí být červená dvojka, jinak by hráč tuto informaci neobdržel. Hráč si však nemůže zapsat, že mu byla napovězena *dvojka*, protože se tak nestalo.

Proto si o každé kartě budeme pamatovat ještě právě tyto kombinace barev a čísel, o kterých máme důvod si myslet, že karta může nabývat. Při rozhodování o hraní nebo zahození poté můžeme zvažovat i tento seznam. O tomto seznamu budeme dál mluvit jako o *Nejistých znalostech*.

Nejprve jsme hráči řekli, že pokud má novou kartu a dostal na ni nápovědu, může ji zahrát. S tím souvisí změna napovídání, kdy se hráč nejprve snaží napovědět hratelnou novou kartu. Až pokud zjistí, že taková karta neexistuje, pokusí se napovědět na hratelnou kartu. U toho si ovšem musí dát pozor na to, aby zároveň nenapověděl na novou kartu, kterou nelze vyložit.

Může se stát, že hráč nemůže napovědět na žádnou kartu, kterou lze zahrát. Zároveň on sám nemůže vyložit ani nechce zahodit některou ze svých karet. V tuto chvíli se pokusí napovědět na některou z unikátních karet, opět u toho nesmí napovědět na novou kartu.

Při všech těchto nápovědách si hráč zapíše do *Nejistých znalostí*, jakých kombinací by karta mohla nabývat, tedy buď takových kombinací, které jsou zrovna hratelné, nebo takových, které jsou nebezpečné. Jinak by nápovědu neobdržel. Pokud hráč obdržel nápovědu na víc než jednu kartu, není schopen rozlišit, na kterou z karet byla nápověda mířena. Proto si všechny unikátní i hratelné kombinace zapamatuje u všech napovězených karet. Jediná výjimka je, pokud mu bylo napovězeno na novou kartu. V tu chvíli ví, že mu bylo napovězeno pouze na ni a o ostatních napovězených kartách neuvažuje.

Proto pokud chce algoritmus vyložit kartu, nejdříve spočítá pravděpodobnost, že je karta hratelná z informací, které mu byly řečeny přímo. Pro přesný popis počítání pravděpodobnosti viz minulý algoritmus. Aby kartu vyložil, musí být pravděpodobnost vyšší, než konstanta *Vyložení karty z řečených informací*. Pokud se pomocí této pravděpodobnosti nerozhodne vyložit, ještě spočítá, jak je bezpečné vyložit z *Nejistých informací* a porovná ji s konstantou *Vyložení karty z Nejistých informací*.

Zároveň bylo potřeba změnit situace, kdy má hráč zahodit kartu. Od teď totiž můžeme kalkulovat s tím, že hráči o některé z karet dlouho nebyla řečena žádná nápověda, a tedy je možné, že není užitečná. Kartu tedy zahodí nejen v případě, že si je o některé z karet jistý, že je zbytečná, ale i v případě, kdy mu na ni dlouho nebylo napovězeno pozitivní nápovědou. Ještě musíme ošetřit případ, kdy některému z hráčů bylo před dlouhou dobou napovězeno na kartu, kterou si má

ještě ponechat. Proto aby zahodil kartu kvůli stáří, musí si být zároveň dostatečně jistý, že je to bezpečné.

Pravděpodobnost, že je kartu bezpečné zahodit, můžeme počítat třemi způsoby. Prvním způsobem je porovnávat kartu s kombinacemi, o kterým víme, že je už určitě nikdy nebudeme potřebovat. To jsou například karty, které už byly zahrány. Z tohoto porovnání dostaneme hodnotu, jak je pravděpodobné, že zahodíme zbytečnou kartu. Druhým způsobem je porovnávat kartu s kombinacemi, které jsou nebezpečné zahodit, protože se už ve hře vyskytují pouze jednou. Z porovnání s těmito kombinacemi získáme pravděpodobnost, že karta je unikátní. Třetím způsobem je porovnávat *Nejisté znalosti*, tedy jestli nám o kartě bylo řečeno, že může být nebezpečná tímto způsobem. Tento seznam zase porovnáváme se seznamem unikátních karet ve hře.

Proto se při rozhodování, jakou kartu má zahodit, hráč nejdříve řídí pravděpodobností, že je karta zbytečná. Jak moc si musí být jistý, že je karta zbytečná na to, aby ji zahodil, určuje konstanta *Zahození zbytečné karty přísný*. Dále se rozhodne kartu zahodit, pokud jsou nápovědy dostatečně staré (tedy jsou starší než konstanty *Stará číselná nápověda* a *Stará barevná nápověda*) a zároveň pokud si je dostatečně jistý, že je karta zbytečná (konstanta *Zahození zbytečné karty jemný*). Pokud si žádnou kartou není jistý, potom se alespoň snaží zahodit kartu, která není unikátní. Pokud zahazuje kartu, která není unikátní, potom nejdříve zjišťuje pravděpodobnost z *Nejistých znalostí*. K rozhodnutí, jestli je pravděpodobnost dostatečná, složí konstanta *Zahození karty z Nejistých informací*. Nakonec rozhoduje o tom, jestli karta není nebezpečná pomocí znalostí, které mu byly řečeny přímo. K tomu slouží konstanty *Zahození bezpečné karty přísný* a poté *Zahození bezpečné karty jemný* spolu s konstantami *Stará číselná nápověda* a *Stará barevná nápověda*. Ty fungují obdobně, jako u zbytečných karet.

Také může nastat situace, kdy je hráč nucený zahodit, což se stane ve chvíli, kdy nechce ani zahrát kartu, ani napovídat. V tu chvíli zahodí tu kartu, o které si je nejvíc jistý, že je zbytečná.

V textu bylo popsáno několik konstant, které je potřeba nastavit. V následujícím seznamu jsou vypsány spolu s tím, jak jsme konstanty nastavili.

- Vyložení karty z řečených informací – 1
- Vyložení karty z *Nejistých informací* – 1
- Zahození zbytečné karty přísný – 1
- Zahození zbytečné karty jemný – 0,1
- Zahození bezpečné karty přísný – 1
- Zahození bezpečné karty jemný – 0,1
- Zahození karty z *Nejistých informací* – 1
- Stará číselná nápověda – 3
- Stará barevná nápověda – 3

6.2 Evoluce

Parametry popsané v minulé sekci u algoritmu Stáří karet a nápověd jsme nastavili pouze odhadem dle našich zkušeností z hry lidských hráčů, což nemusí být rozhodně optimální. Navíc byly tyto parametry nastaveny pro všechny počty hráčů stejné. Pro optimalizaci těchto konstant jsme vytvořili evoluční algoritmus, který bude v této kapitole popsán.

6.2.1 Evoluční algoritmus

Reprezentace jedince

Potřebujeme evolvovat parametry popsané v minulé kapitole. Jedná se tedy o sedm hodnot typu float a dvě hodnoty typu int. Proto v každém jedinci udržujeme dvě pole, jedno typu float a druhé typu int. Hodnoty typu int, tedy počet kol bez nápovědy na kartu aby byla tato karta považována za zbytečnou, mohou teoreticky být hodně velké. Proto jsme se rozhodli je shora omezit hodnotou deset.

Selekce

Algoritmus používá turnajovou selekci, tedy z populace jsou vždy náhodně vylosováni dva jedinci, a ten, který má lepší fitness, je zvolený jako rodič.

Křížení

Jelikož jedince reprezentujeme jako dvě pole rozdílných typů, museli jsme implementovat křížení pro každé zvlášť. Pro pole s hodnotami typu float jsme implementovali aritmetické křížení a pro pole s inty jsme implementovali uniformní křížení.

Mutace

Pro obě pole jsme zvolili zatíženou mutaci. Pokud začne mutovat hodnota typu float, k aktuální hodnotě s se šancí 1/2 přičte náhodná hodnota z intervalu $[0, K]$, s pravděpodobností 1/2 odečte, kde K značí velikost kroku. K je počítáno jako $1 - \frac{aktualní\ generace}{maximální\ generace}$, tedy se postupně snižuje od jedničky do nuly. Pro hodnoty typu int se při mutaci přičte nebo odečte jednička, vždy s šancí 1/2.

Fitness funkce

Podstata problému nám dává jasný způsob, jak nastavit fitness funkci, a to kolik bodů jedinec nahrál ve hře. Musíme však započítat vliv náhody, proto je potřeba počítat průměrný počet bodů z N her.

V průběhu učení jsme řešili, jak optimálně tento počet her nastavit. Pokud jsme N nastavili jako moc nízké číslo (například 10), často se stávalo, že některý jedinec měl štěstí a měl vysokou fitness, i když v realitě dobrý nebyl. Poté měl tedy například větší šanci být vybrán jako rodič než jedinci, kteří byli lepší, ale neměli takové štěstí při hraní. Zároveň jsme si díky takovému jedinci lepší jedince vůbec nezapamatovali.

Na druhou stranu, pokud jsme nastavili N moc vysoké, jedno vyhodnocení fitness trvalo moc dlouho a algoritmus byl velmi pomalý. Tento problém jsme nakonec vyřešili tak, že N je nastaveno na relativně malý počet her. Zároveň ale pokud má jedinec po těchto N hrách lepší fitness než aktuálně nejlepší jedinec, odehraje dalších M her. Jeho fitness se nakonec spočítá jako průměr ze všech $N + M$ her.

6.2.2 Nastavení parametrů

Nyní popíšeme, jak jsme nastavili parametry pro experimenty, jejichž výsledky dále rozebíráme v kapitole Výsledky (8.2). Parametry jsme nechali stejné pro všechny počty hráčů.

Velikost populace jsme nastavili na padesát jedinců. Pro více jedinců běžela evoluce pomalu a zároveň byly výsledky stejné, jako pro námi zvolený počet. Evoluci jsme nechali běžet po 200 generací. Pravděpodobně by jich stačilo méně s podobnými výsledky, jelikož jedinci se často zlepšovali pouze na začátku, později se ustálil jeden nejlepší a jen zřídka se ke konci objevil nějaký lepší. Ke křížení mezi jedinci vybrané turnajovou selekcí dojde s pravděpodobností 0,8. Se stejnou pravděpodobností začne jedinec mutovat. Pravděpodobnost, že bit jedince zmutuje, jsme nastavili na 0,1. Nakonec jsme ještě museli nastavit konstanty N a M popsané výše. Rozhodli jsme se, že průměr ze sta her je dostatečně vypovídající, proto chtěli, aby $N + M = 100$. Pokud jsme nastavili N jako moc nízké (například 10), stávalo se velmi často, že jedinec počítaný z N her měl vyšší fitness, než celkově nejlepší jedinec. To způsobilo, že se průměr z $N + M$ her počítal příliš často a evoluce běžela paradoxně pomaleji. Nakonec jsme nastavili $N = 20$, díky tomu byla fitness přesnější a stávalo se méně, že by se hrálo zbylých M her.

6.3 Hluboké Q-učení

Nakonec jsme se pokusili natrénovat agenta pomocí Q-učení. Jelikož je však stav normálního Hanabi moc velký, rozhodli jsme se ho trénovat na zjednodušené verzi. Zjednodušení má tři části. Zaprvé, hráč vidí nejenom soupeřovy, ale i své karty. Zadruhé, hráč nemůže napovídat. Tím pádem muselo být uvolněno pravidlo, které říká, že hráč nemůže zahodit, pokud má k dispozici plný počet nápověd. A zatřetí, výrazně jsme snížili počet karet v balíčku. Hráči hrají pouze se dvěma barvami, od každé barvy existují tři *jedničky*, dvě *dvojky* a jedna *trojka*. V balíčku je tedy dohromady pouze 12 karet. Hráči zároveň mají na ruce pouze 3 karty ve hře dvou a tří hráčů, 2 karty ve hře čtyř a pěti hráčů. Toto nastavení bylo převzato z práce (Fei Tong, 2020). Ve článku se bohužel nepíše, kolik měli hráči k dispozici žetonů, rozhodli jsme se pro dva žetony bomby a 3 žetony nápovědy. Maximum bodů, které mohou hráči získat, je 6.

Pokud bychom se rozhodli trénovat agenta na standardní verzi Hanabi, určité zlepšení bychom pravděpodobně mohli vidět řádově po desítkách milionů hrách (Bard a kol., 2020). To by i za normálních okolností trvalo velmi dlouho, s memory leakem popsaným v kapitole Implementace (7.3.3) by to bylo naprosto nemožné.

6.3.1 Nastavení modelu

Snažili jsme se natrénovat pouze model hrajícího hru pro dva hráče, jelikož je v tom případě nejmenší stav hry.

Všichni hráči trénují společně jeden model. Velikost vstupní vrstvy, tedy aktuální stav hry, má velikost *počet hráčů* $*$ *počet karet v ruce* $* 2 +$ *počet barev* $+$ *počet zbývajících* žeton*ů bomby*, v našem případě pro hru dvou hráčů tedy $2 * 3 * 2 + 2 + 1 = 15$. Hráč vidí své a spoluhráčovy karty (dohromady šest karet), každá karta má dva atributy (barva a číslo). Výstupní vrstva má velikost $2 *$ *počet karet v ruce* $= 6$, jelikož si hráč vždy vybere jednu ze tří karet a tu buď zahodí, nebo vyloží. Mezi vstupní a výstupní vrstvou jsou dvě plně propojené vrstvy, každá s 256 uzly a aktivační funkcí leakyReLU.

7. Implementace

Program jsem se rozhodla napsat v jazyce C#, jelikož jsem s ním v době, kdy jsem začala program psát, měla nejvíce zkušeností.

7.1 Požadavky na program

Potřebovali jsme vytvořit rozhraní, které by nám umožňovalo testovat námi vytvořené umělé inteligence. Zároveň jsme chtěli, aby program ukládal všechny informace, které pro naše umělé inteligence potřebujeme.

Jelikož nepotřebujeme, aby v našem rozhraní byl schopný hrát člověk, není nutné, aby měl program grafický výstup. Stačí, aby uživatel všechny vstupy zadal jako argumenty příkazové řádky při spuštění programu. Zároveň je potřeba, aby nám program dal zpětnou vazbu o tom, jakých výsledků umělé inteligence dosáhly. To dělá opět textovou formou tím, že vytváří soubory a vypisuje do nich informace. Tyto informace jsou převážně dvojího druhu. Nejdřív jsme potřebovali sledovat, jak probíhají hry hrané agenty. Tedy abychom viděli, jak který agent mění své znalosti o hře na základě nových informací a jak tyto informace využívá. Díky tomu se snáze odstraňovaly chyby v implementaci a zároveň bylo díky tomu snazší agenty dále vylepšovat, neboť jsme přesně viděli, jak se rozhodují. Dále jsme potřebovali získat samotné výsledky agentů pro jejich porovnání a diskuzi toho, jak nové nastavení agenta měnilo jeho výsledky.

7.2 Uživatelská dokumentace

Uživatel pouští program z příkazové řádky a dál už s ním nijak nekomunikuje. Program nepodporuje hru člověka, neboť výstup do konzole je v tomto případě pro člověka velmi nepřehledný a prakticky nehratelný.

Uživatel zadává jednotlivé argumenty oddělěné mezerou ve formátu *jméno argumentu=argument*. Je důležité, aby v rámci jednoho argumentu žádná mezera nebyla. Desetinná čísla je potřeba zadávat v souladu s nastavením prostředí uživatele.

Uživatel si může vybrat mezi třemi módy programu, ve kterých může program spustit, a to *mode=play*, *mode=train* a *mode=evolution*.

Mód hraní

Tento mód slouží k testování toho, jak jsou vytvoření hráči dobří. Tedy vytvoří počet a typy hráčů podle toho, jak jsou jsou zadáni na vstupu, a následně je nechá hrát tolik her, kolik uživatel zadal. Nakonec se vytvoří složka s výstupními soubory. Pokud si uživatel nepřál vypisovat informace o jednotlivých hrách, obsahuje tato složka pouze soubor se shrnutím výsledků. Pokud si uživatel tyto informace vypisovat přál, obsahuje zároveň pro každou hru soubor s jejím průběhem. Některé typy hráčů potřebují ke svému vytvoření data uložená v souboru (konkrétně jde o model pro Q agenta a o načtení konstant pro algoritmus Stáří karet a nápověd). Cestu k těmto souborům uživatel zadává argumenty *model* a *tresholds*.

Jména a datové typy jednotlivých argumentů popsaných výše jsou následující: "int players", "int players_types", "int rounds", "string file_name", "string game_mode", "string print_stats", "string model", "string tresholds". Jako argument players_types uživatel zadá tolik hodnot, kolik hraje hráčů. Tyto hodnoty oddělí středníkem.

Typy hráčů odpovídají číselné hodnotě následovně:

- Náhodné hraní: 0
- Téměř náhodné hraní: 1
- Napovídání: 2
- Lepší počítání karet: 3
- Stáří karet a nápověd: 4
- Q agent: 5

Pokud žádný z hráčů není Q agent, potom uživatel nepotřebuje zadávat jméno modelu. Proto místo tohoto argumentu napíše "none". Stejně tak pokud žádný z hráčů nevyužívá algoritmus Staří karet a nápověd, uživatel nemusí zadávat jméno souboru, odkud se mají načíst konstanty, které tento hráč využívá. Opět místo tohoto argumentu napíše "none". Pokud uživatel nezadá žádný soubor s konstantami i přes to, že některý z hráčů Stáří karet a nápověd využívá, použijí se původní hodnoty toho algoritmu nastavené námi.

Pomocí příkazu *mode=play players=3 players_types=0;0;0 rounds=10000 file_name=random_3players game_mode=normal print_stats=false model=none tresholds=none* uživatel spustí deset tisíc her se třemi náhodnými hráči v normálním módu, výsledky se uloží do složky random_3players a nevytvoří se soubory se záznamem o hrách. Soubor použitý na načtení konstant musí na první řádce obsahovat postupně všechny konstanty oddělené mezerou. Soubory s výsledky, které se vypisují při evoluci, tomuto formátu odpovídají.

Je důležité, aby Q agent byl puštěný pouze s argumentem "mini". Zároveň, pokud je náhodný hráč puštěný s tímto argumentem, neumí napovídat, abychom ho mohli porovnávat s Q agentem, který také nenapovídá.

Mód trénování

V tomto módu se natrénuje nový model Q agenta. Tento mód automaticky předpokládá, že se pouští na módu hry Mini Hanabi a že všichni hráči mají typ Q agenta. Uživatel programu zadá, pro kolik hráčů chce model trénovat a na kolika hrách se mají agenti učit. Dále zadá odměny, které si přeje nastavit a jméno souboru, kam se má model uložit. Jména a typy těchto argumentů jsou následovné: "int players""int rounds""float rewards""string file_name". Jako argument *rewards* uživatel zadá čtyři hodnoty oddělé středníkem. První odpovídá odměně za úspěšně vyloženou kartu, druhý odměně za první neúspěšně vyloženou kartu, třetí odměně za druhou neúspěšně vyloženou kartu a čtvrtý odměně za zahozenou kartu.

Automaticky se vytvoří dvě složky, a to "jméno souboru pro uložení výsledků"_training a "jméno souboru pro uložení výsledků"_model. V první ze

zmíněných složek se vytvoří soubor, do kterého se během trénování po každé hře zapíše výsledek, kterého hráči v té hře dosáhli. Do složky druhé složky se po každých 500 hrách uloží aktuální model a přepíše ten starý.

Tento mód se může pustit například následujícím způsobem: *mode=train players=2 rounds=1000 rewards=20;-1;-10;-1 file_name=mini_2players*. Tedy se natrénuje model pro dva hráče na tisíci hrách s odpovídajícími funkcemi odměn.

Mód evoluce

Tento mód dovoluje spustit evoluci konstant pro algoritmus Stáří karet a nápověd. Uživatel programu zadá, pro hru kolika hráčů se mají konstanty evolvovat, a jméno souboru, kam se mají uložit výsledky. Dále zadá parametry samotné evoluce. Jména a typy argumentů pro tento mód jsou: `"int players""string file_name""int pop_size""int gens""float cross_prob""float mut_prob""float bit_mut_prob""int average""int average_better" "int runs"`.Evoulce se automaticky pustí pro algoritmus Stáří karet a nápověd, tedy není potřeba zadávat typy hráčů. Zároveň není potřeba zadávat mód hry, evoluce se automaticky pustí na normální.

Evoluci je možné spustit například pomocí následujících argumentů: *mode=evolution players=3 file_name=evolution_3players pop_size=50 gens=200 cross_prob=0,8 mut_prob=0,8 bit_mut_prob=0,1 average=20 average_better=100 runs=5*.

Fitness nejlepšího jedince se po každé generaci vypíše na příkazovou řádku. Po konci každého běhu evoluce se vytvoří soubor, do kterého se uloží natrénované parametry, fitness nejlepšího jedince a parametry, se kterými uživatel evoluci spustil.

7.3 Programátorská dokumentace

V následující sekci popíšeme objektový návrh a funkci jednotlivých tříd. Dále nastíníme, jakým způsobem je možné program dále rozšiřovat a představíme knihovnu Tensorflow.NET, kterou v programu používáme.

7.3.1 Objektový návrh

Program je rozdělen do několika tříd, které postupně popíšeme.

Program

Třída Program obsahuje metodu main, ve které zpracuje argumenty na příkazové řádce a zavolá metodu třídy Hanabi podle toho, jaký mód chce uživatel spustit.

Hanabi

Třída Hanabi pouští jednotlivé běhy her (případně běhy evoluce), zpracovává jejich výsledky a vypisuje je do souborů. U módů hraní a trénování zároveň vytváří hráče, které předává hře jako parametr.

Game

Game je třída samotné hry. Udržuje aktuální stav hry a řídí běh celé hry (metody *PlayGame* a *PlayRound*). Udržuje, který hráč je na tahu, ptá se hráče na to, jakou akci chce provést (*AskWhatToDo*), a poté vyhodnotí dopad této akce (*ProcessPlayersAction*). To obnáší mimo jiné aktualizaci hratelných a nebezpečných karet, změnu počtu žetonů, volání metod hráčů, které upravují, co hráči ví o stavu hry (ať už proto, že někomu bylo napovězeno, nebo proto, že si hráč například dobral kartu). Zároveň obsahuje metody, které hlídají, jestli je konec hry, případně jestli začalo poslední kolo. Pomocí metod *GetGameLength* a *GetHintsGiven* lze po skončení hry zjistit, jak byla hra dlouhá a kolik bylo použito nápověd. V konstruktoru nastaví potřebné konstanty, ty se mohou lišit v závislosti na počtu hráčů nebo na módu hry.

Player

Player je abstraktní třída hráče, ze které jsou zděděni všichni ostatní hráči. Udržuje všechno, co hráč ví o svých kartách a o kartách spoluhráčů. Jeho metody jsou převážně různé druhy akcí, které jsme naiplementovali, jako například *Random Hint*, *DiscardARandomCardNotFive* nebo *TryToPlaySafeCard*. Další metody slouží k aktualizaci toho, co hráč ví o svých kartách nebo kartách spluhráčů, například *FigureOutNewInformation*, *ForgetEverythingAboutACard*.

Zděděné třídy jsou RandomPlayer, AlmostRandomPlayer, HintPlayer, GoodPlayer, PlayerUsingAge a CheatingDeepQMiniPlayer. Většinou se jednotlivé třídy liší v tom, jak vypadá jejich metoda *Play*, tedy jakým algoritmem se řídí rozhodování o tom, co zahrají, a které akce implementované ve třídě Player používají. Jednotlivé rozdíly metod *Play* byly popsány v sekci Ručně psané umělé inteligence (6.1).

Třída CheatingDeepQMiniPlayer zároveň umí zakódovat stav hry, ve kterém se nachází, metodou *EncodeState*.

Deck

Třída Deck implementuje balíček karet. Množství a druhy karet v balíčku se liší podle módu, v jakém je hra spuštěna. Metody *CreateDeck* a *ShuffleDeck* společně vytvoří balíček hracích karet. Metoda *GetFirstCard* odstraní vrchní kartu z balíčku a vrátí ji.

Card

Každá karta reprezentovaná třídou Card má čtyři atributy. První dva říkají, jakou má karta hodnotu a barvu (může mít hodnotu 0 a žádnou barvu). Další dva říkají, jakých hodnot a barev může karta teoreticky nabývat. Tato pole se u karty využívají pouze při situacích, kdy hráč potřebuje vědět, které možné kombinace může jeho karta mít. Podle toho, v jakém módu jsme hru pustili, se vytvoří pole o správných velikostech.

Hint

Každá nápověda (třída Hint) má tři atributy. Atribut *player* označuje hráče, kterému byla nápověda dána. Atributy *number* a *suit* označují, na jaké číslo nebo barvu byla nápověda dána.

Třída Hint má dva konstruktory, které se volají podle toho, jestli má být vytvořena číselná, nebo barevná nápověda.

Printer

Třída Printer má na starost vypisování hry do souboru v průběhu hry. Pokud uživatel nechce vypisovat průběh hry do souboru, instance této třídy se nevytvoří.

7.3.2 Rozšíření

Je možné program přirozeně rozšiřovat. K vytvoření nového hráče stačí vytvořit třídu zděděnou z třídy *Player*. V této zděděné třídě stačí implementovat pouze konstruktor, který hráče vytvoří, a metodu *Play*. Při implementování metody *Play* je potřeba, aby v každé situaci hry dokázala vrátit akci, kterou je možné provést. Například pokud by vždy vracela akci nápovědy se může stát, že by se hra zacyklila, pokud by hráč neměl k dispozici žádné žetony nápovědy.

Dalším způsobem, jak hru rozšířit, je přidání nového módu hry. Nový mód se může lišit počtem barev, počtem karet od každé hodnoty, případně jiným počtem žetonů. K tomuto rozšíření je potřeba v konstruktorech hry a karet nadefinovat nové konstanty.

Další možností je vytvořit nový model, který bude schopen hrát jiný mód hry, případně pracovat s jiným stavem hry. Je potřeba správně nastavit velikosti vstupní a výstupní vrstvy, aby odpovídaly velikosti stavu a počtu akcí, které může hráč zahrát. Zároveň je potřeba dopsat hráči, který tento model bude využívat, metodu *EncodeState*.

7.3.3 TensorFlow.NET

Pro implementaci zpětnovazebního učení jsme potřebovali knihovnu, která by měla naimplementované potřebné metody. Rozhodli jsme se pro použití knihovny TensorFlow.NET (Rinne, 2023), což je knihovna obdobná TensorFlow pro Python, jen pro jazyk C#. Toto rozhodnutí však přineslo nemalé problémy kvůli tomu, že knihovna je stále ve vývoji a obecně málo využívaná. Tudíž má velmi stručnou dokumentaci a problémy, na které jsme narazili, před námi nikdo neřešil (nebo aspoň nepopsal na Internetu). Příště by pravděpodobně bylo jednoduší například propojit C# s Pythonem pomocí Serveru a Klienta.

Největší problém, který je bohužel stále v programu, je memory leak, který je z největší pravděpodobností způsoben voláním metody keras.Model.fit. Stejný problém měla i metoda keras.Model.predict, ta však šla nahradit metodou keras.Model.Apply, se kterou žádný takový problém nenastává. Pravděpodobně by šla nahradit i metoda fit, ale bohužel se nám to nepodařilo.

8. Výsledky

8.1 Porovnání ručně psaných algoritmů

V následující sekci se podíváme na výsledky jednotlivých ručně napsaných algoritmů. Rozebereme, jaké rozšíření mělo jaké dopady na hru a jak se dopady měnily v závislosti na počtu hráčů.

Prezentované výsledky jsou průměrné hodnoty z deseti tisíc odehraných her.

8.1.1 Náhodné hraní

Nejdřív jsme vytvořili algoritmus Náhodné hraní. Tyto výsledky nám můžou sloužit pro porovnání, o kolik lepší jsou další algoritmy. Tento algoritmus při každé z her vybuchl. Výsledky, kterých dosáhl, prezentujeme v Tabulce 8.1.

počet hráčů	2	3	4	5
průměr	1,24	1,26	1,25	1,25
směrodatná odchylka	1,27	1,27	1,26	1,27
maximum	9	11	9	10
počet dosažení maxima	1	1	1	1
průměrný počet tahů	10,92	10,91	10,93	10,99
průměrný počet nápověd	4,27	4,23	4,23	4,24

Tabulka 8.1: Výsledky algoritmu Náhodné hraní.

8.1.2 Téměř náhodné hraní

Dalším krokem bylo, aby algoritmus nějakým způsobem bral v potaz nápovědy, které dostal a nehrál kartu, o které si nebyl na sto procent jistý, že ji může zahrát. Dosažené výsledky jsou prezentovány v Tabulce 8.2. Hlavní nevýhodou tohoto algoritmu bylo, že nápovědy stále dával náhodně.

Tento algoritmus dosahoval přibližně o tři body více než algoritmus Náhodné hraní. Tento rozdíl je pravděpodobně způsoben dvěma aspekty. Zaprvé, i přesto, že hráči dávají nápovědy náhodně, se občas stane, že napoví na hratelnou kartu. Zadruhé, jelikož hráči nikdy nemohou udělat chybu a vybuchnout, jsou jeho hry obecně delší o 60–75 tahů v závislosti na počtu hráčů. Díky tomu je zároveň řečeno daleko více nápověd. Například pro hru tří hráčů mají s těmito výsledky k dispozici přibližně 40 nápověd. Rozdílné množství nápověd pro hru 2–5 hráčů odpovídající datům prezentovaných v Tabulce 3.1 se projevuje i ve výsledném získaném množství bodů. Množství nápověd však není jediný aspekt, který ovlivňuje získané množství bodů pro různé počty hráčů. Výsledky dále může ovlivňovat, že ve 2 a 3 hráčích jedna nápověda může označit více karet, jelikož ve dvou a třech hráčích je 5 karet na ruce. Na druhou stranu v nižším počtu hráčů je větší pravděpodobnost na opakování nápověd, jelikož téměř náhodný hráč nekontroluje, jestli stejná nápověda už nebyla použita.

počet hráčů	2	3	4	5
průměr	4,14	4,40	3,89	3,60
směrodatná odchylka	1,99	1,88	1,68	1,59
maximum	13	13	12	10
počet dosažení maxima	3	1	1	5
průměrný počet tahů	84,98	75,79	75,34	68,67
průměrný počet nápověd	43,93	39,60	39,30	36,44

Tabulka 8.2: Výsledky algoritmu Téměř náhodné hraní.

8.1.3 Napovídání

Dosažené výsledky pro algoritmus Napovídání jsou prezentovány v Tabulce 8.3. Stejně jako v předchozím případě tento algoritmus bere v potaz nápovědy, které dostává a hraje pouze karty, u kterých si je na sto procent jistý, že je může zahrát. Hlavní rozdíl je v tom, že tento algoritmus už dává nápovědy na hratelné karty, což přineslo obrovské zlepšení.

Oproti Téměř náhodnému hraní získává Napovídání přibližně o 10 bodů více. To poměrně přesně koreluje s tím, že hry jsou přibližně o 10 tahů kratší, jelikož 10 karet nebylo zahozeno a nebyla za ně získána nápověda, která hru prodlužuje. Zároveň je ale zatím vidět značná neefektivita ve využívání nápověd. Počet použitých nápověd poměrně přesně odpovídá dvojnásobku získaných bodů, tj. na každé zahrané kartě bylo třeba separátně říct její hodnotu i barvu a obecně tedy nebyly příliš efektivně využity nápovědy dané mimochodem. Tedy takové nápovědy, které pokud jsme napovídali např. *červená* na zrovna hratelnou červenou trojku, napověděli jsme na všechny ostatní červené karty v ruce. To je pravděpodobně převážně způsobené tím, že karty jsou zahazovány téměř náhodně, pouze s ohledem na to, aby nebyla zahozena pětka. Algoritmus by nejspíš šel vylepšit například tak, že by přednostně zahazoval karty, o kterých má minimum informací. Zároveň pravidlo, aby nezahazoval pětky, nejspíš nemá smysl a je spíše kontraproduktivní, neboť hráči se k vykládání pětky v naprosté většině her vůbec nedostanou.

Rozdíly mezi jednotlivými počty hráčů jsou ovlivněny několika aspekty, které jdou částečně proti sobě. Můžeme sledovat, že při hře pěti hráčů algoritmus dosahoval nejméně bodů. To je pravděpodobně dáno tím, že hráči mají k dispozici méně nápověd na kartu, jak je prezentováno v Tabulce 3.1. Naopak dva hráči, kteří mají počet nápověd naopak nejvyšší, dosáhli nejlepšího výsledku. Další aspekt se týká rozdílného počtu karet na ruce (pro 2 a 3 hráče 5 karet a pro 4 a 5 hráčů 4 karty). V momentu, kdy má hráč na ruce pět karet, je větší pravděpodobnost, že na některou kartu bude napovězeno *mimochodem*, než když má na ruce pouze čtyři karty. To pravděpodobně ovlivňuje, že ve hře 3 hráčů jsou nejefektivnější nápovědy, tj. je potřeba pouze 1,88 nápovědy na jednu úspěšně zahranou kartu. Proti tomuto aspektu ale naopak působí množství karet, které vidí hráč v rukou spoluhráčů, konkrétně 5 karet ve dvou hráčích, 10 ve třech, 12 ve čtyřech a 16 v pěti. Čím víc karet hráč vidí, tím je větší pravděpodobnost, že mezi nimi je hratelná karta.

počet hráčů	2	3	4	5
průměr	15,11	14,96	14,60	12,87
směrodatná odchylka	2,44	1,38	1,05	1,02
maximum	22	20	19	17
počet dosažení maxima	3	1	1	1
průměrný počet tahů	71,57	64,98	65,60	60,19
průměrný počet nápověd	30,34	28,15	29,27	27,26

Tabulka 8.3: Výsledky algoritmu Napovídání.

8.1.4 Lepší počítání karet

Dosažené výsledky pro algoritmus Lepší počítání karet jsou prezentovány v Tabulce 8.4. Tento algoritmus už bere v potaz všechny informace o kartách, které mohl získat, což se pozitivně projevuje i na počtu získaných bodů. Tento efekt je tím výraznější, s čím více informacemi může hráč pracovat. Tj. pro vyšší počty hráčů je vidět dohromady více karet u spoluhráčů a je možno si dopočítat více informací. Proto rozdíl mezi algoritmy Napovídání a Lepší počítání karet představuje pro 2 hráče pouze zhruba 0,3 bodu, ale pro 5 hráčů už je to téměř 1,8 bodu. S tímto také koreluje, že tento algoritmus již potřebuje méně nápověd na zahrání karty. Pro 3–5 hráčů je to zhruba 1,75 nápovědy. Úměrně tomu také mírně klesl průměrný počet tahů, jelikož opět bylo víc karet použito k zahrání a méně k zahození.

Efekt využití negativních nápověd, který by měl být význačnější pro 2 a 3 hráče (5 karet na ruce), není dle výsledků příliš zřetelný.

počet hráčů	2	3	4	5
průměr	15,40	16,25	16,09	14,65
směrodatná odchylka	2,92	1,64	1,28	1,29
maximum	23	22	21	20
počet dosažení maxima	5	1	1	1
průměrný počet tahů	71,62	64,23	64,71	59,16
průměrný počet nápověd	30,33	27,30	28,23	25,98

Tabulka 8.4: Výsledky algoritmu Lepší počítání karet.

8.1.5 Stáří karet a nápověd

Dosažené výsledky pro algoritmus Staří nápověd a karet jsou prezentovány v Tabulce 8.5. Tento algoritmus je nejpokročilejší ze všech prezentovaných a od algoritmus Lepší počítání karet se liší především tím, že bere v potaz, jak dlouho už jsou jednotlivé karty na ruce. Také jako jediný (mimo Náhodného hraní) hraje karty i bez stoprocentní jistoty bezpečnosti. Z výsledků prezentovaných v Tabulce 8.5 je patrné, že chybovost tohoto algoritmu je poměrně značná ve hře dvou hráčů, kdy průměrně zbude v balíčku 11,21 karty (hráči v tu dobu vybouchnou třetím otočením bomby). Tato chybovost pro více hráčů výrazně klesá a při hře pěti hráčů už průměrně zbude pouze 0,36 karty v balíčku. Vybouchnutí se také samozřejmě výrazně projevují ve směrodatné odchylce, která je výrazně vyšší než u ostatních algoritmů. Dřívější vybouchnutí v méně hráčích je způsobeno

pravděpodobně tím, že v méně hráčích obecně hráči obdrží víc nápověd o svých kartách. Proto snadněji dojde k tomu, že do *Nejistých informací* přidají více nepravdivých informací než ostatní hráči. Proto může snadněji dojít k tomu, že kombinací těchto nápověd je nějaká karta považována za bezpečnou na hraní. Míra chybovosti a vybouchnutí se také výrazně projevuje v počtu získaných bodů.

Ve hře dvou hráčů se celkový průměr sice snížil, ale v ostatních počtech hráčů se naopak výsledky zlepšily, v případě hry čtyř hráčů dokonce o téměř 2,5 bodu. Obecně je také výrazný posun v počtu nápověd na jednu úspěšně vyloženou kartu, který je pro tento algoritmus zhruba 1,4. Zároveň je důležité poznamenat, že ačkoliv se výrazně snížil průměrný počet bodů ve hře dvou hráčů oproti algoritmu Lepší počítání karet, zároveň došlo k výraznému posunu v maximálním množství dosažených bodů. Algoritmu Lepší počítání karet se 5× podařilo dosáhnout 23 bodů, ale zde se dokonce 97× povedlo dosáhnout 25 bodů, tj. maximálního počtu bodů.

počet hráčů	2	3	4	5
průměr	14,33	16,37	18,43	16,19
směrodatná odchylka	6,47	4,76	2,38	2,06
maximum	25	24	24	22
počet dosažení maxima	97	5	1	1
průměrný počet tahů	46,67	55,33	60,95	56,27
průměrný počet nápověd	20,36	22,85	25,18	23,7
průměrný počet karet v balíčku na konci hry	11,21	3,81	0,52	0,36

Tabulka 8.5: Výsledky algoritmu Stáří karet a nápověd.

8.2 Evoluce

Nejpokročilejší algoritmus Stáří nápověd a karet jsme dále ještě vylepšili pomocí evolučního algoritmu, který nám pomohl optimalizovat používané parametry. Tyto parametry optimalizované pro různé počty hráčů spolu s parametry, které jsme nastavili pro algoritmus Staří nápověd a karet jsou prezentovány v Tabulce 8.6. Je patrné, že využití stejných parametrů v algoritmu Staří nápověd a karet pro všechny počty hráčů rozhodně nebylo optimální. Je to například dobře vidět na posledních dvou parametrech, které se poměrně logicky snižují s počtem hráčů, jelikož ve vyšším počtu hráčů je více času během jednoho kola dát informaci na důležitou kartu. Nicméně bližší diskuze efektu jednotlivých parametrů by byla relativně náročná. Evoluční algoritmus jsme pustili pro každý počet hráčů 10× a následně jsme všechny vzešlé sady parametrů otestovali na deseti tisíci hrách. Pozorovali jsme, že ačkoliv se jednotlivé sady parametrů vzešlé z evoluce pro stejný počet hráčů velmi výrazně lišily, výsledná skóre byla velmi podobná. Spíše než konkrétní hodnota u jednotlivých parametrů je tedy pravděpodobně důležitá vzájemnou provázanost v celé sadě parametrů. Obecně je asi možné prohlásit, že parametry Vyložení karty z řečených informací a Vyložení karty z Nejistých informací jsou pravděpodobně nejdůležitější, jelikož byly pro všechny sady parametrů vždy relativně vysoké.

	originál	2	3	4	5
vyložení karty z řečených informací	1	0,97	1	0,78	0,77
vyložení karty z *Nejistých informací*	1	0,94	0,94	0,93	0,93
zahození zbytečné karty přísný	1	0,67	1	0,11	1
zahození zbytečné karty jemný	0,1	0,60	0,85	0,23	0,25
zahození bezpečné karty přísný	1	0,48	0,38	0,12	0,4
zahození bezpečné karty jemný	0,1	0,54	0,12	0,73	0,33
zahození karty z *Nejistých informací*	1	1	1	0,22	1
stará číselná nápověda	3	8	5	2	0
stará barevná nápověda	3	10	6	1	0

Tabulka 8.6: Natrénované parametry.

8.2.1 2 hráči

Pro hru dvou hráčů jsme dosáhli nejvýraznějšího zlepšení, téměř pěti bodů, oproti původnímu nastavení parametrů algoritmu Lepší počítání nápověd a karet. Jak je psáno výše, přesná interpretace parametrů je obtížná, ale výsledky evoluce zde naznačují, že pro 2 hráče je výhodné být odvážnější v zahazování karet (Zahození zbytečné karty přísný, Zahození karty, která není nebezpečná přísný). Hráči tedy pravděpodobně častěji zahodí kartu místo toho, než že někde udělají chybu, což se výrazně podepsalo v nižší chybovosti. Průměrný počet karet na konci hry se snížil z 11,21 na 3,27. Zároveň je zajímavé, že u tohoto hráče už je splněn požadavek na maximální počet nápověd nutný k dosažení maximálního skóre, viz Tabulka 3.1, jelikož tento hráč dosahuje 1,14 nápovědy na zahranou kartu. Kompletní výsledky prezentujeme v Tabulce 8.7

průměr	19,02
směrodatná odchylka	4,97
maximum	25
počet dosažení maxima	172
průměrný počet tahů	59,64
průměrný počet nápověd	21,80
průměrný počet karet v balíčku na konci hry	3,27

Tabulka 8.7: Výsledky evoluce pro dva hráče.

8.2.2 3–5 hráčů

Výsledky pro 3–5 hráčů jsou shrnuty v Tabulce 8.8. U všech těchto hráčů došlo ke zlepšení oproti algoritmu Stáří nápověd a karet, ale už nebylo tak významné jako v případě dvou hráčů. Nejvýznamnější bylo pro tři hráče, kde výsledné skóre se zlepšilo téměř o 2,5 bodu. Nicméně jak už jsme psali výše, tak přesný důvod zlepšení těchto hráčů není jednoduché identifikovat. Společné znak pro všechny hráče je, že se bojí méně zahazovat, než námi navržená sada parametrů pro algoritmus Stáří nápověd a karet.

počet hráčů	3	4	5
průměr	18,77	18,94	16,89
směrodatná odchylka	2,99	2,32	2,05
maximum	24	25	22
počet dosažení maxima	16	2	2
průměrný počet tahů	59,05	60,25	55,41
průměrný počet nápověd	23,27	24,30	22,81
průměrný počet karet v balíčku na konci hry	1,01	0,52	0,4

Tabulka 8.8: Výsledky evoluce pro tři až pět hráčů.

8.2.3 Porovnání s literaturou

Podobné ručně psané algoritmy jsou prezentovány v mnoha článcích (Osawa, 2015), (Eger a kol., 2021), (van den Bergh a kol., 2017), (Canaan a kol., 2018). Ve většině těchto studií se nesnažili vytvořit univerzálního algoritmus pro různé počty hráčů, ale převážně se zaměřovali na vytvoření algoritmu pro hru dvou hráčů. Z těchto algoritmů optimalizovaných pro dva hráče byly nejlepší výsledky dosaženy pro algoritmus *Probabilistic*, který dosahoval 18,56 bodu. Jeho fungování je shrnuto v kapitole Současný stav poznání (5.1.1).

Algoritmy pro různé počty hráčů, které jsou prezentovány v literatuře obecně nedosahují výsledků jako námi vyvinutý algoritmus Staří nápověd a karet vytvořený v rámci této práce. Jako příklad uveďme agenta *IGGI* (Walton-Rivers a kol., 2017). Lepších výsledků algoritmy dosáhly, pokud umělá inteligence vybírala ručně napsané akce jiným způsobem, než pomocí předem napsaného algoritmu. Tento algoritmus se autoři snažili nahradit buď pomocí Monte-Carlo Tree Search (agent *Predictor IS-MCTS*) (Walton-Rivers a kol., 2017) nebo evolucí (agent *MirrorSituational*) (Canaan a kol., 2018). Jak je vidět z Tabulky 8.9, naše umělá inteligence, v Tabulce označena jako *Stáří s evolucí*, o hodně překonává ručně napsaný algoritmus i algoritmus využívající Monte-Carlo Tree Search.

Nejlepších výsledků podle našeho průzkumu z ručně psaných umělých inteligencí dosahoval FireFlower bot (pokud nepočítáme implementace hat guessing strategií). Výsledky z tohoto algoritmu bohužel nejsou prezentovány v žádném autorském článku, ale jde pouze o program na Githubu (Wu, 2018). Stručný popis tohoto algoritmu spolu s výsledky je k nalezení pouze v diskuzi obsažené v článku Bard a kol. (2020). Bohužel zde však nejsou publikovány výsledky pro pět hráčů. Výsledky umělé inteligence Fireflower jsou prezentovány v Tabulce 8.9. Je zřejmé, že Fireflower bot výrazně překonává všechny předchozí umělé inteligence, i ty optimalizované pro určitý počet hráčů. Krátký popis, jak Fireflower funguje je k nalezení v kapitole Současný stav poznání (5). Tato umělá inteligence velmi přesně odpovídá lidské hře, jak je popsána v kapitole Startegie ve hře Hanabi (3) a je do ní implementována i většina komplexnějších heuristik, které pravděpodobně stojí za velkou herní úspěšností, a které například námi vytvořené umělá inteligence postrádá.

Pro porovnání v Tabulce 8.9 uvádíme i výsledky zpětnovazebních agentů, konkrétně *ACHA* (Bard a kol., 2020) a *SAD* (Hu a Foerster, 2021). Také uvádíme výsledky hat guessing strategie *WuTheFWasThat* (Wu, 2).

Z výsledků vidíme, že hat guessing je skutečně nejlepší doposud nalezená strategie pro hru Hanabi a ručně psaný algoritmus založený na ní dosahuje nejlepších

výsledků. Zpětnovazební agenti na tuto strategii zatím nepřišly, hrají *standardně*, ale překonávají nejlepší ručně psané algorimty založené na podobném stylu hry.

počet hráčů	2	3	4	5
Stáří s evolucí	19,02	18,77	18,94	16,89
Probabilistic	18,56	—	—	—
IGGI	11,76	11,29	10,71	10,09
Predictor IS-MCTS	8,36	12,14	11,43	11,02
MirrorSituational	20,07	19,58	19,36	18,29
FireFlower	23,37	22,95	22,83	—
ACHA	22,73	20,24	21,57	16,80
SAD	24,01	23,93	23,81	23,01
WuTheFWasThat (hat guessing)	22,52	24,79	24,93	24,92

Tabulka 8.9: Celkové porovnání námi vyvinutého algoritmu s výsledky z literatury.

8.3 Q učení

Pro lepší porovnání dalších výsledků jsme nejdříve nechali v tomto nastavení hrát náhodného hráče, který nám následně složil k porovnání, jestli se naši agenti něco naučili. Výsledky náhodného hráče jsou shrnuty v Tabulce 8.10.

počet hráčů	2
průměr	1,43
směrodatná odchylka	1,13
maximum	6
počet dosažení maxima	5
průměrný počet tahů	6,09
průměrný počet karet v balíčku na konci hry	0,83

Tabulka 8.10: Výsledky Náhodného hraní pro mód Mini Hanabi.

8.3.1 Výsledky různých funkcí odměn

Zkoušeli jsme měnit funkce odměn a sledovali jsme, jak to mění hráčovo naučené chování a jeho výsledky. Kvůli technickým problémům popsaných v Implementaci 7.3.3 jsme nemohli hráče trénovat na příliš mnoho hrách. Proto jsme se rozhodli, že vyzkoušíme několik funkcí odměn vždy na tisíci hrách, a až nastavení, které bude nejlepší, pustíme na pět tisíc her.

V průběhu učení jsme vyzkoušeli několik různých odměn, které jsou spolu s výsledky prezentované v Tabulce 8.11.

První agent se naučil karty spíše zahazovat, jelikož je to bezpečnější, než hrát. Toto chování bylo popsáno ve článku Grooten (2021), který se zabývá vývojem agenta pro hru Hanabi (s tím rozdílem, že trénoval agenta na standardní verzi Hanabi).

V další iteraci nastavování odměn jsme tedy navíc přidali zápornou odměnu za zahození karty, pomocí které jsme se snažili agenta motivovat spíše ke hraní karet. Agent opravdu dosáhl lepších výsledků. Zároveň je z tabulky vidět, že byl ochoten víc riskovat (vykládat karty), jelikož se snížil průměrný počet tahů.

Dalšího agenta jsme zkoušeli motivovat k vykládání karet ne tím, že bychom penalizovali zahození karty, ale snížením penalizace za první nesprávně zahranou kartu. Výsledný účinek to mělo podobný, dokonce silnější než penalizace zahazování. Agent byl ochotný víc vykládat, což mělo sice za následek průměrně dřívější vybuchnutí, ale jeho průměrné skóre se zlepšilo. V dalším textu budeme o tomto hráči mluvit jako o *Třetím*.

U posledního agenta jsme vyzkoušeli ještě víc zvýšit odměnu za zahrání karty a zároveň jsme penalizovali zahazování. Zde už jsme nepozorovali zlepšení, průměrné skóre pro tohoto agenta je už téměř stejné jako předchozího, proto jsme dál nezkoušeli měnit funkce odměn, ale vyzkoušeli u těchto agentů otestovat efekt množství her. V dalším textu budeme o tomto hráči mluvit jako o *Čtvrtém*.

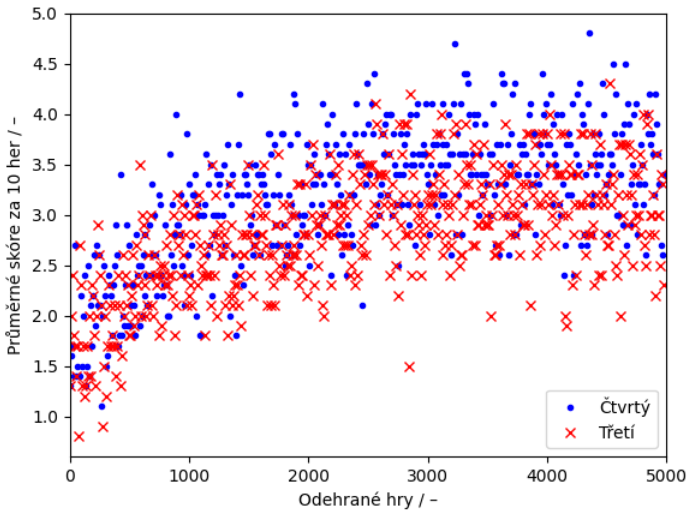
pokus	1.	2.	3.	4.
úspěšné zahrání karty	+5	+5	+5	+20
první neúspěšné zahrání karty	-5	-5	-1	-1
druhé neúspěšné zahrání karty	-10	-10	-10	-10
zahození karty	0	-1	0	-1
průměr	1,71	2,11	2,69	2,76
směrodatná odchylka	0,96	1,04	1,76	1,37
maximum	6	6	6	6
počet dosažení maxima	21	28	149	215
průměrný počet tahů	7,03	6,52	5,68	5,35
průměrný počet karet v balíčku na konci hry	0,43	0,63	1,01	1,18

Tabulka 8.11: Výsledky trénování pro různé odměny.

8.3.2 Trénování na více hrách

Poslední dva agenti, kteří dosáhli během trénovaná na 1000 her téměř shodných výsledků jsme se nechali natrénovat na pěti tisících hrách. Postup jejich učení můžeme vidět na Obrázku 8.1. Každý bod vynesený do grafu jsme získali jako průměr z deseti her, aby byly výsledky lépe čitelné. Průběh učení je pro oba agenty podobný. Oba agenti se poměrně rychle učili v průběhu prvních dvou tisíc her, ale později už průběh učení zpomalil.

V Tabulce 8.12 jsou shrnuty konečné výsledky z hraní již natrénovaných modelů. Je zřejmé, že ačkoliv během prvního tisíce her oba hráči dosáhli téměř stejných výsledků, tak po pěti tisících hrách už se výsledky poměrně význačné liší ve prospěch *Čtvrtého* agenta. A to jak v průměrném dosaženém množství bodů, tak v menší chybovosti (průměrný počet karet v balíčku na konci hry). Je tedy pravděpodobné, že agent, který byl ještě více motivován k hraní se naučil lépe vykládat karty. *Čtvrtý* hráč se oproti náhodnému hráči zlepšil o 2,3 bodu a maximálního skóre dosáhne v 15 % případů.



Obrázek 8.1: Průběh učení agentů.

agent	3.	4.
průměr	3,21	3,77
směrodatná odchylka	1,45	1,54
maximum	6	6
počet dosažení maxima	496	1470
průměrný počet tahů	5,98	6,31
průměrný počet karet v balíčku na konci hry	0,74	0,57

Tabulka 8.12: Výsledky natrénovaných agentů.

Závěr

Tato bakalářská práce se zabývá vývojem umělých inteligencí pro hru Hanabi. V rámci této práce jsme nejprve úspěšně vytvořili implementaci hry Hanabi v jazyce C# a následně jsme opět v jazyce C# vytvořili sedm umělých inteligencí pro tuto hru. Konkrétně šlo o pět ručně psaných umělých inteligencí, pro jednu z nich byly její optimální parametry nalezeny pomocí evolučního algoritmu a jednu umělou inteligence založenou na hlubokém Q učení. Jednotlivé ručně psané inteligence vznikaly od nejjednodušší po nejsložitější s tím, že novější verze vždy přímo vychází ze starší verze. Postupně tedy vznikly algoritmy:

- Náhodné hraní.
- Téměř náhodné hraní, ve kterém bylo implementováno hraní pouze takových karet, které je na 100 % možné vynést.
- Napovídání, ve kterém bylo implementováno napovídání na karty, které je možné vynášet.
- Lepší počítání karet, ve kterém bylo implementováno to, aby hráči o svých kartách získávali veškeré informace, které je možné ze hry získat a využívali je k určení svých karet.
- Staří nápověd a karet, ve kterém je implementováno to, že hráči berou v potaz, před jakou dobou jim byla dána jaká nápověda a využívají tyto informace k efektivnějšímu určení vlastních karet. Tento relativně komplexní algoritmus už obsahuje řadu parametrů, které určují jeho hraní. V první fázi byly tyto algoritmy určeny ručně námi a následně byly optimalizovány pomocí evolučního algoritmu.

Z výsledků ukazujeme, že algoritmus Náhodné hraní dosahuje přibližně 1,2 bodu z 25 možných a vždy relativně brzo vybouchne, kvůli vykládání chybných karet. Algoritmus Téměř náhodné hraní dosažené skóre příliš nevylepšil a dosáhl skóre okolo 4 bodů. Významné zlepšení přineslo až algoritmus napovídání, které dosahovalo skóre okolo 15 bodů. Tyto výsledky byly dále zlepšeny díky algoritmu Lepší počítání karet zhruba o jeden bod. Algoritmus Stáří karet a nápověd, přinesl zhoršení ve hře dvou hráčů kvůli relativně velké chybovosti, ale pro vyšší počty hráčů dosáhl zlepšení, kde pro 4 hráče získal až 18.5 bodů. Pomocí evolučního algoritmu byl algoritmus Stáří karet a nápověd dále vylepšen a pro 2-4 hráče dosáhl okolo 19 bodů. Pro 5 hráčů sice dosáhl tento algoritmus necelých 17 bodů, ale i tak jde o nejvyšší hodnotu prezentovanou v literatuře pro ručně psané umělé inteligence bez využití hat guessing strategií.

Algoritmus hlubokého Q učení jsme nakonec pustili jen na výrazně zmenšené a zjednodušení verzi Hanabi. Šlo především o to, že v plné verzi by na nedostatečně výkonném počítači trvalo učení příliš dlouho a toto učení by ještě bylo výrazně prodlouženo faktem, že jsme v algoritmu hlubokého Q učení měli problémy s meamory leakem, který se bohužel nepovedlo odstranit. Nicméně na pro zmenšené a zjednodušené verzi Hanabi se nám úspěšně povedlo natrénovat agenta, který po 5000 učících hrách dosahoval 3,8 bodu z 6 možných.

Práci je možné rozšiřovat několika způsoby. Mohli bychom pokračovat vylepšováním ručně psaných algoritmů, například implementací pokročilejších strategií nebo lepší prací se získanými informacemi. Dále by bylo možné odstranit problémy se zpětnovazebním učením, což by dovolovalo trénovat agenty i na standardní verzi Hanabi. Také lze například přidat grafické rozhraní, aby bylo možné hru hrát i pro člověka. Díky tomu by nejen bylo pohodlnější analyzovat průběh hry, ale bylo by i možné vyvíjet takové algoritmy, se kterými jsou schopni dobře hrát lidé.

Seznam použité literatury

Baffier, J.-F., Chiu, M., Diez, Y., Korman, M., Mitsou, V., van Renssen, A., Roeloffzen, M. a Uno, Y. (2016). Hanabi is np-complete, even for cheaters who look at their cards. volume 49 of *Leibniz International Proceedings in Informatics (LIPIcs)*. doi: 10.4230/LIPIcs.FUN.2016.4.

Bajaj, P. (2023). Reinforcement learning. *GeeksforGeeks*. URL `https://www.geeksforgeeks.org/what-is-reinforcement-learning/`. Navštíveno 5. května 2023.

Bard, N., Foerster, J. N., Chandar, S., Burch, N., Lanctot, M., Song, H. F., Parisotto, E., Dumoulin, V., Moitra, S., Hughes, E., Dunning, I., Mourad, S., Larochelle, H., Bellemare, M. G. a Bowling, M. (2020). The hanabi challenge: A new frontier for AI research. *Artificial Intelligence*, **280**, 103216.

Bauza, A. (2010). Hanabi. URL `https://www.rexhry.cz/sites/default/files/instructions/hanabi-pravidla.pdf`. Navštíveno 5. května 2023.

Bhatt, S. (2019). Reinforcement Learning 101 - Towards Data Science. URL `https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292`. Navštíveno 5. května 2023.

Brown, N. a Sandholm, T. (2018). Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, **359**(6374), 418–424. doi: 10.1126/science.aao1733. URL `https://www.science.org/doi/abs/10.1126/science.aao1733`.

Canaan, R., Shen, H., Torrado, R., Togelius, J., Nealen, A. a Menzel, S. (2018). Evolving agents for the hanabi 2018 cig competition. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. doi: 10.1109/CIG.2018.8490449.

Chatain, O. (2016). *Cooperative and Non-cooperative Game Theory*, pages 1–3. Palgrave Macmillan UK, London. ISBN 978-1-349-94848-2. doi: 10.1057/978-1-349-94848-2_468-1. URL `https://doi.org/10.1057/978-1-349-94848-2_468-1`.

Cheng, S.-F., Reeves, D., Vorobeychik, Y. a Wellman, M. (2004). Notes on equilibria in symmetric games. In *Proceedings of the 6th International Workshop On Game Theoretic And Decision Theoretic Agents GTDT 2004*, pages 71–78.

Cox, C., Silva, J. D., Deorsey, P., Kenter, F. H. J., Retter, T. a Tobin, J. (2015). How to make the perfect fireworks display: Two strategies for hanabi. *Mathematics Magazine*, **88**(5), 323–336. doi: 10.4169/math.mag.88.5.323. URL `https://doi.org/10.4169/math.mag.88.5.323`.

Davis, M. D. a Brams, S. J. (2023). game theory. *Encyklopedia Britannica*. URL https://www.britannica.com/science/game-theory. Navštíveno 5. května 2023.

Eger, M. a Gruss, D. (2019). Wait a second: Playing hanabi without giving hints. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, FDG '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450372176. doi: 10.1145/3337722.3337744. URL https://doi.org/10.1145/3337722.3337744.

Eger, M., Martens, C. a Cordoba, M. A. (2017). An intentional ai for hanabi. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 68–75. doi: 10.1109/CIG.2017.8080417.

Eger, M., Martens, C., Chacón, P. S., Córdoba, M. A. a Hidalgo-Cespedes, J. (2021). Operationalizing intentionality to play hanabi with human players. *IEEE Transactions on Games*, **13**(4), 388–397. doi: 10.1109/TG.2020.3009359.

FAIR, Bakhtin, A., Brown, N., Dinan, E., Farina, G., Flaherty, C., Fried, D., Goff, A., Gray, J., Hu, H., Jacob, A. P., Komeili, M., Konath, K., Kwon, M., Lerer, A., Lewis, M., Miller, A. H., Mitts, S., Renduchintala, A., Roller, S., Rowe, D., Shi, W., Spisak, J., Wei, A., Wu, D., Zhang, H. a Zijlstra, M. (2022). Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, **378**(6624), 1067–1074. doi: 10.1126/science.ade9097. URL https://www.science.org/doi/abs/10.1126/science.ade9097.

Fei Tong, Masahiro Ichiki, K. N. (2020). Playing mini-hanabi card game with q-learning. volume 2020, pages 41–42.

Foerster, J. N., Song, F., Hughes, E., Burch, N., Dunning, I., Whiteson, S., Botvinick, M. a Bowling, M. (2019). Bayesian action decoder for deep multi-agent reinforcement learning.

Fpvandoorn (2020). URL https://github.com/fpvandoorn/hanabi/blob/master/doc_hat_player.md. Navštíveno 5. května 2023.

Granter, S. R., Beck, A. H. a Papke, David J., J. (2017). AlphaGo, Deep Learning, and the Future of the Human Microscopist. *Archives of Pathology Laboratory Medicine*, **141**(5), 619–621. ISSN 0003-9985. doi: 10.5858/arpa.2016-0471-ED. URL https://doi.org/10.5858/arpa.2016-0471-ED.

Grooten, B. (2021). Deep reinforcement learning for the cooperative card game hanabi. Master's thesis, Eindhoven University of Technology.

Hanabi reference for bga (2023). URL https://docs.google.com/document/d/1VzgN6WoeYwh5NYtHECzUNZqFvpVcg-EdObU-YNMlyhc/edit. Navštíveno 5. května 2023.

Hansen, N., Arnold, D. a Auger, A. (2015). *Evolution Strategies*. doi: 10.1007/978-3-662-43505-2_44.

Hu, H. a Foerster, J. N. (2021). Simplified action decoder for deep multi-agent reinforcement learning.

Mahajan, A. (2022). Using intuitive behavior models to rapidly adapt to and work with human teammates in hanabi. Master's thesis, Pittsburg University.

Matsubara, H., Iida, H., Grimbergen, R. a Laboratory, E. (1996). Chess, shogi, go, natural developments in game research. *ICCA*, **19**, 103–112. doi: 10.3233/ICG-1996-19208.

Osawa, H. (2015). Solving hanabi: Estimating hands by opponent's actions in cooperative game with incomplete information. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Pete (2019). Alphazero crushes stockfish in new 1,000-game match. URL `https://www.chess.com/news/view/updated-alphazero-crushes-stockfish-in-new-1-000-game-match`. Navštíveno 5. května 2023.

Pilát, M. (2023a). Hluboké zpětnovazební učení. URL `http://ktiml.mff.cuni.cz/~pilat/cs/prirodou-inspirovane-algoritmy/hluboke-zpetnovazebni-uceni/`. Navštíveno 5. května 2023.

Pilát, M. (2023b). Evoluční algoritmy - jednoduchý genetický algoritmus, operátory, fitness. URL `http://ktiml.mff.cuni.cz/~pilat/cs/prirodou-inspirovane-algoritmy/evolucni-algoritmy-uvod/`. Navštíveno 5. května 2023.

Pilát, M. (2023c). Zpětnovazební učení - q-učení, sarsa, multi-agentní zpětnovazební učení. URL `http://ktiml.mff.cuni.cz/~pilat/cs/prirodou-inspirovane-algoritmy/zpetnovazebni-uceni/`. Navštíveno 5. května 2023.

Proai (2020). Deep Blue Algorithm: A Detailed Guide. *www.professional-ai.com*. URL `https://www.professional-ai.com/deep-blue-algorithm.html`. Navštíveno 5. května 2023.

Rinne, A. (2023). Github - scisharp/tensorflow.net: .net standard bindings for google's tensorflow for developing, training and deploying machine learning models in c and f. URL `https://github.com/SciSharp/TensorFlow.NET`. Navštíveno 6. května 2023.

Russell, S. a Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition.

Sutton, R. S. a Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA. ISBN 0262039249.

Tesauro, G. a Sejnowski, T. (1989). A parallel network that learns to play backgammon. *Artificial Intelligence*, **39**, 357–390. doi: 10.1016/0004-3702(89)90017-9.

van den Bergh, M. (2015). Hanabi, a co-operative game of fireworks. Bachelor's Thesis, University of Leiden.

van den Bergh, M. J. H., Hommelberg, A., Kosters, W. A. a Spieksma, F. M. (2017). Aspects of the cooperative card game hanabi. In Bosse, T. a Bredeweg, B., editors, *BNAIC 2016: Artificial Intelligence*, pages 93–105, Cham, 2017. Springer International Publishing. ISBN 978-3-319-67468-1.

Walton-Rivers, J., Williams, P. R., Bartle, R., Perez-Liebana, D. a Lucas, S. M. (2017). Evaluating and modelling hanabi-playing agents. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1382–1389. doi: 10.1109/CEC.2017.7969465.

Wu, D. (2018). A rewrite of hanabi-bot in scala. URL `https://github.com/lightvector/fireflower`. Navštíveno 5. května 2023.

Wu, J. (2). State of the art hanabi bots + simulation framework in rust. URL `https://github.com/WuTheFWasThat/hanabi.rs`. Navštíveno 5. května 2023.

Yao, D. (2022). 25 years ago today: How Deep Blue vs. Kasparov changed AI forever. *AI Business*. URL `https://aibusiness.com/ml/25-years-ago-today-how-deep-blue-vs-kasparov-changed-ai-forever`. Navštíveno 5. května 2023.

Zamiell (2023). The hat-guessing convention framework. URL `https://github.com/hanabi/hanabi.github.io/blob/main/misc/hat-guessing.md`. Navštíveno 9. května 2023.

Seznam obrázků

Seznam tabulek

A. Přílohy

Příloha elektronické verze obsahuje následující složky:

- Code – Tato složka obsahuje soubor Hanabi.sln a Hanabi.exe spolu s podsložkou obsahující zdrojové soubory.
- EvolutionResults – Pro každý počet hráčů obsahuje podsložku s výsledky evoluce zmíněné v textu.