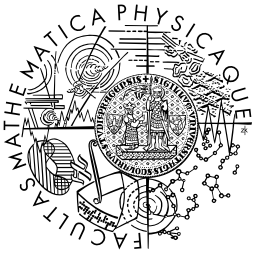


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

**BAKALÁŘSKÁ PRÁCE**

Daniel Novák

# Osadníci z Katanu

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Jan Hric

Studijní program: Informatika

Studijní obor: Obecná informatika

Praha 2020

Prohlašuji, že jsem tuto bakalářskou práci vypracoval 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

Rád bych poděkoval vedoucímu práce RNDr. Janu Hricovi za pomoc při výběru tématu práce a následné rady při jejím vypracování.

Dále bych rád poděkoval své rodině, která mi pomohla výsledný program doladit a otestovat.

Název práce: Osadníci z Katanu

Autor: Daniel Novák

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Jan Hric, Katedra teoretické informatiky a matematické logiky

Abstrakt: V této práci se zabýváme implementací stolní hry Osadníci z Katanu a umělé inteligence hrající tuto hru. Umělá inteligence je založena na kombinaci expectimaxu a zpětnovazebního učení. S využitím zpětnovazebního učení se nám podařilo vyvinout agenta, který zvládá rozumně hrát. Strategii naučenou zpětnovazebním učením jsme úspěšně vylepšili využitím expectimaxu. Výsledný agent je schopný vyhrát proti průměrnému lidskému hráči.

Klíčová slova: umělá inteligence, zpětnovazební učení, expectimax, hra

Title: Settlers of Catan

Author: Daniel Novák

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Jan Hric, Department of Theoretical Computer Science and Mathematical Logic

Abstract: In this thesis, we work on implementation of the board game Settlers of Catan and artifitial intelligence playing this game. The artificial intelligence is based on a combination of expectimax and reinforcement learning. Using reinforcement learning, we have been able to develop an agent who can play reasonably. We managed to improve the policy learned by reinforcement learning using expectimax. The resulting agent is able to win aganist average human player.

Keywords: artificial intelligence, reinforcement learning, expectimax, game

# Obsah

# Úvod

## 0.1 Co jsou Osadníci z Katanu

Osadníci z Katanu je stolní hra navržená Klausem Teuberem a poprvé vydaná roku 1995 v Německu. U nás hru vydává společnost Albi.

Hra je velmi úspěšná, získala mnoho ocenění po celém světě a bylo k ní vytvořeno mnoho rozšíření. V této práci se budeme věnovat pouze základní verzi.

Hráči se dostávají do role osadníků nového ostrova a snaží se ho osídlit svými vesnicemi a městy. V okolí svých vesnic mohou těžit suroviny, které se v dané oblasti nachází. Za získané suroviny mohou hráči stavět silnice a další vesnice a města, nebo kupovat akční karty. Důležitou součástí hry je také obchod se surovinami. Suroviny může hráč měnit buď s jiným hráčem, nebo s obchodníky v přístavu.

## 0.2 Cíl práce

Cílem této práce je naprogramovat prostředí pro hru Osadníci z Katanu, ve kterém proti sobě budou moci hrát na jednom počítači jak lidští hráči, tak umělá inteligence.

Prostředí bude naprogramováno tak, aby co nejpřesněji odpovídalo oficiálním pravidlům přiloženým k této práci.

Umělá inteligence počítačových hráčů bude založena na kombinaci expectimaxu a zpětnovazebního učení.

Expectimax je metoda založená na prohledávání stavového prostoru. Protože jsou Osadníci z Katanu hra s obrovským množstvím stavů, je expectimax sám o sobě nepoužitelný. V naší práci se budeme snažit expectimax využít ke zpřesnění odhadu hodnoty stavu naučeného za pomoci zpětnovazebního učení.

## 0.3 Související práce

Protože jsou Osadníci z Katanu poměrně populární hra, existuje mnoho prací věnujících se samotné hře i umělé inteligenci k jejímu hraní. Můžeme zmínit například bakalářskou práci věnující se umělé inteligenci s využitím genetických algoritmů (Kratochvíl, 2016). V naší práci se zaměříme na metodu zpětnovazebního učení. Tomu se také věnuje například Pfeiffer (2004), který použil hierarchické zpětnovazební učení s modelovými stromy jako aproximační funkcí. V naší práce využijeme jednodušší formu zpětnovazebního učení, navíc se ale zaměříme také na jeho propojení s expectimaxem.

## 0.4 Struktura práce

V prvních kapitolách popíšeme základní principy umělé inteligence použité v práci. Následně popíšeme strukturu programu napsaného v jazyce C++. V další

kapitole vysvětlíme, jakým způsobem může uživatel program využívat. Nakonec ukážeme několik testů, ve kterých zhodnotíme úspěšnost naší umělé inteligence.

# 1. Umělá inteligence

V této kapitole si zatím jenom obecně představíme, čím se zabývá obor umělé inteligence, a ve stručnosti si představíme její podobory, které použijeme dále v naší práci a kterými se budeme podrobněji zabývat v dalších kapitolách.

Umělá inteligence je velmi široký obor informatiky obsahující mnoho podoblastí. Ty všechny spojuje snaha porozumět, napodobit a případně vylepšit inteligentní chování člověka při řešení složitých problémů, jako je například plánování, rozpoznání obrazu nebo přirozeného jazyka, zpracování velkého objemu dat, nebo třeba hraní Osadníků z Katanu. K tomu využívají různé metody od procházení stavového prostoru přes hledání shluků na základě podobnosti až po statistické metody.

Mezi základní pojmy z oboru umělé inteligence, které budeme nadále používat, patří pojmy agent a prostředí.

## 1.1 Agent

Agent je cokoliv, co vnímá okolní pomocí prostřednictvím senzorů a ovlivňuje ho prostřednictvím akčních členů (Russell a Norvig, 2009). Obvykle tedy agent zvolí nějakou akci a předá ji prostředí. Prostředí na základě obdržené akce změní svůj stav a o této změně informuje agenta. Agent rozhodne o své další akci na základě nového stavu prostředí a celý proces se opakuje.

V našem případě je agentem hráč, který může být reprezentovaný člověkem nebo počítačem. Agent svými „senzory“ vidí figurky rozmístěné na hrací ploše, vidí vlastní karty a veřejně viditelné informace o protihráčích. Agent může prostředí ovlivňovat akcemi jako je hod kostkou, stavba, obchod nebo použití akční karty.

## 1.2 Prostředí

Prostředí, ve kterých se agent pohybuje, mohou mít různé vlastnosti. Na jejich základě potom můžeme zvolit vhodný algoritmus pro řešení problému. V následujících odstavcích si představíme různá dělení tak, jak je popisuje Russell a Norvig (2009) a určíme, jaké vlastnosti mají Osadníci z Katanu.

Prostředí může být buď **plně pozorovatelné** nebo **částečně pozorovatelné**. Jak vyplývá z názvu, plně pozorovatelné prostředí znamená, že agent svými senzory vidí celý stav, oproti tomu v částečně pozorovatelném prostředí jsou informace, které agent nemůže svými senzory vidět. Protože v Osadnících z Katanu mají hráči karty se surovinami a akční karty, do kterých protivníci nevidí, je tato hra příkladem částečně pozorovatelného prostředí.

Prostředí se mohou lišit počtem agentů, kteří se v něm pohybují. Osadníci jsou hra pro tři až čtyři hráče, prostředí je proto **multi-agentní**. To můžeme dále dělit na **kooperativní** a **kompetitivní**. Výhra jednoho hráče v Osadnících z Katanu znamená prohra těch ostatních, jedná se tedy o kompetitivní multi-agentní prostředí.

Dále můžeme prostředí dělit na **deterministická** a **stochastická**. Deterministická prostředí jsou taková, ve kterých je následující stav plně určen současným stavem a akcí agenta. To ale u Osadníků z Katanu neplatí, protože například při hodu kostkou může padnout pokaždé jiné náhodné číslo a stejná akce tedy může vést do několika různých stavů. Jedná se tedy o stochastické prostředí.

Prostředí může být **epizodické** nebo **sekvenční** v závislosti na tom, jestli výběr akce agentem závisí na jeho předchozích akcích. Protože v Osadnících agenti navazují na své předchozí akce dokud jeden z nich nevyhraje, jedná se o sekvenční prostředí. [1]

Prostředí dále dělíme na **statická** a **dynamická**. Dynamická prostředí jsou taková, která se mohou měnit zatímco agent přemýšlí. Pokud se prostředí s časem nemění, ale mění se míra výkonu agenta, řekneme, že je prostředí semi-dynamické. Protože v Osadnících se prostředí mění pouze když některý z agentů provede akci a agenti se pravidelně střídají, jedná se o statické prostředí.

Prostředí může být **diskrétní** nebo **spojité**. Protože Osadníci z Katanu mají konečné množství stavů a agenti konečné množství možných akcí, jedná se o diskrétní prostředí.

## 1.3 Metody použité v práci

Cílem práce je sestrojit agenta, který bude hrát hru Osadníci z Katanu za pomocí využití klasických metod, které se zakládají na prohledávání stavového prostoru a mezi které patří expectimax, a zpětnovazebního učení.

Expectimax je stochastickým rozšířením algoritmu minimax. To je metoda, která je ve své základní verzi určena pro deterministické hry dvou hráčů, kteří se pravidelně střídají dokud jeden z nich nevyhraje a nic před sebou neskrývají. Jinými slovy pro plně pozorovatelné, multi-agentní, deterministická, sekvenční, statická a diskrétní prostředí. Osadníci z Katanu několik z těchto vlastností nesplňují. Minimax se ale dá použít s různými úpravami, které si popíšeme v kapitole 2, i na jiné druhy problémů.

Minimax vygeneruje strom hry, tedy takový strom, ve kterém uzly reprezentují stavy hry a hrany jsou mezi uzly těch stavů, mezi kterými existuje nějaká akce. Minimax tento strom prohledává a najde takovou cestu, která vede k co nejlepšímu výsledku pro hráče, tedy k výhře. Takový algoritmus dokáže najít optimální strategii.

Protože jsou ale Osadníci z Katanu poměrně složitá hra, její kompletní strom je příliš velký na to, aby se vešel do paměti a aby se dal v rozumném čase prohledat. V některých tazích, například při obchodování nebo při použití karty „Stavba silnic“, může agent vybírat se stovek různých akcí. V určité hloubce musíme prohledávání ukončit a hodnotu daného stavu nějakým způsobem odhadnout. Na tomto odhadu bude záviset kvalita rozhodování agenta.

Odhad můžeme provést mnoha způsoby. Nejjednodušším způsobem je přiřadit hodnoty (váhy) nějakým vlastnostem stavu (příznakům) a hodnoty těch

[1] Z hlediska jedné hry jde o sekvenční prostředí. Později v kapitole 3, kdy se na problém budeme dívat z pohledu zpětnovazebního učení, řekneme, že jde o epizodický problém, neboť po skončení hry (epizody) začne nová epizoda, na kterou nemá výsledek předchozí hry žádný vliv.

vlastností, které daný stav má, sečíst. Může se hodnotit například počet vesnic hráče nebo pravděpodobnost, se kterou hráč při hodu kostkou získá surovinu.

Protože váhy jednotlivých příznaků neznáme, musíme je nějakým způsobem odhadnout. První možností je, že odhadneme váhy sami a napevno je napíšeme agentovi. S největší pravděpodobností se nám ale tímto způsoben nepodaří získat optimální váhy. Lepší způsob je využít metod strojového učení tak, aby agent na ty správné váhy přišel sám. Podrobněji je popíšeme v kapitole 3.

Hlavní myšlenka tedy nespočívá ve využití expectimaxu jako hlavního nosného algoritmu umělé inteligence, ale spíše jako podpůrného algoritmu, který nám umožní podívat se o několik kroků do budoucnosti a usnadnit tak odhadování hodnoty stavu.

# 2. Expectimax

V této kapitole si stručně popíšeme, co je to expectimax a jakým způsobem je potřeba ho upravit pro problém Osadníků z Katanu.

## 2.1 Minimax

Jak již bylo řečeno v minulé kapitole, minimax je ve své čisté podobě algoritmus vhodný k aplikaci na deterministické hry pro dva hráče s úplnou informací a s nulovým součtem, to znamená, výhra jednoho hráče je „vyvážena“ prohrou druhého hráče. Dále předpokládáme, že počet stavů dané hry není příliš velký.

Minimax nejprve vybuduje kompletní strom hry. V kořeni bude počáteční stav, v listech koncové stavy, tedy ty stavy, ve kterých nějaký hráč vyhrál. Z každého stavu vedou hrany reprezentující akce agenta, který je právě na tahu, a vedoucí do stavů, do kterých se může agent svými akcemi dostat.

Listy stromu jsou ohodnoceny podle toho, jestli v nich hráč, kterého agent reprezentuje, vyhraje nebo prohraje. Stavy, ve kterých je agent na tahu ohodnotíme maximem z potomků daného stavu protože počítáme s tím, že agent zvolí tu nejvýhodnější akci. Naopak hodnotu stavů reprezentujících protivníkovy tahy vypočteme jako minimum z jejich potomků.

Agent potom v každém tahu volí akci vedoucí do stavu s co nejvyšším ohodnocením. Kdyby Osadníci z Katanu splňovali výše popsané podmínky, byla by taková strategie optimální. Vzhledem k tomu, že se Osadníci z Katanu v některých vlastnostech liší, budeme muset algoritmus upravit a tím o optimalitu přijdeme.

## 2.2 Minimax pro hru více hráčů

Nejprve zobecníme minimax pro hru více hráčů, kterou Osadníci z Katanu jsou. Rozšíření je na první pohled přímočaré. Akce prvního protivníka nepovedou do stavů, ve kterých táhne agent, ale do stavů dalšího protivníka. Protože už ale nemusí znamenat, že prohra hráče znamená určitě výhru protihráče, nemusí se nutně jednat o minimum a můžeme na problém pohlížet více způsoby.

Prvním způsobem je vybírat takový tah, který je nejvýhodnější vždy pro daného protihráče, tedy snažit se hodnotit tahy z pohledu protihráče a vybírat maximum z tohoto ohodnocení. Tato možnost nejpřesněji simuluje reálné chování hráčů. Protože ale jsou Osadníci z Katanu hra s neúplnou informací, není jednoduché ohodnotit stav z protihráčova pohledu.

Druhým způsobem je očekávat, že cílem všech protihráčů není jejich výhra, ale agentova prohra, a očekávat vždy ten nejpesimističtější výsledek. Potom můžeme nadále počítat minimum stejným způsobem, jakým jsme to dělali u dvou hráčů.

Ať zvolíme jakoukoli variantu, půjde vzhledem k neúplné informaci vždy jenom o aproximaci. V naší implementaci používáme druhý způsob, tedy počítáme minimum z pohledu agenta, protože je jednodušší výpočetně i implementačně a nedokážeme s jistotou říct, že by některý ze způsobů byl lepší.

## 2.3 Minimax pro stochastické problémy

Pro problémy s náhodnými tahy můžeme algoritmus rozšířit přidáním uzlů náhody. Pokud agent zvolí akci, která je náhodná, dostane se do speciálního vrcholu, který reprezentuje ten samý stav prostředí, jako předchozí vrchol. Jeho potomky ale jsou všechny stavy, do kterých může vést daná stochastická akce. Hodnotu tohoto vrcholu nepočítáme jako maximum ani minimum z potomků, ale spočítáme střední hodnotu potomků. Takto upravenému minimaxu říkáme **expectimax**.

### 2.3.1 Reprezentace akcí

Kromě vyššího větvení a tedy i náročnějšímu výpočtu přináší toto rozšíření pro Osadníky z Katanu ještě jednu komplikaci. Protože některé náhodné akce, jako například hod kostkou, mohou dělit tah hráče na dvě a více částí, není možné počítat s celým tahem hráče jako s jednou akcí. Nabízí se několik možností, jak reprezentovat akce. První možností je počítat s elementárními tahy, jako jsou hod kostkou, stavba budov nebo obchod, ze kterých se skládá celý hráčův tah. Druhou možností je uvažovat vždy co nejdelší posloupnosti těchto elementárních tahů, které jsou ukončeny vždy koncem hráčova tahu nebo náhodnou akcí. Další možností je počítat opravdu s celými tahy a při každé náhodné akci promyslet dopředu všechny možnosti.

V naší práci používáme dva různé přístupy. V první fázi vybere agent množinu tahů, mezi kterými se rozhoduje. Pro reprezentaci těchto tahů jsme až na výjimky zvolili třetí zmiňovaný způsob, protože umožňuje provést následně celý tah najednou bez nutnosti opakovaného počítání. Jednu výjimku tvoří případy, kdy není na výběr mezi více akcemi. Například při povinném hodu kostkou je lepší tuto akci provést rovnou a teprve poté vypočítat zbytek tahu. Druhou výjimkou je, když je prohledávání ukončeno dříve dosažením hloubky prohledávání.

Ve druhé fázi, tedy při hodnocení expectimaxem, jsme zvolili druhou možnost, tedy posloupnosti elementárních akcí ukončené koncem tahu nebo náhodnou akcí. Jednak už si nepotřebujeme nic pamatovat, zajímá nás pouze hodnota stavu a jednak může být výhodné rozdělit tah na více částí, pokud by jeho prohledávání trvalo příliš dlouho.

Agent tedy v první fázi získá strom elementárních akcí reprezentujících tah. Hodnotu tohoto stromu získá jako střední (očekávanou) hodnotu stavů získaných průchodem každé větve stromu (tedy stavů v listech). K ohodnocení těchto stavů využijeme expectimax[1], který dále plánuje akce do budoucna. Tentokrát pro nás ale není užitečné, aby tah nebyl rozdělený do menších částí. Proto tady uvažujeme tah jako posloupnost elementárních akcí ukončených náhodnou akcí.

## 2.4 Evaluační funkce

Na herní mapě je 54 různých míst, kam můžou hráči umisťovat své vesnice nebo města, která mohou být různými způsoby propojena silnicemi, zároveň můžou mít v ruce až 25 akčních karet 4 různých druhů a k tomu skoro libovolný

[1]Expectimax jako myšlenku výběru akce s nejvyšší hodnotou a počítání střední hodnoty u náhodných akcí využíváme už v první fázi, ale teprve tady přímo zavoláme funkci expectimax.

počet surovin[2]. Hra má tedy opravdu mnoho stavů, které se nemohou všechny vejít do paměti a z časových důvodů není ani možné v každém tahu generovat celý strom znovu při prohledávání do hloubky.

Pro takové velké problémy, kdy není možné projít celý herní strom, můžeme výpočet v určené hloubce ukončit a odhadnout hodnotu stavů nějakou evaluační funkcí. V případě této práce použijeme funkci naučenou pomocí zpětnovazebního učení, kterému se budeme věnovat v kapitole 3.

## 2.5 Hra s neúplnou informací

Protože v Osadnících z Katanu drží hráči v ruce karty, nemůžeme přesně určit potomky stavů protihráčových tahů. Vzhledem k tomu, že hodnoty stejně budou následně odhadnuty, není důležité složitě počítat pravděpodobnosti toho, jaké může mít protivník v ruce akční karty a suroviny. Protože vidíme alespoň počet karet surovin a akčních karet v protihráčově ruce, budeme pro jednoduchost předpokládat, že protihráč má v ruce vždy právě ty suroviny, které se mu hodí. Zároveň můžeme některé protivníkovy akce zcela vynechat. Důležité pro agentovo rozhodování je hlavně kam můžou protihráči umisťovat své silnice a vesnice.

## 2.6 Optimalizace a zjednodušování problému

Takto upravený algoritmus je už na Osadníky z Katanu použitelný. Protože počítáme s tím, že strom bude opravdu obrovský a prohledávání náročné, budeme se snažit si práci co nejvíce ulehčit. V této sekci si ukážeme různé způsoby, jakými jsme se snažili problém zjednodušit tak, abychom se mohli podívat o něco dál do budoucnosti třeba za cenu toho, že se můžeme v našem odhadu zmýlit.

### 2.6.1 Alfa-beta ořezávání

Nejčastější optimalizací minimaxu, respektive expectimaxu, je **alfa-beta ořezávání**, který podrobněji popisují například Russell a Norvig (2009).

Pro každý vrchol si navíc budeme pamatovat dvě hodnoty. Pomocí nich průběžně odhadujeme nejvyšší a nejnižší možné hodnoty stavu a pokud usoudíme, že hodnota právě počítaného stavu nemůže být dostatečně vysoká (nebo nízká), nemusíme už zbytečně počítat hodnoty zbylých potomků daného stavu. Tato optimalizace není příliš efektivní pro mělké stromy, se kterými budeme pracovat, její implementace nás ale nic nestojí a může alespoň trochu pomoci.

### 2.6.2 Omezování větvení

Největší překážkou pro využívání expectimaxu u Osadníků z Katanu je obrovský počet stavů hry a s ním související obrovské větvení. V některých stavech může agent volit až stovky různých tahů, z nichž některé mohou vést náhodně do různých stavů. Naší snahou bude omezit větvení a prohledávat tak strom expectimaxu více do hloubky. Toho můžeme dosáhnout například prohledáváním

[2]Oproti stolní verzi hry není počet surovin omezen. S omezením surovin by se ale počet možností příliš nezměnil.

pouze těch nejslibnějších větví stromu, limitováním možných akcí nebo zanedbáním některých méně podstatných informací.

### Dvojí hodnocení stavů

V každém stavu získáme množinu tahů, které může agent zvolit. Vygenerované tahy se skládají z posloupností elementárních akcí rozdělených náhodnými akcemi, viz sekce 2.3.1. Omezit množství tahů, které se budou dále vyhodnocovat expectimaxem, můžeme dvěma způsoby.

Prvním způsobem je hodnotit jednotlivé elementární akce, ze kterých tahy skládáme. Například při umisťování vesnic vybereme pouze ty akce, které staví vesnice na těch nejlepších křižovatkách. Posloupnosti potom skládáme z vytříděných elementárních akcí.

Druhou možností je vytvořit z elementárních akcí posloupnosti, teprve ty ohodnotit nějakou okamžitou ohodnocující funkcí a vybrat z nich ty lepší. Tady můžeme dále rozlišovat, jestli vybíráme skutečně tahy, mezi kterými se agent rozhoduje, nebo tahy protihráčů, které agent se agent snaží odhadnout v rámci expectimaxu. Můžeme tak zachovat vyšší větvení v místech, kde nám jde o výběr nejlepšího tahu a v místech, kde pouze odhadujeme hodnotu předchozího stavu.

V naší práci využíváme oba přístupy. Každý agent může mít počty nejlepších akcí a tahů zvolené individuálně pomocí parametrů popisovaných v sekci 5.5.

### Pevné pořadí akcí v tahu

V dalších odstavcích se podíváme na zjednodušení, která jsou méně obecná a jsou vázána na Osadníky z Katanu.

Do jednoho stavu se můžeme dostat kombinováním podobných akcí v různém pořadí. Na pořadí akcí nám ale ve výsledku nezáleží a bez pevného pořadí akcí v rámci tahu by se stávalo, že bychom mohli hodnotit některé stavy vícekrát. Navíc je implementačně jednodušší přidávat akce do tahu postupně. Je ale důležité promyslet si jejich pořadí, neboť na sebe mohou jednotlivé akce mít vliv.

V naší práci jsme zvolili následující pořadí. Na začátku může agent použít akční kartu. To může učinit nejvýše jednou za tah, může používat pouze karty, které už má z minulého kola a navíc může s pomocí karet získat suroviny pro další akce. Následně musí agent hodit kostkou, pokud ještě neházel. V další fázi může agent obchodovat, čímž může získat další suroviny potřebné pro budoucí akce. V poslední fázi může stavět silnice, vesnice a města (v tomto pořadí). Silnice mohou odemknout nová místa pro stavbu vesnic a ty mohou odemknout nová místa pro stavbu měst. Nakonec může hráč nakoupit akční karty, které může používat až od dalšího kola.

### Obchod s hráči

Za zmínku stojí obchod, obzvláště ten s hráči. Ten totiž můžeme ve stolní verzi Osadníků z Katanu opakovat kolikrát chceme a jediným omezením je počet našich surovin a vůle protihráčů. Problémem v počítačové verzi je, že neomezený počet obchodů by znamenal obrovský počet větvení.

Omezit obchod je zároveň dobré i z pohledu uživatele programu. Pro počítačového agenta je samozřejmě nejvýhodnější vyzkoušet všechny různé možnosti,

které by mu mohly pomoci. Hráč však musí na všechny tyto nabídky reagovat, což při vysokém počtu nabídek může zkazit zážitek ze hry.

Jeden způsob, jakým v naší práci snižujeme počet nabídek, je zvažovat pouze ty obchody, které nám poskytnou suroviny chybějící k vykonání jiné akce.

Kromě toho omezujeme počet nabídek, které může agent za tah naplánovat. V prvních pokusech o implementaci jsme do tohoto limitu nepočítali nabídky, které byly úspěšné. Stávalo se ale, že si dva hráči donekonečna vyměňovali navzájem ty samé suroviny.

Nakonec omezujeme také počet nabídnutých surovin. Na začátku může agent nabídnout výměnu jedné suroviny za jednu jinou. Teprve poté může vyzkoušet směnit dvě suroviny za jednu jinou. Další možnost už není, protože by podle našeho uvážení byla příliš nevýhodná a nevyplatilo by se nad ní uvažovat.

#### Zjednodušování protivníkova tahu

Dalším zjednodušením plánování může být vynechání některých druhů akcí v průběhu protivníkova tahu. Protože nemáme úplnou informaci o stavu protihráčů, nemůžeme ani s jistotou určit všechny jejich možné tahy. Neznámé jsou pro nás akční karty v protivníkově ruce a karty surovin.

U akčních karet známe jejich počty v celé hře a v naší ruce a na základě těchto znalostí bychom mohli odhadovat pravděpodobnostní rozdělení karet v ruce protihráče. U surovin bychom si mohli být o něco jistější, pokud bychom sledovali hody kostkou a pamatovali si, které suroviny si protihráči berou a které utrácí. Stále bychom ale měli problém sledovat suroviny ukradené zlodějem.

Vzhledem k tomu, že i s úplnou znalostí stavu by většinou bylo potřeba spočítat velké množství tahů, propočítávat všechny možnosti by bylo nemožné. Navíc vzhledem k tomu, že expectimax slouží jenom k nakouknutí do blízké budoucnosti, ani nás nezajímají všechny části protihráčova tahu.

Protihráčův tah jsme tedy omezili na hod kostkou a nákup silnice, vesnice nebo města. Při nákupu neznáme jaké suroviny má protihráč v ruce, ale kontrolujeme celkový počet surovin protivníka.

## 2.7 Závěr

Přes všechny zmíněné optimalizace a vylepšení není možné prohledávat více než jeden tah dopředu. To u některých druhů akcí znamená, že se nedozvíme některé důležité informace potřebné k rozhodování. Je to patrné hlavně u stavby silnic, nezjistíme, kterým směrem je výhodné silnici umístit tak, abychom tam mohli v budoucnu postavit nejlepší vesnice.

Proto při hodnocení některých akcí používáme heuristiku, která se snaží uhodnout tyto údaje bez expectimaxu. Po stavbě nové silnice ji zkusíme ještě o jeden díl prodloužit a na místech, které se nám tímto krokem nově odemkly, zkusíme postavit vesnice. Výsledné ohodnocení akce upravíme o hodnoty takto získaných stavů a tím získáme alespoň částečnou informaci o kvalitě zvoleného směru.

Následující pseudokód, který je implementovaný v této práci, je částečně převzatý z pseudokódu alfa-beta minimaxu (Russell a Norvig, 2009), částečně upravený na základě vlastností popsaných v této kapitole:

**Algoritmus 1** Expectimax s $\alpha$-$\beta$ ořezáváním

```
function ALFA-BETA-EXPECTIMAX(stav, h, α, β)                ▷ h jako hloubka

    if hloubka == 0 or KONEC-HRY(stav) then                 ▷ Ukončující podmínka
        return FUNCTION-APPROXIMATION(stav)

    else if NAHODNA-AKCE(stav) then                         ▷ Náhoda
        v ← 0
        for each [pr, potomek] ∈ NAHODNI-POTOMCI(stav) do
            v += pr* ALFA-BETA-EXPECTIMAX(potomek, h − 1, α, β)
        end for
        return v

    else if HRAJE-AGENT(stav) then                          ▷ Max
        v ← −∞
        for each potomek ∈ NEJLEPSI-POTOMCI(stav) do
            v ← MAX(v, ALFA-BETA-EXPECTIMAX(potomek, h − 1, α, β))
            α ← MAX(α, v)
            if α ≥ β then break end if
        end for
        return v

    else                                                    ▷ Min
        v ← ∞
        for each potomek ∈ NEJLEPSI-POTOMCI(stav) do
            v ← MIN(v, ALFA-BETA-EXPECTIMAX(potomek, h − 1, α, β))
            β ← MIN(β, v)
            if α ≥ β then break end if
        end for
        return v
    end if
end function
```

# 3. Zpětnovazební učení

Na odhadování hodnot stavů, viz sekce 2.4, chceme využít funkci naučenou pomocí metod strojového učení, konkrétně pomocí zpětnovazebního učení. V úvodu této kapitoly si nejprve stručně řekneme něco o strojovém učení obecně a jaké jeho oblasti budeme v práci využívat. Následně shrneme základní principy zpětnovazebního učení a upřesníme, které konkrétní metody v práci použijeme.

## 3.1 Strojové učení

Strojové učení je jedním z podoborů umělé inteligence, kde se agent rozhoduje na základě svých vlastních předchozích pozorování prostředí. Dokážeme tedy vyvinout takové agenty, kteří se budou chovat jinak a pravděpodobně lépe, než kdyby měl své chování napevno implementované od programátora.

Strojové učení se dělí podle různých kritérií. Jedno z možných dělení je podle způsobu, jakým agent získává informace o kvalitě svého rozhodování.

Prvním způsobem je **učení bez učitele** (unsupervised learning). Agent nemá žádné informace o správném výsledku. Příkladem takového učení může být například klasifikace příkladů do několika tříd na základě podobnosti mezi nimi. Tento přístup ale nebudeme v práci dále využívat.

Druhým způsobem je **učení s učitelem** (supervised learning). K němu potřebujeme trénovací data, kde ke každému příkladu řekne „učitel“ agentovi i správné řešení. Agent se potom snaží naučit funkci, která mapuje všechny vstupy na správné výstupy s co nejmenší chybou. Metody učení s učitelem budeme využívat dále v sekci 3.4 při odhadu ohodnocující funkce.

Posledním a pro tuto práci nejdůležitějším způsobem získávání zkušeností je **zpětnovazební učení** (reinforcement learning). Agent se učí tím, že zkouší provádět akce. Za každou akci, nebo sérii akcí, dostane nějakou odměnu, ze které zjistí, jaký měla akce úspěch. Na základě této odměny pak může upravit své budoucí chování.

Zpětnovazební učení se tedy zdá být ideální volbou pro hraní her, kdy neznáme optimální tahy, ale dokážeme agentovi po každé hře říci, jak dobře si vedl. Agent může hrát mnoho her třeba sám proti sobě a tím zlepšovat svoji strategii. V následujících sekcích se tedy budeme zabývat právě zpětnovazebním učením.

## 3.2 Zpětnovazební učení

Podobně jako u expectimaxu má zpětnovazební učení mnoho podob, z nichž ne všechny jsou vhodné pro Osadníky z Katanu. Pro jednoduchost budeme zpočátku uvažovat jednodušší verzi zpětnovazebního učení pro takové problémy, které jsou malé a všechny stavy se dají podobně jako u základní verze minimaxu uložit do paměti. Tato verze se dá za pomoci aproximace funkcí jednoduše rozšířit i na větší problémy, jak si ukážeme v sekci 3.4.

Zpětnovazební učení dobře popisuje například Sutton a Barto (2018), my si v následujících sekcích popíšeme ty pro naši práci nejdůležitější části.

Problém zpětnovazebního učení je obvykle formulovaný pomocí **Markovových rozhodovacích procesů (MDP)**. Ty definujeme jako množinu stavů $S$ s nějakým počátečním stavem $s_0$, množinu akcí $A$, přechodovou funkci $p : S \times R \times S \times A \to [0,1]$, a diskontní faktor $\gamma \in [0,1]$.

Přechodová funkce $p(s',r|s,a)$ nám říká pravděpodobnost přechodu do stavu $s'$ s odměnou $r$ za předpokladu, že se agent nachází ve stavu $s$ a použije akci $a$. Důležitou vlastností přechodové funkce je **Markovská vlastnost**. Ta nám říká, že pravděpodobnost každé možné hodnoty $s'$ a $r$ závisí pouze na bezprostředně předchozím stavu a akci, nikoli na stavech předcházejících $s$. Tedy $\sum_{s' \in S} \sum_{r \in R} p(s',r|s,a) = 1$ pro všechny $s \in S$ a $a \in A$ (Sutton a Barto, 2018).

Agent v každém kroku pozoruje stav prostředí, zvolí akci a prostředí poté na základě přechodové funkce $p$ změní svůj stav a vrátí agentovi odměnu.

### 3.2.1 Ohodnocující funkce

Agent volí akce na základě své **strategie** $\pi : S \times A \to [0,1]$. Cílem agenta je nalézt takovou strategii, která agentovi vynese co největší odměnu za nějaký delší časový horizont. Agent se tedy nesnaží maximalizovat okamžitou odměnu v každém stavu, ale očekávanou **kumulovanou budoucí odměnu** (anglicky return) $G_t$, tedy součet okamžitých odměn na cestě z aktuálního stavu až do posledního stavu v čase $T$. Abychom dali větší důraz na bližší akce, můžeme využít **diskontovanou odměnu** s **diskontním faktorem** $\gamma$:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T.$$

Osadníci z Katanu jsou z pohledu zpětnovazebního učení **epizodický problém**[1]. To znamená, že kromě počátečního stavu máme také množinu koncových stavů, tedy stavů, kdy některý z hráčů dosáhl deseti bodů. Po dosažení takového stavu se přesuneme zpět do počátečního stavu a začneme tak novou epizodu. Kumulovanou odměnu počítáme vždy do konce epizody.

Úkolem agenta je tedy najít co nejlepší strategii. Kvalitu strategie můžeme určit za pomocí **ohodnocující funkce** $v_\pi(s)$, která je vypočítána jako očekávaná budoucí kumulovaná odměna, pokud začneme ve stavu $s$ a následujeme strategii $\pi$. Alternativně můžeme použít funkci $q_\pi(s,a)$, která namísto hodnot stavů počítá hodnoty akcí použitých v daném stavu.

Přesnou hodnotu ohodnocující funkce neznáme, pro výpočet jejího odhadu ale máme různé metody. Některé z nich si popíšeme blíže v sekci 3.3. Zakládají se na opakovaném procházení stavů (hraní hry) podle strategie $\pi$ a aktualizaci hodnot navštívených stavů na základě získané odměny. Takovému výpočtu hodnot jedné konkrétní strategie říkáme **iterace hodnot**. Iteraci hodnot budeme využívat v následující sekci při hledání optimální strategie.

### 3.2.2 Optimální strategie

Strategie $\pi^*$ s ohodnocující funkcí $v^*(s)$, respektive $q^*(s,a)$, je **optimální**, pokud splňuje $v^*(s) \leq v_\pi(s)$ pro všechny strategie $\pi$ a stavy $s$.

Pokud bychom znali optimální hodnoty všech akcí $Q^*(s,a)$, snadno bychom nalezli také optimální strategii jako $\pi^*(s) = argmax_a Q^*(s,a)$ nebo v případě naší

[1] Neplést s popisem epizodického a sekvenčního prostředí v sekci 1.2.

práce jako $\pi^*(s) = argmax_a$ALFA-BETA-EXPECTIMAX(Stav($s$,$a$),$h$, $-\infty, \infty$) pro libovolný stav, tedy hladovou strategii vzhledem k hodnotě $Q^*(s,a)$. Taková strategie bude optimální, protože hodnoty budoucích akcí a stavů jsou už uložené v hodnotě $Q^*(s,a)$. Problém hledání optimální strategie tedy můžeme převést na problém hledání optimálního ohodnocení stavů nebo akcí.

Při hledání optimální strategie používáme metodu zvanou **iterace strategie**. Na začátku zvolíme libovolnou strategii a vyhodnotíme ji pomocí iterace hodnot. Jako novou strategii zvolíme hladovou strategii vzhledem k nově získaným hodnotám. Tímto způsobem dostaneme lepší strategii, než jsme měli na začátku a po dostatečně mnoha iteracích se přiblížíme k optimální strategii. Tento postup můžeme zjednodušit tak, že strategii aktualizujeme po každé aktualizaci hodnot, nemusíme tedy čekat na úplné vyhodnocení jedné strategie (Sutton a Barto, 2018).

### 3.2.3 Explorace a exploitace

Abychom správně vypočítali hodnoty všech možných stavů, musíme zajistit, aby agent při výpočtu postupně vyzkoušel všechny možné stavy. Při důsledném následování strategie, obzvláště té hladové, se ale může stát, že se do některých stavů vůbec nedostane. Proto musíme najít nějaký kompromis mezi **exploitací**, tedy využíváním agentových dosavadních znalostí a tedy následováním strategie, a **explorací**, tedy prozkoumáváním málo navštěvovaných stavů.

Nejjednodušší způsob je využít **$\epsilon$-greedy** strategii. Ta s pravděpodobností $\epsilon$ zvolí náhodnou akci, jinak se chová jako hladová strategie. Správné hodnoty získáme, pokud začneme s nějakým větším $\epsilon$, například 0.4, a následně budeme jeho hodnotu snižovat, tedy budeme čím dál méně prozkoumávat a více využívat získané znalosti.

Existuje mnoho dalších metod průzkumu, které se chovají lépe. Například upřednostňují dlouho nenavštívené stavy. Takové metody můžou zaručit rychlejší konvergenci ke správnému výsledku, zároveň jsou složitější na implementaci. Protože $\epsilon$-greedy metoda splňuje naše požadavky na průzkum, rozhodli jsme se zůstat u této jednodušší varianty.

Když máme strategii, která umí prozkoumat všechny možné kombinace stavů a akcí, je potřeba si ještě rozmyslet, jakým způsobem budeme tuto strategii využívat. Rozlišujeme on-policy a off-policy metody.

**On-policy** metody využívají při celém procesu učení jednu strategii, podle které vybírají akce a zároveň tu samou strategii vylepšují.

Oproti tomu **off-policy** metody mají více různých strategií. Jednu strategii podle které vybírají své tahy, například $\epsilon$-greedy strategii. Druhou strategií, která už nemusí vůbec prozkoumávat, se snaží přiblížit k optimální strategii.

V naší práci budeme využívat výhradně on-policy metody, protože se nedá přesně říct, že některý z přístupů je obecně lepší než ten druhý a on-policy metody jsou o něco jednodušší.

### 3.2.4 Okamžitá odměna

Celé agentovo výsledné chování závisí na získaných okamžitých odměnách, které dostává od prostředí. Volbou odměny tedy agentovi říkáme, jak chceme, aby

se choval. Měli bychom si dát pozor, abychom agenta odměňovali skutečně jenom za to, čeho chceme, aby dosáhl, tedy výhry. Pokud bychom odměňovali například za stavbu vesnice, agent by si mohl najít cestu, jak toho zneužít a získávat body, aniž by vyhrál. Obvyklý přístup při hraní her je dát agentovi kladnou odměnu při výhře, zápornou nebo nulovou odměnu při prohře a nulovou odměnu ve všech ostatních stavech.

V Osadnících z Katanu si kromě samotného vítěze mohli vést dobře i ostatní hráči, pokud získali hodně bodů a vítěz vyhrál jenom o kousek. Je tedy vhodné dát všem hráčům nějakou odměnu za získané body. Je ale potřeba zvolit vhodný poměr velikosti odměny za vítězství a velikosti odměny za herní body.

V naší práci jsme uvažovali dvě možnosti. V prvním případě jsme dali agentovi 100 bodů odměny za každý získaný herní bod, za výhru dalších 800 bodů a za prohru jsme mu 200 bodů sebrali. Ve druhém případě jsme se rozhodli dát větší důraz na primární cíl, tedy výhru hry. Vítězi jsme dali 1000 bodů za výhru, za každý herní bod ale dostali agenti jenom 10 bodů odměny.

K výběru vhodnějšího způsobu jsme naučili hrát (se stejným nastavením všech ostatních parametrů) dva agenty oběma zmíněnými způsoby, zafixovali jejich naučené váhy a nechali je hrát proti sobě. Ze 440 her vyhrál agent naučený s prvním způsobem 271 her, zatímco agent s druhým způsobem odměňování vyhrál jenom 169 krát.

Protože se jedná o nedeterministickou hru a v metodách učení používáme spoustu odhadů a zjednodušení, mohlo se stát, že by se agenti při dalším experimentu naučili odlišné váhy a experiment by dopadl jinak. Jsme ale přesvědčeni, že rozdíl by v dalších pokusech nebyl příliš velký.

Na konci tedy agenta odměníme 100 body odměny za každý herní bod a navíc 800 body za výhru nebo $-200$ body za prohru.

Kromě hlavní odměny na konci může v naší práci každý agent v průběhu hry získávat malou zápornou odměnu za každý tah. Smyslem je, aby pro agenta bylo výhodné ukončit hru co nejrychleji. Tato odměna je pro každého agenta nastavitelná individuálně v parametru `default_reward` (viz sekce 5.5).

## 3.3 Odhad ohodnocující funkce

V této sekci si představíme dva hlavní přístupy k výpočtu ohodnocující funkce. Budeme uvažovat funkci $Q(s,a)$, tedy odhad hodnoty akcí v daném stavu. Metody se ale dají stejně tak použít i pro výpočet funkce $V(s)$, tedy odhad hodnot samotných stavů.

Obě metody se dají použít jak pro výpočet ohodnocující funkce nějaké konkrétní strategie $Q_\pi$, tak pro výpočet optimálních ohodnocující funkce $Q^*$. Záleží, podle jaké strategie bude agent při výpočtech volit své akce. Protože my chceme primárně najít optimální strategii, budeme používat hladovou strategii vzhledem k aktuálnímu odhadu. S každým zlepšením odhadu hodnot stavů nebo akcí tedy vylepšíme také strategii.

Jak jsme již naznačili dříve, hodnotu budeme odhadovat opakovaným hraním hry a průběžným aktualizováním odhadu hodnot použitých akcí v navštívených stavech. Při každém průchodu zjistíme hodnotu odměn a vypočítáme z nich celkovou kumulovanou odměnu $G$, nebo její odhad. Odečtením hodnoty současného odhadu ohodnocující funkce získáme chybu, o jejíž $\alpha$-násobek následně upravíme

odhad hodnoty stavu nebo akce:

$$Q(S_t,A_t) \leftarrow Q_t(S_t,A_t) + \alpha[G_t - Q_t(S_t,A_t)]$$

Parametr $\alpha \in (0,1)$ je **velikost kroku**. Čím větší je $\alpha$, tím větší váhu mají novější aktualizace. Je dobré nastavit $\alpha$ podle očekávaného počtu průchodů jedním stavem. Pokud bychom zvolili například $\alpha = \frac{1}{10}$, naučil by se agent hodnotu stavu $s$ přibližně po deseti průchodech daného stavu. Protože volba parametru závisí částečně také na volbě příznaků u aproximace funkcí v sekci 3.4, ještě se k ní vrátíme později.

### 3.3.1 Monte Carlo

V metodě Monte Carlo budeme procházet celé epizody najednou a na konci zpětně ohodnotíme navštívené stavy a použité akce. Tímto způsobem získáme přesnou hodnotu kumulované odměny $G$ získané průchodem navštívených stavů. Aktualizace tedy bude vypadat přesně tak, jak je uvedeno výše.

### 3.3.2 Temporální diference

Trošku jiný přístup k aktualizacím odhadů hodnot akcí nabízí temporální diference. Zatímco Monte Carlo metody čekají na kumulovanou odměnu získanou z epizody a dají se tedy použít pouze u epizodických problémů, metody temporální diference vylepšují svůj odhad pokaždé, když prostředí přejde do dalšího stavu. Základními algoritmy využívající temporální diferenci jsou off-policy metoda **Q-learning** a on-policy **SARSA**.

Protože před koncem epizody nemůžeme znát přesnou hodnotu kumulované odměny, odhadujeme její hodnotu na základě získané okamžité odměny a současného odhadu hodnoty další akce. Aktualizace hodnot metodou Sarsa vypadá následovně:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

. Rozdíl mezi Q-learningem a Sarsou je ten, že Sarsa využívá odhad hodnoty tahu, který agent skutečně použil. Oproti tomu při Q-learningu odhadneme hodnoty všech možných akcí a vybereme tu nejlepší, tedy $G_{t+1} = argmax_a(S_{t+1}, a)$.

### 3.3.3 $\lambda$-return

Aktualizaci temporální diference můžeme dále rozšiřovat různými způsoby. Nejpřímočařejší rozšíření je namísto jednokrokového odhadu budoucí kumulativní odměny použít n-krokový odhad budoucí kumulativní odměny, tedy podívat se na n příštích odměn a zbytek odhadnout s pomocí aktuálního odhadu ohodnocující funkce. Takový odhad značíme $G_{t:t+n}$. V extrémním případě se tak můžeme vrátit zpět k Monte Carlo přístupu.

Tuto myšlenku můžeme dále rozšířit. Budeme uvažovat různé n-krokové odhady budoucí kumulativní odměny, jejichž hodnoty zprůměrujeme. Tuto myš-

lenku využívá **$\lambda$-return**[2]:

$$G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n}.$$

Po dosažení koncového stavu jsou všechny další odhady budoucí kumulované odměny nahrazeny skutečnou kumulovanou odměnou $G_t$.

Různými nastaveními parametru $\lambda$ můžeme měnit chování algoritmu. Pokud nastavíme $\lambda = 0$, algoritmus přiřadí všem vícekrokovým odhadům kumulované odměny nulovou váhu a získáme tedy jednokrokový odhad stejně jako při temporální diferenci popsané v sekci 3.3.2. Naopak když nastavíme $\lambda = 1$, získáme celou kumulativní odměnu stejně jako při Monte Carlu.

Nejjednodušším algoritmem využívajícím $\lambda$-return je **off-line $\lambda$-return**. Off-line proto, že abychom mohli aktualizovat hodnoty, musíme počkat až do konce epizody. Tím ale přijdeme o jednu z výhod temporální diference, která upravuje hodnoty průběžně. Abychom dosáhli častějších aktualizací, můžeme namísto celého $\lambda$-return použít jenom jeho prvních n kroků k horizontu $h$, tedy

$$G_{t:h}^\lambda \doteq (1-\lambda)\sum_{n=1}^{h-t-1}\lambda^{n-1}G_{t:t+n} + \lambda^{h-t-1}G_{t:h}, \quad 0 \le t \le h \le T.$$

Výhody obou přístupů můžeme sloučit do jednoho algoritmu za cenu vyššího výpočetního výkonu. Získáme tak **on-line $\lambda$-return** algoritmus. V každém kroku aktualizujeme hodnoty s pomocí všech dosavadních znalostí. V dalším kole tuto aktualizaci zapomeneme a provedeme novou, lepší.

Tento algoritmus je zatím příliš výpočetně náročný, později v sekci 3.6 si ale ukážeme, jak se dá s využitím eligibility trace zefektivnit.

## 3.4 Aproximace funkcí

Osadníci z Katanu mají velmi mnoho stavů a není tedy možné si pro každý z nich pamatovat nějakou hodnotu. Proto budeme odhadovat jejich hodnotu na základě vlastností, které má společné více stavů.

K ohodnocení stavů tedy budeme využívat funkci $\hat{v}(s,w)$, která na vstupu dostane kromě stavu k ohodnocení také **váhový vektor** $w$. Úkolem agenta je najít takové váhy, aby funkce $\hat{v}$ odhadovala hodnotu skutečné ohodnocující funkce s co nejmenší **chybou** $\overline{VE}(w) \doteq \sum_{s\in S}\mu(s)\left[v_\pi(s) - \hat{v}(s,w)\right]^2$. $\mu$ značí rozdělení, které odpovídá výskytu stavů během hry. Častější stavy pro nás tedy mají větší váhu.

### 3.4.1 Stochastic Gradient Descent

K učení těchto vah budeme využívat metod učení s učitelem. Jednou z nejpoužívanějších a nejvhodnějších metod pro zpětnovazební učení je metoda **Stochastic Gradiend Descent** (SGD), případně **Semi-Gradient metoda**, pokud v příkladech počítáme s odhady tak jako při temporální diferenci (Sutton a Barto, 2018). Tu budeme nadále využívat v naší práci.

[2]return, tedy kumulovaná budoucí odměna

Tyto metody po každém příkladu upraví váhový vektor pokaždé o $\alpha$-násobek chyby směrem, který „je nejvíce zodpovědný“ za způsobenou chybu, tedy ve směru gradientu: $w_{t+1} = w_t + \alpha \left[ v_\pi(S_t) - \hat{v}(S_t, w_t) \right] \nabla \hat{v}(S_t, w_t)$. Protože hodnotu $v_\pi$ neznáme, nahradíme ji odhadem získaným průchodem hrou tak, jako u temporální diference. S využitím tohoto pravidla můžeme jednoduše upravit metody Monte Carlo a Temporální diference ze sekce 3.3.

### 3.4.2 Aproximační funkce

Aproximační funkce $\hat{v}$ může mít mnoho podob. Nejjednodušší je **lineární funkce**, tedy skalární součin **vektoru příznaků** stavu s $x(s)$ a váhového vektoru $w$, tedy $\hat{v}(s,w) = w^\top x(s)$.

V případě použití lineární aproximace je gradient aproximační funkce rovný vektoru příznaků, tedy $\nabla \hat{v}(s,w) = x(s)$. S každou aktualizací tedy nejvíce upravujeme váhy těch vektorů, které měly na ohodnocení stavu největší vliv.

Existuje mnoho dalších metod. Jednou z nich jsou například **neuronové sítě**. Jejich výhodou je hlavně schopnost určit různé vztahy mezi jednotlivými příznaky, díky kterým dokáže odhadnout ohodnocující funkci přesněji. Nevýhodou neuronových sítí je velká náročnost na učení.

Další možností můžou být například **modelové stromy**, které ve své práci využívá k řešení problému Osadníků z Katanu Pfeiffer (2004).

V naší práci jsme se rozhodli využívat lineární aproximaci. Výhodou je její jednoduchost a snadné propojení s eligibility trace, které si představíme v sekci 3.6. Naopak její největší nevýhodou je, že neumí znázornit vztahy mezi různými příznaky. Tuto nevýhodu ale můžeme částečně odstranit přidáním nových příznaků, jak si ukážeme v další sekci.

### 3.4.3 Volba příznaků

Příznaky, tedy vlastnosti stavů, podle kterých se agent rozhoduje, je nutné zvolit tak, aby reprezentovaly všechny možné vlastnosti stavu, podle kterých se má agent rozhodovat. Nemělo by jich ale být příliš mnoho, učení by pak nebylo efektivní.

V naší práci jsme rozdělili příznaky do tří skupin. První dvě reprezentují informace o protihráčích, ta třetí potom vlastnosti samotného hráče. V Osadnících z Katanu nemáme pevný počet protihráčů, nemůžeme každému z nich věnovat jeho vlastní sadu příznaků, a pravděpodobně by to ani nebylo příliš užitečné. Proto jsme zvolili dva reprezentativní „protihráče“. Prvním sada příznaků reprezentuje vlastnosti protihráče s nejvíce body, tedy protihráče, který nejvíce ohrožuje agentovo vítězství. Druhá sada průměruje vlastnosti všech protihráčů.

Seznam všech příznaků:

- Nejlepší protihráč a průměrný protihráč (příznaky jsou u obou stejné)
  - Body
  - Počet surovin v ruce
  - Délka cesty
  - Velikost vojska

  - Počet volných křižovatek[3]
  - Pravděpodobnost zisku surovin (za každou surovinu jeden příznak)

- Hráč
  - Body
  - Délka cesty
  - Velikost vojska
  - Hráč má nejdelší cestu (bool)
  - Hráč má největší vojsko (bool)
  - Počet silnic
  - Počet vesnic
  - Počet měst
  - Počet volných křižovatek
  - Pravděpodobnost zisku surovin (za každou surovinu jeden příznak)
  - Pravděpodobnost zisku surovin na nákup (4 příznaky: silnice, vesnice, město, akční karta)
  - Počet surovin (za každou surovinu jeden příznak)
  - Hráč má suroviny na nákup (bool, 4 příznaky: silnice, vesnice, město, akční karta)
  - Karty v ruce (bool, 4 příznaky za každý druh karty)
  - Ceny obchodu s přístavem (za každou surovinu jeden příznak)

Některé příznaky mohou mít různou hodnotu pro agenta v závislosti na čase nebo jiných příznacích. Například mít vesnici v přístavu pro obchod s vlnou může být výhodný pouze za předpokladu, že má agent zároveň nějaký zdroj vlny, aby měl co prodávat. Stejně tak vlastnit suroviny je dobré, ale pokud jich má agent moc, riskuje okradení zlodějem. Některé aproximační funkce, například neuronové sítě, si s tím dokáží poradit samy o sobě. Protože my jsme se rozhodli používat lineární aproximační funkci, musíme přímo zvolit příznaky tak, aby tyto závislosti už zohledňovaly. K původním příznakům můžeme přidat polynomy, které budou reprezentovat vztahy mezi těmi původními příznaky.

Jednou možností by bylo vytipovat takové n-tice příznaků, o kterých si myslíme, že budou užitečné. Druhou možností potom je přidat všechny možné dvojice příznaků. Druhá možnost zvětší počet příznaků $O(n^2)$-krát, což negativně ovlivní rychlost učení. Oproti tomu první možnost odporuje původní myšlence, že by si agent měl sám přijít na to, co je pro něj dobré.

V naší práci tedy k základní sadě vektorů přidáme jedničku a roznásobíme každý příznak s každým (včetně sebe samého), tedy

$$x(s) = (1, s_1, s_2, \cdots, s_1^2, s_2^2, \cdots, s_1 s_2, \cdots)$$

. Jedinou výjimku jsme udělali u kombinací surovin potřebných k nákupu. Protože například na stavbu vesnice jsou potřeba 4 druhy surovin, což je více než dvojice, přidali jsme tyto kombinace do základní sady příznaků vypsané dříve.

[3]Počet míst, kam hráč může umístit vesnici. Jinými slovy počet míst, kam vede hráčova cesta a které nesousedí s jinou vesnicí.

Další možností by samozřejmě bylo použít všechny trojice a čtveřice příznaků. To by ale znamenalo už příliš velký nárůst příznaků.

Každý příznak ze základní sady jsme se rozhodli ještě před roznásobením znormalizovat, tedy odečíst od každého příznaku odhad jeho střední hodnoty a vydělit směrodatnou odchylkou tak, aby střední hodnota výsledných příznaků byla nulová a směrodatná odchylka se rovnala jedné. Jednak tím zaručíme, že všechny příznaky budou mít při určování hodnoty stavu zpočátku stejný význam, jednak tím napomůžeme tomu, aby odhadované hodnoty nebyly příliš vysoké. Abychom zaručili, že ani po roznásobení nedostaneme žádnou příliš vysokou hodnotu, rozhodli jsme se výsledný násobek ještě upravit tak, aby se jeho hodnota nacházela v intervalu $[-1,1]$. K tomu jsme použili funkci $f(x) = \frac{1}{1+e^x}$.

## 3.5 Parametr $\alpha$

Nyní se pojďme podívat na volbu vhodné velikosti kroku $\alpha$. Jak píší Sutton a Barto (2018), není dána žádná optimální volba parametru $\alpha$. Doporučují ale zvolit $\alpha \doteq \left(\tau \mathbb{E}\left[x^\top x\right]\right)^{-1}$, kde $\tau$ je očekávaný počet průchodů stavem $s$, než agent správně odhadne jeho hodnotu $V(s)$. Hodnotu $\mathbb{E}\left[x^\top x\right]$ jsme odhadli na 400.

V naší práci můžeme každému agentovi nastavit různou hodnotu $\tau$, (sekce 5.5). Čím menší číslo, tím agent upravuje hodnoty vah rychleji. Pokud ale nastavíme $\tau$ příliš nízké (nižší než 10), mohou hodnoty vah divergovat.

## 3.6 Eligibility trace

V sekci 3.3 jsme si ukázali dva hlavní přístupy k výpočtu odhadu ohodnocující funkce. V každém kroku jsme se podívali na odměny získané v budoucnu a na jejich základě jsme upravovali váhy. Eligibility trace nám umožní podívat se na problém opačně.

**Eligibility trace** $z_t \in \mathbb{R}^d$ je vektor, ve kterém si pamatujeme příznaky minulých stavů. V každé aktualizaci použijeme namísto gradientu, tedy příznaků aktuálního stavu, právě vektor eligibility trace. Aktualizujeme tedy zpětně váhy ze všech minulých stavů, na které měla právě získaná odměna vliv.

Nejznámějším algoritmem, který eligibility trace využívá, je **TD($\lambda$)** algoritmus, který se chová podobně jako off-line $\lambda$-return algoritmus ze sekce 3.3.3 (Sutton a Barto, 2018). V každém stavu zmenší hodnoty v $z_t$ $\lambda$-krát a přičte příznaky současného stavu.

Dalším algoritmem využívajícím eligibility trace je **True Online TD($\lambda$)** (algoritmus 2). Přestože to na první pohled nemusí být patrné, bylo dokázáno, že aktualizuje váhy stejným způsobem jako s on-line $\lambda$-return algoritmem popisovaným v sekci 3.3.3 (van Seijen a kol., 2016).

## 3.7 Závěr

Po prvních pokusech o implementaci zpětnovazebního učení do našeho programu nefungovalo všechno podle našich představ. Ukázalo se, že většina problémů byla způsobována chybami v kódu. Jedním z problémů se ale ukázalo být

**Algoritmus 2** True Online TD($\lambda$)

**procedure** TRUE-ONLINE-TD-LAMBDA($\alpha, \gamma, \epsilon, \lambda$)
  $x(s) \to \mathbb{R}^d$ t. ž. $x(koncový) = 0$
  $w \leftarrow 0$
  $\pi \leftarrow \epsilon$-greedy strategie vzhledem k $\hat{v}(s,w)$
  **for each** Epizoda **do**
    Inicializuj počáteční vektor příznaků $x$
    $z \leftarrow 0$ ▷ $d$-rozměrná eligibility trace
    $V_{old} \leftarrow 0$ ▷ skalární proměnná
    **repeat**
      Zvol akci $A$ podle strategie $\pi$
      Použij akci $A$, pozoruj $R, x'$ ▷ Vektor příznaků příštího stavu
      $V \leftarrow w^\top x$
      $V' \leftarrow w^\top x'$
      $\delta \leftarrow R + \gamma V' - V$
      $z \leftarrow \gamma\lambda z + (1 - \alpha\gamma\lambda z^\top x)x$
      $w \leftarrow w + \alpha(\delta + V - V_{old})z - \alpha(V - V_{old})x$
      $V_{old} \leftarrow V'$
      $x \leftarrow x'$
    **until** $x' = 0$
  **end for**
**end procedure** ▷ Sutton a Barto (2018)

to, že jsme v aktualizacích využívali všechny stavy, jimiž jsme za epizodu prošli. Jeden tah se totiž může dělit na několik částí v závislosti na počtu náhodných akcí, případně můžeme do stavů zahrnout i ty, ve kterých agent reagoval na protihráčovu nabídku k obchodu. Agent se tedy neučil vyhrát v co nejmenším počtu kol, ale s co nejmenším počtem použitých akcí, což mohlo být kontraproduktivní. Poté, co jsme váhy začali aktualizovat pouze ve stavech, po kterých agent svůj tah ukončil, učení se výrazně vylepšilo.

Všechny zmíněné metody (Monte Carlo, Sarsa, TD($\lambda$), True Online TD($\lambda$)) jsme zkusili implementovat. Ve finální verzi jsme se rozhodli používat jenom True Online TD($\lambda$) (algoritmus 2). Tento algoritmus je popisovaný jako v současnosti nejlepší algoritmus využívající temporální diferenci (Sutton a Barto, 2018). Navíc změnou parametru $\lambda$ dokážeme simulovat chování, které se podobá chování

# 4. Programátorská dokumentace a objektový návrh

V následující kapitole popíšeme, jakým způsobem jsme přistupovali k implementaci samotné hry. Jaký jazyk jsme použili a jaké knihovny nám pomohly. Zmíníme nejdůležitější třídy a vztahy mezi nimi.

## 4.1 Jazyk a knihovny

Pro implementaci prostředí i umělé inteligence Osadníků z Katanu jsme zvolili jazyk C++. Očekáváme totiž, že počítání tahů expectimaxem a hraní velkého množství trénovacích her bude náročné na čas a protože jazyk C++ na rozdíl od Javy, C# nebo Pythonu vytváří přímo strojový kód cílového procesoru, je pro takovou aplikaci vhodným kandidátem. Navíc máme s jazykem C++ největší zkušenosti.

Nevýhoda C++ je, že přímo nepodporuje okenní aplikace, které v naší práci využíváme. Proto využíváme grafickou knihovnu, která umí vytvořit okno a spravovat jeho události. Knihoven, které se dají využít je opravdu hodně, například SFML, Qt, GLFW. Každá má trochu jiné vlastnosti, protože ale grafika není pro tuto práci stěžejní, zvolili jsme bez přílišného rozebírání možností knihovnu SFML[1], protože umí to, co potřebujeme a má přehlednou dokumentaci[2].

Kromě grafické knihovny jsme v práci použili cizí funkci pro čtení konfiguračního souboru agenta[3], funkci pro zalomení příliš dlouhého textu[4] a funkci hešující dvojice čísel[5]

## 4.2 Návrh programu

Při psaní programu jsme nejprve vytvořili konzolovou aplikaci simulující Osadníky z Katanu a reagující na slovní příkazy uživatele. Protože takový program bylo nemožné odladit, přidali jsme grafické rozhraní za pomoci knihovny SFML, které umožnilo uživateli sledovat stav mapy a ovládat hru klikáním na tlačítka. V poslední fázi jsme přidali agenty využívající umělou inteligenci popsanou v předchozích kapitolách.

### 4.2.1 Hra

Při každé hře Osadníků z Katanu se ve funkci `main` vytvoří nová instance třídy `Hra` a zavolá se její metoda `hraj`. Třída `Hra` vytvoří a pamatuje si všechno potřebné: informace o hráčích, mapě a balíčku karet. Navíc má v sobě uložené základní objekty sloužící ke komunikaci s uživatelem, jako je font nebo popisky.

[1] `https://www.sfml-dev.org` (přístup 31. 5. 2020)

[2] `https://www.sfml-dev.org/learn.php` (přístup 31. 5. 2020)

[3] `https://www.walletfox.com/course/parseconfigfile.php` (přístup 4. 6. 2020)

[4] `https://en.sfml-dev.org/forums/index.php?topic=20346.0` (přístup 4. 6. 2020)

[5] `https://www.geeksforgeeks.org/how-to-create-an-unordered_map-of-pairs-in-c/` (přístup 4. 6. 2020)

Metoda `hraj` opakovaně volá tahy hráčů, dokud některý z nich nevyhraje.

Vlastnosti objektu `Hra` mohou být v průběhu hry ovlivňovány jednotlivými hráči, kteří používají své metody v rámci svého tahu. Výjimkou je hod kostkou, který má vliv na všechny hráče najednou a proto jsme ho přidali jako samostatnou metodu hry.

### 4.2.2 Mapa

**Mapa** Druhou velmi důležitou součástí programu je třída `Mapa`. Mapa nám poskytuje možnost ukládat a vykreslovat stav herního plánu.

V původní myšlence jsme chtěli ukládat všechny objekty políček, cest a křižovatek do jednoho pole indexovaného herními souřadnicemi za využití polymorfismu. To se v pozdější fázi hry ukázalo jako neefektivní, protože například při procházení bylo potřeba kontrolovat, jestli se zrovna díváme na cestu nebo na něco jiného. Proto ukládáme políčka, cesty i křižovatky do samostatných map, které indexujeme pomocí zvolených souřadnic. Souřadnice jednotlivých políček jsou znázorněné v přílohách na obrázku A.2. Původní souřadnice ale zůstaly zachovány. Všechny objekty si navíc pamatují odkazy na své sousedy, aby se na ně dalo rychle odkazovat.

Důležitou součástí mapy byl původně převod kliknutí myši na nějaký objekt na mapě, tedy pole, cestu nebo křižovatku. Snažili jsme se počítat vzdálenost myši od středu jednotlivých objektů a vracet minimum. S postupným nabýváním zkušeností s knihovnou SFML jsme namísto toho začali využívat jejích metod, které umožňují se přímo zeptat, jestli je pozice myši uvnitř nějakého nakresleného tvaru.

**Vlastníci** Původně měly v sobě tyto objekty uložené také informace o tom, který hráč na nich má postavenou svoji vesnici. Při implementaci expectimaxu se ale tento přístup ukázal jako nevhodný. Při plánování budoucích stavů totiž bylo potřeba kopírovat zbytečně celou mapu. Proto jsme vytvořili třídu `Vlastnici`, která v sobě ukládá právě ty informace, které se s akcemi hráčů mohou měnit. Při plánování budoucích stavů tedy stačí kopírovat a měnit tuto třídu a zbytek mapy předávat konstantním odkazem.

### 4.2.3 Balíček karet

Důležitou součástí hry je balíček akčních karet. Jednotlivé karty jsou uloženy ve vektoru v `shared_ptr`. Jednak abychom mohli s pomocí polymorfismu ukládat různé druhy karet do jednoho vektoru a jednak aby bylo možné mít dvě kopie vektoru karet v ruce hráče. Do jedné se ukládají nově líznuté karty, které je ale možné použít až od příštího kola. Ve druhé jsou ty karty, které hráč vlastnil už od začátku kola a může je tedy použít.

Každá karta obsahuje funkce `liznuto` zavolanou ve chvíli, kdy si hráč kartu lízne a `zahrano`, která je zavolána při zahrání karty během tahu hráče. Funkce jsou různě přetížené v závislosti na druhu karty.

Podobně jako objekty na mapě se i karty dají graficky nakreslit tak, aby si uživatel mohl snadněji vybrat, kterou kartu z ruky chce použít.

### 4.2.4 Hráč

Poslední důležitou součástí třídy `Hra` je seznam hráčů. Třída `Hrac` reprezentuje vlastnosti a metody hráče. Z vlastností například suroviny nebo akční karty v ruce.

Kromě své stěžejní metody `tah`, která je volána každé kolo třídou `Hra` má hráč další metody, které reagují na neočekávané události ve hře mimo hráčův tah, například okrádání zlodějem nebo obchod s jiným hráčem.

Všechny hráčovy akce jsou řízeny objektem třídy `Agent`.

### 4.2.5 Agent

**Agent** `Agent` je abstraktní třída, jejíž hlavním úkolem je volit `Akce`, na jejichž základě bude hráč konat. V naší práce rozlišujeme dva druhy agentů. Prvním je `HumanAgent`, který vybírá akce na základě uživatelových kliknutí myší. Druhým je `RLAgent`, který se rozhoduje na základě umělé inteligence.

Třída `RLAgent` má v sobě navíc třídu `ReinforcementLearning`, která může ovlivnit způsob, jakým agent přistupuje k aktualizaci vah. Můžeme tak porovnávat různé metody.

**Akce** Objekt třídy `Akce` má v sobě vždy uložené informace o typu akce, například stavba vesnice nebo obchod, a parametry akce. Parametry můžou být například suroviny, které chce agent směnit v obchodu, karta, kterou chce agent použít nebo souřadnice cesty, na které chce agent postavit silnici. V každém druhu akce jsou jiné druhy parametrů.

Parametry by se daly ukládat jako jeden vektor čísel. Protože jako parametry často předáváme souřadnice a předávat jednotlivé souřadnice oddělené by mohlo vést k chybám, rozhodli jsme některé druhy parametrů namísto vektoru čísel předávat pomocí vektoru dvojic čísel. Třída `Akce` tedy obsahuje vektor čísel `parametry` a vektor dvojic čísel `dvojice_parametru`.

**Naplánovaný tah** Agentova metoda `tah` vrací objekt třídy `NaplanovanyTah`. Ta je reprezentována dvojicí vektoru akcí a mapy (`std::map`) s dalšími naplánovanými tahy. Na prvním místě ukládá posloupnost akcí naplánovaných buď na celý, nebo na část hráčova tahu. Pokud je tato posloupnost ukončena akcí s náhodou, slouží mapa na druhém místě dvojice k uložení všech možných pokračování tahu.

### 4.2.6 Učení

V hlavičkovém souboru `Learning.hpp` nalezneme různé třídy, které jsou využívány při učení a plánování akcí za pomoci umělé inteligence.

#### Stav

Nejdůležitější je třída `Stav`, která ukládá všechny proměnlivé informace o aktuálním stavu hry z pohledu jednoho hráče.

Hlavní dvě funkce stavu při plánování jsou schopnost vygenerovat všechny možné akce ze stavu a schopnost simulovat použití těchto akcí sám na sobě. Tomu jsou přizpůsobeny i různé konstruktory stavu, které umožňují kromě vytvoření

zcela nového stavu také vytvořit kopii jiného stavu, na které bude následně provedena akce. Pokud je tato akce náhodná, zavolá se jiný konstruktor, který navíc přijímá argument s informací, co se přesně má stát.

### Vektor příznaků

Druhou důležitou třídou je třída `FeatureVector`, která v konstruktoru dostane stav a vytvoří si seznam příznaků daného stavu. Jejím potomkem je třída `Weights`, která má konstruktor schopný načíst váhy uložené v souboru.

Protože s těmito třídami hodně počítáme, mají definované operátory skalárního součinu, sčítání a odečítání.

### Zpětnovazební učení

Poslední, už zmiňovanou třídou je `ReinforcementLearning` a její potomci. Tato třída má různé funkce pro ohodnocení stavů, na základě své strategie umí pro každý stav vybrat nějakou akci a hlavně umí ohodnocení stavů upravovat na základě získaných zkušeností.

V průběhu práce jsme získali potřebu mít několik různých ohodnocujících funkcí. `ReinforcementLearning` má hned tři různé způsoby, jakým je možné nějaký stav ohodnotit.

První způsob je `v_s_expectimaxem`, které používáme při výběru příštího tahu. Zajímá nás, který tah může vést do stavu s nejlepším ohodnocením. Projdeme tedy expectimax do určené hloubky a na konci odhadneme hodnotu stavu.

Druhým způsobem je `v_bez_expectimaxu`. Ten používáme kdykoli potřebujeme rychlé ohodnocení stavu, které může být méně přesné. Například při dvojím ohodnocování popisovaném v sekci 2.6.2, kdy pomocí expectimaxu generujeme všechny potomky, ohodnotíme je funkcí bez expectimaxu a dále rozvíjíme jenom ty nejslibnější.

Posledním způsobem je `q_s_heuristikou`. Využíváme ji při plánování konkrétních elementárních akcí, například při rozmýšlení, kam umístit silnici. Některé akce totiž po jejich použití neposkytují samy o sobě dostatečnou informaci o své kvalitě. Nejzřetelněji je to vidět právě na stavbě silnice, kde potřebujeme vědět, co budeme moci v budoucnu postavit směrem, kterým silnice povede. Ideální by bylo využít k tomuto účelu expectimax. Problémem je, že se nedokážeme nikdy dostat dostatečně hluboko. Proto namísto toho vybereme několik nejlepších akcí při ohodnocení s heuristikou, a s těmi následně počítáme dále při expectimaxu.

Metody `update` a `update_konec_hry` implementují v různých potomcích třídy `ReinforcementLearning` různé způsoby aktualizace vah popsané v kapitole 3.

Součástí třídy `ReinforcementLearning` je i metoda *policy* pro výběr agentovy příští akce. Protože tento výpočet je velmi náročný a zahrnuje výpočet hodnot různých na sobě nezávislých stavů, je možné počítat hodnoty akcí paralelně.

Máme proto také metodu `policy_async`, ve které počítáme hodnoty jednotlivých stavů v `std::future`, které mohou spustit funkci v jiném vlákně. Výhodou je kromě rychlosti také to, že během čekání můžeme aktivně vykreslovat okno hry, čímž ho udržujeme aktivní a tedy může reagovat na různé události.

Počítání ve více vláknech by mohlo být pomalejší, pokud je potomků málo a většina času by byla využita na tvorbu nového vlákna. Zároveň nám paralelní výpočty znemožní některé optimalizace, které jsme implementovali do sériové

| Prohledávání | do hloubky 1 | do hloubky 2 | do hloubky 3 |
|---|---|---|---|
| Sériově | 30.75 sekund/hru | 117.69 sekund/hru | 353.98 sekund/hru |
| Paralelně | 17.46 sekund/hru | 45.3 sekund/hru | 253.48 sekund/hru |

Tabulka 4.1: Porovnání rychlosti sériového a paralelního výpočtu.

verze výpočtu strategie, například možnost pamatovat si hodnotu všech mezivýsledků, které mohou být pro různé stavy shodné.

Proto jsme oba způsoby výpočtu zkusili spustit na počítači[6] a porovnat čas výpočtu, viz tabulka 4.1. Výsledky jsou počítány jako průměr z délky pěti náhodných her a samozřejmě jsou závislé na mnoha parametrech. Jde nám ale spíše o porovnání sériového a paralelního výpočtu vzhledem ke hloubce prohledávání. Z tabulky je vidět, že metoda využívající `std::future` je vždy rychlejší.

### 4.2.7 Uživatelské rozhraní

V samostatném hlavičkovém souboru máme s pomocí knihovny SFML definované různé objekty, které pomohou uživateli s ovládáním programu, jako jsou tlačítka, textová pole a popisky. Možná by bylo jednodušší využít už hotové objekty se stejnou funkcí z nějaké jiné knihovny, bylo ale velmi zajímavé vyzkoušet si, jak takové věci mohou být implementované.

### 4.2.8 Global

Poslední důležitou součástí kódu je hlavičkový soubor `Global.hpp`. Je využíván napříč celým kódem a jsou v něm definovány různé výčtové typy, konstanty, pomocné funkce nebo aliasy použitých datových struktur.

[6] Dell G5 5587
Windows 10
Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz, 2304 Mhz, jádra: 4, logické procesory: 8
RAM 8,00 GB

# 5. Uživatelská dokumentace

## 5.1 Pravidla hry

Hru je vytvořena podle oficiálních pravidel[1]. Pravidla implementovaná v našem programu jsou téměř totožná, liší se pouze v několika detailech, které souvisí hlavně s jejím převodem do počítačové verze.

Nejdůležitější rozdíl spočívá ve způsobu obchodování mezi hráči. Desková verze hry počítá s živými hráči, kteří mezi sebou mohou různě smlouvat. V počítačové verzi hry je smlouvání samozřejmě dosti omezené. Pokud spolu hrají lidští hráči, mohou samozřejmě smlouvat v reálném světě a výsledek jenom zadat do počítače, při hře s umělou inteligencí by ale smlouvání bylo velmi složité. Proto je obchod mezi hráči omezen na kladení nabídek, které protihráči mohou přijmout nebo odmítnout, aniž by mohli učinit nějakou protinabídku. Pro zjednodušení jsme navíc stanovili limit počtu nabídnutých surovin. Není možné směnit více než dvě suroviny za jednu jinou surovinu. Myslíme si, že toto zjednodušení neovlivní hru nijak výrazně, neboť ostatní obchody nejsou příliš výhodné.

Ve stolní verzi by také bylo možné, aby se v případě více zájemců o obchod hráč na tahu rozhodl, s kým bude obchod proveden. Hráči se mohou překřikovat a na jejich pořadí přitom příliš nezáleží. Zde jsme se, opět pro jednoduchost, rozhodli, že se hráč bude na obchod ptát postupně a suroviny smění s prvním protihráčem, který nabídku přijme.

Druhý rozdíl je spíše detail – počet surovin ve hře není omezen počtem natištěných karet. Oproti stolní verzi hry se tedy může stát, že si hráči nalížou desítky surovin. Myslíme si, že to není ani tak pravidlo, jako spíš omezení fyzické verze hry. Protože kartičky navíc dojdou jen výjimečně, rozhodli jsme se toto pravidlo vynechat.

## 5.2 Nastavení hry

Uživatel má možnost zvolit způsob rozmisťování kostiček na začátku hry a může vybrat hráče, kteří se hry budou účastnit. Tyto volby může uživatel provést dvěma způsoby.

### 5.2.1 Běžní uživatelé

Méně zkušených uživatelů se program na začátku zeptá, jestli si přeje sestavit začátečnickou mapu, tedy takovou, jaká je znázorněna v přílohách na obrázku A.1, nebo jestli si přeje rozmisťovat políčka a přístavy náhodně způsobem popsaným v almanachu[2] v sekci „Výstavba variabilního plánu“. Toto nastavení je možné zvolit pouze na začátku hry a zůstane platné až do ukončení programu.

Na začátku každé hry bude uživatel vyzván k výběru hráčů, kteří se hry zúčastní. Vybírat může mezi lidskými hráči a předpřipravenou sadou hráčů počítačových. Při výběru lidských hráčů uživatel napíše hráčovo jméno do textového

[1] Dohledatelné například na oficiálních Českých stránkách hry (Albi)

[2] Součást oficiálních pravidel hry: Podrobný výklad pravidel a příklady ke hře

pole a potvrdí stiskem klávesy Enter. Textové pole je aktivní a reaguje na události, pokud je jeho pozadí žluté, jinak je potřeba ho označit kliknutím myši. Počítačoví protihráči už mají přezdívky zvolené. Každý protihráč může využívat jiný způsob ohodnocení stavů. Jejich nastavení si více popíšeme v sekci 5.5. Ve hře se počítá se třemi nebo čtyřmi hráči. Pokud uživatel při výběru čtvrtého hráče stiskne klávesu Escape, bude hra spuštěna pouze se třemi hráči.

### 5.2.2 Zkušenější uživatelé

Zkušenější uživatelé mohou využít možnosti spustit program s argumenty. První argument je tvaru `true/false` a odpovídá na otázku, jestli chce uživatel rozmisťovat mapu náhodně. Pokud je tedy první argument roven hodnotě `"true"` nebo `"t"`, bude každou hru mapa rozmisťována náhodně. Pokud bude první argument roven čemukoli jinému, bude použito začátečnické rozvržení mapy.

Pokud uživatel spustí program jenom s jedním argumentem, bude stejně jako při spuštění bez argumentů na začátku každé hry dotazován na hráče. Program spuštěný s více než jedním argumentem hledá přezdívky možných protivníků a přidává je do seznamu. Pokud mezi zbylými argumenty budou jména třech až čtyř hráčů (jména se mohou opakovat), budou se každé hry účastnit právě tito hráči. Pokud bude tímto způsobem zvolen jenom jeden nebo dva hráči, je ve hrách používán vícekrát tak, aby hru hráli celkem čtyři hráči. Pro ilustraci uvádíme příklady spuštění programu s argumenty:

- `OsadniciZKatanu.exe false Modrozoubek` spustí hru se začátečnickým rozmístěním políček na mapě a čtyřmi Modrozoubky.
- `OsadniciZKatanu.exe t Modrozoubek Beruska` spustí hru s náhodným rozmisťováním políček na mapu se dvěma Modrozoubky a se dvěma Beruškami.
- `OsadniciZKatanu.exe true Modrozoubek Turbo Beruska Boruvka` spustí hru s náhodným rozmístěním políček na mapě, kde proti sobě budou hrát Modrozoubek, Turbo, Beruška a Borůvka.

Tento způsob přidávání hráčů je myšlený hlavně pro opakované hry při učení hráčů, u kterých by opakované vybírání hráčů zbytečně zdržovalo. Myslíme si, že při hře lidských hráčů takový přístup nemá moc smysl a proto je možné tímto způsobem přidávat pouze počítačové hráče.

## 5.3 Popis grafického rozhraní

Po počátečním nastavování se uprostřed okna objeví rozmístěná mapa. V horní části jsou vypsaná jména všech hráčů. Po najetí myší na jméno některého z nich se zobrazí základní veřejně viditelné informace o daném hráči.

V pravé části okna jsou tlačítka, kterými uživatel může volit své akce. Některé vyžadují dodatečné upřesnění. Může jít o volbu nějaké pozice na mapě. V takovém případě se zvýrazní všechny možné pozice a uživatel některou vybere přímo kliknutím do mapy.

Některé akce vyžadují volbu surovin. V takovém případě se v pravé části obrazovky otevře okno s tlačítky, jejichž stisknutím hráč suroviny vybere.

Věříme, že použití většiny akcí je intuitivní a není potřeba dále rozebírat. Za zmínku ale stojí obchod s hráčem, který možná tak úplně intuitivní být nemusí. Uživatel nejprve zvolí první surovinu, kterou chce nabídnout protihráči. Následně může zvolit druhou surovinu. Pokud chce protihráči nabídnout jenom jednu surovinu může uživatel stisknout klávesu Escape. Následně může zvolit surovinu, kterou od protihráče požaduje.

Obecně platí, že pokud si uživatel rozmyslí nějakou svoji volbu ještě předtím, než se celá akce použije, může se vrátit stisknutím klávesy Escape.

V levé části jsou vypsány veškeré informace o hráči, který je aktuálně na tahu. Text úplně nahoře vždy říká, co se právě děje. To se hodí hlavně u volby složitějších akcí, jako například zmiňovaný obchod s hráčem.

## 5.4 Zobrazení okna

Při spuštění programu se skryje konzole a otevře se okno, které svojí velikostí odpovídá velikosti obrazovky. S oknem je možné manipulovat podobně, jako tomu jsme zvyklí u jakéhokoli jiného okna. Tato nastavení můžeme za běhu programu upravit.

Stisknutím klávesy **F1** můžeme kdykoli znovu zobrazit nebo skrýt konzoli. Klávesa **F5** slouží k přepínání mezi fullscreen a okenním režimem.

## 5.5 Nastavení protihráčů

Kromě uživatelem ovládaného hráče je do hry možné přidat jednoho z 11 připravených agentů. Ti jsou rozděleni do tří kategorií podle způsobu výběru akcí a učení.

- **True Online TD($\lambda$)** Učí se za pomoci algoritmu True Online TD($\lambda$) (algoritmus 2). Mezi tyto agenty patří Modrozoubek, Turbo, Anča, Beruška, Borůvka a Drátek.
- **Fixní hráč** Tento agent využívá váhy, které převzal od jiného agenta, sám se neučí. Je tedy vhodný pro testování a porovnávání agentů. Fixními hráči jsou Houba, Piškot, Žofka a Lišák.
- **Náhodný hráč** Stejně jako fixní hráč se neučí žádné váhy, ale ani žádné nevyužívá. Každý jeho tah je náhodný. Jediným náhodným hráčem ve hře je Bahňák.

Při výběru hráčů s pomocí konzole je potřeba napsat jejich jména bez diakritiky.

Data každého z hráčů jsou uložena ve složce `../Hraci/Typ_Jmeno`. Nejdůležitějšími soubory ve složce jsou `weights.txt` a `config.txt`.

V souboru `weights.txt` jsou uloženy všechny agentovy nabyté vědomosti o hře. Jejich úpravou můžeme změnit chování hráče. Váhy jednotlivých příznaků jsou uloženy ve stejném pořadí, v jakém jsou vypsány v sekci 3.4.3. Uživatel by měl mít na paměti, že přepsáním vah může znehodnotit zkušenosti z tisíců tréninkových her.

| Klíč | Defaultně | Popis |
|---|---|---|
| hloubka | 1 | Hloubka plánování tahu a hodnocení expectimaxem |
| pocet_akci | 3 | Počet nejlepších akcí, se kterými bude při plánování dále počítáno |
| pocet_tahu | 9 | Počet nejlepších posloupností akcí, se kterými bude počítáno dále do hloubky |
| pocet_tahu_exp | 3 | Počet nejlepších posloupností akcí vygenerovaných v průběhu expectimaxu, se kterými bude dále počítáno do hloubky |
| tau | 10 | Odhadovaný počet průchodů jedním stavem, než se agent naučí správně jeho váhy. Parametr $\alpha$ se následně vypočítá jako $\alpha = \frac{1}{400\tau}$ |
| gamma | 0.9 | Diskontní faktor |
| epsilon | 0.1 | Pravděpodobnost náhodného tahu |
| default_reward | -5 | Odměna, kterou agent dostane po každém tahu kromě posledního. |
| lambda | 0.8 | Parametr pro TD($\lambda$) |

Tabulka 5.1: Tabulka parametrů agenta

V souboru `config.txt` jsou uloženy všechny parametry využívané agentem při učení a rozhodování. Parametry jsou v souboru uloženy vždy na samostatném řádku ve tvaru `klíč=hodnota`. Řádky začínající znakem `'#'` jsou považovány za komentář a bílé znaky jsou ignorovány.

Parametry, které může uživatel upravovat, jsou vypsány v tabulce 5.1.

V přílohách této práce je uvedený příklad, jak může soubor `config.txt` vypadat (příloha B).

Ostatní soubory ve složce nemají žádný vliv na chování agenta a slouží hlavně pro sběr statistik.

## 5.6 Nastavení grafiky

Textury k obrázkům surovin, políček pevniny a pobřeží s přístavy jsou uloženy ve složce `../Grafika`. Obrázky jsme kreslili v Gimpu. Pokud by uživatel chtěl hrát hru s vlastním vzhledem, může si nakreslit vlastní, vložit je do příslušné složky a pojmenovat stejně jako původní textury.

**Políčka surovin** V podsložce `policka_surovin` jsou umístěny textury políček surovin, ze kterých je poskládána herní plocha. Pojmenovány jsou vždy ve tvaru `policko_surovina.jpg`, kde „surovina“ je nahrazena názvem suroviny bez diakritiky.

**Políčka pobřeží** V podsložce `Pristavy` jsou umístěny textury pobřeží a přístavů. Barva moře je zvolena tak, aby se shodovala s barvou pozadí okna se hrou, která je zvolena napevno. Vzhledem k možnosti náhodného rozmístění přístavů

okolo ostrova jsou všechny textury malovány tak, jako by měly přiléhat k hornímu okraji ostrova a jsou rotovány později podle potřeby.

Některé přístavy přiléhají k levému spodnímu rohu, jiné k pravému spodnímu rohu. Jejich pozice byla zvolena podle obrázků v oficiálních pravidlech a je nutné ji zachovat, aby obrázky ve hře odpovídaly skutečným pozicím přístavů.

Obrázky pobřeží bez přístavu mají jméno `policko_more_dve_strany.jpg` nebo `policko_more_jedna_strana.jpg` podle toho, jestli mají pobřeží na obou spodních hranách nebo jenom na té pravé. Obrázky pobřeží s přístavy jsou pojmenovaná ve tvaru `policko_more_pristav_surovina.jpg`. „Surovina“ je opět nahrazena názvem suroviny bez diakritiky. V případě obecného přístavu je namísto suroviny `obecny_1` nebo `obecny_2` opět v závislosti na tom, jestli má políčko pobřeží na jedné nebo na dvou stranách.

**Obrázky surovin** V podsložce `Znacky_surovin` jsou uloženy obrázky samotných surovin. Obrázky pojmenované jako `surovina.png` nejsou použity přímo ve hře, sloužily hlavně k vytváření ostatních textur.

Textury pojmenované ve tvaru `surovina_s_mistem` jsou obrázky surovin zmiňované v předchozím odstavci vložené do obdélníku s poměrem stran 240x180 tak, aby v levé části bylo vynechané místo pro doplňující text. Tyto textury jsou využity pro tlačítka, kterými si hráč vybírá suroviny.

**Font** Kromě textur je ve složce s grafikou uložený také font používaný všude ve hře. Využili jsme zdarma dostupný font Dosis[3]. Protože rozměry a umístění mnoha objektů byly zvoleny podle tohoto fontu a jiné fonty mohou mít odlišné rozměry znaků, není uživateli doporučeno font měnit.

[3]https://fonts.google.com/specimen/Dosis (přístup 31. 5. 2020)

# 6. Výsledky

V této kapitole si řekneme, jak se nám povedlo metody popsané v předchozích dvou kapitolách propojit a implementovat. Popíšeme si, na jaké problémy jsme narazili, co jsme museli ještě upravit a nakonec zhodnotíme, jak úspěšní jsme byli.

## 6.1 Rozmístění herního pole

V prvních verzích našeho programu jsme využívali pouze fixní rozmístění herního pole (viz obrázek A.1). Předpokládali jsme, že když se agent nebude učit odhad hodnot konkrétních pozic, ale jejich vlastností podle sousedních polí, snadno se dokáže adaptovat i pro jiná rozmístění mapy a naučit agenta hrát s fixním rozmístěním políček je jednodušší. Ukázalo se ale, že takový přístup nefunguje zcela správně.

Tím, že se agenti učí na jedno konkrétní rozmístění mapy, dojde k přetrénování. Agenti se například naučili, že pokud mají vysokou pravděpodobnost zisku vlny, je zároveň velmi výhodné mít nízkou cenu obchodu s vlnou. U ostatních surovin to za tak výhodné nepovažovali. To proto, že na základním rozmístění se nachází přístav s vlnou hned na dvou políčkách s vlnou, zatímco u ostatních surovin museli agenti mít vesnice na různých místech, což se zdaleka nestávalo tak často.

Proto jsme začali pole rozmisťovat náhodně způsobem, jaký je popsaný v pravidlech hry.

## 6.2 Testy

Protože agenti mají mnoho nastavitelných parametrů ovlivňujících rychlost a kvalitu učení, nemohli jsme otestovat všechna možná nastavení a vybrat to skutečně optimální. Rozhodli jsme se proto optimální hodnotu některých parametrů odhadnout nebo zvolit na základě doporučení z literatury a otestovat jenom některé.

V první části jsme vyzkoušeli různá nastavení parametru $\lambda$, tedy různá nastavení algoritmu pro učení vah.

Ve druhé části experimentu jsme otestovali, jestli expectimax dokáže (v rozumném čase) vylepšit chování agenta. Vybrali jsme vítěze z předchozího testování, jeho váhy jsme předali dalším agentům a každému jsme zkusili nastavit jinou hloubku expectimaxu nebo parametry větvení.

V poslední části testování jsme vyzkoušeli doposud nejlepšího počítačového hráče proti člověku.

### 6.2.1 Učení

#### Self-play

V první fázi testování jsme naučili hrát tři agenty s různě zvoleným parametrem $\lambda$. Agenti se učili nezávisle na sobě tak, že hráli sami proti sobě vždy

| **Počet odehraných her** | $\epsilon$ |
|---|---|
| 0-500 | 0.4 |
| 500-1000 | 0.2 |
| 1000-2000 | 0.1 |
| 2000-5000 | 0.05 |

Tabulka 6.1: Nastavení parametru $\epsilon$

ve čtyřech hráčích (self-play). Na každou hru se tedy agenti dívali z pohledu čtyř hráčů a naučili se čtyři odlišné váhové vektory. Ty se po každé hře zprůměrovaly do nového pro další hru. Celkově jsme každého agenta nechali tímto způsobem odehrát 5200 her.

**Parametr $\lambda$** Hodnoty parametru $\lambda$ jsme zvolili následovně:

- $\lambda = 0.0$: Algoritmus TD(0) odpovídá jednokrokové temporální diferenci ze sekce 3.3.2.
- $\lambda = 0.8$: Podle výsledků experimentů z literatury na problému náhodné procházky je toto nejvhodnější volba (Sutton a Barto, 2018).
- $\lambda = 1.0$: Algoritmus TD(1) odpovídá metodě Monte Carlo ze sekce 3.3.1

**Parametr $\epsilon$** Parametr $\epsilon$ jsme zpočátku nastavili na vyšší hodnotu, aby agenti více prozkoumávali neznámé tahy. V průběhu učení jsme ho v závislosti na počtu odehraných her postupně snižovali způsobem znázorněným v tabulce 6.1.

**Parametr $\tau$** Rychlost učení jsme se vzhledem k počtu příznaků snažili nastavit co nejrychlejší. Proto jsme zvolili $\tau = 10$ (viz sekce 3.5).

**Parametr $\gamma$** Parametr $\gamma$ určuje diskontní faktor, tedy jakou váhu bude agent přikládat budoucím stavům a odměnám. Protože agent dostává největší odměnu až v posledním stavu hry, chceme, aby agent sledoval spíše hodnoty vzdálenějších stavů. Zvolili jsme tedy $\gamma = 0.9$.

**Odměny během hry** Pro parametr `default_reward` jsme zvolili hodnotu -5 tak, aby byla záporná a v poměru k odměně získané na konci hry malá.

**Parametry expectimaxu** Cílem tohoto experimentu bylo otestovat hlavně zpětnovazební učení. Chtěli jsme tedy odehrát velké množství her v rozumném čase. Proto jsme nastavili hloubku prohledávání na 1.

Počet nejlepších akcí jsme nastavili na tři. Tím jsme zredukovali jejich množství a zrychlili tím rozhodování, zároveň jsme nechali nějaký prostor k vygenerování více možných tahů.

Nastavení počtu nejlepších tahů není v tomto experimentu vzhledem k nízké hloubce důležité.

**Výsledky** Během učení jsme u každého agenta měřili délku hry a průměrný počet bodů poražených hráčů. Délku hry proto, že čím rychleji dokáže agent vyhrát hru, tím těžší je ho porazit. Průměrný počet bodů poražených jsme měřili, abychom zjistili, jestli agent hraje podobně pokaždé, nebo má při výhře jenom štěstí.

Z naměřených čísel jsme načrtli dva grafy porovnávající průběh algoritmů s různými nastaveními $\lambda$. Protože grafy ze samotných naměřených čísel by byly nepřehledné, v grafech jsme zobrazili klouzavý průměr hodnot přes 200 her, tedy na prvním místě je průměr první až dvousté hodnoty. Výsledky měření jsou v příloze C.1. Z obou grafů je vidět, že se všichni agenti s rostoucím časem zlepšují.

Nakonec jsme zkusili váhy zafixovat a nechali jsme proti sobě hrát tři různě naučené agenty a navíc jednoho, který hraje náhodně. Jako nejlépe naučený vyšel agent, který se učil s parametrem $\lambda = 1.0$. Z přibližně dvou tisíců her jich vyhrál 55 %, tedy více než ostatní hráči dohromady. To neznamená, že je tato metoda nejlepší, protože jsme zastavili učení v náhodném čase a z přiložených grafů je patrné, že kvalita agentů se v čase měnila. Tyto vítězné váhy ale budeme nadále využívat v příštích experimentech.

#### Učení více agentů v jedné hře

V další části experimentu jsme se rozhodli nechat agenty naučit se váhy od začátku, tentokrát ale budou hrát všichni dohromady a čtvrtý hráč bude zafixovaný agent s váhami z minulého experimentu. Tím budou mít o něco ztížené podmínky, protože zpočátku nebudou dostávat kladné odměny za výhru. Navíc může být učení pomalejší a nestabilnější vzhledem k tomu, že se váhy neprůměrují napříč agenty.

Parametry agentů jsme nastavili stejně jako v předchozím pokusu. Parametry $\lambda$ byly opět rovné 0.0, 0.8 a 1.0, parametr $\epsilon$ jsme opět snižovali s rostoucím časem. Protože hráli všichni agenti ve stejné hře, nebylo nutné počítat počty her před změnou $\epsilon$ tak důkladně, jako při minulém experimentu.

Experiment jsme provedli třikrát a výsledky jsme počítali z průměru hodnot získaných v těchto pokusech. Na obrázku C.3 jsou znázorněny počty výher jednotlivých agentů v průběhu dalších 100 her. Je vidět, že všichni agenti zlepšují své chování. Nedokázali se ale dostat na takovou úroveň, aby vyhrávali více, než původní hráč.

### 6.2.2 Expectimax

Přestože agenti dokáží hrát poměrně dobře i bez využití expectimaxu, není toto chování optimální, protože nedokáže vždy správně odhadnout, co se může stát v budoucnosti. Pokud bychom se dokázali podívat expectimaxem o několik tahů dopředu, nepochybně by došlo ke zlepšení chování agenta. Protože ale doba výpočtu tahu roste s hloubkou exponenciálně rychle, je možné podívat se vždy jen o malý kousek dopředu. Cílem tohoto experimentu je zjistit, jestli má tato limitovaná hloubka prohledávání na kvalitu agenta nějaký vliv.

Pro experiment jsme vybrali nejlepší váhy z prvního experimentu a předali je různým zafixovaným agentům, aby měli všichni během učení stejné váhy a rozhodovalo pouze nastavení expectimaxu.

**Parametry** Protože v tomto experimentu se agenti nic neučí, nemají parametry učení ($\alpha, \gamma, \lambda$, `default_reward`) na chování agenta žádný vliv. Parametr $\epsilon$ vliv na agentovu strategii má, ale protože pro nás není důležité prozkoumávat, nastavíme ho na nízkou hodnotu ($\epsilon = 0.05$).

Důležitější pro nás bylo nastavení hloubky prohledávání a větvení. Cílem bylo primárně vyzkoušet různé hloubky prohledávání. Aby výpočet proběhl v rozumném čase, budeme u vyšších hloubek muset více omezit větvení.

Zároveň jsme vyzkoušeli, jestli se agent bez expectimaxu bude chovat lépe, pokud méně omezíme počet nejlepších akcí. Jak jsme psali v závěru kapitoly 2, při plánování stavby silnic využíváme heuristiku, která částečně nahrazuje prohledávání do hloubky. Proto může být lepší volit menší počet uvažovaných akcí.

Volby parametrů expectimaxu a větvení čtyř agentů v tomto experimentu jsou vypsány v tabulce 6.2.

| | **Hloubka** | **Počet akcí** | **Počet tahů** | **Počet tahů exp.** |
|---|---|---|---|---|
| **Houba** | 1 | 3 | 1 | 1 |
| **Piškot** | 1 | 9 | 1 | 1 |
| **Žofka** | 3 | 3 | 3 | 2 |
| **Lišák** | 5 | 2 | 2 | 2 |

Tabulka 6.2: Tabulka s nastavením jednotlivých hráčů pro test expectimaxu

Protože využití expectimaxu zabere mnohem více času, nebyli jsme schopni provést tolik testovacích her jako u minulého experimentu. Výhodou naopak je, že se agenti nemění v čase, takže jednotlivé hry jsou na sobě navzájem nezávislé.

| | **Počet vítězství** | **Procentuálně** |
|---|---|---|
| **Houba** | 214 | 20% |
| **Piškot** | 235 | 22% |
| **Žofka** | 292 | 28% |
| **Lišák** | 304 | 29% |

Tabulka 6.3: Výsledky měření expectimaxu

V tabulce 6.3 jsou uvedeny počty vítězství jednotlivých agentů během experimentu. Z tabulky je patrné, že agenti s vyšší hloubkou prohledávání mají lepší výsledky.

Jelikož se jedná o nezávislé pokusy, můžeme z výsledků experimentu spočítat intervalový odhad pravděpodobnosti vítězství Lišáka, tedy hráče s nejvyšší

hloubkou prohledávání. Zvolili jsme 95% interval spolehlivosti.

$$
\begin{aligned}
t_{1-\frac{\alpha}{2}} &= t_{0.975} = 1.960 \\
IntervalSpolehlivosti &= [\overline{X}_n - \sqrt{\frac{S_n^2}{n}} t_{1-\frac{\alpha}{2}}; \overline{X}_n + \sqrt{\frac{S_n^2}{n}} t_{1-\frac{\alpha}{2}}] \\
&\doteq [0.291 - \sqrt{\frac{0.206}{1045}} 1.960; 0.291 + \sqrt{\frac{0.206}{1045}} 1.960] \\
&\doteq [0.291 - 0.028; 0.291 + 0.028] \\
&\doteq [0.263; 0.318]
\end{aligned}
$$

Kdyby neměl expectimax na kvalitu agenta žádný vliv, měli by všichni agenti stejnou pravděpodobnost výhry, tedy 0.25. Zároveň můžeme říct, že pravděpodobnost výhry s prohledáváním stavů do hloubky je s pravděpodobností 95 % vyšší než 0.26. S velkou pravděpodobností je tedy agent používající expectimax skutečně lepší, než agent, který se řídí pouze hodnotami ze zpětnovazebního učení.

Naopak zvýšení počtu generovaných nejlepších akcí nepřineslo žádné znatelné zlepšení. To poukazuje na účinnost heuristiky výběru nejlepších akcí.

### 6.2.3 Lidský protihráč

V posledním kroku jsme otestovali, jak dobře si vede námi implementovaná umělá inteligence proti průměrnému lidskému hráči. Rozhodli jsme se využít podobný způsob testování, jaký zvolil Pfeiffer (2004) s tím rozdílem, že proti umělé inteligenci nebude hrát expert na Osadníky z Katanu, ale několik hráčů od začátečníka až po mírně pokročilého hráče. Výsledky proto můžou být oproti předloze trochu zkreslené ve prospěch umělé inteligence.

Lidský hráč má v naší implementaci tu výhodu, že může nabízet obchod kolikrát chce, což jsme u počítačového hráče omezili z důvodu úspory času.

V experimentu jsme nechali člověka hrát několik her proti třem agentům s nejlepším nastavením z minulého testování. Agenti se lišili pouze jménem, aby byla hra uživatelsky přívětivější. V každé hře jsme měřili maximální, průměrný a minimální počet bodů získaných počítačovými hráči. Z těchto hodnot jsme nakonec spočítali průměr přes všech čtrnáct odehraných her. Výsledky jsou zobrazeny v tabulce 6.4.

| | |
|---|---|
| Nejlepší body: | 9.1 |
| Průměrné body: | 6.8 |
| Nejhorší body: | 4.4 |

Tabulka 6.4: Výsledné body počítačových agentů při hře proti lidskému hráči

Přestože mají počítačoví hráči omezený počet nabídek k obchodu, dokázali hrát obstojně a ve více než polovině her se jim podařilo lidské hráče porazit.

# Závěr

Podařilo se nám úspěšně implementovat Osadníky z Katanu v jazyce C++. Program běží stabilně a nepadá. I přes pár drobných nedostatků je uživatelské rozhraní dle vyjádření lidí, kteří hru testovali, přehledné a použitelné.

V další fázi jsme úspěšně aplikovali zpětnovazební učení na Osadníky z Katanu a integrovali ho s expectimaxem.

Nyní se zamyslíme nad oblastmi, které by bylo možné dále vylepšovat a rozvíjet. Největší rezervy jsou podle našeho názoru v tom, že agent nemá kompletní informace o stavu hry. Tím, že expectimax nedokáže prozkoumat hru dostatečně do hloubky, není schopen například naplánovat správně směr budované silnice. Tomu jsme se snažili zabránit heuristikou, pomocí které jsme se dívali kousek směrem, kterým cesta vede. To nám ale stále nezaručí, že agent nakonec zvolí nejlepší možnou cestu.

Navíc veškeré informace agenta o stavu musí být uloženy v množině příznaků, které jsou vyhodnocovány pomocí lineární funkce. Přestože počítáme i s příznaky znázorňujícími jednoduché vztahy mezi nimi, stále nemáme všechny informace. Řešením by mohlo být namísto lineární funkce využít neuronové sítě, které jsou ale samy o sobě poměrně složitou metodou a která už je mimo původně určené cíle práce.

Další možností by mohlo být přidat nové příznaky, které by doplňovaly informace, chybějící v současné verzi. Mohla by v nich být například počítána nějaká heuristika, díky které by se dále zmenšila potřeba prohledávat stavy do větší hloubky. Mohla by se třeba počítat vzdálenost všech křižovatek od nejbližší silnice hráče vážená atraktivitou dané křižovatky.

Přestože námi vyvinutá umělá inteligence není dokonalá, nakonec se naučila poměrně obstojně hrát a je schopná postavit se průměrnému lidskému protihráči.

# Seznam použité literatury


Albi. Osadníci z Katanu. https://www.albi.cz/hry-a-zabava/osadnici-z-katanu. Přístup: 3. 6. 2020.

Kratochvíl, P. (2016). Umělá inteligence pro osadníky. Bakalářská práce, MFF UK. URL https://is.cuni.cz/webapps/zzp/detail/154992/.

Pfeiffer, M. (2004). Reinforcement learning of strategies for settlers of catan. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.561.6293&rep=rep1&type=pdf.

Russell, S. a Norvig, P. (2009). *Artificial Intelligence: A Modern Approach (3rd Edition)*. Pearson. ISBN 0-13-604259-7.

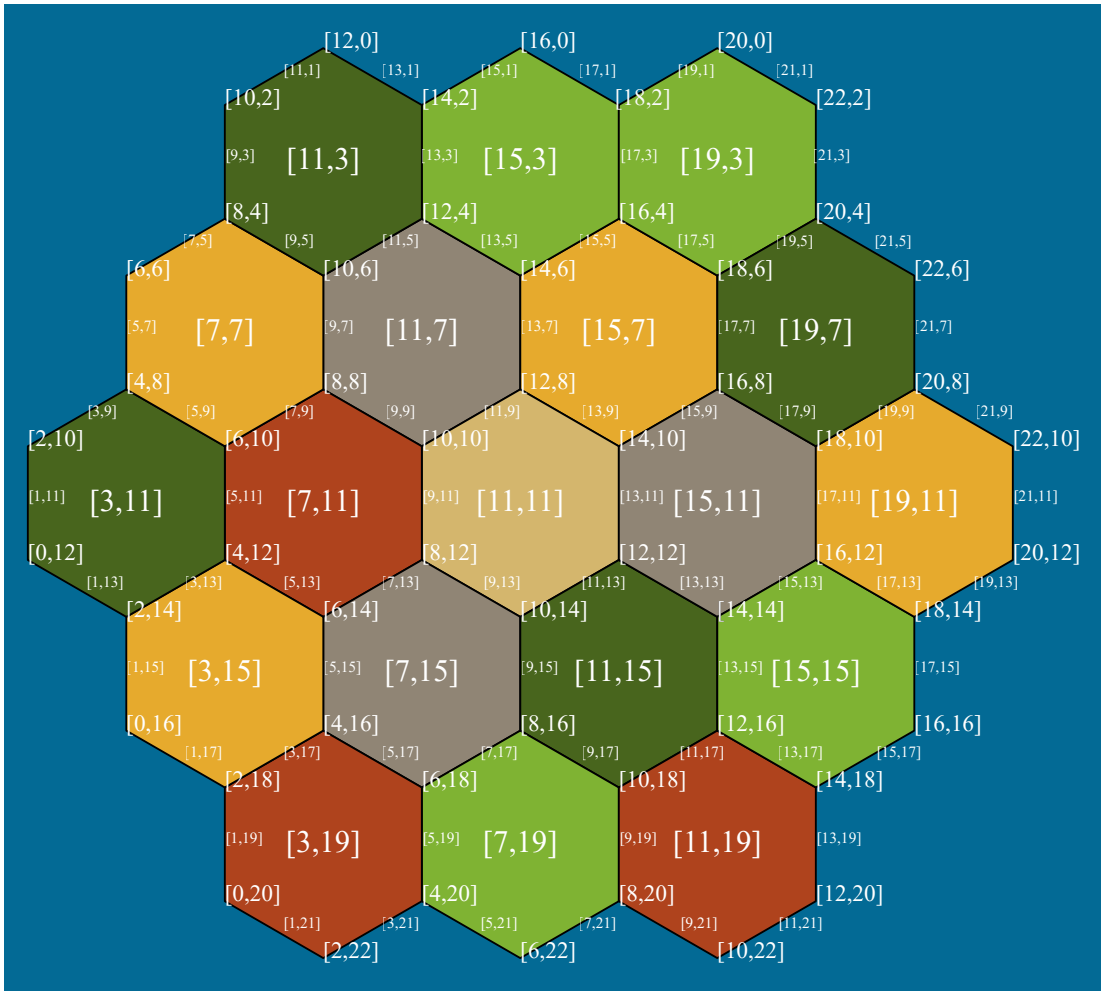
Sutton, R. S. a Barto, A. G. (2018). *Reinforcement Learning*. The MIT Press. ISBN 9780262039246. URL http://incompleteideas.net/book/RLbook2020.pdf.

van Seijen, H., Mahmood, A. R., Pilarski, P. M., Machado, M. C. a Sutton, R. S. (2016). True online temporal-difference learning. *Journal of Machine Learning Research*, **17**(145), 1–40. URL http://jmlr.org/papers/v17/15-599.html.

# Přílohy

# A. Rozvržení mapy


Obrázek A.1: Obrázek ze hry se základním rozmístění polí



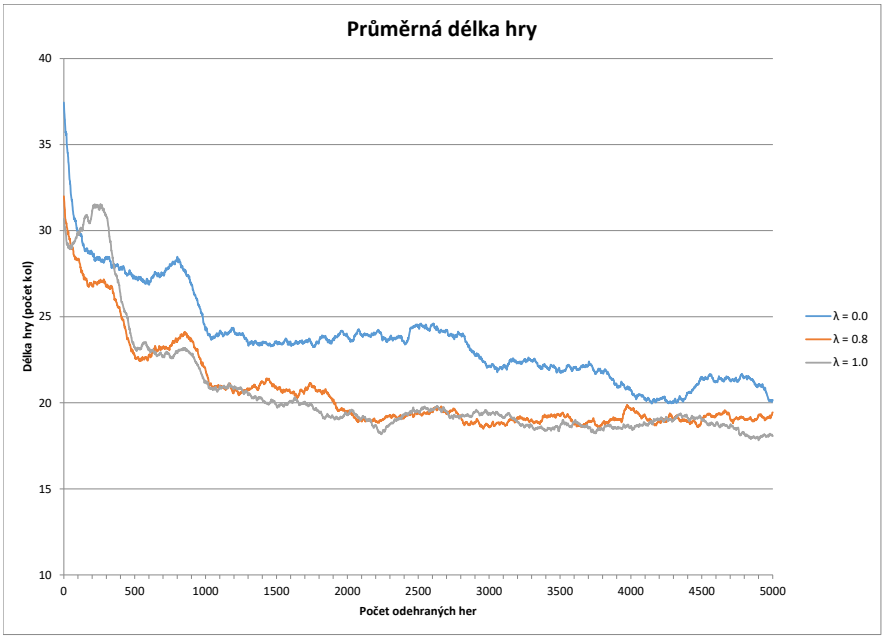

Obrázek A.2: Plánek znázorňující systém souřadnic v naší implementaci

# B. Příklad souboru s parametry agenta

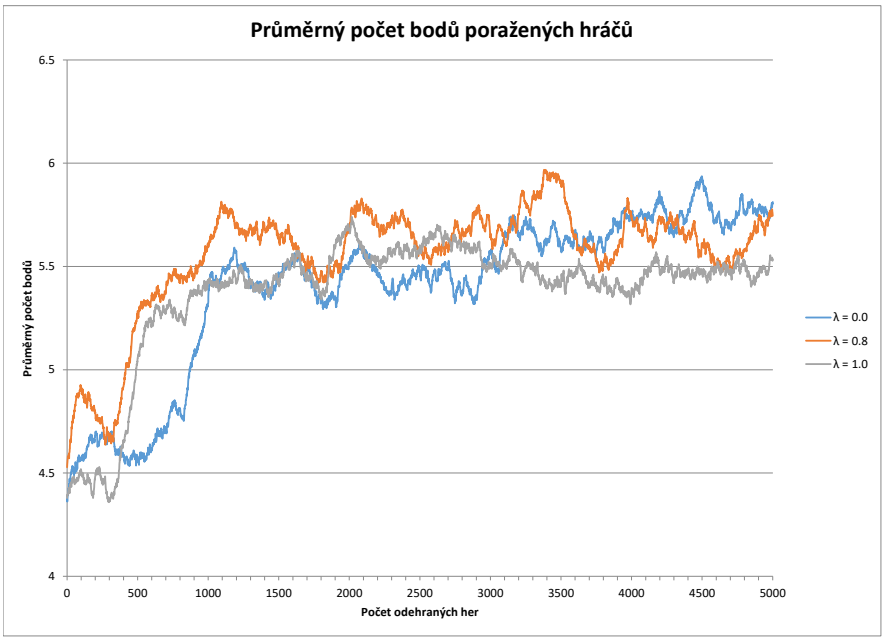
```
# hloubka expectimaxu
        hloubka=1
# Pocet nejlepsich akci, se kterymi bude pri planovani
# dal pocitano
         pocet_akci=3
# Pocet nejlepsich posloupnosti akci, se kterymi bude
# pri planovani dal pocitano
         pocet_tahu=9
# Pocet nejlepsich posloupnosti akci vygenerovanych
# v prubehu expectimaxu, se kterymi bude dal pocitano
        pocet_tahu_exp=3

# odhadovany pocet pruchodu jednim stavem, nez se nauci
# vahy. Alpha = 1 / (400 * tau)
        tau=10
# Diskontni faktor
        gamma=0.9
# Pravdepodobnost nahodneho tahu
        epsilon=0.05
# Trest za dlouhou hru. Tuto odmenu dostane agent kazdy
# tah krome posledniho
        default_reward=-5

# parametr pro TD(lambda)
         lambda=0.8
```

# C. Výsledky testování

## C.1 Průběh učení agentů s různým $\lambda$

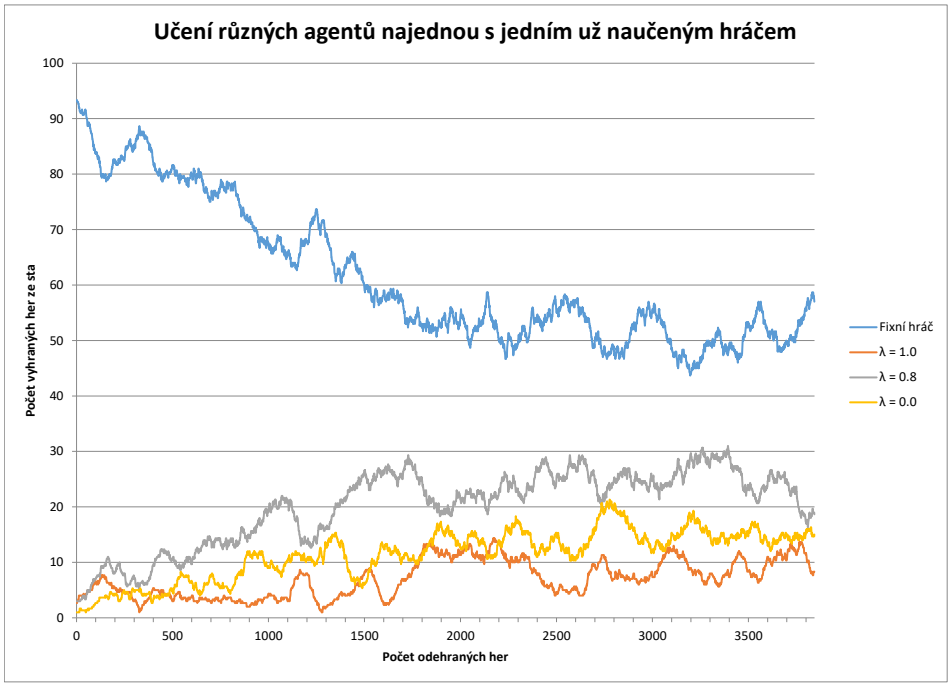



Obrázek C.1: Průměrná délka hry v závislosti na počtu odehraných her.




Obrázek C.2: Průměrný počet bodů v závislosti na počtu odehraných her.

## C.2 Porovnání agentů s různým $\lambda$ v jedné hře




Obrázek C.3: Průběh učení agentů při hře proti už naučenému hráči.

# D. Seznam elektronických příloh

- **Projekt C++** – Ve složce `OsadniciZKatanu` je projekt v jazyce C++ včetně zkompilovaného binárního souboru (ve složce Release), obrázků využitých ve hře (ve složce Grafika) a příkladů agentů s naučenými vahami (ve složce Hraci). Binární soubor byl zkompilován pro 32-bitový operační systém Windows.
- **Dokumentace** – Ve složce `Dokumentace` je výstup z programu Doxygen v podobě HTML stránky.
- **Výsledky experimentů** – Složka `Vysledky experimentu` obsahuje `*.csv` soubory s čísly, ze kterých byly vypočítány výsledky experimentů v této práci.