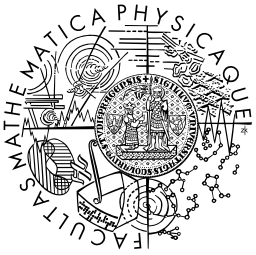


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Ondřej Motlíček

Umělá inteligence pro hru Carcassonne – Objevitelé

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(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

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

V ..... dne ....

Podpis autora

Chtěl bych poděkovat RNDr. Janu Hricovi za vedení této práce a nespočet podnětů na zlepšení. Dále děkuji své rodině, přítelkyni a blízkým, kteří mě podporovali při psaní práce, zejména mému kamarádovi Jaroslavu Kořínkovi za pomoc při testování programů vytvořených v rámci této práce a užitečné poznámky k textu práce.

Název práce: Umělá inteligence pro hru Carcassonne – Objevitelé

Autor: Ondřej Motlíček

Katedra: Katedra teoretické informatiky a matematické logiky

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

Abstrakt: Bakalářská práce se zabývá vývojem umělých inteligencí pro hru Carcassonne – Objevitelé. V této práci jsou představeny různé metody pro vytvoření umělé inteligence. Jsou zde vytvořeny heuristické funkce zaměřující se na různé aspekty hry. Pro prohledávání stavového prostoru hry jsou použity Monte Carlo metody a algoritmus Expectiminimax. Navržené metody umělých inteligencí jsou implementovány a experimentálně porovnány pomocí simulací vzájemných střetnutí. Jsou zde představeny a vysvětleny získané výsledky her umělých inteligencí. V rámci simulační prostředí jsou implementovány programy umožňující hru umělých i lidských hráčů s důrazem na automatickou dávkovou simulaci her umělých inteligencí.

Klíčová slova: hra, Carcassonne – Objevitelé, umělá inteligence, Monte Carlo metody, Expectiminimax

Title: Artificial intelligence for Carcassonne – The Discovery

Author: Ondřej Motlíček

Department: Department of Theoretical Computer Science and Mathematical Logic

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

Abstract: The bachelor paper deals with the development of an artificial intelligence for the game Carcassonne – The Discovery. Different approaches for designing an artificial intelligence are presented. Heuristic functions based on various aspects of the game. Monte Carlo methods and the Expectiminimax algorithm are used for state space of the game. The designed methods are implemented and experimentally tested and compared by simulations of the game between the artificial players. Results of the experiment are presented and explained. The simulation environment consists of multiple programs for the game simulation of both artificial and human players. A batch simulation of the artificial intelligence is emphasized.

Keywords: game, Carcassonne – The Discovery, artificial intelligence, Monte Carlo methods, Expectiminimax

Obsah

Úvod

Carcassonne – Objevitelé je společenská strategická desková hra. Vytvořil ji v roce 2005 Leo Colovini na motivy populární hry *Carcassonne* od Klause-Jürgena Wredeho. Objevitelé jako varianta klasické hry nedosáhla takové obliby jako původní hra. Kvůli svým odlišnostem se však jedná o strategicky zcela odlišnou hru, za což je často, a dle mého názoru neprávem, kritizována.

„Klasický“ Carcassonne analyzoval ve své bakalářské práci Michal Nekvinda (Nekvinda, 2018). Má práce se inspiruje námětem na pokračování a zabývá se hrou Carcassonne – Objevitelé, která má sice podobná pravidla, ale je strategicky velmi odlišná.

Cílem této práce je navrhnout a implementovat umělé inteligence pro hru Carcassonne – Objevitelé, které používají různé přístupy. Konkrétně se zaměříme na algoritmus Expectiminimax, strojové učení a Monte Carlo metody. Experimentálně vyhodnotíme jejich výkonnost. Součástí práce je také implementovat program pro vizualizaci hry a možnost hry člověka.

Tato práce je rozdělena na 5 kapitol. V kapitole 1 jsou analyzovány herní prvky. Kapitola 2 je zaměřena na návrh umělých inteligencí podle jednotlivých metod. V kapitole 3 jsou představeny a zhodnoceny výsledky her mezi umělými inteligencemi. Kapitola 4 dokumentuje prostředí, ve které probíhá simulace her. V závěru práce jsou shrnuty dosažené výsledky.

1. Analýza

V této kapitole nejprve stručně shrneme pravidla hry a porovnáme je s pravidly „klasického“ Carcassonne. Poté definujeme pojmy, které budeme používat při návrhu umělých inteligencí.

1.1 Pravidla hry

V této části popíšeme základní pravidla hry. Úplné znění pravidel v českém jazyce je k dispozici v elektronické příloze práce.

Hru hraje 2 až 5 hráčů. Před začátkem hry se zamíchají kartičky s prvky krajiny na hromádku tak, aby byla vidět zadní strana kartiček. Jedna předem určená (startovní) kartička se položí přední stranou vzhůru. Každý z hráčů dostane 4 figurky jedné barvy, každý hráč má vlastní barvu figurek. V průběhu hry se hráči střídají v tazích, po celou dobu hry je dodržováno stejné pořadí hráčů na tahu.

Hráč si v každém tahu vylosuje kartičku a položí ji k již položeným kartičkám tak, aby nově přidaná kartička sousedila stranou s alespoň jednou již položenou kartičkou. Pokud kartička nelze umístit, hráč si vylosuje jinou kartičku. Poté hráč dokončí tah jedním ze tří možných způsobů:

- Pokud má hráč k dispozici figurku, může ji položit na některý region na kartičce, kterou položil v tomto tahu. Region nesmí být obsazený žádnou figurkou. Po položení figurky následuje tah dalšího hráče.
- Pokud má hráč položenou figurku, může ji odebrat. Za odebrání figurky získává body podle pravidel bodování. Po odebrání figurky následuje tah dalšího hráče.
- Pokud hráč nechce nebo nemůže položit nebo odebrat figurku, předává tah dalšímu hráči.

Jakmile nejsou na hromádce žádné další kartičky, probíhá koncové zúčtování. Odebírají se všechny umístěné figurky a hráči za ně získávají body podle pravidel bodování. Po konečném zúčtování vyhrává hráč s nejvyšším počtem bodů.

Bodování se řídí typem regionu a uzavřeností. Rozlišujeme 3 typy regionů: hory, nížiny a moře. Region je uzavřený, pokud na regionu neexistuje místo, na kterém by region pokračoval do „prázdna“. Pro účely bodování se za neuzavřené regiony považují i regiony velikosti 2. Při závěrečném zúčtování se za neuzavřené považují všechny regiony.

Za odebrání figurky z nížiny získává hráč bod za každou kartičku, která tvoří tuto nížinu. Pokud je nížina uzavřená, hráč získává dvojnásobný počet bodů. Za odebrání figurky z hor hráč získává bod za každé město, které se nachází na tomto regionu nebo na nížině sousedící s těmito horami. Pokud jsou hory uzavřené, hráč získává dvojnásobný počet bodů. Za odebrání figurky z moře získává hráč bod za každé město, které se na moři nachází. Pokud je moře uzavřené, hráč získává navíc bod za každou kartičku, které tvoří toto moře.

1.2 Srovnání pravidel s Carcassonne

Odlišnosti v pravidlech „klasického“ Carcassonne a Carcassonne – Objevitelé znázorňuje tabulka 1.1.

Pravidlo nebo herní prvek	Carcassonne	Carcassonne – Objevitelé
počet figurek hráče	8	4
počet kartiček	72	84
regiony na kartičkách	města (a štíty), cesty, louky, kláštery	hory, nížiny, moře
odebírání figurek	automaticky při uzavření regionu	hráčem
zisk bodů	pouze hráč s převahou na regionu	každá figurka je bodována zvlášť
uzavřenost regionů	každý typ regionu má vlastní pravidla pro uzavření	uzavřenost odpovídá uzavřenosti měst z Carcassonne

Tabulka 1.1: Srovnání pravidel Carcassonne a Carcassonne – Objevitelé.

Pro hráče klasického Carcassonne jistě obtížné přizpůsobit se pravidlům Carcassonne – Objevitelé hry. Rozdíl ve způsobu bodování a odebírání figurek je však natolik velký, že Carcassonne a Carcassonne – Objevitelé jsou strategicky odlišné hry.

1.3 Definice

Kartička obsahuje prvky krajiny. Kartičku označíme jako *umístěnou*, pokud byla v nějakém tahu vylosována a umístěna, v opačném případě nazveme kartičku jako *neumístěnou*. Množinu nevylosovaných a neumístěných kartiček označíme jako *balíček*. Množinu umístěných kartiček nazýváme jako *herní desku*. *Souřadnice* kartičky určujeme podle jejich umístění. *Počátek souřadnic* je dán startovní kartičkou.

Figurka je *umístěná*, pokud byla umístěna na nějaký region, v opačném případě je figurka *volná*.

Region je souvislá plocha krajiny vytvořená spojením jedné nebo více kartiček. *Velikost regionu* je počet kartiček, na kterých se region nachází. Region je jednoho ze tří *typů*: hory, nížiny, moře. Region nazveme *uzavřený*, pokud neexistuje místo, ve kterém by region pokračoval do volného prostoru. V opačném případě je region *otevřený*. Pro otevřený region označíme souřadnice, na které je třeba umístit kartičky tak, aby se z regionu stal uzavřený region, jako *otevřené souřadnice regionu*. Pokud se na regionu nachází libovolná figurka, nazveme region

jako *obsazený*, v opačném případě je region *neobsazený*. Region, který je otevřený a neobsazený nazveme jako *obsaditelný*. Mezi některými regiony se nachází *města*.

Skóre figurky je počet bodů, který získá hráč při odebrání figurky. *Skóre regionu* r v okamžik t je skóre figurky, která by byla umístěna na regionu r a v okamžik t by byla odebrána. Okamžik t explicitně neuvádíme, je známý z kontextu použití.

Tah je dvojice skládající se z *tahu s kartičkou* a *tahu s figurkou*. Tah s kartičkou je umístění kartičky na herní desku. Tah s figurkou označuje položení figurky na region, odebrání figurky z regionu nebo vynechání položení nebo odebrání figurky.

2. Návrh umělých inteligencí

V této kapitole formálně upřesníme cíl vytvoření umělé inteligence a popíšeme metody, které lze použít pro splnění tohoto cíle.

2.1 Formální zadání

Hlavním cílem této práce je vytvořit umělé inteligence pro hru Carcassonne – Objevitelé. Samotný pojem *umělá inteligence* je široký, proto je třeba ho zkonkretizovat. Pod pojmem „umělá inteligence“ budeme uvažovat (racionálního) agenta dle definice P. Norviga a S. Rusella (Russell a Norvig, 2009).

Agent je cokoliv, co vnímá okolí prostřednictvím senzorů a ovlivňuje ho *akcemi* konaných prostřednictvím akčních členů. Agenta lze formálně popsat *agentovou funkcí*: $V^\star \to A$, kde V je množina vjemů, $V^\star$ je posloupnost vjemů a A je množina akcí. Agentová funkce je abstraktní matematický popis, při konstrukci agenta tuto funkci implementujeme jako *agentový program*. *Racionální agent* je poté agent, který maximalizuje očekávanou hodnotu výkonu vzhledem k dané míře. *Míra výkonu* je funkce, která by měla reflektovat požadovaný cíl. Racionální agent není *vševědoucí*, proto nemůže maximalizovat skutečnou hodnotu výkonu.

Agenti se nachází v určitém *prostředí*. Prostředí se v každý okamžik nachází v určitém *stavu*. Prostředí, stejně jako agent, může mít různé podoby. Prostředí agenta klasifikujeme:

- jako *diskrétní* nebo *spojité* podle charakteristiky veličin, ze kterých se skládá stav,
- jako *částečně pozorovatelné*, pokud ve stavu prostředí existuje skrytá informace, kterou agenti nemohou svými senzory vnímat, v opačném případě jako *plně pozorovatelné*,
- jako *epizodické*, pokud jednotlivé akce agenta jsou na sobě nezávislé, v opačném případě jako *sekvenční*,
- jako *deterministické*, pokud přechody mezi stavy prostředí jsou jednoznačně určené provedenou akcí, v opačném případě jako *stochastické*,
- jako *statické*, pokud se v době přemýšlení agenta prostředí nemění, *semidynamické*, pokud se mění pouze hodnota míry výkonu, jinak prostředí označujeme jako *dynamické*,
- jako *jednoagentové*, pokud se v prostředí nachází pouze jeden agent, v opačném případě jako *multiagentní*,
- v případě multiagentního prostředí rozlišujeme prostředí *kooperativní*, kde agenti mají společnou míru výkonu, a *kompetitivní*, kde agenti mají protichůdné míry výkonu.

Naše agenty budeme sestrojovat jako počítačové programy chovající se jako hráči. Jejich vstupem bude aktuální stav hry, výstupem bude tah k provedení. Cílem hráčů je na konci hry získat vyšší skóre než ostatní hráči. Hráč by tedy měl ve výsledném pořadí figurovat co nejvýše, jako případný druhotný cíl by měl získat co nejvyšší počet bodů. Losování kartiček lze považovat jako součást provedení tahu.

Prostředí hry Carcassonne – Objevitelé je:

- diskrétní, protože pracujeme s konečnou množinou kartiček a konečnou množinou figurek, tedy i počet stavů je konečný,
- plně pozorovatelné, protože neexistuje skrytá informace – víme, které kartičky byly umístěny a které mohou být ještě vybrány,
- sekvenční, protože tahy v rámci jedné hry se navzájem ovlivňují,
- stochastické, protože dochází k losování kartiček,
- statické, protože se stav hry nemění v době přemýšlení hráče (losování kartičky je součástí změny prostředí po provedení tahu),
- multiagentní kompetitivní, protože hraje více hráčů proti sobě.

Stav prostředí se skládá z umístěných a neumístěných kartiček, umístěných figurek a dosaženého skóre hráčů. Prováděné akce odpovídají tahům hráčů.

2.2 Prohledávání stavového prostoru a heuristické funkce

Základní metodou při sestrojování agentů je prohledávání *stavového prostoru*. Tato metoda na základě stavu hry a proveditelných akcí vytváří *strom hry*, což je orientovaný graf, ve kterém vrcholy odpovídají stavům prostředí a hrany odpovídají akcím. Mezi dvěma vrcholy V_1 a V_2, které reprezentují stavy S_1 a S_2, vede orientovaná hrana, která reprezentuje akci A, právě když pomocí akce A lze přejít ze stavu S_1 do stavu S_2. Stavům reprezentovanými vrcholy nacházejícím se v listech tohoto stromu přiřadíme *užitek* – reálné číslo reprezentující odhad hodnoty míry výkonu. Funkci, která přiřazuje stavům užitek, označíme jako *heuristickou funkci*. Užitek budeme nazývat také jako *heuristickou hodnotu* stavu.

Nejjednodušším případem prohledávání stavového prostoru je prozkoumání všech bezprostředně následujících stavů. Vybíráme takovou akci, po které je heuristická hodnota stavu nejvyšší možná. Tuto metodu nazveme jako *heuristický přístup* nebo též *jednokrokový přístup*. Hráče využívající tento přístup nazveme jako *heuristický hráč*. Pseudokód algoritmu pro výběr tahu heuristického hráče je uveden v pseudokódu 1.

Heuristický hráč byl implementován ve třídě `HeuristicsAi<T>` s typovým parametrem `T` implementující rozhraní `IHeuristics`, které reprezentuje heuristickou funkci. Samotná implementace místo časově náročnějšího kopírování stavu hry využívá aplikování tahu na stav a následného vracení tohoto tahu. Třídy heuristických funkcí mohou využít toho, že k prohledávání tahů dochází

Pseudokód 1 Heuristický hráč

```
1: funkce NajdiNejlepšíTah(stavHry)
2:     hodnota_max := −∞
3:     tah_max := ∅
4:     pro všechny tah_K v TahSKartičkou(stavHry) proveď
5:         stavHry_K := Aplikuj(tah_K, stavHry)
6:         pro všechny tah_F v TahSFigurkou(stavHry_K) proveď
7:             stavHry_F := Aplikuj(tah_F, stavHry_K)
8:             hodnota := HeuristickáFunkce(stavHry_F)
9:             když hodnota > hodnota_max tak
10:                hodnota_max := hodnota
11:                tah_max := ⟨tah_K, tah_F⟩
12:            konec když
13:        konec pro všechny
14:    konec pro všechny
15:    vrať tah_max
16: konec funkce
```

v systematickém pořadí, a tak lze některé hodnoty relevantní pro heuristickou funkci předpočítat, a tím předejít opakovanému výpočtu totožných hodnot.

Nyní popíšeme několik způsobů, jak navrhnout heuristickou funkci.

2.2.1 Náhodná heuristická funkce

Náhodná heuristická funkce přiřadí stavu reálné číslo vygenerované generátorem pseudonáhodných čísel. Tato heuristika simuluje hráče, který pouze dodržuje pravidla hry a provádí tahy bez rozmýšlení. Pseudokód náhodné heuristické funkce je uveden jako pseudokód 2.

Pseudokód 2 Náhodná heuristická funkce

```
1: funkce HeuristickáFunkce(stavHry)
2:     vrať NáhodnéČíslo()
3: konec funkce
```

Tento přístup má zřejmě velkou nevýhodu v tom, že nijak nezohledňuje stav hry, a tedy nemaximalizuje očekávanou hodnotu míry výkonu. Tím není splněn požadavek na racionalitu agenta. Výhodou je rychlý výpočet. Hráče využívajícího náhodnou heuristickou funkci využijeme především jako tzv. základní čáru. Žádný jiný hráč by neměl být horší, než hráč využívající náhodnou heuristiku.

Náhodná heuristika byla implementována ve třídě `RandomHeuristics`.

2.2.2 Hladová heuristická funkce

Lepším přístupem, než je náhodný výběr tahu, je ze všech tahů vybrat ten, po kterém bude heuristická hodnota výsledného stavu co nejlepší pro hráče provádějícího tah. Tento přístup nazveme v souladu s konvencemi jako *hladový*. Hladová heuristická funkce tedy přímo odpovídá míře výkonu, ale místo očekávaného výkonu po konci hry maximalizujeme skutečný výkon po provedení tahu.

Jako druhotné kritérium lze použít rozdíl skóre od soupeře figurujícího v pořadí nejblíže danému hráči. Pseudokód výpočtu hodnoty heuristické funkce je uveden jako pseudokód 3.

Pseudokód 3 Hladová heuristická funkce

```
1: funkce HeuristickáFunkce(stavHry)
2:     skóre_H := Skóre(stavHry)
3:     pro všechny hráč v Hráči(stavHry) proveď
4:         pro všechny fig v UmístěnéFigurky(hráč, stavHry) proveď
5:             skóre_F := SkóreFigurky(fig, stavHry)
6:             skóre_H[hráč] += skóre_F
7:         konec pro všechny
8:     konec pro všechny
9:     vrať PočetHráčů(stavHry)                    ▷ maximalizujeme hodnotu
                - PořadíHráče(skóre_H, HráčNaTahu())
10: konec funkce
```

Výhodou hladového přístupu je přiblížení se cíli maximalizovat míru výkonu. Výpočet hodnoty je relativně rychlý. Nevýhodou je, že se ohodnocení stavu zaměřuje pouze na skóre figurek v daný okamžik. Hráč nepočítá se stavem herní desky a budoucími tahy. V tom případě hrozí, že hráč vytvoří ostatním soupeřům příležitost ke snadnému získání bodů v následujícím tahu.

Hladová heuristika byla implementována ve třídě `GreedyHeuristics`. Vlastní implementace využívá systematičnosti prohledávání tahů a pro stav předpočítává skóre figurek. Pokud nedojde ke změně skóre po umístění kartičky, není třeba skóre figurky znovu počítat. Jako míra výkonu bylo zvoleno pořadí a sekundárně nižší rozdíl skóre od o příčku lepšího soupeře nebo v případě vedoucího hráče vyšší rozdíl skóre od druhého hráče.

2.2.3 Potenciálová heuristická funkce

Pokusíme se odstranit nedostatek v neuvažování stavu herní desky u hladového přístupu. Každému regionu přiřadíme hodnotu *potenciálního skóre*, které bude odhadovat maximální skóre regionu dosažitelné do konce hry. Funkci přiřazující potenciální skóre nazveme *potenciálová funkce*.

Hráčům určíme hodnotu *potenciálního skóre* jako součet jejich aktuálního skóre, potenciálního skóre pro jejich umístěné figurky a potenciálu do budoucích kol. Idea určení potenciálu v budoucích kolech je taková, že hráči se snaží obsadit regiony s co nejvyšším potenciálem, dokud mají k dispozici figurky. Pokud figurku k dispozici nemají, odeberou ji z nějakého vhodného regionu, a mohou ji využít v budoucích kolech pro obsazení dalších regionů. Postupně simulujeme obsazování těchto regionů v pořadí od toho s nejvyšším potenciálem normalizovaným pravděpodobností zisku kartičky, pomocí které lze region obsadit, dokud jsou nějaké regiony k dispozici nebo dokud neprovedeme tolik kroků, kolik je zbývajících kol ve hře. Pseudokód potenciálové heuristické funkce je uveden jako pseudokód 4.

Potenciálová heuristika byla implementována ve třídě `PotentialHeuristics`. Vlastní implementace využívá předpočítání potenciálního skóre regionů a figurek před umístěním kartičky. Pokud nedojde k přiložení kartičky do okolí regionu, není potřeba přepočítávat potenciální skóre těchto regionů.

Pseudokód 4 Potenciálová heuristická funkce

```
1: funkce HeuristickáFunkce(stavHry)
2:     potenciál_H := Skóre(stavHry)
3:     pro všechny hráč v Hráči(stavHry) proveď
4:         pro všechny fig v UmístěnéFigurky(hráč, stavHry) proveď
5:             potenciál_F := PotenciálováFunkce(Region(fig), stavHry)
6:             potenciál_H += potenciál_F
7:         konec pro všechny
8:     konec pro všechny
9:     potenciál_R := VytvořMaximovouHaldu()
10:    pro všechny region v Regiony(stavHry) proveď
11:        když Neobsazený?(region) ∧ Neuzavřený?(region) tak
12:            Přidej(potenciál_R, PotenciálováFunkce(region, stavHry))
13:        konec když
14:    konec pro všechny
15:    figurky := VolnéFigurkyHráčů(stavHry)
16:    tahy := PočetZbývajícíchFigurek(stavHry)
17:    hráčNaTahu := NásledujícíHráč(HráčNaTahu(), stavHry)
18:    dokud tahy > 0 ∧ ¬ Prázdný?(potenciál_R) proveď
19:        když figurky[hráčNaTahu] > 0 tak
20:            figurky[hráčNaTahu] -= 1
21:            potenciál_H += OdeberMaximum(potenciál_R)
22:        jinak
23:            figurky[hráčNaTahu] += 1
24:        konec když
25:        tahy -= 1
26:    konec dokud
27:    vrať PočetHráčů(stavHry)                    ▷ maximalizujeme hodnotu
                - PořadíHráče(potenciál_H, HráčNaTahu())
28: konec funkce
```

Zbývá sestrojit vzorec pro výpočet potenciálního skóre regionu. Z pravidel vyplývá, že skóre figurky má tři složky. První složka souvisí s velikosti regionu, druhá složka se týká počtu měst na regionu a třetí složka se týká počtu měst na přilehlých nížinách. Jednotlivé složky mají váhu dle typu regionu, uzavřenosti regionu a velikosti regionu. Na horách je bodováno pouze počet měst na regionu a přilehlých nížinách, v případě uzavření je počet bodů dvojnásobný. Na nížinách je hodnocena pouze velikost regionu, v případě uzavření je počet bodů dvojnásobný. Na mořích je bodován počet měst na regionu, v případě uzavření se připočítává skóre za velikost regionu. Regiony o velikosti 1 nebo 2 se pro účely bodování považují za neuzavřené.

Vzorec se bude skládat z výše uvedených složek. Jednotlivé složky budou mít parametry zohledňující typ regionu, uzavřenost regionu a velikost regionu. Typ regionu zohledníme tak, že všechny koeficienty ve vzorci budou mít vlastní hodnoty pro různé typy regionů, při výpočtu se použije hodnota koeficientu odpovídajícího typu regionu. Uzavřenost regionu zohledníme tak, že v každé složce přidáme člen odhadující pravděpodobnost uzavření regionu. Velikost regionu zohledníme indikátorem velikosti regionu.

Potenciální skóre regionu r určíme podle vzorce

$$\begin{aligned}\Phi(r) = \quad & \Big(a_t^V + b_t^V \cdot (i + p^U(R)\Big) \cdot \Big(v + c_t^V \cdot p^R(R)\Big) + \\ & + \Big(a_t^M + b_t^M \cdot (i + p^U(R)\Big) \cdot \Big(m + c_t^M \cdot p^R(R) \cdot h_t\Big) + \\ & + \Big(a_t^N + b_t^N \cdot (i + p^U(R)\Big) \cdot \Big(n + c_t^N \cdot p^N(R) \cdot h_{nížina}\Big),\end{aligned}$$

kde

- t označuje typ regionu,
- v velikost regionu,
- m počet měst na regionu,
- n počet měst na přilehlých nížinách regionu,
- i je indikátor vyjadřující velikost a uzavření regionu,
- p^U je odhad pravděpodobnosti uzavření regionu,
- p^R je střední hodnota přidaných kartiček k regionu,
- p^N je střední hodnota přidaných kartiček k přilehlým nížinám regionu,
- h_t je odhad pravděpodobnosti výskytu města u regionu typu t na kartičce,
- a_t^V, b_t^V, c_t^V, a_t^M, b_t^M, c_t^M, a_t^N, b_t^N, c_t^N jsou konstanty závislé na typu regionu.

Parametry t, v, m, n jsou vstupní. Konstanty a_t^V, b_t^V, c_t^V, a_t^M, b_t^M, c_t^M, a_t^N, b_t^N, c_t^N a h_t určíme v souladu s pravidly bodování. Hodnotu i určíme podle podmínek:

$$i = \begin{cases} 1 & \text{pokud je region uzavřený a o velikosti alespoň 3,} \\ 0 & \text{jinak.} \end{cases}$$

Ve vzorcích pro p^U, p^R a p^N budeme používat následující značení: I vyjadřuje množinu otevřených souřadnic regionu, K_i počet zbývajících kartiček umístitelných na danou otevřenou souřadnici $i \in I$, K počet zbývajících kartiček, H počet hráčů ve hře. Dále označíme $k = \lfloor \frac{K}{H} \rfloor$ zbývající počet kartiček pro každého hráče a p pravděpodobnost zisku kartičky na otevřenou souřadnici, kterou určíme jako geometrický průměr pravděpodobností. Geometrický průměr je motivován použitím p v binomickém rozdělení. Vzorec pro výpočet p je

$$p = \sqrt[|I|]{\prod_{i \in I} \frac{K_i}{K}}.$$

Hodnotu p^U vypočítáme pomocí binomického rozdělení

$$p^U(r) = \sum_{j=|I|}^{k} \binom{k}{j} (1-p)^{k-j} \cdot p^j,$$

kdy odhadujeme pravděpodobnost, že hráč ve svých zbývajících tazích si vylosuje kartičku umístitelnou na otevřené souřadnice regionu alespoň tolikrát, kolik volných míst regionu existuje. Pokud je region uzavřený, definujeme $p^U(r) = 0$.

Zbývá definovat funkce p^R resp. p^N, které budou vyjadřovat očekávaný počet kartiček, kterými rozšíříme region resp. přilehlé nížiny. Budeme odhadovat rozšíření regionu na každé otevřené souřadnici nezávisle na sobě, zanedbáme tedy skutečnost, že některé kartičky z balíčku lze umístit na více otevřených souřadnic. Přesnou hodnotu by bylo možné vypočítat pomocí principu inkluze a exkluze, který je však výpočetně náročnější. Při velkém počtu kartiček bude však odchylka od správné hodnoty nízká a při malém počtu kartiček bude hodnota mít nízkou váhu na celkovou hodnotu $\Phi(r)$. Výsledný vzorec pro p^R je ve tvaru

$$p^R(r) = \sum_{i \in I} \left(1 - (1 - \tfrac{K_i}{K})^k\right).$$

Pokud region r je uzavřený, nelze rozšířit a platí $p^R(r) = 0$. Vzorec pro p^N je totožný, ale místo otevřených souřadnic regionu I uvažujeme všechny otevřené souřadnice přilehlých nížin.

Výhodou použití potenciálové heuristické funkce je uvažování do budoucnosti. Nevýhodou je, že všechny použité vzorce jako vstup používají jen určité parametry stavu hry a nemohou uvažovat všechny okolnosti. Výkon hráče používajícího potenciálovou funkci závisí na počátečním určení koeficientů. Tento nedostatek zkusíme vylepšit v metodě učení koeficientů v části 2.5.

Potenciálová funkce byla implementována třídou `PotentialFormula`. Vlastní implementace je rozšířena o další prvky, které využívá metoda učení koeficientů, ale v rámci této potenciálové heuristické funkce nejsou tyto prvky použity.

2.3 Expectiminimax

Navážeme na předchozí část, ve které jsme navrhli heuristiky pro použití v metodách prohledávání stavového prostoru. Představíme si *Expectiminimax*, což je algoritmus prohledávání stavového prostoru používaný v multiagentních, diskrétních, stochastických prostředích. Postupně tento algoritmus odvodíme

ze základního algoritmu *Minimax* a přizpůsobíme ho pro použití pro Carcassonne – Objevitelé. Postupovat budeme opět podle P. Norviga a S. Russela (Russell a Norvig, 2009).

2.3.1 Minimax a alfa-beta ořezávání

Algoritmus Minimax ilustrujeme na nejjednodušším příkladu. Máme deterministickou, plně pozorovatelnou hru dvou hráčů, kteří se střídají v tazích. Hráče nazveme Max a Min. První tah provádí Max. Hra končí výhrou jednoho z hráčů nebo remízou.

Vybudujeme strom hry z počátečního stavu. Koncovým stavům přiřadíme hodnotu *užitku* podle výsledku hry. Stavy, ve kterých vyhrál Max, ohodnotíme +1. Stavy, ve kterých vyhrál Min, ohodnotíme -1. Remízové stavy ohodnotíme 0. Max se tedy snaží maximalizovat hodnotu užitku a Min se snaží minimalizovat hodnotu užitku. Pro nekoncové stavy hry určíme *minimaxovou hodnotu*, která označuje nejvyšší dosažitelný užitek pro Maxe, pokud hra dosáhne daného stavu. Minimaxovou hodnotu stavu s určíme pomocí rekurzivního vzorečku:

$$Minimax(s) = \begin{cases} Užitek(s) & \text{pokud je } s \text{ koncový,} \\ \max_{a \in Akce(s)} Minimax(Aplikuj(s,a)) & \text{pokud je na tahu Max,} \\ \min_{a \in Akce(s)} Minimax(Aplikuj(s,a)) & \text{pokud je na tahu Min.} \end{cases}$$

Indukcí lze dokázat, že minimaxová hodnota je optimální. Pokud hráči budou vybírat optimální akce, tak je zaručeno, že dosáhnou nejlépe užitku rovnému minimaxové hodnotě stavu. V případě, že soupeř provede neoptimální tah, může hráč dosáhnout pouze lepších výsledků.

Pseudokód algoritmu Minimax je uveden jako pseudokód 5.

Tento algoritmus poskytuje optimální řešení, ale za velkou cenu. Výpočet algoritmu musí projít celý strom hry, aby zjistil přesnou minimaxovou hodnotu. Pokud označíme b *větvící faktor*, tedy průměrný počet proveditelných tahů v každém stavu, a m nejvyšší hloubku ve stromu, časová složitost algoritmu je $\mathcal{O}(b^m)$, tedy exponenciální. Tento nedostatek lze vylepšit pomocí dvou metod.

První metodou je *alfa-beta* ořezávání. Při procházení stromem hry si v paměti uchováváme, jaké nejlepší hodnoty užitku může dosáhnout každý z hráčů v některém koncovém stavu, který je součástí podstromu kořenícím v aktuálním stavu. Označíme α nejlepší hodnotu, které lze dosáhnout po tahu Maxe, tedy se jedná o nejvyšší hodnotu užitku. Dále označíme β nejlepší hodnotu, která lze dosáhnout po tahu Mina, tedy se jedná o nejnižší hodnotu užitku. Na počátku jsou hodnoty α resp. β rovny $-\infty$ resp. $+\infty$. V každém stavu tedy máme interval $[\alpha,\beta]$, který postupně aktualizujeme podle získaných hodnot. Pokud se v některý okamžik stane, že $\alpha > \beta$, tak víme, že jeden z hráčů nebude provádět tento tah, protože existuje tah, který mu zaručí lepší výsledek. V tom případě není třeba procházet zbývající následníky stavu a provedeme *řez*, tedy se vrátíme k výpočtu minimaxové hodnoty v předchozím stavu. Budeme mít zaručeno, že vrácená hodnota nemůže být optimální pro daný stav. Pseudokód alfa-beta ořezávání v Minimaxu je uveden jako pseudokód 6.

Množství ořezaných větví závisí na pořadí prohledávání. V nejlepším případě lze dosáhnout časové složitosti $\mathcal{O}(b^{m/2})$, v nejhorším případě nedochází k ořezání žádné větve a časová složitost zůstává $\mathcal{O}(b^m)$.

Pseudokód 5 Minimax

```
1: funkce MINIMAX(stav)
2:     ⟨hodnota_max, akce_max⟩ := MAXHODNOTA(stav)
3:     vrať akce_max
4: konec funkce
5:
6: funkce MAXHODNOTA(stav)
7:     když KONEČNÝ?(stav) tak
8:         vrať ⟨UŽITEK(stav), ∅⟩
9:     konec když
10:    ⟨hodnota_max, akce_max⟩ := ⟨−∞, ∅⟩
11:    pro všechny akce v PROVEDITELNÉAKCE(stav) proveď
12:        stav_N := APLIKUJ(akce, stav)
13:        ⟨hodnota_N, akce_N⟩ := MINHODNOTA(stav_N)
14:        když hodnota_N > hodnota_max tak
15:            ⟨hodnota_max, akce_max⟩ := ⟨hodnota_N, akce⟩
16:        konec když
17:    konec pro všechny
18:    vrať ⟨hodnota_max, akce_max⟩
19: konec funkce
20:
21: funkce MINHODNOTA(stav)
22:    když KONEČNÝ?(stav) tak
23:        vrať ⟨UŽITEK(stav), ∅⟩
24:    konec když
25:    ⟨hodnota_min, akce_min⟩ := ⟨+∞, ∅⟩
26:    pro všechny akce v PROVEDITELNÉAKCE(stav) proveď
27:        stav_N := APLIKUJ(akce, stav)
28:        ⟨hodnota_N, akce_N⟩ := MAXHODNOTA(stav_N)
29:        když hodnota_N < hodnota_min tak
30:            ⟨hodnota_min, akce_min⟩ := ⟨hodnota_N, akce⟩
31:        konec když
32:    konec pro všechny
33:    vrať ⟨hodnota_min, akce_min⟩
34: konec funkce
```

Pseudokód 6 Alfa-beta ořezávání v Minimaxu

```
1: funkce AlfaBeta(stav)
2:     ⟨hodnota_max, akce_max⟩ := MaxHodnota(stav, −∞, +∞)
3:     vrať akce_max
4: konec funkce
5:
6: funkce MaxHodnota(stav, α, β)
7:     když Konečný?(stav) tak
8:         vrať ⟨Užitek(stav), ∅⟩
9:     konec když
10:    ⟨hodnota_max, akce_max⟩ := ⟨−∞, ∅⟩
11:    pro všechny akce v ProveditelnéAkce(stav) proveď
12:        stav_N := Aplikuj(akce, stav)
13:        ⟨hodnota_N, akce_N⟩ := MinHodnota(stav_N, α, β)
14:        když hodnota_N > hodnota_max tak
15:            ⟨hodnota_max, akce_max⟩ := ⟨hodnota_N, akce⟩
16:            α := Max(α, hodnota_max)
17:            když α ≥ β tak                                  ▷ řez
18:                vrať ⟨hodnota_max, akce_max⟩
19:            konec když
20:        konec když
21:    konec pro všechny
22:    vrať ⟨hodnota_max, akce_max⟩
23: konec funkce
24:
25: funkce MinHodnota(stav, α, β)
26:    když Konečný?(stav) tak
27:        vrať ⟨Užitek(stav), ∅⟩
28:    konec když
29:    ⟨hodnota_min, akce_min⟩ := ⟨+∞, ∅⟩
30:    pro všechny akce v ProveditelnéAkce(stav) proveď
31:        stav_N := Aplikuj(akce, stav)
32:        ⟨hodnota_N, akce_N⟩ := MaxHodnota(stav_N, α, β)
33:        když hodnota_N < hodnota_min tak
34:            ⟨hodnota_min, akce_min⟩ := ⟨hodnota_N, akce⟩
35:            α := Min(β, hodnota_min)
36:            když α ≥ β tak                                  ▷ řez
37:                vrať ⟨hodnota_min, akce_min⟩
38:            konec když
39:        konec když
40:    konec pro všechny
41:    vrať ⟨hodnota_min, akce_min⟩
42: konec funkce
```

Druhým způsobem urychlení je provedení *přerušení* prohledávání místo prohledávání stromu až do koncových stavů. Dosažené nekoncové stavy ohodnotíme *heuristickou funkcí*, která určuje *heuristickou hodnotu* odhadující užitek stavu. Přerušení můžeme provést podle různých kritérií, nejčastěji se používá daná hloubka.

Výhodou je snížení časové náročnosti. Místo nejvyšší hloubky m prohledáváme do nižší hloubky. Nevýhodou je, že výsledná minimaxová hodnota přímo závisí na kvalitě heuristické funkce. Pokud heuristická funkce neodpovídá očekávanému užitku, získáváme zkreslené hodnoty.

Při použití testu přerušení se také může projevit *efekt horizontu*. V průběhu výpočtu se může stát, že heuristická funkce ohodnotí stav jako lepší pro jednoho z hráčů, ale ve skutečnosti se bez ohledu na výběr tahů nelze vyhnout určité události, která způsobí výhodu pro druhého hráče. Původní heuristická hodnota je tedy v přímém rozporu se skutečnou dosažitelnou minimaxovou hodnotou.

2.3.2 Rozšíření minimaxu

Minimax používáme pro hry dvou hráčů v deterministickém prostředí. Potřebujeme jej však modifikovat i pro hry více hráčů ve stochastickém prostředí.

Vyšší počet hráčů zohledníme tak, že místo jedné minimaxové hodnoty budeme uvažovat vektor minimaxových hodnot pro každého hráče. Cílem hráče na tahu je maximalizovat vlastní složku vektoru. Dle R. Korfa (Korf, 1991) lze použít alfa-beta ořezávání, pokud existuje spodní odhad pro každou složku vektoru minimaxových hodnot a horní odhad pro součet všech složek vektoru minimaxových hodnot.

Náhodné události ve stochastickém prostředí zohledníme tak, že ve stromu hry zavedeme *vrcholy náhody*. Z nich povedou hrany do dalších stavů podle toho, do jakého stavu se lze dostat při výskytu náhodné události, která má známou pravděpodobnost. Výsledné vrcholy následující po vrcholu náhody musí být *disjunktním rozkladem* – součet pravděpodobností jednotlivých přechodů musí být roven 1 a nesmí existovat žádný jiný přechod, který by nebyl zohledněn. Při výpočtu minimaxové nebo heuristické hodnoty pro vrchol náhody použijeme střední hodnotu minimaxové nebo heuristické hodnoty následujících stavů. Tento algoritmus se nazývá Expectiminimax (nebo též *Expectimax*), podle anglického překladu pojmu „střední hodnota“ – *expected value*.

Kromě uvedených rozšíření algoritmu Minimax se používají různé *urychlovací techniky*. *Knihovny zahájení* a *knihovny koncovek* se používají pro hry, ve kterých často dochází k opakování podobných počátečních nebo koncových akcí. Jedná se o slovník, kde vybraným stavům – podle vhodného klíče – je přiřazena předpočítaná optimální akce. U her, kde lze pomocí přehození pořadí akcí dojít do stejného stavu, se používají *transpoziční tabulky*. Při použití alfa-beta ořezávání lze použít *heuristiky* pro brzké nalezení větve s vysokou heuristickou hodnotou, které způsobí ořezání více větví. Pro Carcassonne – Objevitelé by bylo možné použít pouze heuristiky pro alfa-beta ořezávání, ostatní techniky nelze použít kvůli charakteru hry.

2.3.3 Použití Expectiminimaxu

Použijeme algoritmus Expectiminimax s určitou heuristickou funkcí a jeho rozšíření pro více hráčů. Jako test přerušení použijeme pevně danou hloubku, která je parametrem daného hráče. Alfa-beta prořezávání nebudeme používat, protože v obecném případě nemusí být splněny předpoklady pro jeho použití. Pseudokód použitého algoritmu Expeciminimax je uveden jako pseudokód 7.

Výhodou použití Expectiminimaxu na rozdíl od jednokrokového přístupu je uvažování budoucích možností vylosovaných kartiček. Pokud by prohledávání bylo prováděno do hloubky koncových stavů, získali bychom tah s nejlepší možnou očekávanou hodnotou užitku. Nevýhodou je velká časová náročnost daná velkým větvícím faktorem a exponenciální složitostí vzhledem k hloubce prohledávání.

Expectiminimaxový hráč byl implementován ve třídě `ExpectiminimaxAi<T>` s typovým parametrem `T` třídy, která implementuje heuristickou funkci. Implementace je založena na původní implementaci heuristického hráče `HeuristicsAi<T>` s rozšířením o prohledávání do hloubky.

2.4 Monte Carlo metody

Monte Carlo metody (MC) se řadí mezi metody zpětnovazebního učení. Představíme si obecný model zpětnovazebního učení, jeho modifikaci pro MC a algoritmus implementovaný pro agenta.

2.4.1 Zpětnovazební učení

Představíme si model zpětnovazebního učení *Markovův rozhodovací proces* dle definice od R. Suttona a A. Barta (Sutton a Barto, 2018). Budeme uvažovat pouze diskrétní, statické prostředí, které máme i v případě Carcassonne – Objevitelé.

Již známé pojmy agent, prostředí a akce definované v části 2.1 doplníme o pojem *odměna* (anglicky *reward*). Označíme t libovolný časový okamžik a T časový okamžik ukončení hry. Po každé své akci A_t získá agent odměnu R_t, což je číselná hodnota vyjadřující, jak „dobrá“ byla akce A_t v daném stavu hry S_t, a stav prostředí se změní na S_{t+1}. Agent má za cíl maximalizovat celkový *užitek* G_0 v průběhu času, kde

$$G_i = \sum_{i}^{T} R_i.$$

V případě neepizodického prostředí chceme navíc zohlednit to, aby nedocházelo k plýtvání času při učení. Pokud by existovalo více sekvencí tahů vedoucí ke stejnému cíli se stejnou odměnou, vybrali bychom tu, kterou dosáhneme co nejdříve. Zavedeme *koeficient znevýhodnění* $\gamma \in (0; 1]$ a upravíme vzorec pro užitek:

$$G_i = \sum_{j=i}^{T} \gamma^{j-i} \cdot R_i.$$

Dále předpokládejme, že odměna R_t i výsledný stav S_{t+1} po provedení akce se chovají jako diskrétní náhodné veličiny a existuje tedy jejich rozdělení pravděpodobnosti. To znamená, že pro každý stav s a akci a existuje rozdělení náhodných veličin odměny r a následujícího stavu s', které nazveme *dynamikou*

Pseudokód 7 Expectiminimaxový hráč

```
1: funkce NajdiNejlepšíTah(stavHry)
2:     ⟨hodnota, tah⟩ := MaxHodnotaProHráče(stavHry, limitHloubky)
3:     vrať tah
4: konec funkce
5:
6: funkce MaxHodnotaProHráče(stavHry, hloubka)
7:     když Konečný?(stav) tak
8:         vrať ⟨Užitek(stav), ∅⟩
9:     jinak když hloubka == 0 tak
10:        vrať ⟨HeuristickáFunkce(stav), ∅⟩
11:    konec když
12:    hráč := HráčNaTahu(stav)
13:    ⟨hodnota_max, tah_max⟩ := ⟨(..., −∞, ...), ∅⟩
14:    pro všechny tah_K v TahSKartičkou(stavHry) proveď
15:        stavHry_K := Aplikuj(tah_K, stavHry)
16:        pro všechny tah_F v TahSFigurkou(stavHry_K) proveď
17:            stavHry_F := Aplikuj(tah_F, stavHry_K)
18:            hodnota := ProhledejNásledující(stavHry_F, hloubka)
19:            když hodnota[hráč] > hodnota_max[hráč] tak
20:                ⟨hodnota_max, tah_max⟩ := ⟨hodnota, ⟨tah_K, tah_F⟩⟩
21:            konec když
22:        konec pro všechny
23:    konec pro všechny
24:    vrať ⟨hodnota_max, tah_max⟩
25: konec funkce
26:
27: funkce ProhledejNásledující(stavHry, hloubka)
28:    suma := (..., 0, ...)
29:    možnosti := 0
30:    hloubka_N := hloubka − 1
31:    pro všechny kartička v ZbývajícíKartičky(stavHry) proveď
32:        když LzeUmístit?(kartička, stavHry) tak
33:            stav_N := ZahájeníTahu(stavHry, kartička)
34:            ⟨hodnota, tah⟩ := MaxHodnotaProHráče(stav_N, hloubka_N)
35:            suma += hodnota                          ▷ sčítání po složkách
36:            možnosti += 1
37:        konec když
38:    konec pro všechny
39:    vrať suma / možnosti                             ▷ dělení po složkách
40: konec funkce
```

prostředí a označíme ho jako $p(s',r|s,a)$. Tento model označujeme jako Markovův rozhodovací proces (*MDP*, z anglického *Markov Decision Process*). Definice Markovova rozhodovacího procesu je velmi abstraktní a lze jej snadno adaptovat pro konkrétní účely.

Označíme $\pi(a|s)$ *strategii*, funkci vyjadřující pravděpodobnost výběru akce a ve stavu s. Dále stav hry s ohodnotíme funkcí *očekávaného užitku ve stavu*

$$v_\pi(s) = \mathsf{E}_\pi[G_t|S_t = s]$$

a akci a proveditelnou ve stavu s ohodnotíme funkcí *očekávaného užitku při výběru akce*

$$q_\pi(s,a) = \mathsf{E}_\pi[G_t|S_t = s,A_t = A],$$

v obou případech rozhodování probíhá za pomocí strategie π. Cílem zpětnovazebního učení je získat *optimální strategii* $\pi_\star$.

Odpovídající optimální hodnotu pro funkci očekávaného užitku ve stavu určíme jako $v_\star(s) = \max_\pi v_\pi(s)$ a optimální hodnotu pro funkci očekávaného užitku při výběru akce určíme jako $q_\star(s,a) = \max_\pi q_\pi(s,a)$. Díky rovnosti

$$G_t = R_{t+1} + \gamma G_{t+1}$$

a pozorování

$$v_\star(s) = \max_a q_{\pi_\star}(s,a)$$

platí pro každý stav s rovnost

$$\begin{aligned}
v_\star(s) &= \max_a q_{\pi_\star}(s,a) = \\
&= \max_a \mathsf{E}_{\pi_\star}[G_t|S_t = s,A_t = a] = \\
&= \max_a \mathsf{E}_{\pi_\star}[R_{t+1} + \gamma G_{t+1}|S_t = s,A_t = a] = \\
&= \max_a \mathsf{E}\,[R_{t+1} + \gamma v_\star(S_{t+1})|S_t = s,A_t = a] = \\
&= \max_a \sum_{s',r} p(s',r|s,a)(r + \gamma v_\star(s')),
\end{aligned}$$

kterou nazveme *Bellmanovou rovnicí optimality*.

Pokud máme prostředí s relativně malým počtem stavů, můžeme použít techniku *dynamického programování* pro iterativní výpočet hodnot v_i a π_i. Lze dokázat, že posloupnosti v_i resp. π_i konvergují k optimálním hodnotám $v_\star$ resp. $\pi_\star$. Bohužel, stavový prostor pro Carcassonne – Objevitelé je exponenciálně velký vzhledem k počtu kartiček, a tak nelze opakovaně určit hodnoty v a π pro všechny stavy.

2.4.2 MC predikce

Jak uvádí Sutton a Barto (Sutton a Barto, 2018), MC na rozdíl od zmíněného dynamického programování ke svému učení nepotřebuje kompletní znalost dynamiky prostředí p, potřebují pouze určité *zkušenosti*. Zkušenosti jsou vzorek stavů, akcí a odměn získaných ze simulovaných interakcí s prostředím.

Nechť máme danou strategii π a počáteční stav S_0. Provedeme velký počet simulací s počátečním stavem S_0 za použití strategie π. V každé simulaci získáme

Pseudokód 8 Monte Carlo predikce

```
1: funkce MCPredikce(π, kroky)
2:     pro všechny stav v StavovýProstor() proveď ▷ inicializace hodnot
3:         v_stav := 0
4:         g_stav := PrázdnýSeznam()
5:     konec pro všechny
6:     pro všechny i od 1 do kroky proveď
7:         odměny := PrázdnýSeznam()
8:         stavy := PrázdnýSeznam()
9:         Přidej(stavy, PočátečníStav())
10:        dokud ¬ UkončitSimulaci?(stavy, odměny proveď
11:            s, r := AplikujSOdměnou(Poslední(stav), π)
12:            Přidej(s, stavy)
13:            Přidej(r, odměny)
14:        konec dokud
15:        G := 0
16:        pro všechny t od Délka(stavy) - 1 do 0 proveď
17:            s := stavy[t]
18:            G := G · γ + odměny[t]
19:            Přidej(G, g_s)
20:            v_s := Průměr(g_s)
21:        konec pro všechny
22:    konec pro všechny
23:    vrať v
24: konec funkce
```

odhad očekávaného užitku všech stavů navštívených v simulaci. Po provedení simulací získáváme odhad užitku všech navštívených stavů. Pseudokód je uveden jako pseudokód 8.

Tento algoritmus použijeme pro odhad užitků stavů, které jsou přímými následníky aktuálního stavu. Poté vybereme ten tah, pomocí kterého se dostaneme do stavu s nejvyšší hodnotou užitku. Simulace budou využívat strategii π, která se řídí heuristickou funkcí. Strategie π tedy vybírá akci, pomocí které dosáhneme stavu s nejvyšší heuristickou hodnotou. Odměnou R bude právě heuristická hodnota stavu, protože splňuje vlastnosti odměny.

Jako test ukončení simulace použijeme limit pro váhu nově přidávané odměny. Pokud koeficient znevýhodnění γ umocněný na pořadí akce v simulaci i klesne pod určitou hranici γ_T, ukončíme simulaci. K ukončení simulace dojde také, pokud se dostaneme do cílového stavu. Hodnotu γ_T nazveme *hranicí významnosti*. Celkový počet simulací, koeficient znevýhodnění γ a hranice významnosti γ_T jsou parametry hráče.

Hráč využívající MC predikci byl implementován třídou `MCPredictionAi<T>` s typovým parametrem `T` třídy, která implementuje heuristickou funkci. Implementace je založena na původní implementaci heuristického hráče `HeuristicsAi<T>` s rozšířením o generování a vyhodnocování simulací.

2.5 Metody strojového učení

Zatím jsme navrhli agenty, kteří se v průběhu času chovají totožným způsobem. I když se Monte Carlo metody řadí mezi metody zpětnovazebního učení, k *učení* dochází pouze v tom smyslu, že se naučíme vybrat nejlepší tah pro jeden konkrétní stav.

V části 2.2.3 jsme navrhli heuristiku, která pro svůj výpočet používala funkci, která v průběhu hry určovala potenciální skóre, tedy predikci skóre regionu na konci hry. Koeficienty byly vybrány podle pravidel bodování a předpokladů chování hráčů. Pokud byl výběr koeficientů na počátku „špatný“, hráč používající tuto funkci nebude dosahovat „dobrých“ výsledků a nebude v jeho možnostech tento nedostatek odstranit.

Pro takový případ se jeví vhodné použít metody *klasického strojového učení*. Představíme si základní myšlenku strojového učení, algoritmus *poklesu dle gradientu* (*GDA* z anglického *Gradient Descent Algorithm*) používaný pro učení a poté sestrojíme *učícího se agenta*.

2.5.1 Klasické strojového učení

Definicí strojového učení od T. Mitchella (Mitchell, 1997) zní „Počítačový program se *učí* ze zkušeností E, vzhledem k zadané úloze T a míře výkonu P, pokud se jeho výkon v řešení úlohy T dle míry P zlepšuje s přibývající zkušeností E“. Pojem *klasické* slouží především pro samostatné vyčlenění metod *hlubokého učení*, které také řadíme do metod strojového učení.

Úlohu T máme obvykle danou. Vstupem pro úlohu je vektor čísel (z dané množiny), výstupem je obvykle číslo (z dané množiny). Je-li výstupem číslo ze spojité množiny, hovoříme o *regresní úloze*. Pokud je výstupem číslo z diskrétní množiny, hovoříme o *klasifikační úloze*. Míru výkonu P vhodně volíme podle druhu úlohy

T. Zkušenosti E jsou představovány množinou *příkladů*. Příklad je tvořen *vektorem příznaků*, který je vstupem úlohy T, a skutečnou *výstupní hodnotou*, pokud je známa.

Samotný proces učení probíhá pomocí systematického použití vybrané *metody* strojového učení s vhodnými parametry na příklady. Výsledkem učení je *model* zahrnující metodu a množinu získaných parametrů, které používáme k predikci nebo regresi na nových datech.

2.5.2 Gradient Descent Algorithm

Zaměříme se na regresní úlohy, pro metody používající funkce více proměnných f_Θ nad reálnými čísly. Cílem učení je určit vhodné parametry Θ tak, aby byla minimalizována hodnota *ztrátové funkce*. Ztrátová funkce určuje, k jak velké chybě dochází mezi predikovanou a skutečnou hodnotou. U regresních úloh jako ztrátovou funkci obvykle používáme *čtvercovou chybu* (RSS). Označíme-li $\{y_i\}_{i=1}^n$ posloupnost skutečných hodnot příkladů a $\{\hat{y}_i\}_{i=1}^n$ posloupnost predikovaných hodnot pomocí funkce $f_{\hat{\Theta}}$, určíme hodnotu ztrátové funkce jako

$$RSS(\hat{\Theta}) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

Optimálními parametry $\Theta^\star$ jsou takové, pro které je dosažena nejnižší hodnota ztrátové funkce, tedy

$$\Theta^\star = \text{argmin}_\Theta RSS(\Theta).$$

U obecné funkce f_Θ je obtížné určit $\Theta^\star$ exaktně, používáme numerické metody.

Gradient Descent Algorithm (*GDA*) je optimalizační algoritmus, který nalezne lokální minimum funkce f. Začneme s náhodným vektorem vstupních hodnot a snažíme se určit následující bod, který je blíže k nějakému lokálnímu minimu, než původní hodnota. Následující bod určíme jako součet původního bodu a gradientu funkce (vektoru parciálních derivací vzhledem k jednotlivým proměnným) vynásobeným konstantou $\alpha \in (0; 1)$. Tento postup opakujeme, dokud vzdálenost gradientu od nulového vektoru není menší než dostatečně nízké číslo $\varepsilon > 0$. Koeficient α nazýváme *koeficientem učení* (anglicky *learning rate*).

Pseudokód GDA je uveden jako pseudokód 9.

Pseudokód 9 Gradient Descent Algorithm

```
1: funkce GradientDescentAlgorithm(f, α, ε, Θ)
                    ▷ Θ je počáteční bod, v jehož okolí hledáme lokální minimum
2:     Δ_f := Gradient(f, Θ)
3:     dokud Velikost(Δ_f) > ε proveď
4:         Θ := Θ − α · Δ_f
5:         Δ_f = Gradient(f, Θ)
6:     konec dokud
7:     vrať Θ
8: konec funkce
```

2.5.3 Agent s učící se potenciálovou heuristikou

Potenciálovou heuristickou funkci z části 2.2.3 vylepšíme o prvky učení. Jako základ použijeme totožnou potenciálovou funkci, ale provedeme dvě změny. Zásadní změnou je, že místo pevně určených koeficientů budeme koeficienty průběžně učit. Abychom získali příklady pro učení, necháme agenta odehrát celou hru. Při každém použití potenciálové funkce si agent do paměti uloží všechny vstupní argumenty funkce. Na konci hry je známa skutečná bodová hodnota regionů, a tak uloženým vstupním argumentům lze přiřadit skutečnou výstupní hodnotu. Poté aplikujeme Gradient Descent Algorithm a tím získáme nové koeficienty, které použijeme jako výchozí pro další hry.

Druhou změnou je nahrazení předpočítaných hodnot indikátorů, pravděpodobností a středních hodnot i, p^R, p^M, p^N. Tyto hodnoty jsou určeny za předpokladu, že hráč je soustředěn na rozšiřování nebo uzavírání daného regionu a nevěnuje pozornost ostatním regionům. Všechny výše uvedené hodnoty budeme *normalizovat* pomocí nově zavedených koeficientů A_X a B_X, kde X označuje určitou výše uvedenou předpočítanou hodnotu. Každou hodnotu X nahradíme výrazem $A_X + B_X \cdot X$, abychom lépe zohlednili, že předpoklad soustředění se na jeden region není platný. Při hodnotách $A_X = 0$ a $B_X = 1$ dostáváme původní výraz. Nicméně, všechny koeficienty A_X a B_X budeme také učit.

Heuristický hráč používající učící se potenciálovou funkci byl implementován ve třídě `PotentialLearningAi`. Implementace je založena na původní implementaci `HeuristicsAi<PotentialHeuristics>` s rozšířením o logiku načítání a ukládání koeficientů potenciálové funkce. Implementace potenciálové funkce `PotentialFormula` a logika ukládání koeficientů `PotentialFormulaTape` byla zohledněna pro potřeby učení.

2.6 Další metody

Představíme si další metody, které by bylo možné použít pro vytvoření umělých hráčů, ale které nebyly implementovány v této práci.

Zajímavým přístupem je použití *genetických algoritmů*. V této metodě pracujeme se *stavy agenta*, který obsahuje různé číselné parametry. Na počátku máme *populaci*, což je určitý počet agentů s náhodně vybranými stavy. Populace je určitou metodou vyhodnocena pomocí *fitness funkce*. Z populace jsou vybrány dvojice agentů podle jejich výkonu – *fitness hodnoty* – a provede se *reprodukce* spočívající ve výměně několika parametrů mezi dvojicí. U získaných nových agentů se provede *mutace* spočívající v náhodné změně v některém parametru. Tímto postupem získáme novou populaci a opakujeme stejný postup. V průběhu času získáváme populaci s lepší fitness hodnotou. Genetické algoritmy jsou inspirovány *přirozeným výběrem*, ve kterém přežívají silnější jedinci a ti mají možnost reprodukce. Nevýhodou je delší časová doba, než získáme populaci s lepšími výsledky, než je např. u metod strojového učení. Pro Carcassonne – Objevitelé by bylo možné tento přístup použít např. pro koeficienty potenciálové funkce. Kvůli použití učení potenciálové funkce nebyl genetický algoritmus implementován.

V posledních letech je populární metodou při vytváření umělé inteligence *hluboké učení*. Pro Carcassonne – Objevitelé by bylo možné použít metody konvolučních neuronových sítí (CNN) a rekurentních neuronových sítí (RNN).

Vstupem by byl aktuální stav herní desky ve vhodném formátu (např. bitmapě herní desky) a několika číselných hodnot skóre hráčů, případně sekvence několika posledních stavů. Výstupem by byla souřadnice pro umístění kartičky a tah s figurkou, popřípadě heuristická hodnota.

Poslední metodou, kterou zde uvedeme, je *imitace lidského hráče*. Ve skutečné hře lze zařadit tahy hráčů do několika kategorií. Hráč se snaží obsazovat regiony v různých časových horizontech, od vytvoření malých, rychle uzavřených regionů až po vytvoření velkých, bodově výnosných regionů (*krátkodobé* a *dlouhodobé investice*). Vhodné je při tom obsazovat regiony různých typů (*diverzifikace*), protože tím zvyšuje pravděpodobnost výběru vhodné kartičky. Kombinací umístění figurek na hory a přilehlé nížiny zvyšuje hráč bodový zisk sobě (*synergie*) nebo ostatním hráčům (*parazitismus*). Spojením dvou obsazených regionů lze zvýšit bodový zisk pro více figurek najednou (*multiplikace*). Pokud hráč nemůže nebo nechce rozšiřovat nebo uzavírat vlastní regiony, snaží se alespoň omezit možnosti ostatním hráčům (*kažení*). Tyto vypozorované taktiky by se vybíraly podle různých kritérií (skóre, počet volných figurek apod.) vhodnou metodou (např. rozhodovacím stromem).

3. Vyhodnocení

V této kapitole popíšeme postup experimentu, představíme jeho výsledky a zhodnotíme je.

3.1 Cíl a postup experimentu

Cílem experimentu je porovnat výkon jednotlivých heuristik a přístupů navržených v předchozí kapitole.

Stěžejní částí experimentu je simulace většího počtu her dvou hráčů, kdy se utká každý hráč s každým včetně sebe samotného. Pro zhodnocení vlivu počtu hráčů na výsledky hry provedeme i menší počet simulací her tří hráčů.

Porovnání výkonnosti ve dvojici hráčů provedeme pomocí Studentova párového t-testu. Předpokladem je, že dva stejně výkonní hráči budou ve vzájemných střetnutích získávat stejný počet bodů. Nulová hypotéza tedy tvrdí, že střední hodnota rozdílu skóre mezi hráči je rovna nule. V případě zamítnutí nulové hypotézy lze tvrdit, že jeden z hráčů je lepší než druhý. Porovnání provedeme pro všechny dvojice hráčů, a to zvlášť pro hry s daným pořadím hráčů na tahu a zvlášť pro všechny hry mezi dvojicí hráčů bez ohledu na pořadí na tahu. V t-testech a určování intervalů spolehlivosti volíme koeficient spolehlivosti $\alpha = 0{,}01$.

U her tří hráčů budeme pozorovat, jak pořadí hráčů na tahu ovlivňuje skóre hráčů. Výsledky her více hráčů nemusí odpovídat výsledkům her dvou hráčů. Můžeme očekávat, že hráč provádějící slabší tahy bude dávat výhodu následujícímu hráči na tahu. V tom případě potenciálně nejlepší hráč nemusí vyhrát, protože on sám nebude mít takovou výhodu. Porovnání výkonnosti pomocí t-testů nebudeme provádět s ohledem na nízký počet simulací.

Pro porovnání vlivu počtu hráčů na výsledky budeme sledovat celkové skóre všech hráčů dohromady i skóre jednotlivých hráčů. Kromě konečného výsledku budeme také sledovat vývoj skóre hráčů v jednotlivých kolech hry. V různých fázích hry může být různý trend v zisku skóre a na základě toho může být preferovaný jiný přístup.

Simulace budou probíhat na dvou osobních počítačích. První osobní počítač má procesor Intel Core i7-5500U s taktovací frekvencí 2,40 GHz a operační paměť 16 GB. Druhý osobní počítač má čtyřjaderný procesor Intel Core i7-4810MQ s taktovací frekvencí 2,80 GHz a operační paměť 16 GB. Kvůli rozdílné výkonnosti nebudeme omezovat časový limit na přemýšlení a provádění tahů. Na druhou stranu, není žádoucí, aby simulace her trvaly nepřiměřeně dlouhou dobu. Obvyklá doba trvání „živé“ hry je půl až tři čtvrtě hodiny. Simulace by obvyklou dobu trvání hry neměly výrazně překračovat. Tuto skutečnost zohledníme ve výběru hráčů.

Hráči mají možnost přemýšlet pouze tehdy, když jsou na tahu. Z hlediska implementace tato skutečnost znamená, že v každý okamžik simulace jedné hry je spuštěný výpočet pouze pro jednoho hráče. V jedné simulaci je tedy pro delší dobu aktivní nejvýše jedno vlákno. Tato skutečnost umožňuje provádět paralelně více simulací na jednom stroji. Počet simulací, který dává smysl provádět aniž by docházelo k vzájemnému omezování výpočetních prostředků mezi simulacemi, je dán počtem jader CPU.

3.2 Výběr hráčů

Hráče vybereme dle dvou hlavních kritérií. Prvním kritériem je reprezentativnost. Hráči by měli být vybráni tak, aby bylo možné mezi sebou porovnat jednotlivé heuristiky a jednotlivé přístupy. Druhým kritériem je časová a paměťová náročnost. Časové trvání hry by nemělo překračovat tři čtvrtě hodiny na strojích, na kterých budou prováděny simulace. Zároveň by nemělo být využito nepřiměřeně mnoho paměti. Konkrétní limit využití paměti nebudeme stanovovat.

Porovnání jednotlivých heuristik – náhodné, hladové a potenciálové – provedeme pomocí hráčů využívající jednokrokový heuristický přístup. Pro porovnání jednotlivých přístupů – heuristického, Expeciminimaxu, Monte Carlo a učení – vybereme jednoho zástupce. Heuristický přístup už má tři zástupce, není důvod přidávat dalšího hráče pro tento přístup. Pro učící se přístup je třeba vybrat koeficient učení, vstupní koeficienty potenciálové funkce a určit pravidla, jak má učení probíhat. Pro Expectiminimax je třeba vybrat hloubku prohledávání a aplikovanou heuristiku v listech. Pro Monte Carlo přístup je třeba vybrat aplikovanou heuristiku, koeficient znevýhodnění a hranici významnosti.

3.2.1 Volba parametrů učícího se hráče

Pro volbu počátečních koeficientů jsou dvě možnosti. První možností je vyjít z navržených koeficientů pro „neučící se“ potenciálovou funkci. Druhou možností je vybrat počáteční koeficienty náhodně pro dané rozmezí, např. interval $[-1,2]$. Rozhodl jsem se zvolit první možnost. V případě náhodné volby počátečních koeficientů by k různému počtu nutných simulací, než dojde k přizpůsobení koeficientů. Použití totožných počátečních koeficientů je proto spravedlivější.

Ve hrách dvou hráčů budou učící se hráči používat a učit sadu koeficientů specializovanou na protihráče a pořadí na tahu. Učící se hráč A hrající i-tou hru proti hráči B bude využívat koeficienty získané po $(i-1)$-té hře mezi hráči A a B. Ve hře s opačným pořadí hráčů na tahu je použita jiná sada koeficientů. V prvním kole budou použity vstupní koeficienty. Ve hrách mezi dvěma učícími se hráči budou použity dvě sady koeficientů, pro každé pořadí na tahu jedna sada.

Ve hrách tří hráčů budeme používat tři sady koeficientů, pro každé pořadí na tahu jedna sada. Výsledné koeficienty se tedy na rozdíl od her dvou hráčů nespecializují na jednotlivé protihráče.

Do celkových výsledků budeme započítávat všechny hry, nebudeme provádět žádné hry předem. K získání optimálních parametrů tedy může dojít až v průběhu simulací. Funkce učení bude zapnutá po celou dobu, hráč se bude učit s každou hrou.

Z předběžných simulací byl zvolen koeficient učení 0,005. Jedná se o hodnoty z obvyklého rozsahu koeficientu učení používaném ve strojovém učení.

3.2.2 Volba parametrů Expectiminimaxového hráče

Jako heuristika byla zvolena hladová heuristika. Použití náhodné heuristiky nedává smysl, protože náhodný tah lze provést bez prohledávání do hloubky. Potenciálová heuristika nebyla použita, protože podobně jako Expectiminimax predikuje budoucnost. Při použití dvou různých prostředků pro predikci budoucnosti

můžeme dostat zkreslené výsledky vzhledem jednomu vybranému prostředku. Princip potenciálu z heuristické funkce by tedy mohl skrýt princip očekávané hodnoty z Expectiminimaxu.

Na základě předběžných simulací byla zvolena hloubka prohledávání 2. Bohužel, při vyšší hloubce prohledávání dochází k nárůstu časového trvání hry nad přiměřenou mez.

3.2.3 Volba parametrů Monte Carlo hráče

Heuristika pro Monte Carlo hráče byla vybrána dle totožných argumentů, jako u Expectiminimaxového hráče.

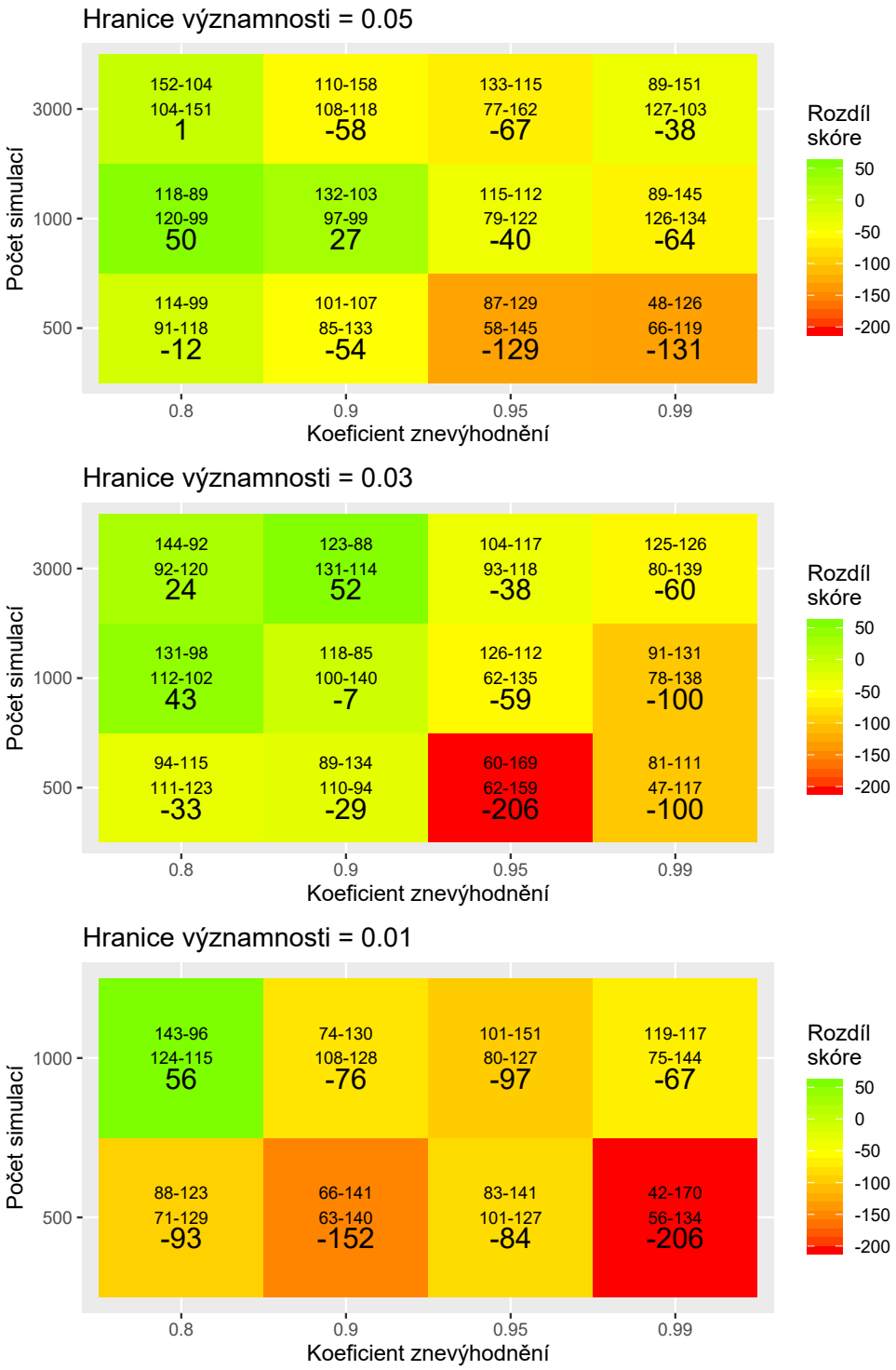
Pro zvolení parametrů hráče využívajícího Monte Carlo metody byly pro všechny kombinace potenciálních parametrů provedeny simulace dvou her hráče s danými parametry a heuristického hráče s hladovou heuristikou. Jako kandidátní parametry byly použity

- pro počet simulací hodnoty 500, 1000 a 3000,
- pro koeficient znevýhodnění hodnoty 0.99, 0.95, 0.90 a 0.80,
- pro hranici významnosti hodnoty 0.05, 0.03, 0.01.

Doba trvání provedení jednoho tahu roste s vyšším počtem simulací, s vyšším koeficientem znevýhodnění a nižší hladinou významnosti.

Výsledky simulací jsou znázorněny v obrázku 3.1. V každé buňce je postupně uveden výsledek hry, kdy začínal MC hráč, dále výsledek hry, kdy MC nezačínal, a nakonec celkový rozdíl skóre mezi MC hráčem a soupeřem. Výsledky her vždy jako první obsahují skóre MC hráče, skóre protihráče je uvedeno jako druhé.

Kombinace vysokého počtu simulací a nižší hladiny významnosti prodloužily herní dobu nad únosnou míru, proto simulace nebyly ani dokončeny a výsledky nejsou k dispozici.



Obrázek 3.1: Přehled výsledků her pro určení parametrů MC hráče.

I přes nízký počet simulací pro jednotlivé kombinace parametrů můžeme v grafu pozorovat určité trendy ve výsledcích. Hráči provádějící vyšší počet simulací dosahují lepších výsledků než hráči s nižším počtem simulací. Hráči s nižším koeficientem znevýhodnění mají lepší výsledky než hráči s vyšším koeficientem znevýhodnění. Nižší hranice významnosti mírně snižuje výkonnost hráče, a především zvyšuje časovou náročnost.

Mírně překvapující jsou lepší výsledky u hráčů s nižším koeficientem zne-

výhodnění. Při výpočtu celkového užitku dostávají vyšší váhu stavy pro dřívěji provedené tahy. To znamená, že hráč by se měl soustředit na koncepci tahů v blízké budoucnosti, než vzdálenějších tahů.

Na základě pozorování byly vybrány následující parametry: počet simulací 3000, koeficient znevýhodnění 0,80 a hranice významnosti 0,05.

3.3 Úvodní hypotézy

V porovnání hráčů očekáváme, že zásadní roli bude mít schopnost přemýšlení do budoucnosti. Tato schopnost se může projevovat více způsoby. Slabší tahy jednoho hráče dávají výhodu druhému hráči, proto je třeba odhadnout, jaké možnosti mají ostatní hráči. Některé regiony mohou být špatně uzavíratelné a nemusí se vyplatit plýtvat tahy pro jejich uzavírání.

Nejhorší výsledky bude mít beze sporu náhodná heuristika. V porovnání zbylých dvou heuristik očekáváme lepší výsledky u potenciálové heuristiky než u hladové heuristiky. Potenciálová heuristika uvažuje budoucí tahy v rámci potenciálu obsaditelných regionů. Hladová heuristika zohledňuje stav pouze prostřednictvím skóre a nikoliv stavu na herní desce.

V porovnání jednotlivých metod bude nejhorší heuristická metoda. Hráči využívající Expectiminimax i Monte Carlo metody provádějí prohledávání a tím mohou lépe odhadovat budoucnost. Vzhledem k nízké hloubce prohledávání u Expectiminimaxu odhadujeme lepší výsledky pro Monte Carlo přístup. Učící se hráč bude v prvních hrách proti jednotlivým soupeřům zaostávat, ale postupem času se stane lepším, než jeho soupeři.

Celkový součet skóre všech hráčů by však měl být mírně vyšší při vyšším počtu hráčů, protože se zvýší celkový počet příležitostí k odebrání figurky. Průměrné skóre jednoho hráče bude poměrově menší vzhledem k počtu hráčů.

Ve hrách více hráčů bude mít roli uspořádání hráčů na tahu. Výsledky ve dvojicích po sobě táhnoucích hráčů budou odpovídat výsledkům z her dvou hráčů.

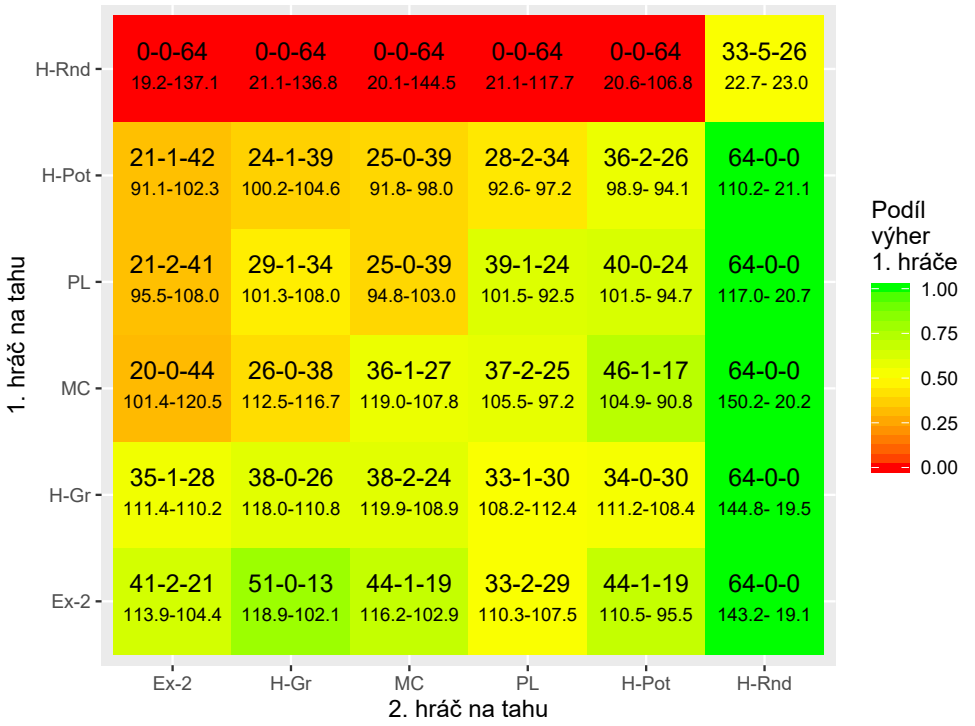
3.4 Výsledky her dvou hráčů

Pro každou uspořádanou dvojici hráčů bylo provedeno 64 simulací, celkově tedy bylo provedeno $6^2 \cdot 64 = 2304$ simulací. V simulacích hráči používají standardizovaná jména. Přehled hráčů je uveden v tabulce 3.1.

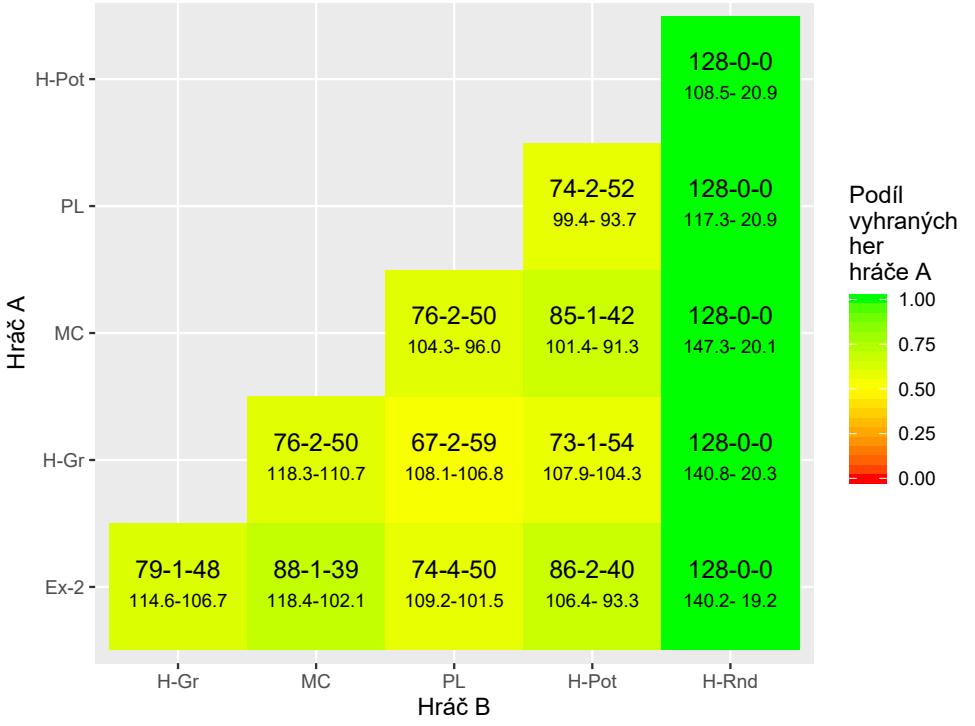
Zkratka	Plný název
H-Rnd	HeuristicsAi-Random
H-Gr	HeuristicsAi-Greedy
H-Pot	HeuristicsAi-Potential
Ex-2	ExpectiminimaxAi-2-Greedy
MC	MonteCarloAi-1000-0,8-0,05-Greedy
PL	PotentialLearningAi-FromDefault-0,005

Tabulka 3.1: Přehled hráčů.

Souhrnné výsledky her mezi jednotlivými dvojicemi jsou znázorněny na obrázcích 3.2 a 3.3. V buňkách křížové tabulky je uveden počet výher hráče nalevo, počet remíz a počet výher hráče napravo a dále průměrné výsledné skóre hráčů.



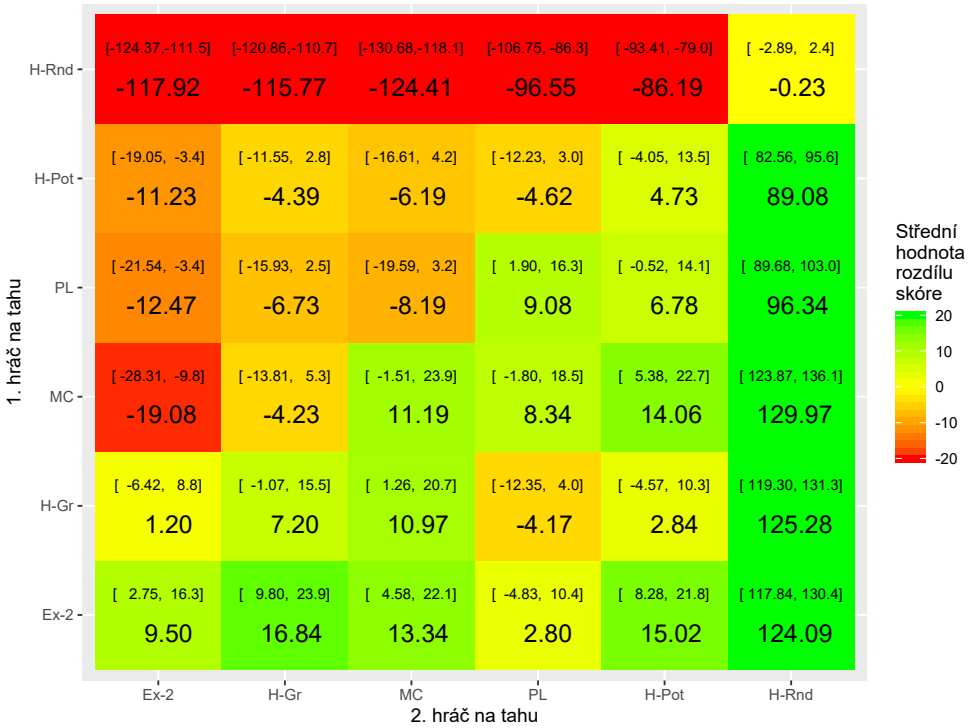
Obrázek 3.2: Křížová tabulka výsledků s ohledem na pořadí hráčů.



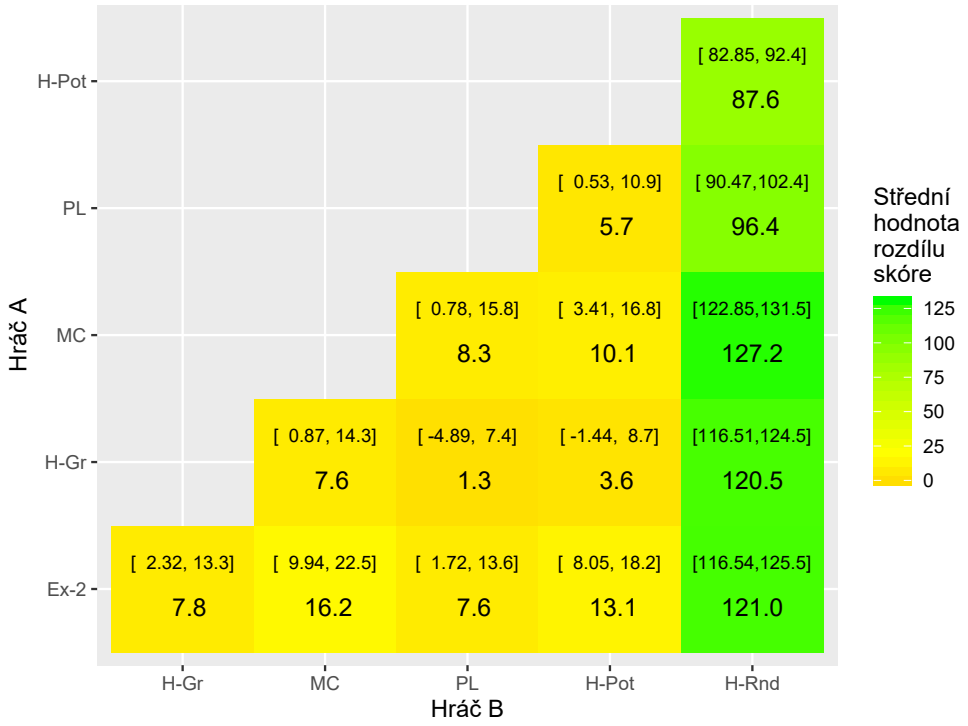
Obrázek 3.3: Křížová tabulka výsledků bez ohledu na pořadí hráčů.

Z křížové tabulky dle pořadí hráčů pozorujeme, že v případě her dvou totožných hráčů, jejichž výsledky se nacházejí na diagonále tabulky, má převahu začínající hráč. U většiny zbylých her má převahu jeden z hráčů bez ohledu na pořadí hráčů na tah. Jedinou výjimku tvoří dvojice hráčů (Ex-2, H-Gr). Objevily se i dva případy, kdy hráč s vyšším počtem výher dosáhl menšího průměrného skóre než jeho soupeř, a to u dvojic (H-Gr, PL) a (H-Rnd, H-Rnd).

Na křížové tabulce výsledků her bez ohledu na pořadí hráčů pozorujeme, že hráče lze jednoznačně seřadit tak, že každý výše umístěný hráč porazil všechny ostatní níže umístěné hráče, alespoň v průměrném rozdílu skóre. Pomocí t-testů určíme, zda jsou rozdíly mezi hráči statisticky významné. Výsledky párových t-testů na rozdíl skóre ve hrách jsou uvedeny na obrázcích 3.4 a 3.5. V buňkách křížové tabulky je uveden interval spolehlivosti pro rozdíl skóre mezi hráči a střední hodnota rozdílu skóre.



Obrázek 3.4: Výsledky t-testů her s ohledem na pořadí hráčů.



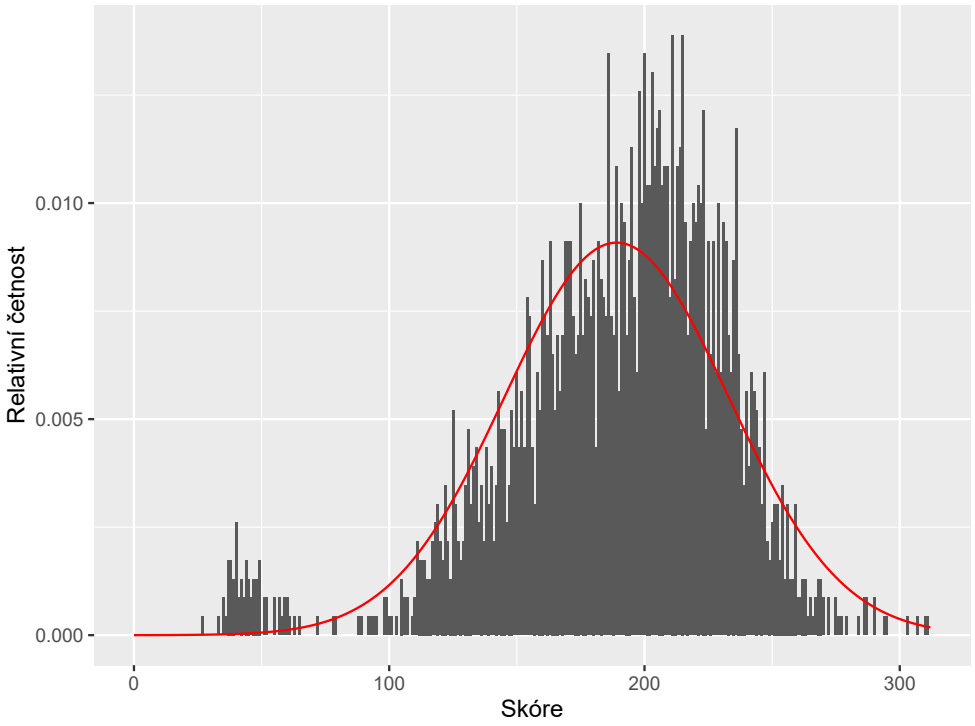
Obrázek 3.5: Výsledky t-testů her bez ohledu na pořadí hráčů.

Nulovou hypotézu nezamítáme, pokud 0 náleží do intervalu spolehlivosti. U her s rozlišením hráčů dle pořadí na tahu je poměrně vysoký počet zamítnutí nulové hypotézy. V případě započítání her bez ohledu na pořadí na tahu byla nulová hypotéza zamítnutá u dvou dvojic hráčů. Velký rozdíl v množství takových případů lze vysvětlit velikostí souboru hodnot. Při rozlišení her vzhledem k pořadí na tahu máme poloviční počet vzorků než v případě bez rozlišení pořadí hráčů na tahu, a tak je interval spolehlivosti relativně širší a častěji obsahuje nulu.

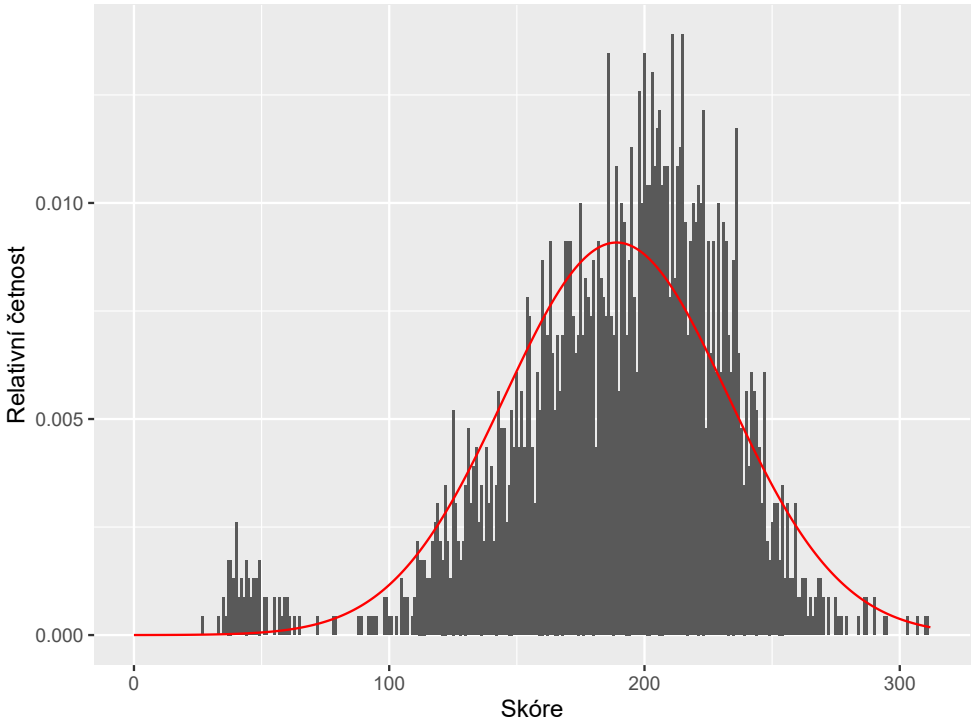
Nulovou hypotézu zamítáme u dvojic hráčů (H-Gr, PL) a (H-Gr, H-Pot). Velmi blízko zamítnutí jsou také dvojice (H-Gr, MC), (MC, PL) a (PL, H-Pot).

Na předchozích grafech jsme pozorovali velké rozdíly ve skóre hráče H-Rnd a skóre ostatních hráčů. Proto vyhodnocení skóre her provedeme zvlášť pro všechny hry a zvlášť pro hry neobsahující hráče H-Rnd. Obě varianty porovnáme mezi sebou.

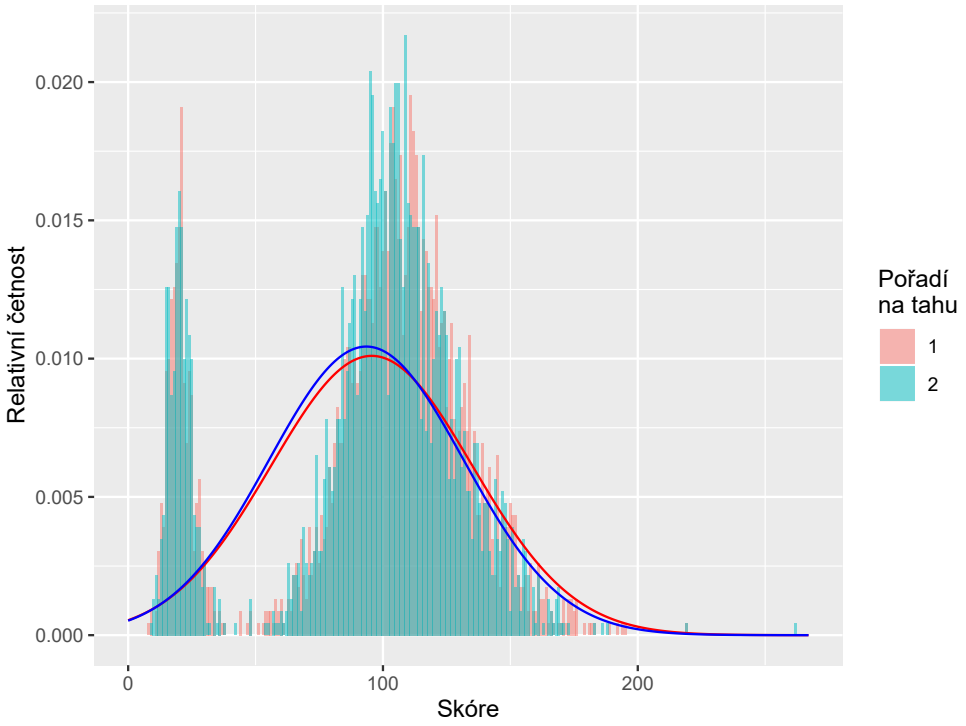
Histogram celkového skóre hráčů dohromady a jednotlivých hráčů zvlášť podle pořadí na tahu znázorňují grafy na obrázcích 3.6 až 3.9 a tabulky 3.2 a 3.3. V grafech je křivkami znázorněno rozdělení pravděpodobnosti normálního rozdělení se střední hodnotou a rozptylem odpovídajícími naměřeným hodnotám.



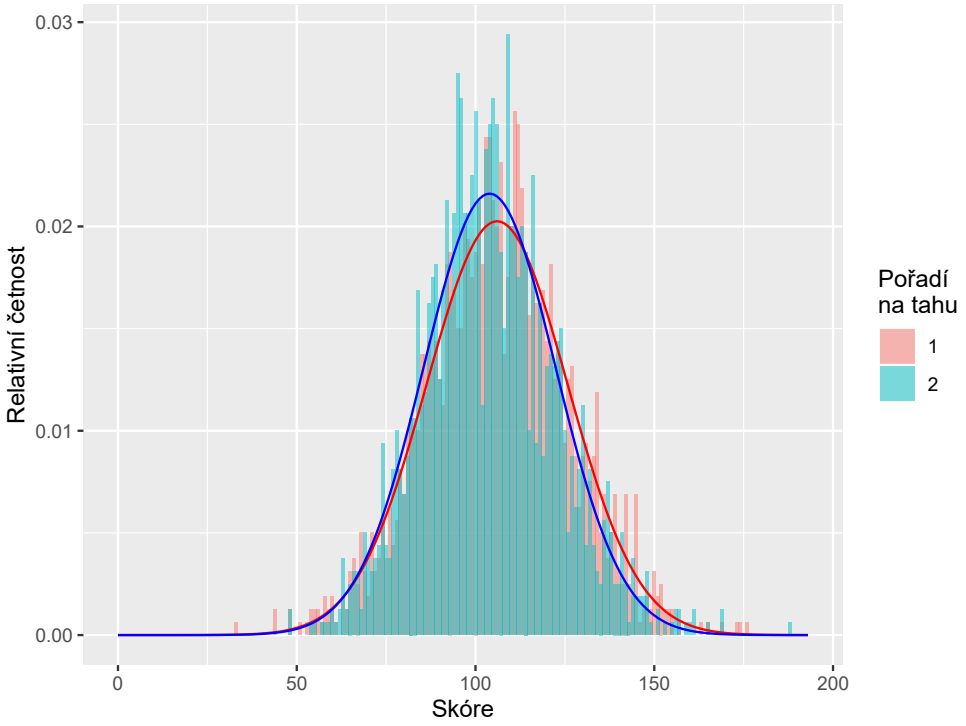
Obrázek 3.6: Histogram součtu výsledného skóre obou hráčů.



Obrázek 3.7: Histogram součtu výsledného skóre obou hráčů po vyloučení her hráče H-Rnd.



Obrázek 3.8: Histogram výsledného skóre hráčů dle pořadí na tahu.



Obrázek 3.9: Histogram výsledného skóre hráčů dle pořadí na tahu po vyloučení her hráče H-Rnd.

	Střední hodnota	Rozptyl	Interval spolehlivosti
Celkové skóre	189,06	43,91	[186,70 ; 191,42]
Skóre 1. hráče	95,59	39,50	[93,47 ; 97,71]
Skóre 2. hráče	93,47	38,23	[91,42 ; 95,52]
Rozdíl skóre	+ 2,12	64,14	[−1,33 ; 5,56]

Tabulka 3.2: Statistiky výsledného skóre.

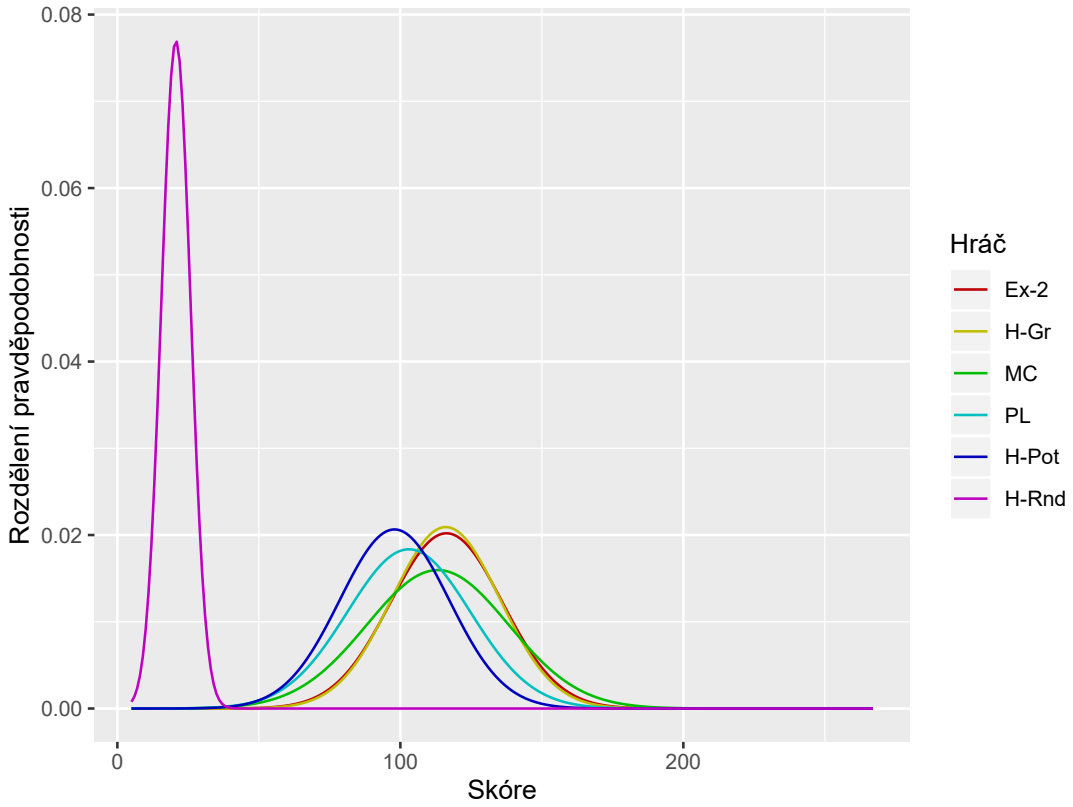
	Střední hodnota	Rozptyl	Interval spolehlivosti
Celkové skóre	209,98	26,27	[208,28 ; 211,67]
Skóre 1. hráče	106,04	19,70	[104,77 ; 107,31]
Skóre 2. hráče	103,94	18,46	[102,75 ; 105,13]
Rozdíl skóre	+ 2,10	27,70	[0,32 ; 3,89]

Tabulka 3.3: Statistiky výsledného skóre po vyloučení her hráče H-Rnd.

Rozdíly mezi započítáním všech her a her neobsahujících hráče H-Rnd jsou zřejmé. V histogramu celkového skóre i histogramech skóre hráčů dle pořadí na tahu pozorujeme shluky hodnot v levé části, které odpovídají výsledkům hráče H-Rnd. Díky vyřazení nízkých hodnot se zvýšila střední hodnota skóre a snížily rozptyly celkového skóre a rozdílů skóre.

Předpoklad normálního rozdělení skóre hráčů se jeví jako správný. Křivky náhodného rozdělují kopírují histogram. Rozdíl skóre mezi začínajícím a nezačínajícím hráčem je malý, ale i přesto je statisticky významný, alespoň po vyloučení her H-Rnd, které způsobují velký rozptyl hodnot.

Histogramy výsledného skóre pro jednotlivé hráče jsou uvedeny na obrázku 3.10 a v tabulce 3.4.



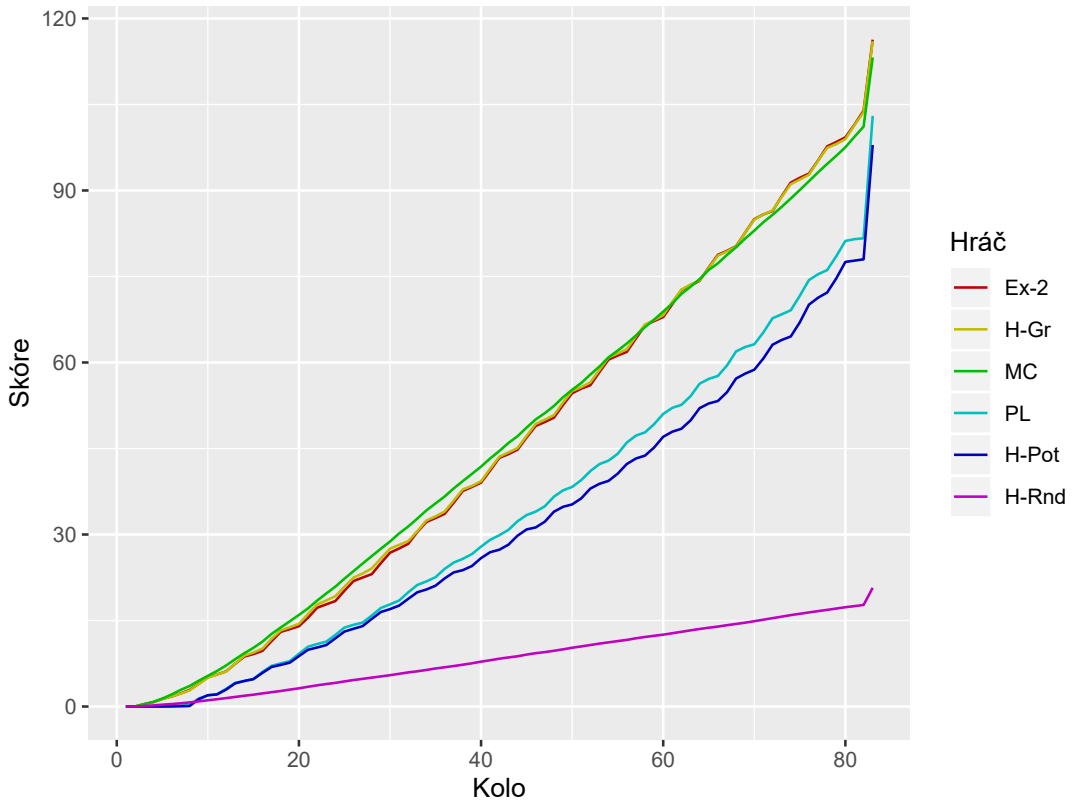
Obrázek 3.10: Rozdělení skóre všech hráčů.

Hráč	Střední hodnota	Rozptyl	Interval spolehlivosti
Ex-2	116,30	19,75	[114,46 ; 118,14]
H-Gr	116,05	19,07	[114,27 ; 117,82]
MC	113,20	25,00	[110,87 ; 115,53]
PL	103,01	21,73	[100,98 ; 105,03]
H-Pot	97,92	19,32	[99,12 ; 99,72]
H-Rnd	20,70	5,18	[20,22 ; 21,19]

Tabulka 3.4: Statistiky skóre hráčů.

Seřazení hráčů podle střední hodnoty jejich skóre odpovídá seřazení hráčů podle výsledků z křížové tabulky. Intervaly spolehlivosti pro skóre hráčů se částečně překrývají u hráčů Ex-2, H-Gr a MC. Mezi ostatními hráči jsou dostatečně velké rozdíly v hodnotách.

Vývoj průměrného získaného skóre hráčů v jednotlivých kolech jsou uvedeny na obrázku 3.11.



Obrázek 3.11: Vývoj skóre hráčů v průběhu hry.

Vývoj skóre u hráče MC je na počátku strmější než u hráčů Ex-2 a H-Gr, v průběhu hry ale hráči Ex-2 a H-Gr získávají vyšší skóre. Hráč PL dostahuje mírně vyššího skóre než H-Pot, což je pravděpodobně způsobeno přizpůsobením koeficientů pro jednotlivé soupeře.

Na konci hry pozorujeme skokový nárůst skóre u všech hráčů, díky odebrání zbývajících figurek. U hráčů H-Pot a PL dochází k vyššímu skokovému nárůstu skóre než u ostatních hráčů. To může být způsobeno tím, že potenciálové heuristiky nadhodnocují potenciální skóre regionů, a tak mají jinou strategii pro zisk skóre.

Vývoj skóre u hráčů MC a H-Rnd je více rovnoměrný, u ostatních dochází k „pilovitému“ vývoji. To lze vysvětlit tak, že hráči střídají tahy s položením a odebráním figurky. K přičtení bodů tedy dojde jednou za 4 kola.

Celkový průběh až na úplný začátek a konec hry připomíná konvexní křivku velmi podobnou úsečce. Mimo sklon, „pilovitost“ křivky a skokové rozdíly na konci hry nepozorujeme žádné další významné rozdíly mezi křivkami.

3.5 Výsledky her tří hráčů

Pro každou uspořádanou trojici hráčů bylo provedeno 6 simulací, celkově tedy bylo provedeno $6^3 \cdot 6 = 1296$ simulací. U her dvou hráčů jsme pozorovali velké rozdíly ve výsledcích při započítání všech her a výsledcích her neobsahující hráče H-Rnd. Z toho důvodu budeme vyhodnocovat zvlášť výsledky všech her a zvlášť výsledky her neobsahující hráče H-Rnd.

Souhrnné výsledky her jsou znázorněny v tabulkách 3.5 a 3.6.

Hráč	1. na tahu				2. na tahu			
	Skóre			Pořadí	Skóre			Pořadí
Ex-2	75,90	-	60,88	1,64	76,84	-	62,24	1,74
H-Gr	77,62	-	63,60	1,77	75,78	-	65,25	1,88
MC	74,38	-	62,59	1,78	75,24	-	62,81	1,74
PL	74,60	-	58,60	1,60	72,22	-	59,62	1,77
H-Pot	70,04	-	61,16	1,86	67,18	-	60,25	1,99
H-Rnd	15,45	-	69,42	2,81	14,68	-	69,12	2,84

Hráč	3. na tahu				Všechny hry			
	Skóre			Pořadí	Skóre			Pořadí
Ex-2	73,15	-	63,75	1,90	75,30	-	62,29	1,76
H-Gr	71,56	-	65,22	2,14	74,98	-	64,69	1,93
MC	70,37	-	63,42	1,91	73,33	-	62,94	1,81
PL	72,85	-	61,50	1,81	73,22	-	59,91	1,73
H-Pot	68,32	-	61,49	1,96	68,51	-	60,97	1,94
H-Rnd	14,33	-	69,59	2,85	14,82	-	69,37	2,83

Tabulka 3.5: Souhrnné výsledky her tří hráčů.

Hráč	1. na tahu				2. na tahu			
	Skóre			Pořadí	Skóre			Pořadí
Ex-2	74,11	-	69,10	1,75	73,10	-	70,47	1,92
PL	73,44	-	65,98	1,68	70,50	-	66,91	1,88
H-Gr	73,63	-	71,58	1,93	72,17	-	74,39	2,09
MC	71,67	-	70,47	1,94	71,95	-	71,28	1,89
H-Pot	69,47	-	69,40	1,99	64,40	-	68,58	2,22

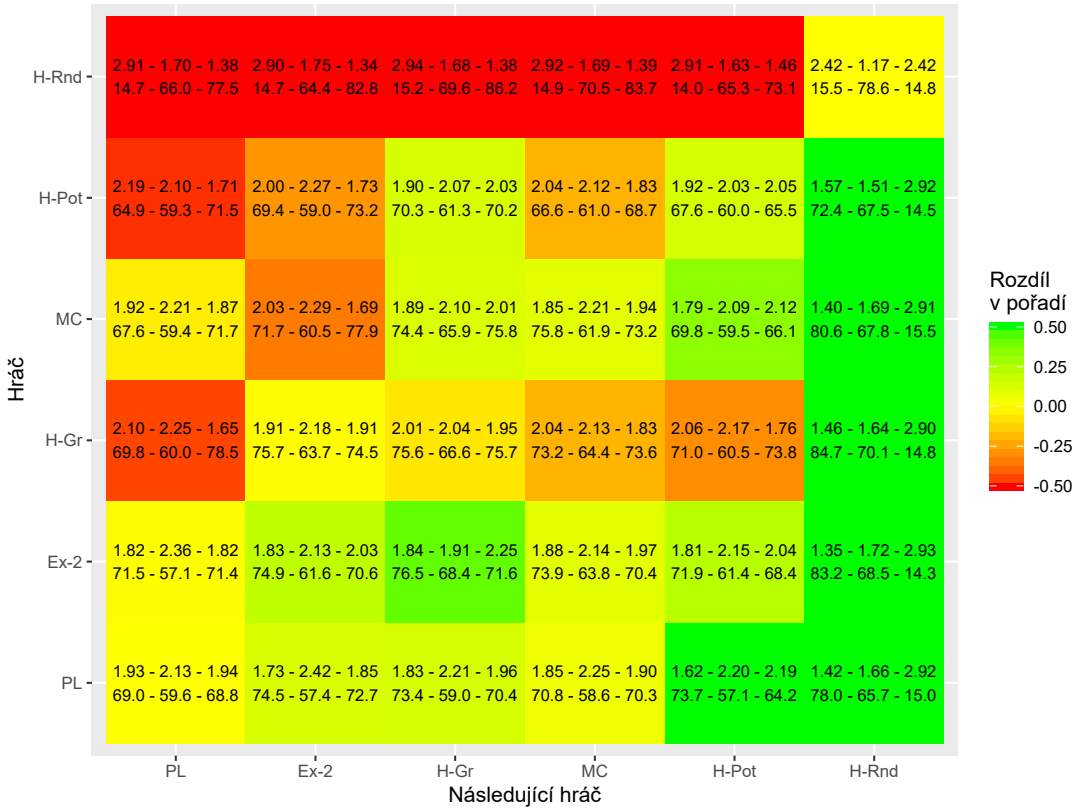
Hráč	3. na tahu				Všechny hry			
	Skóre			Pořadí	Skóre			Pořadí
Ex-2	70,03	-	73,06	2,11	72,41	-	70,88	1,92
PL	70,18	-	69,69	2,01	71,37	-	67,53	1,86
H-Gr	66,88	-	74,11	2,39	70,90	-	73,36	2,14
MC	66,97	-	71,27	2,12	70,20	-	71,00	1,98
H-Pot	66,87	-	69,10	2,09	66,92	-	69,03	2,10

Tabulka 3.6: Souhrnné výsledky her tří hráčů neobsahující hráče H-Rnd.

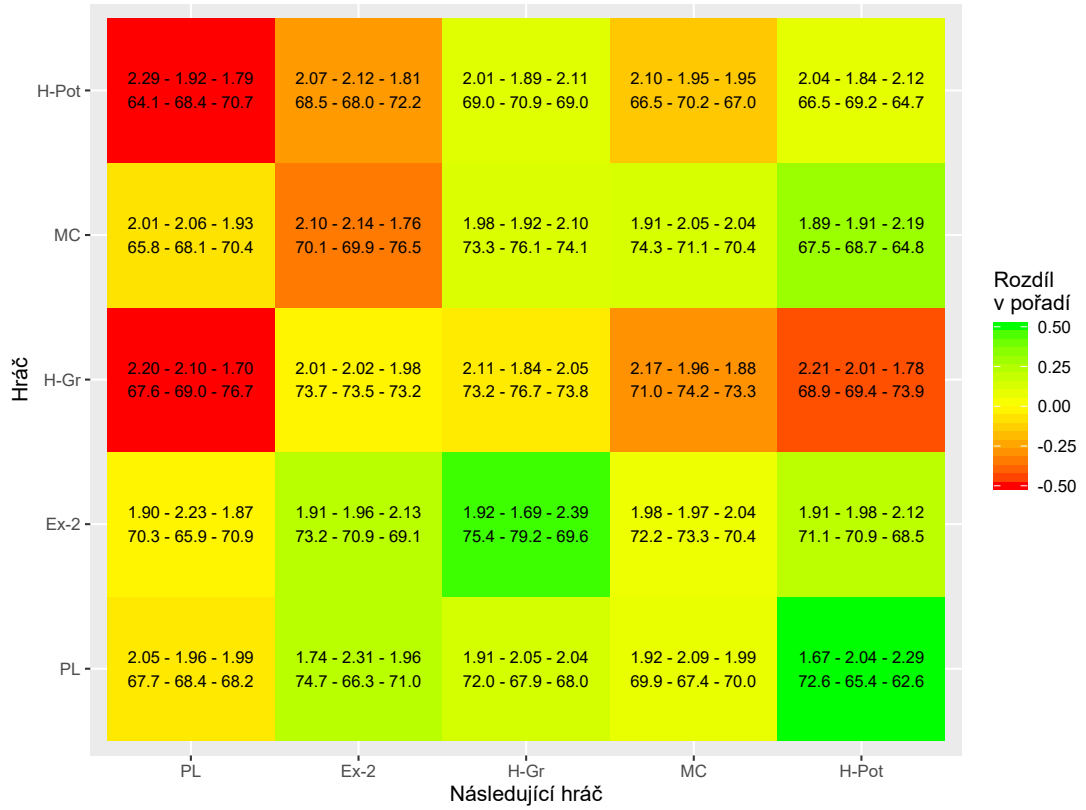
Na rozdíl od her dvou hráčů pozorujeme, že průměrné skóre hráčů není přímo úměrné jejich pořadí. Při vyloučení her hráče H-Rnd je odlišné uspořádání hráčů podle získaného skóre, uspořádání podle průměrného pořadí je přibližně zachováno.

Porovnání výsledků dvojic po sobě táhnoucích hráčů je znázorněno na obrázcích 3.12 a 3.13. První graf opět zohledňuje výsledky všech her, druhý graf

nezapočítává hry hráče H-Rnd. Každá buňka grafu obsahuje na prvním řádku průměrné umístění hráče ve hře a na druhém řádku je uvedeno průměrné skóre hráčů. Vždy je nejprve uvedena hodnota pro hráče táhnoucího ve dvojici jako první, poté hodnota pro třetího – neuvedeného – hráče a nakonec hodnota pro hráče táhnoucího ve dvojici jako druhý.



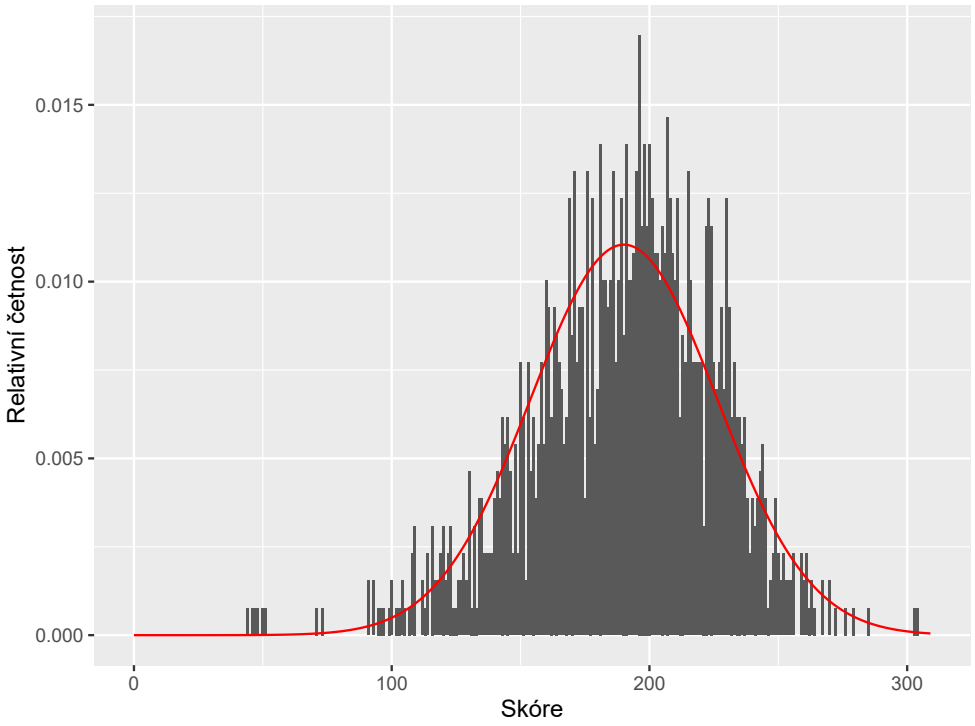
Obrázek 3.12: Křížová tabulka výsledků her podle dvojic po sobě táhnoucích hráčů.



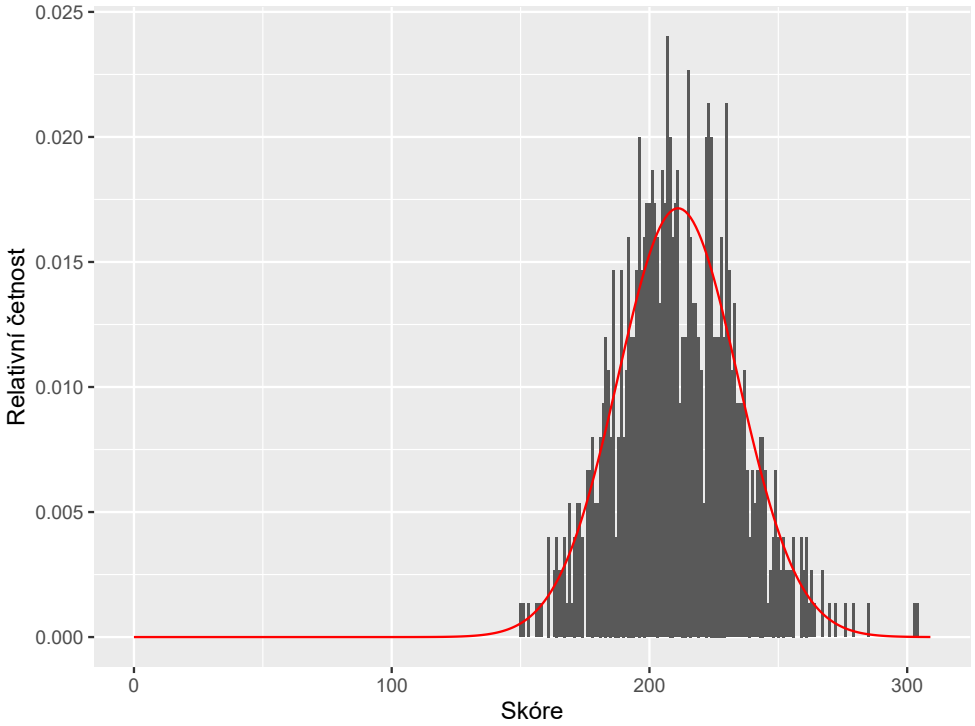
Obrázek 3.13: Křížová tabulka výsledků her neobsahující hráče H-Rnd podle dvojic po sobě táhnoucích hráčů.

Výsledky všech her a výsledky her bez započítání her H-Rnd se od sebe příliš neliší, znatelným rozdílem je pouze vyšší skóre třetího hráče. Naopak zřetelné jsou rozdíly mezi obdobnými výsledky ve hrách dvou hráčů. Nejsilnějším hráčem je dle výsledků hráč PL, naopak hráč H-Gr se řadí mezi nejslabší (s výjimkou H-Rnd).

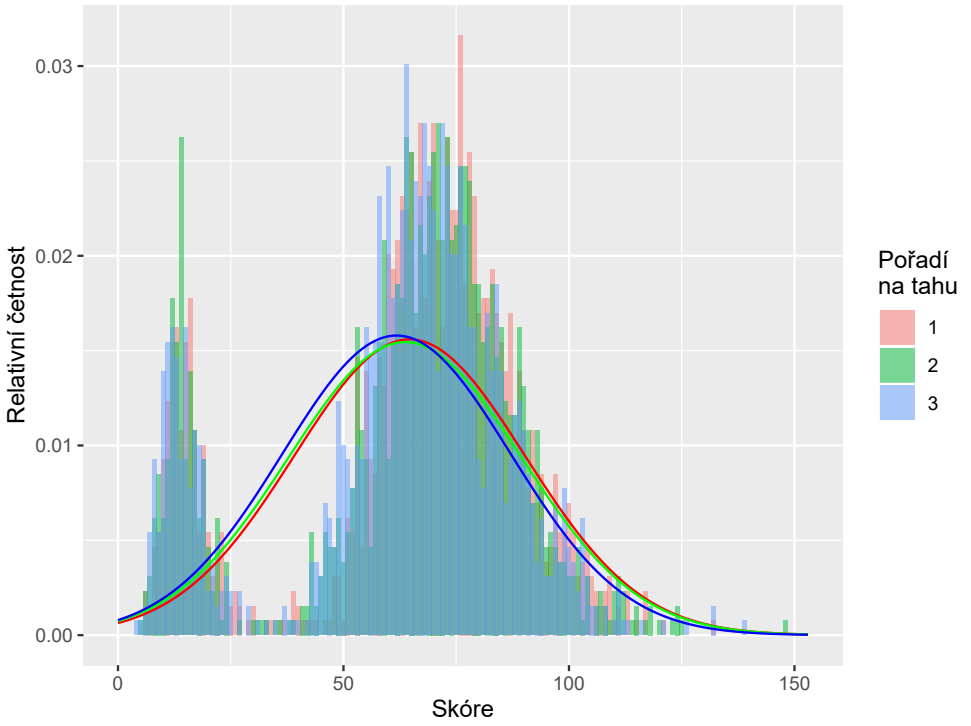
Histogram celkového skóre hráčů dohromady a jednotlivých hráčů zvlášť podle pořadí na tahu znázorňují grafy na obrázcích 3.14 až 3.17 a tabulky 3.7 a 3.8. Opět je v grafech znázorněno rozdělení pravděpodobnosti normálního rozdělení se střední hodnotou a rozptylem odpovídajícími naměřeným hodnotám.



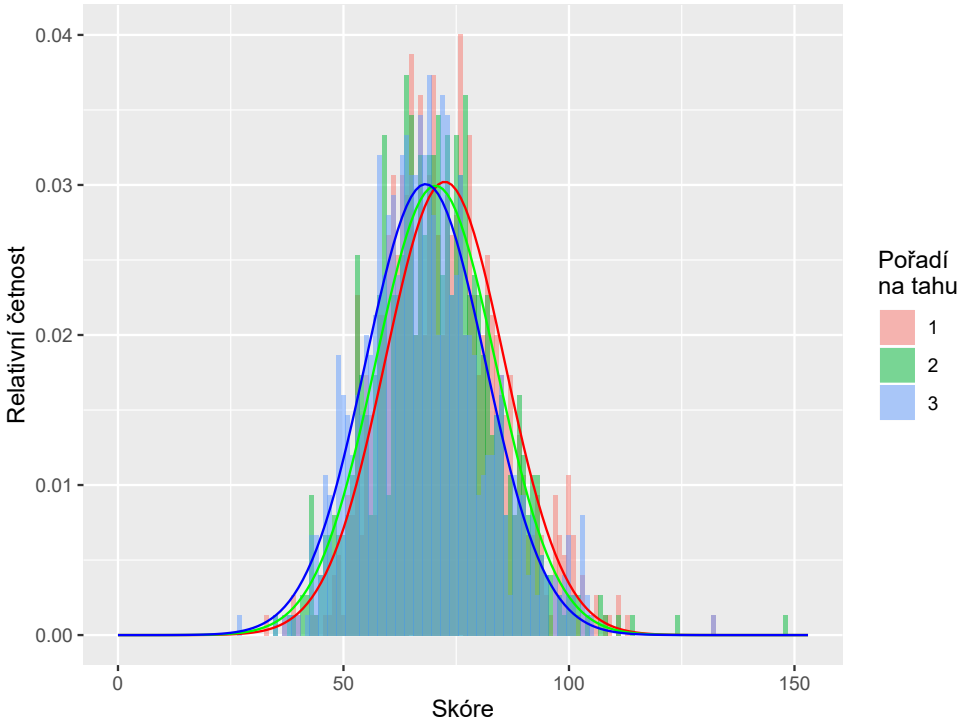
Obrázek 3.14: Histogram součtu výsledného skóre všech tří hráčů.



Obrázek 3.15: Histogram součtu výsledného skóre obou všech tří po vyloučení her hráče H-Rnd.



Obrázek 3.16: Histogram výsledného skóre hráčů dle pořadí na tahu.



Obrázek 3.17: Histogram výsledného skóre hráčů dle pořadí na tahu po vyloučení her hráče H-Rnd.

	Střední hodnota	Rozptyl	Interval spolehlivosti
Celkové skóre	190,08	36,12	[187,75 ; 192,67]
Skóre 1. hráče	64,59	25,58	[62,83 ; 66,52]
Skóre 2. hráče	63,47	25,82	[61,80 ; 65,51]
Skóre 3. hráče	61,47	25,25	[59,95 ; 63,57]

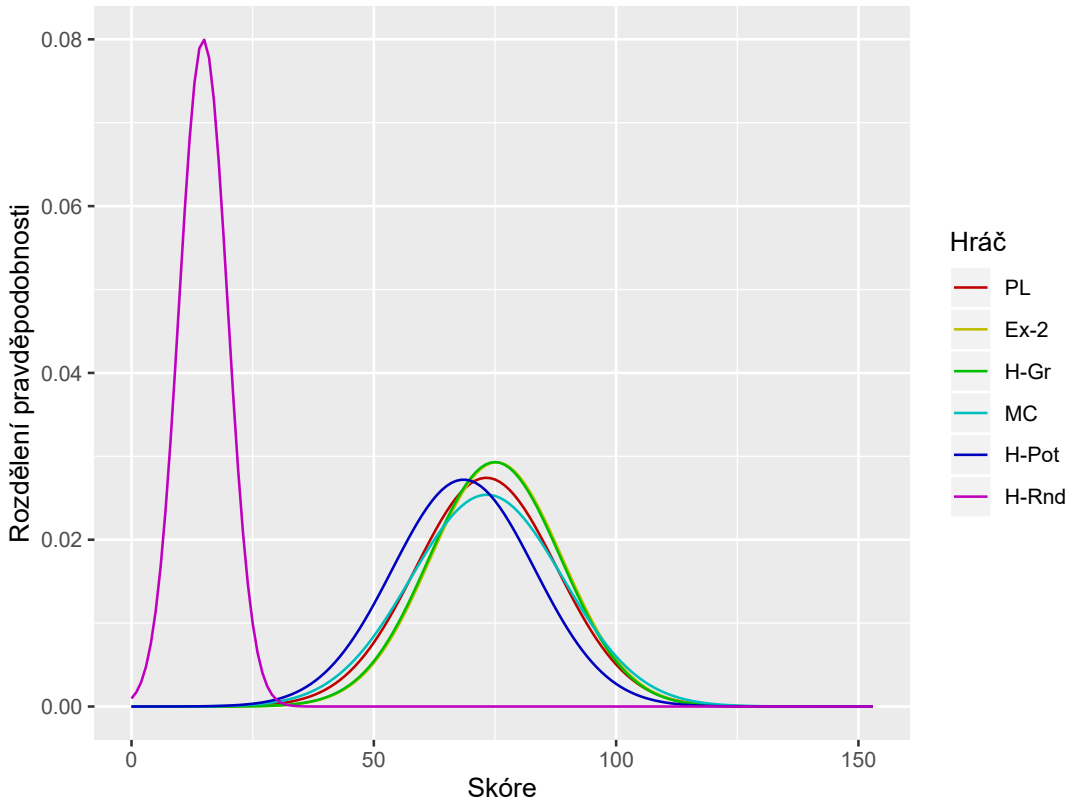
Tabulka 3.7: Statistiky výsledného skóre her tří hráčů.

	Střední hodnota	Rozptyl	Interval spolehlivosti
Celkové skóre	211,08	23,26	[208,88 ; 213,27]
Skóre 1. hráče	72,47	13,20	[71,22 ; 73,71]
Skóre 2. hráče	70,42	13,32	[69,17 ; 71,68]
Skóre 3. hráče	68,19	13,27	[66,94 ; 69,44]

Tabulka 3.8: Statistiky výsledného skóre her tří hráčů po vyloučení her hráče H-Rnd.

Pozorujeme totožné jevy jako při porovnání skóre u her dvou hráčů. Rozdíly mezi výsledky všech her a výsledky her neobsahující hráče H-Rnd se projevují vyšší rozptylem hodnot a vyšší četností nízkých hodnot skóre. Četnost hodnot skóre se přibližuje normálnímu rozdělení. Hráči dříve na tahu mají výhodu nad ostatními hráči.

Histogram skóre jednotlivých hráčů je znázorněn na obrázku 3.18 a v tabulce 3.9. Uvádíme výsledky všech her.



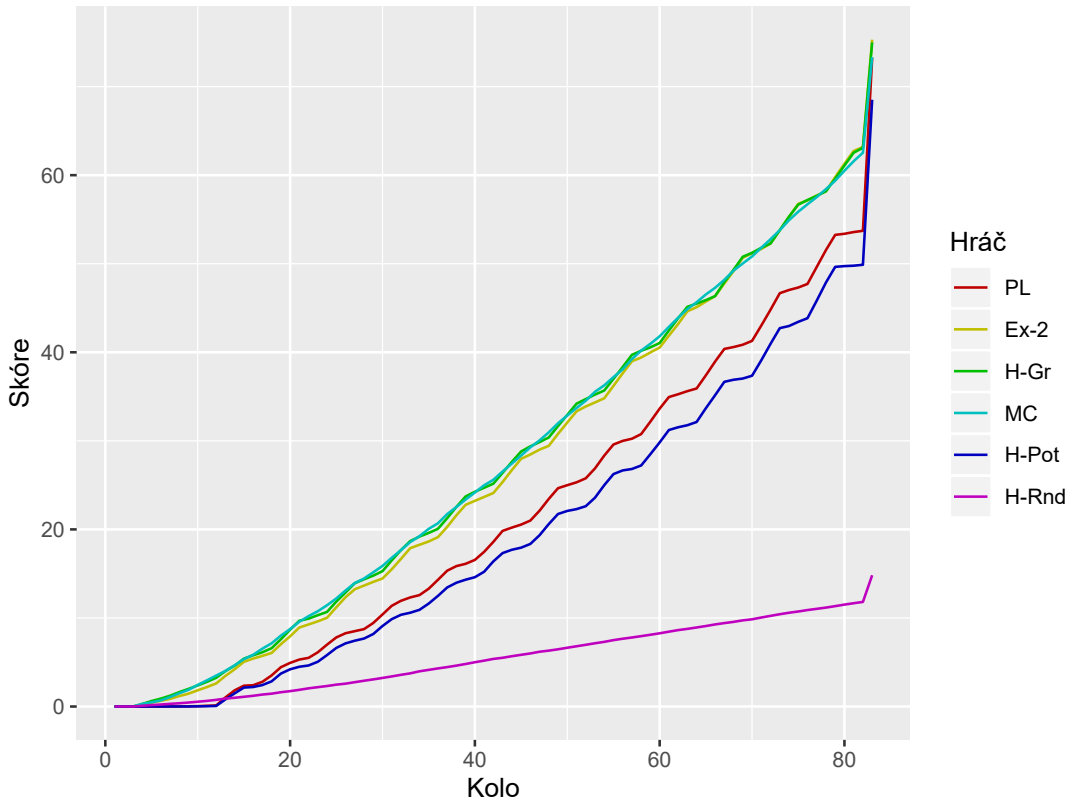
Obrázek 3.18: Rozdělení skóre hráčů ve hrách tří hráčů.

Hráč	Střední hodnota	Rozptyl	Interval spolehlivosti
Ex-2	75,30	13,62	[73,92 ; 76,68]
H-Gr	74,98	13,61	[73,60 ; 76,37]
MC	73,33	15,71	[71,73 ; 74,62]
PL	73,22	14,55	[71,75 ; 74,70]
H-Pot	68,51	14,66	[67,03 ; 70,00]
H-Rnd	14,82	4,99	[14,32 ; 15,33]

Tabulka 3.9: Statistiky skóre hráčů ve hrách tří hráčů.

Skóre hráčů je vyrovnanější než u her dvou hráčů, intervaly spolehlivosti se částečně překrývají u čtyř hráčů. To je nejspíše způsobeno nižším počtem simulací a tedy i vzorků.

Vývoj průměrného získaného skóre hráčů v jednotlivých kolech jsou uvedeny na obrázku 3.19.



Obrázek 3.19: Vývoj skóre jednotlivých hráčů v průběhu hry tří hráčů.

Vývoj skóre ve hrách tří hráčů velmi připomíná vývoj skóre ve hrách dvou hráčů. Skóre hráčů Ex-2, H-Gr a MC se více překrývá. Bodový rozdíl mezi touto trojicí a dvojicí hráčů PL a H-Pot je nyní nižší, pravděpodobně kvůli nižšímu podílu na celkovém skóre způsobené třetím hráčem ve hře.

Opět se objevuje „pilovitá“ křivka u některých hráčů, ale nyní jsou jednotlivé „zuby“ větší. To podporuje hypotézu o strategii střídání přikládání a odebírání figurek. Místo periody 4 kol v případě dvou hráčů je nyní perioda délky 6 kol.

3.6 Závěr

Ve hrách dvou hráčů jsme určili pořadí hráčů dle výkonnosti podle jejich vzájemných výsledků (graf na obrázku 3.3), a to v pořadí Ex-2, H-Gr, MC, PL, H-Pot a H-Rnd. U her tří hráčů bylo umístění dle průměrného skóre hráčů totožné (tabulka 3.9), ale neodpovídalo průměrnému umístění hráčů ve hře (tabulky 3.5 a 3.6). Pořadí hráčů podle jejich průměrného umístění by bylo PL, Ex-2, MC, H-Gr, H-Pot a H-Rnd.

Dle očekávání je nejslabší náhodná heuristika. Hladová heuristika je silnější než potenciálová heuristika při hrách méně hráčů. U her více hráčů hladová heuristika hůře zohledňuje stav hry. Hráč s učící se potenciálovou heuristikou má lepší výsledky než hráč s potenciálovou heuristikou.

V porovnání metod je nejsilnější Expectiminimax, přestože hráč používající tuto metodu dosahoval nízké hloubky prohledávání. S ohledem na pokles výkonnosti hladové heuristiky při vyšším počtu hráčů by musel hráč použít vyšší hloubku prohledávání, aby dosahoval obdobných výsledků i ve hrách více hráčů.

Monte Carlo metoda, metoda učení a jednokroková heuristická metoda dosáhly odlišných výsledků pro hry se dvěma hráči a pro hry se třemi hráči. Obecně platí, že při vyšším počtu hráčů provádí každý z hráčů méně tahů a tak je rozhodující lepší schopnost odhadu budoucnosti. Hrám tří hráčů se metoda učení přizpůsobila tak, že byla srovnatelná s Expectiminimaxem. Naopak u her dvou hráčů dosahovala lepších výsledků „úsporná“ jednokroková.

Průměrný součet skóre ve hrách dvou hráčů (189,06 resp. 209,98 bez započítání her náhodné heuristiky) je mírně nižší než průměrný součet skóre ve hrách tří hráčů (190,08 resp. 211,08). Předpoklad s mírně vyšším průměrným skóre se tedy potvrdil. Na druhou stranu, intervaly spolehlivosti pro součty skóre mají netriviální překryv, nemůžeme s jistotou určit, která hodnota je vyšší.

3.7 Zdrojová data

Konfigurační soubory pro spuštění simulací, data s výsledky simulací a R skripty použité pro zpracování výsledků jsou v elektronické příloze této práce.

4. Simulační prostředí

V této kapitole zdokumentujeme návrh a implementaci simulačního prostředí. Uvedeme také uživatelskou dokumentaci k ovládání programů vytvořených v rámci této práce.

4.1 Specifikace

Nároky na simulační prostředí se pokusíme minimalizovat, budeme se soustředit na možnost automatické simulace her umělých inteligencí. Na simulační prostředí budeme klást tyto nároky:

1. V prostředí existuje entita, která řídí a rozhoduje simulace her, tedy vykonává herní úkony na základě požadavku hráčů a kontroluje dodržování pravidel. Tato entita se v prostředí chová jako *server*.

2. Hráči resp. agenti jsou samostatné entity, které se chovají jako *klienti*. V rámci simulace hry jsou hráči identifikovaní svou barvou, pro účely vyhodnocení je dále třeba uchovávat další identifikaci hráčů, např. jméno.

3. Uživatel má možnost zakládat nové hry a dotazovat se na výsledky odehraných her. Tyto úkony provádí pomocí specializovaných klientů.

4. Server umožňuje klientům pomocí vhodně definovaného *aplikačního rozhraní* provádět základní herní úkony a ovládání simulací popsané v předchozích bodech. Komunikace pomocí vybraného běžného *komunikačního protokolu*. Pro účely této práce není nutné dbát na autentizaci a autorizaci, tedy klienti mohou provádět libovolné úkony a serverem nebude ověřována jejich identita.

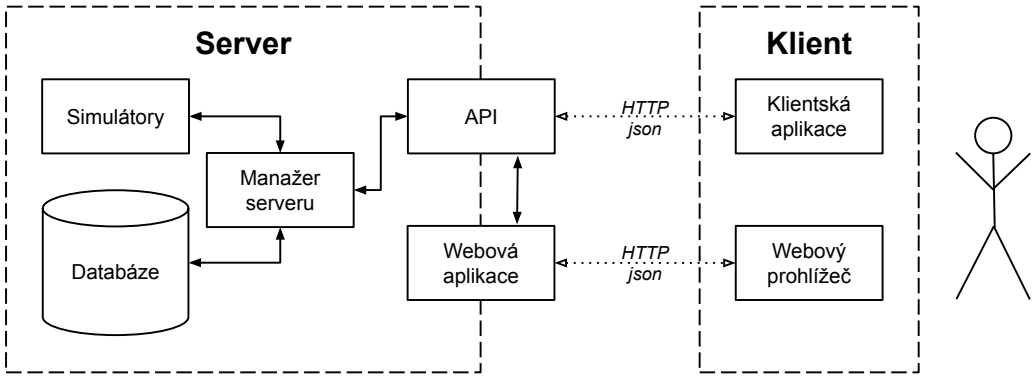
5. Výsledky a průběh simulací je možné zpětně zrekonstruovat, aby bylo možné vyhodnotit výkonnost hráčů.

6. Technologie, které lze použít pro implementaci klientů, jsou omezeny pouze komunikačním protokolem.

Pro účely práce bude implementován server umožňující simulaci her, autonomní klienti umělých inteligencí a klient pro vizualizaci hry s možností hry uživatele.

4.2 Návrh

Na základě specifikace požadavků na simulační prostředí byl proveden jeho návrh. Fungování simulačního prostředí znázorňuje schéma na obrázku 4.1.



Obrázek 4.1: Schéma simulačního prostředí.

Prostředí je rozděleno na serverovou a klientskou část. Klienti a server komunikují prostřednictvím HTTP protokolu s obsahem zpráv kódovaným pomocí JSON.

Server má hlavní řídící entitu *manažer serveru*. Manažer serveru zpracovává požadavky klientů předané pomocí aplikačního rozhraní. Podle druhu požadavku manažer komunikuje s dalšími částmi serveru.

Požadavky na provádění herních úkonů jsou dále předány na simulátory her. Každý simulátor má na starosti simulaci právě jedné hry. Životnost simulátoru je dána stavem hry. Po ukončení simulace hry již není třeba uchovávat simulátor v operační paměti. Konstrukci nových simulátorů nebo destrukci simulátorů ukončených her obstarává manažer serveru.

Zpětně rekonstruovatelná data, tedy provedené herní akce a výsledky her, jsou ukládány do databáze. Dotazy na ukládání dat do databáze a získávání dat z databáze provádí manažer serveru.

Jako nezávislá část serveru je webová aplikace, kterou může uživatel používat pomocí webového prohlížeče. Tato aplikace umožňuje provádění všech úkonů aplikačního rozhraní.

Chování klientských programů je dáno druhem klienta. Samostatné programy umělých inteligencí a program pro automatické simulování her fungují po zadání vstupu autonomně. Program pro hru uživatele je ovládán uživatelem interaktivně.

4.3 Technická dokumentace

4.3.1 Použité technologie

Všechny části simulačního prostředí byly implementovány v jazyce *C#*, na platformě *.NET Core*, za použitích standardních knihoven a *NuGet* balíčků.

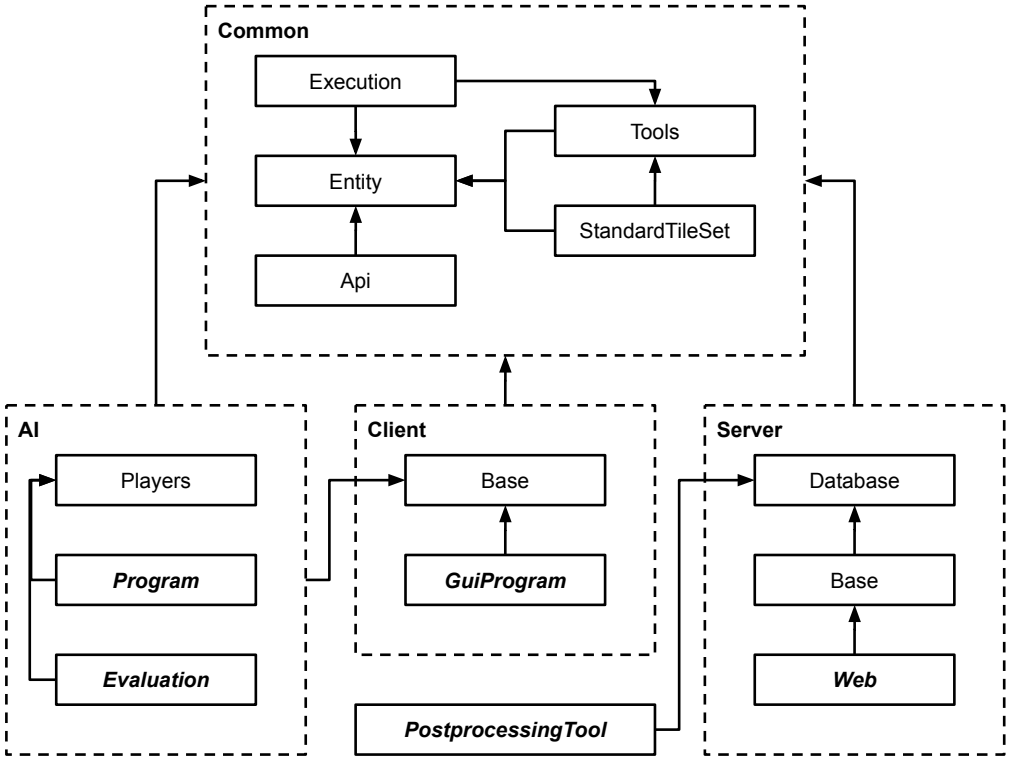
Aplikační rozhraní serverové části používá technologii *ASP.NET Core*. Pro komunikaci serveru s databází se používá *Microsoft Entity Framework Core*. Webová aplikace používá JavaScriptový framework *Angular* a knihovnu *Bootstrap* pro stylování webových stránek. Klient pro vizualizaci hry využívá technologii *Windows Presentation Foundation* (*WPF*) pro grafické uživatelské rozhraní. Databáze používá technologii *SQLite*.

Technologie byly vybrány podle osobních preferencí a cílily na použití v operačním systému Microsoft Windows 10, nicméně lze použít libovolný operační systém podporující vybrané technologie.

Pro vývoj bylo použito vývojové prostředí *Visual Studio 2019 Community*.

4.3.2 Rozdělení do knihoven a programů

Implementace byla rozdělena do knihoven dle schématu na obrázku 4.2.



Obrázek 4.2: Schéma řešení ve Visual Studio.

- Společné knihovny
  - `CarcassonneDiscovery.Entity.dll` obsahuje entity reprezentující herní prvky, které mají povahu datových objektů bez logiky manipulace.
  - `CarcassonneDiscovery.Tools.dll` obsahuje pomocné entity pro manipulaci s herními prvky jako prohledávání v mapě položených kartiček nebo práci se souřadnicemi a dále obsahuje „kanonickou“ implementaci schémat kartiček.
  - `CarcassonneDiscovery.Execution.dll` obsahuje logiku provádění herních akcí a rozhodování her.
  - `CarcassonneDiscovery.StandardTileSet.dll` obsahuje výčet standardní sady kartiček a logiku jejich losování.

  - `CarcassonneDiscovery.Api.dll` obsahuje implementaci datových objektů použitých v aplikačním rozhraní.

- Serverová část
  - `CarcassonneDiscovery.Server.Base.dll` obsahuje logiku simulačního
    prostředí poskytované na serverové části.
  - `CarcassonneDiscovery.Server.Database.dll` obsahuje databázové entity.
  - `CarcassonneDiscovary.Server.Web.exe` je program chovající se jako server, jehož součástí je i webová aplikace.

- Klientská uživatelská část
  - `CarcassonneDiscovery.Client.Base.dll` obsahuje logiku klientské části komunikace a rozhraní pro příjem vstupu od uživatele nebo umělé inteligence.
  - `CarcassonneDiscovary.Client.GuiProgram.exe` je program umožňující vizualizaci hry a hru uživatele prostřednictvím pomocí interaktivního grafického rozhraní.

- Klientská část pro umělé inteligence
  - `CarcassonneDiscovery.Ai.Players.dll` obsahuje entity reprezentující hráče umělé inteligence.
  - `CarcassonneDiscovary.Ai.Program.exe` obsahuje implementaci autonomního klientského programu pro jednoho hráče umělé inteligence.
  - `CarcassonneDiscovary.Ai.Evaluation.exe` obsahuje implementaci klientského programu organizující dávkové simulace her mezi umělými inteligencemi.

- Další části
  - `CarcassonneDiscovery.PostprocessingTool.exe` je pomocný program použitý při zpracování výsledků.

4.3.3 Implementace herních prvků

Knihovně `CarcassonneDiscovery.Entity.dll` obsahuje entity reprezentující herní prvky.

Herní prvek	Odpovídající entita
stav hry	`class GameState`
parametry hry	`class GameParams`
fáze hry	`enum GamePhase`
barva hráče	`enum PlayerColor`
schéma kartičky	`interface ITileScheme`
souřadnice kartičky	`struct Coords`
umístění kartičky na herní desce	`class TilePlacement`
rotace kartičky	`enum TileOrientaton`
typ regionu na kartičce	`enum RegionType`
umístění figurky na herní desce	`class FollowerPlacement`
losování kartiček	`interface ITileSupplier`

Tabulka 4.1: Přehled herních prvků a odpovídajících entit.

Města a regiony jsou ve schématu kartičky identifikovány svým pořadovým číslem. Sady kartiček a schémata kartiček v rámci sady jsou identifikovány prostřednictvím identifikátorů typu `string`. To umožňuje variabilitu při implementaci alternativních způsobů práce s kartičkami (např. pomocí identifikátoru můžeme popsat náhodně vygenerovanou kartičku atp.). Pro účely práce byla rozhraní `ITileScheme` a `ITileSupplier` implementována s ohledem na standardní sadu kartiček. Kanonickou implementací schématu kartičky je třída `TileScheme` v knihovně `CarcassonneDiscovery.Tools.dll`. Standardní sada kartiček je implementována v rámci knihovny `CarcassonneDiscovery.StandardTileSet.dll`.

Hráči jsou v rámci hry jednoznačně identifikováni pomocí svých barev. Jména hráčů pro účely zpracování výsledků jsou uložena do databáze, nijak se jméno hráče pro účely simulace nepoužívá.

4.3.4 Implementace rozhodování a řízení hry

Řízení průběhu hry je vykonáváno simulátory typu `GameSimulator`. Po zahájení hry je nutné umístit první kartičku a poté vylosovat kartičku pro dalšího hráče. Po každém vykonání herní akce je třeba provést změnu fáze hry. Poté, co hráč provede obě fáze tahu (položí kartičku a provede tah s figurkou nebo předá tah), je třeba vylosovat kartičku pro dalšího hráče. Po výběru všech kartiček je třeba vypočítat celkové skóre hry podle umístěných figurek.

Simulace hry může být v jednom ze tří stavů:

- Simulace je průběhu. V tom případě existuje instance simulátoru, která udržuje stav hry, provádí herní akce podle příkazů manažeru serveru.
- Simulace je ukončena po skončení hry. V tom případě je hra archivována – simulátor hry je destruován, záznam o hře je uložen v databázi. V případě požadavků na získání informací o proběhlé hře manažer vytvoří náhradu simulátoru `FinishedGameSimulator`, která poskytuje informace o proběhlé hře.
- Simulace byla ukončena, aniž by došlo k ukončení hry. V tom případě je hra označena za „mrtvou“, nelze pokračovat v její simulaci.

Samotné změny ve stavu hry provádí vykonavatel herních akcí implementován třídou `GameExecutor`. Pro každou herní akci (zahájení a ukončení hry, vylosování kartičky pro hráče a všechny akce prováděné hráči) existují metody dvou typů podle jejich významu.

- Metoda s předponou `Set` provede samotnou herní akci bez provedení kontroly pravidel. Tato metoda je použita po kontrole pravidel (následující metodou) a na klientech (kde nelze vybírat kartičky a tedy zcela kontrolovat pravidla).
- Metoda bez předpony `Set` provede kontrolu pravidel při provedení akce a v případě úspěchu provede změnu herního stavu (pomocí korespondující metody s předponou`Set`).

Pro účely umělých inteligencí je často vhodné umět herní akce vracet (aby neylo třeba vytvářet kopie herního stavu), k těmto účelům slouží pomocná entita pro vykonávaní opačných akcí `GameRevertExecutor`. Při umisťování kartiček, umisťování figurek a počítání skóre při odebírání figurek je třeba prohledávat v mapě umístěných kartiček. Prohledávání v mapě umístěných kartiček pomocí algoritmu DFS je implementováno ve třídě `GridSearch`.

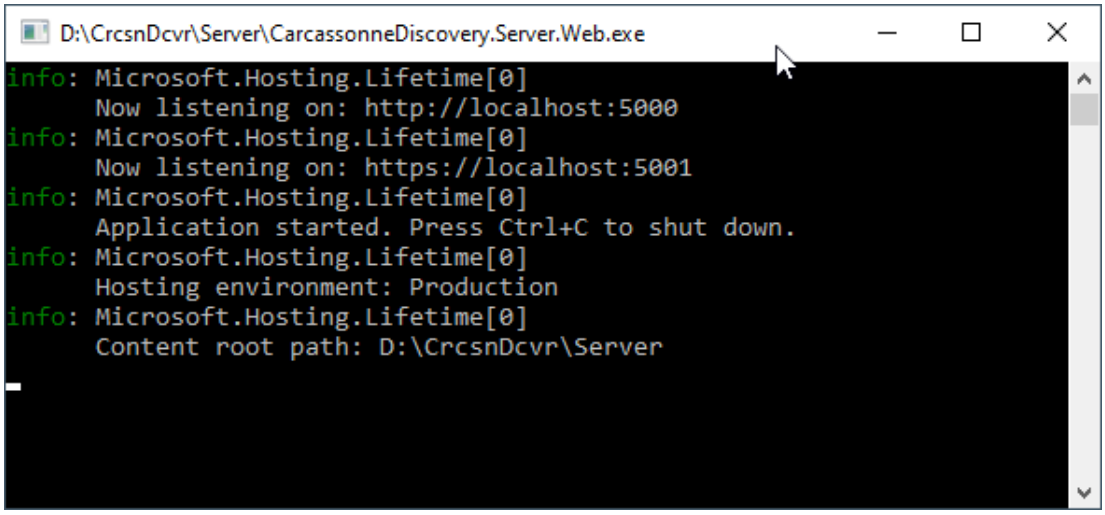
4.4 Uživatelská dokumentace

Před prvním spuštěním je vhodné ověřit, zda je nainstalovaná běhová podpora pro .NET Core 3.1 a ASP.NET Core 3.1. Odkazy pro stažení instalátorů běhové podpory jsou uvedeny v elektronické příloze této práce.

4.4.1 Ovládání serveru a webové aplikace

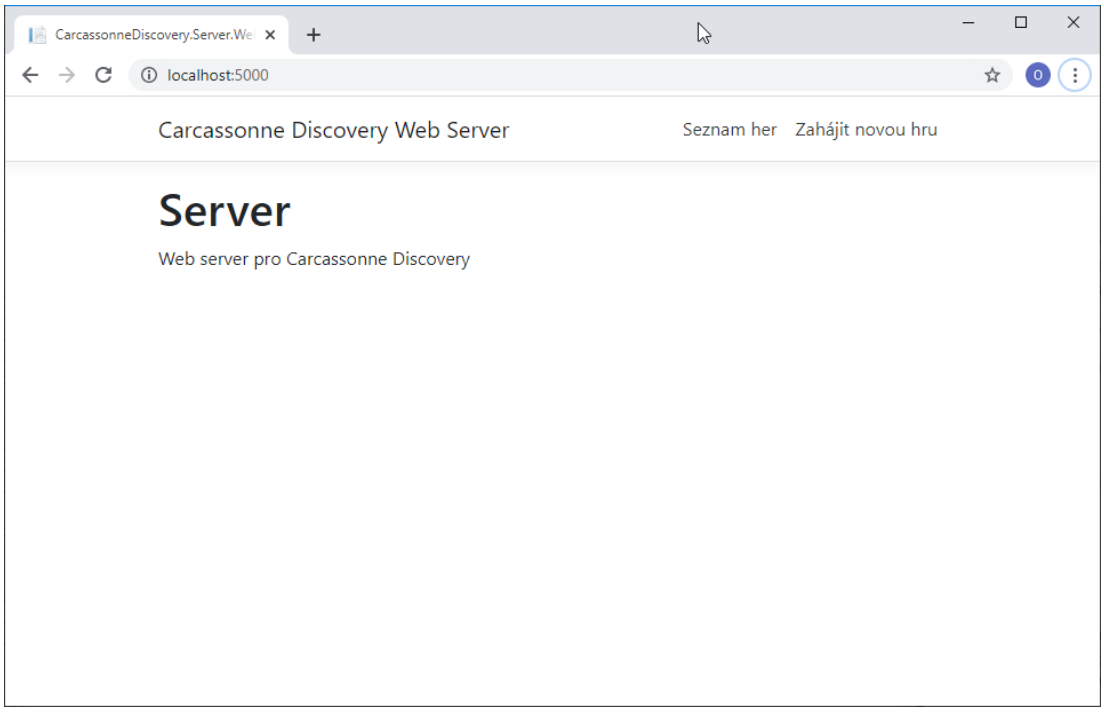
Pro běh serveru slouží aplikace `CarcassonneDiscovery.Server.Web.exe`. Po korektním spuštění se v příkazové řádce objeví se základní informace o serveru. Zavřením okna příkazové řádky (nebo klávesovou zkratkou Ctrl+C v příkazové řádce) se ukončí běh serveru.

Pokud není k dispozici běhové prostředí pro ASP.NET Core, není k dispozici port 5000 nebo dojde k jiné chybě, aplikace serveru se okamžitě ukončí.



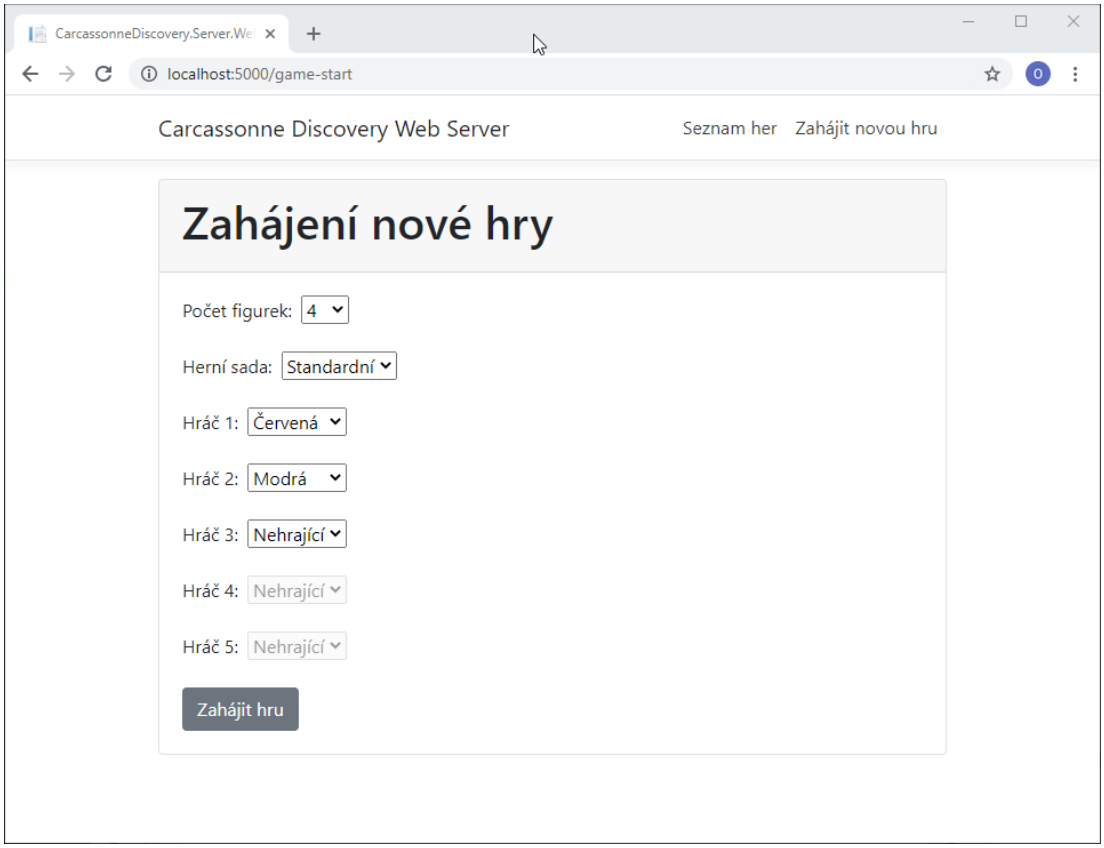
Obrázek 4.3: Okno serveru po spuštění.

Webová aplikace je k dispozici po spuštění serveru na adrese uvedené v základních informacích o serveru. Výchozí adresa je `http://localhost:5000`.



Obrázek 4.4: Úvodní stránka webové aplikace.

Uživatel má možnost prostřednictvím webové aplikace zahájit novou hru, zobrazit seznam her v průběhu a hrát hru pomocí přiloženého formuláře.



Obrázek 4.5: Formulář pro vytvoření nové hry.

4.4.2 Ovládání klientů umělých inteligencí

Program `CarcassonneDiscovery.Ai.Program.exe` slouží pro simulaci hry jedné umělé inteligence. Program přijímá argumenty příkazové řádky. Pokud nejsou zadány argumenty při spuštění programu, je uživatel dotázán na jednotlivé argumenty pomocí interaktivního vstupu. Pro různé hráče jsou vyžadovány různé parametry.

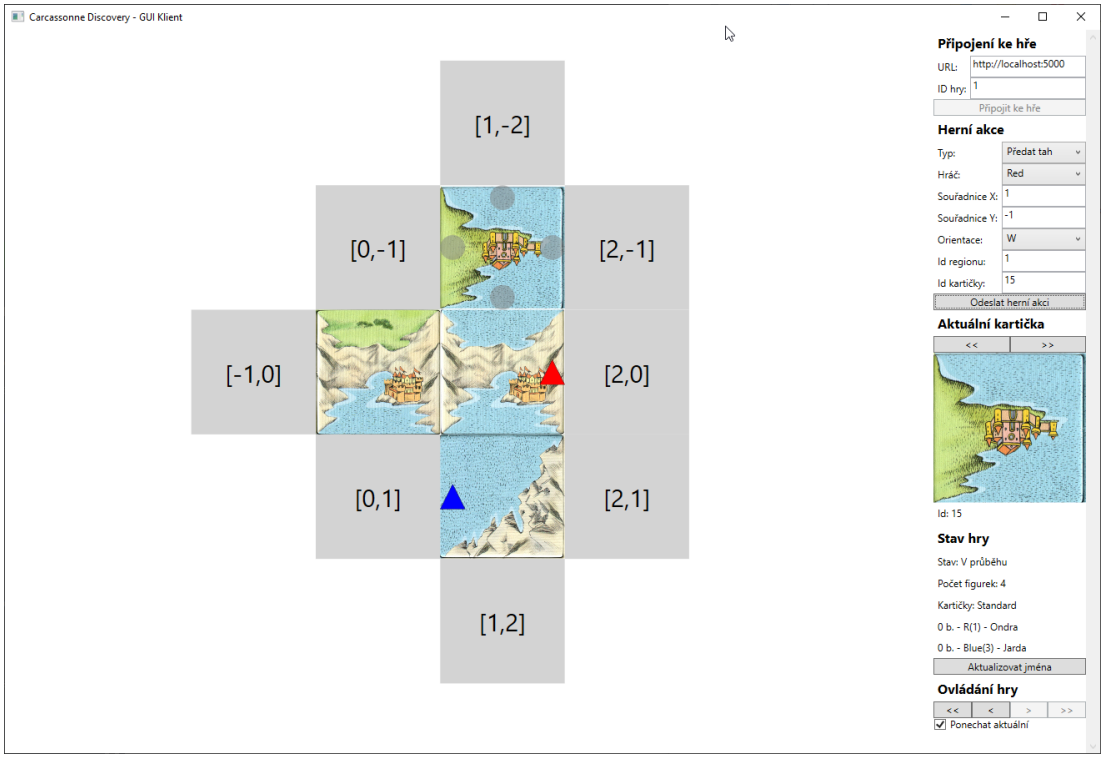
argument	rozsah hodnot	význam
baseUrl	libovolný	adresa serveru
aiKind	rnd, gr, pot, pl, ex, mc	druh umělé inteligence
playerColor	1-5, r, g, b, y, k	barva hráče
gameId	kladné celé číslo	identifikátor hry
savePath	libovolný	cesta k souboru pro uložení koeficientů učícího se hráče
loadPath	libovolný	cesta k souboru pro načtení koeficientů učícího se hráče
depth	kladné celé číslo	hloubka prohledávání Expectiminimaxového hráče
simulationCount	kladné celé číslo	počet simulací MC hráče
gamma	desetinné číslo	koeficient znevýhodnění MC hráče
gammaLimit	desetinné číslo	hranice významnosti u MC hráče

Tabulka 4.2: Argumenty klienta pro hru umělé inteligence.

Program `CarcassonneDiscovery.Ai.Evaluation` slouží pro dávkovou simulaci her umělých inteligencí. Program při spuštění očekává argument reprezentující cestu ke konfiguračnímu souboru, který obsahuje parametry simulací. Příklady konfiguračních souborů jsou v elektronické příloze této práce.

4.4.3 Ovládání klienta pro hru uživatele

Program `CarcassonneDiscovery.Client.GuiProgram` slouží pro visualizaci hry a hru lidských hráčů. Hlavní okno programu se skládá ze dvou částí, v levé části se nachází herní deska, v pravé části se nachází ovládací panel.



Obrázek 4.6: Program pro hru lidského hráče.

Uživatel se připojí k běžící hře pomocí formuláře *Připojení ke hře*, kde je třeba zadat adresu serveru a identifikátor hry. Po připojení se v části s herní deskou zobrazí již položené kartičky.

Uživatel může provádět v každý okamžik libovolné herní úkony, nezáleží na tom, jakého hráče reprezentuje. Každý herní úkon se provádí pomocí formuláře *Herní akce*. Po vyplnění jednotlivých položek a stisknutí tlačítka *Odeslat herní akci* se na server odešle požadavek na provedení akce. Pro pohodlnější ovládání hry je herní deska interaktivní. Na herní desce lze provádět následující úkony:

- Po kliknutí na šedou oblast se souřadnicemi se formulář vyplní tak, aby reprezentoval akci položení kartičky na danou souřadnici. Orientace kartičky lze změnit pomocí tlačítek v části *Aktuální kartička*.
- Po kliknutí na libovolnou figurku se formulář vyplní tak, aby reprezentoval akci odebrání dané figurky.
- V druhé části tahu po kliknutí na šedé kolečko na naposledy položené kartičce se formulář vyplní tak, aby reprezentoval akci položení figurky na daný region.
- Akce předání tahu bez položení nebo odebrání figurky se provádí automaticky po položení kartičky. Pokud mezitím uživatel vybere jinou akci, musí vybrat předání tahu ve formuláři ručně.
- Pomocí stisknutí pravého tlačítka myši lze pohybovat s herní deskou. Pomocí kolečka myši lze herní deska přibližovat a oddalovat.

Pomocné nástroje slouží pouze pro usnadnění ovládání, vždy jsou závazné hodnoty ve formuláři.

Pro zpětné přehrávání tahů slouží tlačítka v části *Ovládání hry*. Uživatel se může přesunout na první, předchozí, následující nebo poslední provedený tah. Navíc je možné zapnout volnu *Ponechat aktuální*, pomocí které se bude automaticky zobrazovat další provedené akce.

V části *Stav hry* se zobrazují parametry hry, jména hráčů a jejich aktuální skóre.

Závěr

V rámci této bakalářské práce jsme navrhli několik různých metod pro vytvoření umělé inteligence pro hru Carcassonne – Objevitelé. Tyto metody jsme implementovali a experimentálně vyhodnotili pomocí simulací her umělých hráčů. Zjistili jsme, které metody mají výhody v různých aspektech hry.

V rámci simulačního prostředí jsme implementovali server pro simulaci her, klienta pro samostatnou hru umělé inteligence, program pro automatickou dávkovou simulaci her programy pro samostatnou hru umělé inteligence také implementovali program pro vizualizaci hry s možností hry lidských hráčů.

Po vyhodnocení výsledků jsem měl možnost hrát s nejlepšími umělými inteligencemi. Vytvořené umělé inteligence jsou více než důstojnými soupeři. Žádná umělá inteligence se sebelepšími výsledky však nemůže nahradit lidskost, která je daná hrou živých bytostí. Atmosféra při hře přátel či členů rodiny je nenahraditelná.

Seznam použité literatury

Korf, R. E. (1991). Multi-player alpha-beta pruning. *Artif. Intell.*, **48**, 99–111.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill international editions - computer science series. McGraw-Hill Education. ISBN 0070428077.

Nekvinda, M. (2018). *Umělá inteligence a herní strategie v deskové hře Carcassonne*. Vedoucí práce RNDr. Tomáš Holan. Univerzita Karlova.

Russell, S. a Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition. ISBN 0136042597.

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

Seznam obrázků

Seznam tabulek

Seznam pseudokódů

A. Obsah elektronické přílohy

- Adresář `Ilustrace` obsahuje obrázky použité v této práci a zdrojové soubory pro jejich vytvoření.
- Adresář `SestaveneBinarniSoubory` obsahuje veškeré implementované programy a knihovny.
- Adresář `ZdrojoveSoubory` obsahuje zdrojové kódy a projekt pro Visual Studio.
- Adresář `ZpracovaniVysledku` obsahuje databáze s výsledky her a R skripty použité pro jejich zpracování.
- Soubor `DavkoveSimulace.cfg` obsahuje příklad konfiguračního souboru pro dávkovou simulaci her umělých inteligencí.
- Soubor `OdkazyProBehoveProstredi.txt` obsahuje odkazy na instalátory pro běhové prostředí .NET Core nutné pro spuštění programů.
- Soubor `PravidlaCZ.pdf` obsahuje pravidla hry Carcassonne – Objevitelé v českém jazyce.