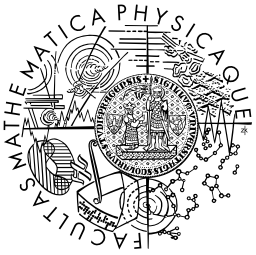


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Michal Nekvinda

Umělá inteligence a herní strategie v deskové hře Carcassonne

Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Holan, Ph.D.
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2018

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

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

V ..... dne ..... Podpis autora

Rád bych poděkoval RNDr. Tomáši Holanovi, Ph.D. za cenné rady, věcné připomínky a čas, který mi během psaní práce věnoval. Dále děkuji své rodině za podporu při studiu.

Název práce: Umělá inteligence a herní strategie v deskové hře Carcassonne

Autor: Michal Nekvinda

Katedra: Katedra softwaru a výuky informatiky

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

Abstrakt: Bakalářská práce se zabývá problematikou vývoje umělé inteligence pro deskovou hru Carcassonne ve variantě pro dva až pět hráčů. Jsou zde představeny různé možnosti přístupů k vytvoření herní strategie. Vybrané z nich jsou implementovány pro testování. Největší prostor je věnován vývoji pomocí genetických algoritmů. Práce uvádí detailní výsledky porovnání mezi všemi implementovanými inteligencemi. Navíc se zaměřuje i na vysvětlení způsobu rozhodování té nejúspěšnější v rámci testů. Nejpokročilejší umělé inteligence jsou schopné porazit i lidské hráče.
Dále vznikla implementace herního prostředí pro tuto hru. K dispozici je varianta s grafickým uživatelským rozhraním, která nabízí možnost hry člověka proti počítači a jednoduchá konzolová aplikace pro provádění testů.

Klíčová slova: Carcassonne, hra, umělá inteligence, genetický algoritmus

Title: Artificial intelligence and game strategy in Carcassonne board game

Author: Michal Nekvinda

Department: Department of Software and Computer Science Education

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

Abstract: The bachelor paper deals with the development of artificial intelligence for the Carcassone board game for two to five players. Different approaches to creating a suitable game strategy are presented. Then, selected strategies are tested. Most of the paper is dedicated to the development by means of genetic algorithms. The paper presents detailed results of the comparison of all implemented artificial intelligences. In addition, it explains how the most successful decision-making method works. The most advanced artificial intelligences are capable of defeating human players.
Additionally, the game environment for this game was implemented. It provides a graphical user interface application that offers a human-to-computer gameplay and a simple console application to perform the artificial intelligence performance tests.

Keywords: Carcassonne, game, artificial intelligence, genetic algorithm

Obsah

Úvod

Carcassonne je taktická společenská stolní hra pro dva až pět hráčů. Jejím autorem je Klaus-Jürgen Wrede. Hra byla vydána v roce 2001 a během následujících let pak vzniklo množství nejrůznějších rozšíření. Díky své popularitě se Carcassonne dočkalo i přesunu na různé herní platformy.

Hráči se střídají v pokládání karet s motivy krajiny a vytvářejí tak vlastní podobu středověkého města. Na kartičky mohou umísťovat figurky, za které získávají body. Vyhrává hráč, který má po vyložení všech kartiček na svém kontě nejvíce bodů.

Cílem bakalářské práce je vytvoření umělé inteligence pro hru Carcassonne a implementace jejího herního prostředí. To bude nabízet možnost hraní dvou až pěti hráčů na jednom počítači. Libovolný počet těchto hráčů bude možné nahradit umělou inteligencí. Softwarová implementace hry se omezí pouze na základní variantu hry Carcassonne bez rozšíření.

Práce se zabývá různými možnostmi, jak vytvořit vhodnou umělou inteligenci. Důraz je kladen především na sestavení komplexní ohodnocující funkce, která dokáže pomocí předem definovaných pravidel vybrat v každé pozici co nejvýhodnější tah. Optimální nastavení vah pro jednotlivá pravidla se hledá pomocí genetického (evolučního) algoritmu. Díky tomu pak nalezené řešení poskytuje přehled o důležitosti různých aspektů hry a dává lidskému hráči informaci, jaká je optimální strategie z pohledu počítače.

Na úvod 1 se zmíníme o drobné změně pravidel, kterou jsme v práci uvažovali a definujeme některé nově zavedené pojmy (kapitola 2). V kapitole 3 této práce nejprve projdeme různé návrhy na fungování umělé inteligence a představíme ty, které jsme implementovali. Následně provedeme první experiment pro nalezení vhodné umělé inteligence (kapitola 4).

V další části práce se budeme věnovat vývoji pomocí genetického algoritmu. Proto se zaměříme na jeho specifika a parametry (kapitola 5). Následují výsledky testů zachycující zlepšování se vyvíjené inteligence (kapitola 6) a porovnání všech umělých inteligencí v kapitole 7. Nejlepší vytvořenou strategii rozebereme v kapitole 8. Vliv vybraných modifikací pravidel zmiňujeme v kapitole 9. Kapitola 10 obsahuje uživatelskou dokumentaci s popisem ovládání programu. V kapitole 11 najdeme programátorskou dokumentaci.

Jak již bylo řečeno, hra Carcassonne se dočkala vydání napříč herními platformami. Níže je uveden stručný přehled existujících programů, umožňujících hru proti umělé inteligenci.

- **Verze pro PC**
  - *Carcassonne – Tiles & Tactics* [5] – komerční software pro Windows, vydávaný společnost Asmodee Digital.
  - *JCloisterZone* [6] – program ke stažení zdarma, pro Windows, Linux i Mac OS.

  - *Carcassonne – Last Stend* [7] - možné hrát zdarma na internetu, jedná se o mírně upravenou verzi hry. Je pouze pro dva až čtyři hráče a má jiné počítání bodů.

- **Verze pro Xbox**
  - *Carcassonne* [8] – komerční software od společnosti Sierra Online, s možností dokoupení dalších rozšíření.

- **Verze pro Android**
  - *Carcassonne: Official Board Game – Tiles & Tactics* [9] – komerční software od společnosti Asmodee Digital.

Popisem a programováním umělé inteligence se pak zabývá například diplomová práce C. Heyden [10]. V této práci se autorka zaměřuje především na použití algoritmů Monte Carlo Tree Search a expectimax.

1. Úprava pravidel

Podrobná pravidla hry Carcassonne jsou k dispozici na webových stránkách vydavatele [2]. Uvedená pravidla pochází z doby vydání první verze této hry. Během let pak došlo k jedné změně při počítání bodů, kterou v práci používáme. Uzavřené město tvořené dvěma kartičkami bylo podle uvedených pravidel ohodnoceno dvěma body. V novějších verzích se přešlo ke čtyřbodovému ohodnocení.

Zároveň ale neexistují novější pravidla, která by byla vytvořena pouze pro základní variantu hry se 72 kartičkami. Z marketingových důvodů se nyní původní verze prodává vždy s dalšími mini rozšířeními zdarma.

V souladu s novými pravidly a názorem autora hry [11], budeme v této práci uzavřené město tvořené dvěma kartičkami počítat za čtyři body. O dvoubodové variantě a rozdílech, které přináší, se zmíníme v kapitole 9.

2. Základní pojmy

Pro lepší porozumění a pochopení textu zavádíme seznam pojmů, které v práci budeme používat v souvislosti s hrou Carcassonne. Definované pojmy rozšiřují terminologii uvedenou v oficiálních pravidlech hry.

Definice 1 (herní objekt)**.** *Pro slova město, cesta, louka, klášter používáme slovo nadřazené* (herní) objekt*.*

Definice 2 (tah)**.** Tah *je dvojice (umístění kartičky, umístění figurky).* Umístění kartičky *udává souřadnice na hrací ploše a rotaci kartičky.* Umístění figurky *udává pozici figurky na kartičce. Umístění figurky může být vynecháno (označeno jako null), pokud pokládáme pouze kartičku bez figurky.*

Definice 3 (tah s figurkou)**.** *Pojmem* tah s figurkou *rozumíme* tah*, jehož* umístění figurky *není vynecháno. V opačném případě se jedná o* tah bez figurky*.*

Definice 4 (majoritní objekt)**.** *Řekneme, že herní objekt je pro hráče H* majoritní*, pokud má v objektu položenou alespoň jednu figurku a zároveň žádný jiný hráč nemá v objektu více figurek než hráč H.*

Definice 5 (výhra)**.** *Řekneme, že hráč H* vyhrál *hru h, pokud po skončení hry pro všechny hráče platí, že jejich bodový zisk je menší, nebo roven bodovému zisku hráče H.*

3. Analýza

V této kapitole analyzujeme možné přístupy k vytvoření umělé inteligence a vybíráme několik z nich pro implementaci a další zkoumání.

3.1 Náhodný přístup

Nejjednodušší přístup k tvorbě umělé inteligence je vybírat tahy čistě náhodně. Přínos takto se rozhodující umělé inteligence spočívá zejména v tom, že slouží jako jednoduchý ukazatel výkonnosti pro sofistikovanější strategie.

Implementace tohoto přístupu je přímočará. Potřebujeme pouze umět najít seznam všech tahů, které je možné v dané pozici zahrát. Z tohoto seznamu pak na základě náhody zvolíme příslušný tah. Strategie je v práci implementována pod názvem *Nahodna UI*.

Cílem má být přiblížení se chování lidského hráče, který vždy položí kartičku na první místo, které ho napadne (a zároveň tím neporušuje pravidla hry). Následně pak buď položí, nebo nepoloží figurku.

K dosažení výše zmíněného cíle drobně upravíme náhodné vybrání ze seznamu tahů. Je-li vybrán tah s figurkou, pak s pravděpodobností 50 % figurku odstraníme. Tím dojde k přesnější simulaci toho, co chápeme jako náhodné rozhodování.

3.2 Hladový přístup

Výkonnost a úspěšnost herní strategie můžeme dobře posuzovat pomocí počtu získaných bodů vůči ostatním hráčům. Snaha mít na konci co nejvyšší skóre vede přirozeně k nápadu získávat co nejvíce bodů při každé vhodné příležitosti. Dostáváme se tak k myšlence použití hladového algoritmu pro vyřešení problému. K realizaci opět potřebujeme seznam všech dostupných tahů a také schopnost umět spočítat, kolik bodů nám zahrání určitého tahu přinese.

Měřítko počtu bodů (těch, co se objeví na herním počítadle bodů) se zde ale jeví jako příliš hrubé. Snadno se nám může stát, že v určité pozici žádný tah nevede ke zvýšení bodového zisku. Neznamená to ovšem, že by v tom případě byly všechny tahy rovnocenné. Kromě ihned započítatelných bodů se nám na hrací ploše vytváří tzv. dočasné body. Ty se mohou s vyšší či nižší pravděpodobností proměnit v body počítané do skóre.

Jako typický příklad uveďme založení nového města. Toto město se dále může vyvíjet podle tří následujících scénářů.

První možnost: město se podaří zavřít. Za každou kartičku tvořící město se nám tedy započítají dva body.

Druhá možnost: město se až do konce hry zavřít nepodaří, ale i na konci zůstaneme majoritním vlastníkem města. Pak za každou kartičku tvořící město dostaneme jeden bod.

Třetí možnost: během hry ztratíme majoritní podíl ve městě. Potom žádné body nedostaneme.

Předpoklad bodového zisku při obsazení nějakého objektu je proto složité odhadnout. Zkusíme tedy metodu vyhodnocení „právě teď“. To znamená, že uvažujeme počty bodů, které by se započítaly, pokud by okamžitě po našem tahu hra skončila.

Implementace hladového algoritmu projde seznam všech tahů, zjistí pro každý stav body v pozici a vybere tah, který maximalizuje počet bodů proti ostatním hráčům. Strategie má v práci jméno *Hladova UI*.

3.3 Optimalizace funkce dvou proměnných

Dá se snadno nahlédnout, že použití hladového algoritmu nemůže obecně dávat optimální řešení. Při maximalizaci tzv. dočasných bodů popsaných u hladového přístupu preferujeme v drtivé většině tahy s figurkou. Pokud nám figurky dojdou, ztrácíme možnost hrát kvalitní tahy. Projeví se zde slabina hladových algoritmů, které nemyslí na budoucnost.

Důvodem k nezahraní tahu s figurkou je pouze fakt, že předpokládáme její lepší využití v některém z dalších tahů. Snažíme se najít rovnováhu mezi počtem bodů, které za tah získáme a počtem figurek, které si chceme ponechat pro budoucí použití. Maximalizujme tedy funkci se dvěma parametry. Jeden parametr udává aktuální zisk, druhý vyjadřuje potenciál do budoucích tahů. Aktuální zisk se dá počítat stejně jako v hladovém přístupu, potenciál do dalších tahů je asi nejjednodušeji možné vyjádřit poměrem počtu volných figurek ku počtu zbývajících tahů.

Optimální poměr můžeme hledat hrubou silou. Provedený experiment popisujeme v kapitole 4. Jeho výsledkem je umělá inteligence pojmenovaná v práci jako *Optimalizacni UI*.

3.4 Ohodnocující funkce

Zobecněním předchozí myšlenky je vytvoření funkce, která na základě nejrůznějších kritérií umí ohodnotit hru v nějakém stavu. Různými tahy se pak mění charakter herní pozice, a tudíž i její ohodnocení. Projitím všech možných tahů v dané pozici, pak najdeme ten, ve kterém je hodnota funkce nejvyšší a zahrajeme ho.

Věnujme se nyní tvorbě ohodnocující funkce. Její hodnotu v určitém okamžiku můžeme počítat jako součet různých dílčích částí, kde každá z nich si všímá nějakého zajímavého rysu dané pozice. Jednotlivé části si označme jako pravidla. Hodnoty pravidel uvažujeme především kvantitativní, ale můžeme mít i kvalitativní, pokud jsme schopni porovnat je od nejlepšího po nejhorší. V této práci umělá inteligence používá celkem devatenáct různých pravidel. Jejich výčet a stručný popis ukazuje tabulka 3.1. Funkce je vždy počítána z pohledu konkrétního hráče.

Je zřejmé, že každé z pravidel má pro ohodnocení celkové kvality pozice různou váhu. Výkonnost umělé inteligence pak zcela závisí na správném nastavení poměru vah.

Protože není z časových důvodů možné zkoušet všechny kombinace vah (těch je nekonečně mnoho), nabízí se cesta hledat je pomocí genetického algoritmu. Na

začátku vybereme váhy náhodně a pak se s použitím evoluce budeme snažit vyvinout co nejlepší hodnoty vah pro zmíněná pravidla. O genetických algoritmech pojednává například kniha Russella a Norviga [1] (v části II, kapitole 4.1.4).

Číslo pravidla	**Popis pravidla**
1	Body za uzavíraná města.
2	Body za uzavírané cesty.
3	Body za kláštery.
4	Body za rozestavěná města.
5	Body za rozestavěné cesty.
6	Body za louky.
7	Kolik zbývá hráči figurek vůči počtu předpokládaných tahů, které bude hrát.
8	Zda hráč provedl tah s figurkou (tedy je buď 0, nebo -1, přičemž je-li více volných figurek, než tahů do konce, pak vrací 0).
9	Počet figurek, které byly právě vráceny soupeřům.
10	Návratnost jednotlivých figurek kromě sedláků. Návratností je nekladné číslo, které označuje minimální počet kartiček, který je nutný k uzavření objektu.
11	Počet majoritních měst.
12	Počet majoritních cest.
13	Počet majoritních klášterů.
14	Počet majoritních luk.
15	Počet volných figurek.
16	Kvalita měst hráče – poměr velikosti města ku uzavřenosti.
17	Návratnost položených sedláků. Počítáme pomocí vzorce $-1 \times (počet\_sedláků/počet\_tahů\_do\_konce)$.
18	Počet figurek, které hráč získá po tahu zpět.
19	Kvalita rozestavěných hradů na majoritních loukách.

Tabulka 3.1: Seznam ohodnocujících pravidel.

3.4.1 Různé fáze hry

Ohodnocující funkce počítá kvalitu tahů stejně během celé hry. Nabízí se ale otázka, zda se váhy jednotlivých pravidel nemohou měnit v průběhu hry. S blížícím se koncem bude pravděpodobně docházet k růstu preferencí některých pravidel a naopak poklesu preferencí pravidel jiných.

Uvažujme tedy drobné vylepšení a rozdělme hru do několika fází. V každé této fázi by pak umělá inteligence uplatňovala jiné váhy pro ohodnocení pravidel. Například rozdělení hry na dvě fáze by znamenalo, že po odehrání přibližně poloviny tahů (tj. $72/2 = 36$) se umělá inteligence může začít chovat jinak a upřednostňovat jiné tahy.

Obecná implementace této myšlenky je implementována pod názvem *UIkFazi*. Je nutné do ní dodat údaj o počtu fází a hodnoty všech vah. Používá se při vývoji v genetickém algoritmu. Její potomek využívající váhy, které byly nalezeny v evoluci, je označován jako *UI 19/3*.

3.4.2 Rozhodování na základě pravděpodobnosti

Naše ohodnocující funkce umí posuzovat pozici na základě aktuálního stavu. Zatím ale nevyužívá znalosti ještě neodehraných kartiček. Tato informace může být důležitá ze dvou důvodů. Za prvé můžeme ze zbývajících kartiček spočítat pravděpodobnost s jakou půjde určitý objekt rozšířit (přidáním těchto kartiček). Za druhé se podle toho, jak dobře (a zda vůbec) lze objekt rozšířit, dá ocenit jeho kvalita – čím snadněji se může objekt rozrůst, tím více bodů nám přinese a pravděpodobně půjde uzavřít snadněji než objekt, ke kterému lze přidat jen velmi málo doposud neodehraných kartiček.

Navrhovaná úprava najde uplatnění pro pravidla číslo 10 až 13 z tabulky 3.1. U pravidel číslo 11 až 13 vynásobíme počet majoritních objektů jejich výše zmíněnou kvalitou. U pravidla číslo 10 pak budeme předpokládat, že stojí-li figurka v objektu s vyšší šancí na rozšíření, má také větší pravděpodobnost vrátit se svému majiteli zpět. V tomto případě se ale jedná spíše o heuristiku.

Obecná implementace má označení *UIkFaziPr*. Používá se opět jen v genetickém algoritmu. Umělé inteligence využívající konkrétní hodnoty z evoluce jsou implementovány pod názvy *UI 19/3 pr* a *UI 19/5 pr*.

3.5 Deterministický rozhodovací strom

Zajímavou myšlenkou je vymyslet strategii, která vybírá své tahy na základě nějakého rozhodovacího stromu. Ukázalo se ale, že výběr vhodného tahu mnohem více závisí na stavu herní plochy, než na kartičce, kterou otočíme. Zároveň při splnění nějaké podmínky nemáme k dispozici jiné akce než *polož kartičku* a *polož figurku na kartičku*.

Tato strategie proto není v práci implementována.

3.6 Monte Carlo Tree Search a algoritmus expectimax

Tvorbu umělé inteligence za použití heuristických prohledávacích algoritmů a algoritmu expectimax popisuje práce C. Heyden [10].

4. Optimalizace funkce dvou proměnných – testy

Nalezení vhodného poměru mezi aktuálním ziskem a budoucím potenciálem pro umělou inteligenci *Optimalizacni UI* provedeme experimentálně pomocí počítače.

Ohodnocení o tahu t spočítáme dle vzorce:

$$o(t) = body\_nyní + k * potenciál^{exp}. \tag{4.1}$$

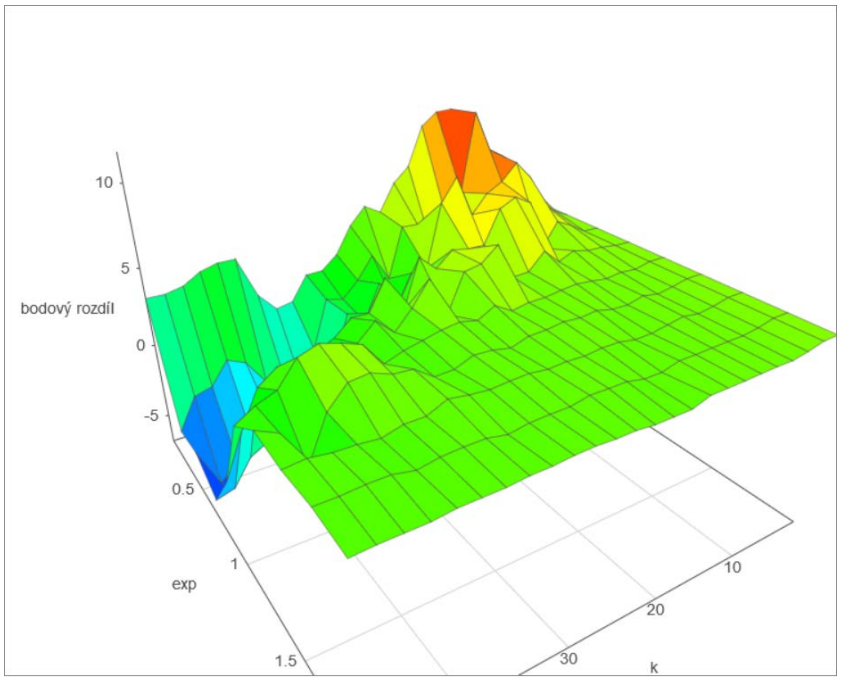
Proměnná *body_nyní* odpovídá aktuálnímu zisku z hladového přístupu, proměnná *potenciál* značí poměr volných figurek hráče ku počtu tahů, které zbývají do konce. My hledáme hodnoty proměnných k a exp tak, aby umělá inteligence hrála co nejlépe. Jinými slovy hledáme vhodné koeficienty, které určí důležitost potenciálu ve vzorci 4.1.

Testujeme pro hodnoty proměnných:

- $k = 1, 3, 5, ..., 49$,
- $exp = 0{,}1; 0{,}3; 0{,}5; ...; 1{,}9$.

Pro každou kombinaci vytvoříme umělou inteligenci, která s těmito hodnotami sehraje sto her proti hráči *Hladova UI* a spočítá průměrný bodový rozdíl za jednu hru.

Výsledek testování shrnuje graf na obrázku 4.1 nakreslený v softwaru pro tvorbu 3D grafů [12]. Nejlepšího výsledku bylo dosaženo s hodnotami $k = 11$ a $exp = 0{,}3$. Toto nastavení tedy používá umělá inteligence *Optimalizacni UI*.



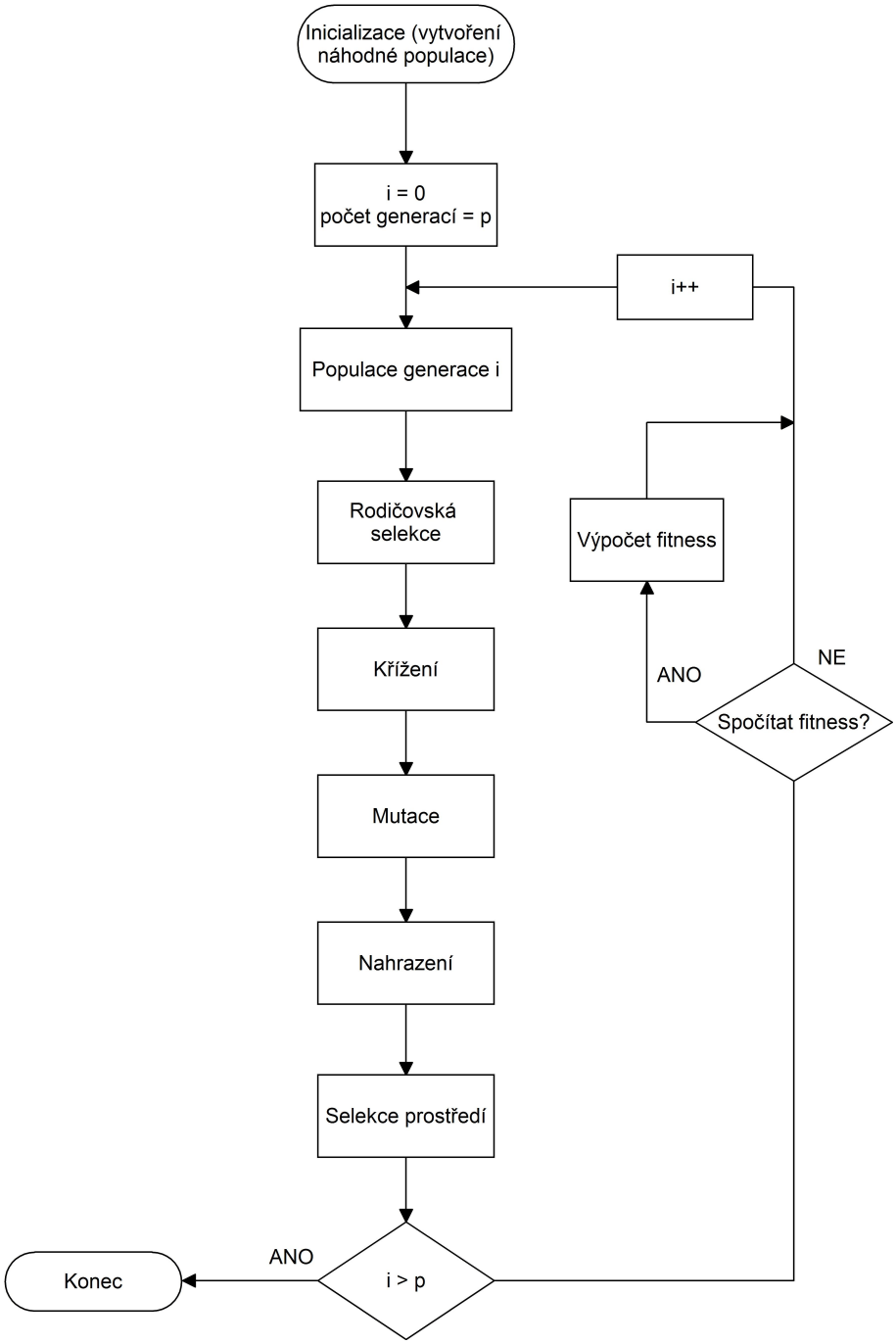
Obrázek 4.1: Hledání maxima parametrů k, exp.

5. Genetický algoritmus

Pro nalezení co nejlepších vah ohodnocující funkce použijeme genetický algoritmus. Dále rozepíšeme jeho průběh, jedince, druhy použitých operátorů a způsob počítání fitness funkce.

5.1 Průběh genetického algoritmu

Vývin populace probíhá dle schématu, které je znázorněno na obrázku 5.1.



Obrázek 5.1: Průběh genetického algoritmu.

5.2 Jedinec

Jedince kódujeme vektorem reálných čísel. Délka vektoru je počet pravidel krát počet fází hry. Povolené hodnoty, které může jedinec obsahovat, jsme nastavili v intervalu $\langle -2, 20 \rangle$. Očekávání je takové, že většina genů bude kladná, proto máme interval posunutý směrem do kladných hodnot. Díky použití desetinných čísel se nemusíme obávat, že by meze intervalu byly příliš malé.

5.3 Selekce

Genetický algoritmus rozlišuje dva druhy selekce.

- **Rodičovská selekce** – z populace s j jedinci vybírá j jedinců pro křížení.
- **Selekce prostředí** – vybere j jedinců do $i + 1$-ní generace. Počet jedinců, ze kterých vybírá, závisí na druhu zvoleného nahrazení.

V práci používáme při selekci tyto konkrétní metody selekce.

- **Turnajová selekce** – princip námi použité turnajové selekce je následující. Nechť vyvíjíme populaci pro hru Carcassonne s n hráči. Potom z populace náhodně vybereme n jedinců. Ty použijeme jako váhy ve zvolených umělých inteligencích. Poté s nimi sehrajeme předem určený počet her m. Vždy dodržujeme, že n dělí m. Každý počítačem řízený hráč pak sehraje $\frac{m}{n}$ her jako n-tý v pořadí. Tím je zajištěna větší objektivita při posouzení kvality jedinců. Během všech m her sčítáme body, kterých umělé inteligence dosáhly. Jedinec, který dodal váhy do nejlepší z nich, je označen a vybrán jako rodič ke křížení. Pro populaci j jedinců provádíme výběr j krát. Turnajovou selekci je možné použít jako rodičovskou i jako selekci prostředí.

  Turnajová selekce se nám velmi hodí, protože nepracuje s fitness funkcí, jejíž výpočet je komplikovaný. Při její implementaci také bývá zvykem s určitou pravděpodobností (deset až dvacet procent) vybrat horšího jedince z turnaje. My tuto možnost nahrazujeme tím, že s jedinci sehrajeme pouze omezené množství her. S určitou pravděpodobností, která se zmenšuje s počtem her, ale naopak zvětšuje s klesajícím rozdílem mezi porovnávanými jedinci, tak může dojít k výběru jedince, který by byl při dlouhodobém testování posouzen jako horší.

- **Diferenciální selekce** – používáme ji jako selekci prostředí, navíc pouze v případě, kdy jsme použili slučovací nahrazení (viz sekce 5.6). Do jisté míry slouží jako náhrada pro zachování elit. Nejprve vybere jednoho jedince z j-té generace a jednoho nově vytvořeného jedince (vzniklého z křížení a upraveného mutací). Každý člen populace je vybrán právě jednou. Vyvíjíme-li populaci pro hru Carcassonne s n hráči, je k dvěma vybraným jedincům přidáno $n-2$ náhodných. Dále postupujeme stejně jako u turnajové selekce. Na konci vybereme lepšího ze dvou vybraných jedinců do populace generace $j + 1$.

5.4 Křížení

Při hledání optimálních vah jsme použili vždy jedno ze dvou níže uvedených křížení.

- **Uniformní křížení** – představuje křížení po jednotlivých genech. Nově vytvořený jedinec má vždy dva rodiče. V jeho genomu se na libovolné pozici nachází s pravděpodobností 50 % gen prvního rodiče a s pravděpodobností 50 % gen druhého rodiče.
- **Speciální n-bodové křížení** – dá se použít pro jedince, kteří kódují strategii používající více herních fází. Pro jedince s délkou $p \times f$, kde p označuje počet pravidel a f značí počet fází, se jedná o $f - 1$ bodové křížení v místech na hranicích jednotlivých fází. Potomek tedy dostane pro k fází hry strategii od prvního rodiče a $f - k$ fází hry bude hrát podle svého druhého rodiče.

5.5 Mutace

V našem genetickém algoritmu pracujeme se dvěma různými mutacemi.

- **Klasická mutace** – klasická mutace prochází geny jedince a některé z nich náhodně změní. Protože měníme reálná čísla, rozlišujeme mutaci zatíženou a nezatíženou. Zatížená mutace přihlíží k dosavadní hodnotě. Nezatížená mutace volí novou hodnotu náhodně z celého intervalu. Zde použitá implementace používá oba způsoby. S pravděpodobností 80 % se provede zatížená mutace, pro 20 % případů nastane nezatížená varianta.
- **Adaptivní mutace** – adaptivní operátory mění některé své hodnoty během evoluce. Tato mutace mění na základě počtu generací svůj rozptyl (je zatížená) a svoji pravděpodobnost. S rostoucím počtem generací klesá pravděpodobnost a snižuje se rozptyl.

5.6 Nahrazení

Nahrazení vybírá populaci, která projde k selekci prostředím. Aplikujeme dva přístupy lišící se šancí rodičů přežít do další generace.

- **Jednoduché nahrazení** – k selekci prostředím postoupí jen nově vytvoření potomci, rodiče jsou zahozeni.
- **Slučovací nahrazení** – sloučí rodiče a potomky. Vzniká populace dvojnásobné velikosti. Příslušná selekce pak musí vybrat polovinu jedinců do další generace.

5.7 Fitness funkce

Fitness funkce není v námi řešeném problému explicitně daná. Vyvíjený jedinec se totiž používá jako hráč ve hře Carcassonne a výsledek každé sehrané hry je relativní vůči množství a kvalitě soupeřů.

V našem případě definujeme fitness funkci jedince jako průměrný bodový rozdíl za jednu hru vůči průměru ostatních hráčů ve hře. Ostatní hráči jsou pevně dané umělé inteligence neměnící se během evoluce.

Výpočet fitness funkce jedince j tedy probíhá následovně. Vezmeme jedince j a vytvoříme umělou inteligenci s vahami, které j reprezentuje. Přidáme jednoho až čtyři pevně dané soupeře (dle toho, pro hru kolika hráčů strategii vyvíjíme). S těmito počítačem řízenými hráči sehrajeme h her. Nechť umělá inteligence používající váhy jedince j získala během h her celkem b bodů. Nechť jejích n soupeřů získalo během h her b_1,b_2,...,b_n bodů. Potom hodnotu fitness funkce jedince j označenou $f(j)$ spočítáme dle vzorce:

$$f(j) = \frac{b - \frac{1}{n}(b_1 + b_2 + ... + b_n)}{h}. \tag{5.1}$$

Díky použití turnajové a diferenciální selekce, které fitness funkci nepoužívají, není její výpočet nutný pro správné fungování genetického algoritmu. Proto fitness jedinců počítáme z důvodu úspory času pouze občasně (jednou za deset až dvacet generací). Tato funkce nám tak slouží jen jako ukazatel růstu kvality populace.

6. Výsledky genetického algoritmu

V této kapitole uvedeme výsledky, kterých jsme pomocí evoluce dosáhli. Genetický algoritmus jsme pustili s různými druhy nastavení a sledovali jeho chování.

Všechny volitelné parametry jsou uvedené v tabulce 6.1 a to včetně výchozích hodnot. Není-li dále uvedeno jinak, potom každý parametr byl v konkrétním běhu algoritmu nastaven na tuto hodnotu.

Parametr	**Popis parametru**	**Výchozí hodnota**
vyvijenaUI	Název vyvíjené umělé inteligence.	UIkFaziPr
popSize	Velikost populace.	20
maxGenerations	Počet prováděných generací.	1000
pravidla	Počet vyvíjených pravidel.	19
faze	Počet fází.	3
mutProb	Pravděpodobnost, že jedinec bude vybrán k mutaci.	0,8
mutProbPerBit	Pravděpodobnost, že konkrétní gen jedince bude změněn mutací.	0,7[1]
xOverProb	Pravděpodobnost, že jedinec bude vybrán ke křížení.	0,75
matingGames	Počet her, který se sehraje při rodičovské selekci pro posouzení, který z jedinců je lepší.	20
environmentGames	Počet her, který se sehraje při selekci prostředí pro posouzení, který z jedinců je lepší.	16
fitnessFrequent	Po kolika generacích se počítá fitness (četnost).	14
players	Pro hru kolika hráčů je umělá inteligence vyvíjena.	2
parSel	Druh rodičovské selekce.	turnajová
envSel	Druh selekce prostředí.	diferenciální
mutOp	Druh použité mutace.	adaptivní
crossOp	Druh použitého křížení.	speciální n-bodové
replacement	Druh použitého nahrazení.	slučovací
vuciUI	Umělá inteligence, vůči které se počítá fitness funkce jedinců.	Vazena UI

Tabulka 6.1: Seznam parametrů genetického algoritmu.

Protože máme relativně velké množství parametrů a běh algoritmu je časově náročný, není možné vyzkoušet všechny jejich kombinace. Vybrali jsme tedy dvacet různých nastavení, u kterých předpokládáme největší přínos pro nalezení

[1] Jedná se o počáteční hodnotu, u adaptivní mutace v průběhu evoluce klesá až na 0,05.

vhodných vah. Celkem tak bylo sehráno přes deset milionů her. Není však zaručeno, že vybrané kombinace parametrů jsou ty nejlepší.

Při výběru hodnot pro velikost populace a pravděpodobnost křížení jsme se opírali o výzkum uvedený v knize M. Mitchell [13] (v kapitole 5.6). Pravděpodobnost adaptivní mutace byla zase zvolena na základě dobrých výsledků v optimalizaci spojitých funkcí z BBOB benchmarku [14].

Vybrané konfigurace s hodnotami parametrů, které se liší od výchozích, uvádíme níže. Každá odkazuje na graf v příloze A, ve kterém je uveden růst fitness funkce v daném běhu.

Konfigurace A.1 – všechny hodnoty jsou stejné s výchozími.

Konfigurace A.2 – *faze = 5.*

Konfigurace A.3 – *popSize = 15, faze = 5; mutProbPerBit = 0,6.*

Konfigurace A.4 – *mutProbPerBit = 0,5; matingGames = 6; environmentGames = 6; crossOp = uniformní.*

Konfigurace A.5 – *xOverProb = 0.8; envSel = turnajová; crosOp = uniformní; replacement = jednoduché; vuciUI = UI 19/3 pr.*

Konfigurace A.6 – *players = 3.*

Konfigurace A.7 – *vyvijenaUI = UIkFazi; crossOp = uniformní.*

Konfigurace A.8 – *popSize = 15; crossOp = uniformní; vuciUI = vůči nejlepší UI z předchozí konfigurace.*

Konfigurace A.9 – *vyvijenaUI = UIkFazi; popSize = 15; mutProb = 0,5; crossOp = uniformní.*

Konfigurace A.10 – *players = 5.*

Konfigurace A.11 – *popSize = 15; xOverProb = 0,9; mutProb = 0,9; mutProbPerBit = 0,1; matingGames = 10; environmentGames = 10; mutOp = klasická.*

Konfigurace A.12 – *popSize = 15; xOverProb = 0,65; mutProb = 0.5; mutProbPerBit = 0,8; environmentGames = 10.*

Konfigurace A.13 – *popSize = 15; mutProbPerBit = 0,2; envSel = turnajová; mutOp = klasická; replacement = jednoduché; crossOp = uniformní.*

Konfigurace A.14 – *maxGeneration = 750* – začínala s populací, kterou vytvořila předchozí konfigurace a rozšířila ji o pět jedinců.

Konfigurace A.15 – *players = 4.*

Konfigurace A.16 – *faze = 5; mutProb = 0,5; mutProbPerBit = 0,2.*

Konfigurace A.17 – *mutProb = 0,9; mutProbPerBit = 0,1; xOverProb = 0,85; mutOp = klasická; crosOp = uniformní.*

Konfigurace A.18 – *popSize = 15; maxGeneration = 700; mutProbPerBit = 0,1; matingGames = 30; environmentGames = 24; mutOp = klasická.*

Konfigurace A.19 – *popSize = 15; maxGeneration = 700; mutProbPerBit = 0,1; mutOp = klasická* – začínala s populací, kterou vytvořila předchozí konfigurace, bylo zvýšeno rozpětí jedince na interval $\langle -5, 30 \rangle$.

Konfigurace A.20 – *popSize = 15; faze = 1; crossOp = uniformní.*

Při provedených experimentech jsme nezjistili téměř žádný rozdíl ve výběru operátoru ani pro křížení, ani pro mutaci. Na druhé straně ale vývoj zlepšilo rozšíření pravidel o počítání pravděpodobností a zvýšení počtu fází hry z jedné na tři. Další posun ze tří na pět fází už ale viditelné zlepšení nepřinesl. Zhoršení úspěšnosti evoluce přineslo snížení počtu her při selekcích. Rovněž se ukázalo jako velmi dobré zvolit u adaptivní mutace vysokou počáteční pravděpodobnost.

Na základě porovnání výsledků z daných běhů jsme vybrali konkrétní jedince jako váhy do umělých inteligencí. Volili jsme dle nejvyšší hodnoty fitness funkce v posledních sto generacích genetického algoritmu.

Umělá inteligence *UI 19/3 pr* v sobě obsahuje vybraného jedince z první, šesté, desáté a patnácté uvedené konfigurace (v závislosti na počtu hráčů). *UI 19/5 pr* pak implementuje jedince získaného z druhé konfigurace a *UI 19/3* jedince ze sedmé konfigurace.

7. Testování umělých inteligencí

Zde se budeme zabývat porovnáním úspěšnosti vytvořených umělých inteligencí. Úspěšnost měříme jednak počtem výher, jednak průměrným množstvím bodů získaných v jedné hře. Dále také sledujeme průměrné počty bodů získané za jednotlivé herní objekty.

7.1 Parametry testů

Umělé inteligence testujeme pomocí programu `CarcassonneTestyUI.jar`. V rámci každého testu je sehráno vždy $100 \times k$ her, kde k udává počet hráčů, kteří hru hrají. Důvod je takový, že v rámci co největší objektivnosti požadujeme, aby každá z umělých inteligencí při testování sehrála vždy stejný počet her na i-tém místě ($i = 1,2,...,k$). Tedy chceme, aby stokrát hru začínala, stokrát hrála jako druhá v pořadí atd.

Výsledek každého testování shrnuje příslušná tabulka. Každý sloupec je nadepsán jménem umělé inteligence, pro kterou výsledky uvádíme. Její název se shoduje s názvem použitým v programu. V řádcích pak máme údaje s následujícím významem.

- **Body za města/louky/kláštery/cesty** – průměrný počet bodů za daný herní objekt v jedné hře (spočítaný jako $\frac{celkové\_body\_za\_objekt\_během\_všech\_her}{počet\_her}$).
- **Body celkem** – průměrný počet bodů v jedné hře.
- **Počet výher** – celkový počet výher dle definice 5.

Všechny číselné údaje v tabulkách jsou zaokrouhleny na dvě desetinná místa.

7.2 Náhodný přístup

Náhodný přístup je implementován pod názvem *Nahodna UI*. Ve hře dvou takových umělých inteligencí dosahuje každá z nich průměrně kolem dvaceti bodů za hru (viz tabulka 7.1). Na základě tohoto údaje pak můžeme vnímat úspěšnost dalších netriviálních umělých inteligencí.

	Nahodna UI	**Nahodna UI**
Body celkem	20,85	20,36
Body za města	8,99	7,61
Body za kláštery	1,91	2,11
Body za cesty	4,8	6,37
Body za louky	5,16	5,28
Počet výher	102	103

Tabulka 7.1: Nahodna UI vs. Nahodna UI.

7.3 Hladový přístup

Hladový přístup má v programu název *Hladova UI*. Jedná se o první netriviální strategii, která v souboji s *Nahodna UI* zcela dominovala (viz tabulka 7.2).

	Hladova UI	**Nahodna UI**
Body celkem	97,12	17,11
Body za města	46,66	5,78
Body za kláštery	9,28	1,96
Body za cesty	21,96	3,94
Body za louky	19,23	5,43
Počet výher	200	0

Tabulka 7.2: Hladova UI vs. Nahodna UI.

7.4 Optimalizace funkce dvou proměnných

Výsledkem tohoto přístupu je umělá inteligence *Optimalizacni UI*. Myšlení na budoucnost, které přidává tato strategie oproti *Hladova UI*, se vyplácí (viz tabulka 7.3).

	Optimalizacni UI	**Hladova UI**
Body celkem	89,05	83,23
Body za města	37,08	35,06
Body za kláštery	16,36	10,65
Body za cesty	18,41	19,42
Body za louky	17,21	18,11
Počet výher	124	79

Tabulka 7.3: Optimalizacni UI vs. Hladova UI.

7.5 Ohodnocující funkce

Tento přístup v našem programu využívají následující umělé inteligence.

- ***Vazena UI*** – základní soupeř pro porovnání v genetickém algoritmu, používá pravidla číslo 1 až 7 z tabulky 3.1 . Jejich váhy nastavuje na hodnotu 1.
- ***UI 19/3*** – implementuje všechna pravidla z tabulky 3.1, pracuje se třemi fázemi hry.
- ***UI 19/3 pr*** – používá rozhodování na základě pravděpodobnosti, pracuje se třemi fázemi hry.
- ***UI 19/5 pr*** – používá rozhodování na základě pravděpodobnosti, pracuje s pěti fázemi hry.

Jak vidíme v tabulce 7.4, tak už *Vazena UI* stačí na to, aby *Optimalizacni UI* porazila. Pomocí přidání pravidel a evoluce jejich vah jsme dokázali vyvinout umělou inteligenci *UI 19/3*, která je lepší ve více než 75% her a dosahuje o 13,5 bodu lepšího skóre na hru než základní *Vazena UI* (viz tabulka 7.5). Zajímavý je rozdíl v bodech za města, kde *UI 19/3* jasně vítězí, naopak se ale nestará tolik o louky, na kterých zase bodově zaostává.

Vylepšení pravidel o počítání pravděpodobností pak přináší další mírné zlepšení. Opět se zdokonalilo získávání bodů za města a vylepšil se také počet bodů získaných z luk (viz tabulka 7.6). Na druhou stranu *UI 19/3 pr* oproti svému konkurentovi ztrácí na cestách.

Přidání dalších dvou fází v umělé inteligenci *UI 19/5 pr* už zlepšení nepřineslo (viz tabulka 7.7). Jak již bylo patrné v genetickém algoritmu, vývin této strategie byl pomalejší, zřejmě v důsledku větší délky genu. Při déle trvající evoluci by nejspíš tento přístup dokázal poměr sil obrátit ve svůj prospěch, nejednalo by se však zřejmě o příliš dramatický nárůst výkonnosti.

Pro *UI 19/3 pr* jsme dále vyvinuli váhy navržené pro hru tří, čtyř a pěti hráčů. Tabulka 7.8 ukazuje (řádky a sloupce jsou z důvodu nedostatku místa prohozeny) porovnání vybraných pěti umělých inteligencí, kde *UI 19/3 pr(5)* v sobě obsahuje váhy vyvinuté pro hru pěti hráčů. V aplikaci s grafickým rozhraním i při zadávání údajů do konzole se váhy této umělé inteligence nastaví automaticky dle počtu hráčů. Při spouštění testů s nastavením údajů ze souboru je možné vynutit konkrétní váhy volbou *UI 19/3 pr(X)*, kde X je počet hráčů, pro které chceme váhy nastavit.

Vidíme, že při počtu pěti hráčů se začínají výrazně vyrovnávat průměrné počty dosažených bodů. Špatně nedopadla *Hladova UI*, u které s klesajícím počtem tahů mizí její hlavní handicap. Nejlépe dopadla specializovaná *UI 19/3 pr(5)*. Sloupec s počtem průměrných bodů ale ukazuje, že byly sehrány velmi těsné hry s minimálními rozdíly mezi vítězi a poraženými. Velkou roli už zřejmě hraje štěstí (nebo smůla) na vytažené kartičky (protože každý má během hry k dispozici jen čtrnáct až patnáct tahů). Při jiném zamíchání kartiček tak skóre může vypadat jinak.

	Vazena UI	**Optimalizacni UI**
Body celkem	100,03	93,79
Body za města	41,73	38,98
Body za kláštery	17,39	16,48
Body za cesty	22,26	18,33
Body za louky	18,66	20,01
Počet výher	124	80

Tabulka 7.4: Vazena UI vs. Optimalizacni UI.

	UI 19/3	**Vazena UI**
Body celkem	108,02	94,59
Body za města	50,32	34,52
Body za kláštery	17,81	15,81
Body za cesty	23,26	20,42
Body za louky	16,64	23,85
Počet výher	151	52

Tabulka 7.5: UI 19/3 vs. Vazena UI.

	UI 19/3	**UI 19/3 pr**
Body celkem	101,14	105,16
Body za města	39,68	47,1
Body za kláštery	18,13	18,67
Body za cesty	23,37	17,88
Body za louky	19,97	21,53
Počet výher	87	118

Tabulka 7.6: UI 19/3 vs. UI 19/3 pr.

	UI 19/3 pr	**UI 19/5 pr**
Body celkem	103,85	99,82
Body za města	46,46	43,02
Body za kláštery	19,52	16,09
Body za cesty	18,97	19,56
Body za louky	18,92	21,15
Počet výher	119	86

Tabulka 7.7: UI 19/3 pr vs. UI 19/5 pr.

	Body celkem	Počet výher
Hladova UI	41,16	72
UI 19/3	43,37	95
UI 19/5 pr	42,96	100
UI 19/3 pr	44,54	111
UI 19/3 pr(5)	46,34	147

Tabulka 7.8: Hladova UI vs. UI 19/3 vs. UI 19/5 pr vs. UI 19/3 pr vs. UI 19/3 pr(5).

7.6 Monte Carlo Tree Search a algoritmus expectimax

V práci [10], kde byly tyto algoritmy implementovány, dosahovala nejlepší umělá inteligence na bázi Monte Carlo Tree Search v jedné hře průměrně 70 až 80 bodů v závislosti na protihráči. Nejlepší umělá inteligence používající algoritmus expectimax dokázala v jedné hře získat průměrně okolo 85 bodů.

7.7 Hra proti člověku

Během ladění programu a testování chování umělých inteligencí jsme sehráli několik her proti některým umělým inteligencím jako lidský hráč. Ukázalo se, že ty nejlepší z nich mohou být člověku důstojným soupeřem a jsou schopné ho opakovaně porážet.

8. Rozbor nalezených vah

V této kapitole se podíváme na to, co nám říkají nalezené váhy vzhledem ke strategii hry. Uvádíme váhy umělé inteligence *UI 19/3 pr* pro hru dvou hráčů. Jejich číselné hodnoty ukazuje tabulka 8.1. Řádky odpovídají číslům pravidel uvedených v tabulce 3.1. Sloupce ukazují tzv. fáze hry. Ty se řídí podle počtu kartiček, které se již odehrály. Celkem je ve hře 72 kartiček, fáze rozdělujeme rovnoměrně. Tedy:

- **1. fáze** – do 24 odehraných kartiček (včetně),
- **2. fáze** – od 25 do 48 odehraných kartiček (včetně),
- **3. fáze** – 49 odehraných kartiček až do konce hry.

Číslo pravidla	**1. Fáze**	**2. Fáze**	**3. Fáze**
1	17,909	19,592	18,015
2	7,58	17,5267	9,7
3	19	11,637	19,238
4	19,9	17,423	15,822
5	12,624	15,101	12,062
6	6,301	6,662	15,863
7	5,791	1,724	9,695
8	2,048	18,032	16,323
9	16,927	4,758	15,862
10	0,534	0,584	6,651
11	20	5,021	11,407
12	18,406	18,406	12,729
13	4,952	2,366	4,9
14	-1,764	9,627	1,876
15	8,824	2,899	0,964
16	19,95	13,370	17,225
17	-0,405	0,3	-1,893
18	19,292	19,935	17,553
19	4,97	2,599	1,538

Tabulka 8.1: Váhy umělé inteligence UI 19/3 pr.

Jak je vidět, tak ze začátku hry je pro počítač nejdůležitější obsazovat kláštery a města (pravidla 3 a 4). Právě městům přikládá zdaleka největší váhu, protože si vysoce cení i jejich počtu (pravidlo 11) a kvality (pravidlo 16). Nejméně si váží počtu svých luk (pravidlo 14). Na druhou stranu ale také vyvrací předpoklad o nutnosti penalizace za položené sedláky (pravidlo 17).

Ve střední fázi hry se zvyšuje důraz především na uzavírání cest (pravidlo 2) a vysoce stoupá penalizace za tahy, ve kterých je položena figurka (pravidlo 8). To je zřejmě způsobeno tím, že umělé inteligenci začínají docházet figurky. Přestávají ji také zajímat počty a kvalita majoritních měst (pravidla 11 a 16). To je

pravděpodobně proto, že už jich má obsazeno dostatek a nehodlá přidávat další. Výrazně roste její zájem o počet majoritních luk (pravidlo 14).

V závěru hry stoupá zejména váha bodů za louky (pravidlo 6). V rámci herních objektů je druhá nejvyšší (po klášteru), což znamená, že v závěru hry počítač raději pokládá figurky na louky než na stejné bodově ohodnocené cesty nebo města. Celou hru pak logicky klesá váha pravidla o počtu volných figurek (pravidlo 15), což nám říká, že není dobré mít v závěru hry příliš volných figurek.

9. Modifikace pravidel a vliv na umělé inteligence

V této kapitole se zaměříme na některé zajímavé modifikace standardních pravidel a podíváme se, jak působí na výkonnost vyvinutých umělých inteligencí.

9.1 Malé město za dva body

Za malé město považujeme takové, které je uzavřené a skládá se z právě dvou kartiček. Ve všech novějších verzích hry se ohodnocuje čtyřmi body. Původně se však počítalo pouze za dva body. Podívejme se tedy, jak zapůsobí tato (již nepoužívaná) varianta na naše umělé inteligence.

Tabulka 9.1 zobrazuje dosažené body umělých inteligencí ve stejných hrách jako tabulka 7.4. Jedinou změnou je právě bodové ohodnocení malého města. Obdobně tabulka 9.2 ukazuje hodnoty ze stejných her jako tabulka 7.7.

Pokles průměrného počtu celkových bodů i počtu bodů za města je znatelný u všech umělých inteligencí. Pohybuje se v rozmezí od pěti do deseti bodů. V ostatních kategoriích není změna příliš znát. Všimnout si můžeme poklesu bodů z luk u *UI 19/5 pr* o dva body. To může souviset s tím, že mu jeho soupeř nezavírá tolik měst (protože buduje větší), za které by *UI 19/5 pr* na loukách získala body. Rozdíl je ale velmi malý (dva body z luk představují necelé jedno uzavřené město).

	Vazena UI	**Optimalizacni UI**
Body celkem	89,76	85,45
Body za města	32,91	31,04
Body za kláštery	16,03	17,04
Body za cesty	22,77	18,04
Body za louky	18,06	19,34
Počet výher	119	84

Tabulka 9.1: Vazena UI vs. Optimalizacni UI.

	UI 19/3 pr	**UI 19/5 pr**
Body celkem	98,85	93,71
Body za města	42,03	38,09
Body za kláštery	19,75	16,52
Body za cesty	18,61	19,95
Body za louky	18,47	19,16
Počet výher	123	80

Tabulka 9.2: UI 19/3 pr vs. UI 19/5 pr.

9.2 Spravedlivá hra

Základní varianta hry Carcassonne obsahuje celkem 72 kartiček. Jedna kartička je označená jako startovní a vykládá se na herní plochu ještě před začátkem hry. Hráči si tedy vybírají a umísťují 71 kartiček. Číslo 71 je prvočíslo, takže pokud nedojde k zahození některých kartiček z důvodu nemožnosti položení (což není příliš časté), nepodaří se kartičky mezi hráče rovnoměrně rozdělit.

Někteří jsou tak zvýhodněni tím, že umísťují do hry o jednu kartičku víc než jíní. Spravedlivá hra má v našem pojetí následující pravidla. Hru budeme chápat jako posloupnost kol. V jednom kole položí každý hráč do hry jednu kartičku. Před začátkem každého kola se podíváme, zda ještě zbývá dostatek kartiček ve hře. Pokud je jich méně než hráčů, nastává konec hry.

Tímto postupem zařídíme, že pokud nedojde k vyřazení některé z kartiček v posledním kole (což je velmi nepravděpodobné), budou mít všichni hráči ve hře stejný počet kartiček k zahrání.

Z provedeného experimentu vyplývá, že při hře dvou stejných umělých inteligencí má spravedlivá hra minimální vliv na počet výher (viz tabulka 9.3 a 9.4). Na druhou stranu se ale ukazuje výhoda začínajícího hráče, který bez ohledu na druh hry, vyhrává zhruba v 55% případů.

Při hře čtyř vesměs různých umělých inteligencí se situace změnila (viz tabulka 9.5). Každá umělá inteligence má oproti hře dvou hráčů poloviční počet tahů. Rozdíl oné jedné kartičky navíc se tak již projeví výrazněji.

Nejvíce získala *UI 19/3 pr*, která hrála jako čtvrtá v pořadí. Při normální hře byla jediná, která pokládala o jednu kartičku méně. Spravedlivá hra uškodila strategii *Optimalizacni UI*. Nalezený poměr, který používá, bude nejspíš také závislý na počtu hráčů.

Ve všech třech pokusech jsme sehráli 500 klasických a 500 spravedlivých her, které byly ve všech testech stejné.

	Počet výher	**Počet výher – spravedlivá hra**
UI 19/3 pr	272	273
UI 19/3 pr	236	238

Tabulka 9.3: UI 19/3 vs. UI 19/3 – normální a spravedlivá hra.

	Počet výher	**Počet výher – spravedlivá hra**
Hladova UI	281	273
Hladova UI	223	235

Tabulka 9.4: Hladova UI vs. Hladova UI – normální a spravedlivá hra.

	Počet výher	Počet výher – spravedlivá hra
Hladova UI	108	109
Optimalizacni UI	104	79
UI 19/3	161	163
UI 19/3 pr	148	172

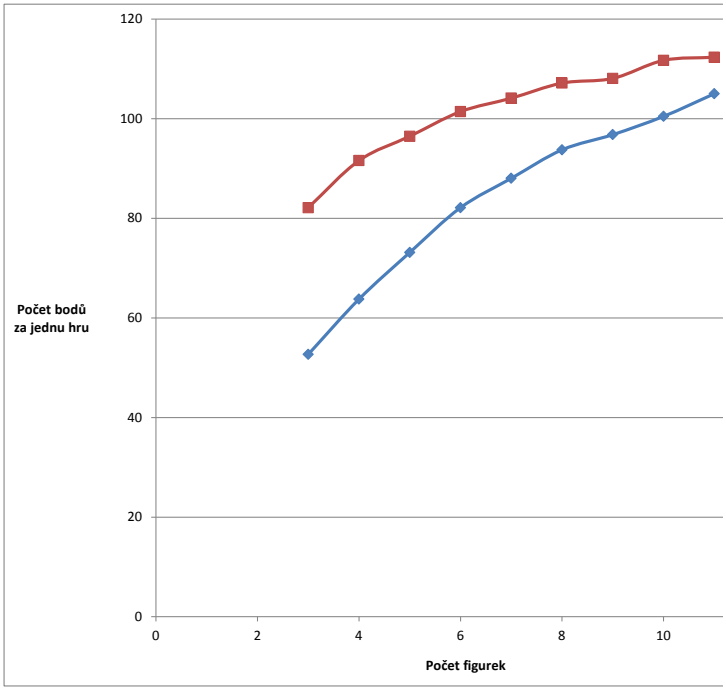
Tabulka 9.5: Hladova UI vs. Optimalizacni UI vs. UI 19/3 vs. UI 19/3 pr – normální a spravedlivá hra.

9.3 Vliv počtu figurek

Figurky (družiníci) představují jediný zdroj bodů ve hře. Podle pravidel má každý hráč na začátku k dispozici sedm figurek. Můžeme se podívat na to, co se stane s výší dosažených bodů pokud figurky přidáme, či naopak ubereme.

Na obrázku 9.1 vidíme, jak se mění počet bodů získaných za jednu hru u umělých inteligencí *Optimalizacni UI* a *UI 19/3 pr* v závislosti na počtu figurek, které měly pro hru k dispozici. Počet bodů za jednu hru byl spočítán jako průměr během sto her.

Růst počtu bodů není ani v jednom případě čistě lineární. Je vidět, že umělá inteligence *Optimalizacni UI* roste rychleji a dohání *UI 19/3 pr*. Naučené váhy buď neumí tak pružně reagovat na přidání dalších figurek nebo se na úrovni 120 bodů nachází hranice možností, za kterou je těžké se dostat bez ohledu na počet figurek. Dále si můžeme všimnout, že pro obě umělé inteligence se nejvyšší nárůsty odehrávají hned ze začátku, přičemž strategii *Optimalizacni UI* vydrží trend rychlého růstu o něco déle (až do přidání šesté figurky).



Obrázek 9.1: Závislost počtu bodů na počtu figurek.

10. Uživatelská dokumentace

10.1 Minimální požadavky na systém

Pro spuštění a správný běh programu je vyžadována alespoň instalace Java SE Runtime Environment ve verzi 8 nebo vyšší. Pro pořizování záznamů a obrázků je nutné mít ve vybrané složce práva na vytváření souborů a zápis.

10.2 Úvodní obrazovka

Po spuštění programu se objeví úvodní obrazovka (viz obrázek 10.1). Pro vytvoření nové hry postupujeme následujícím způsobem. Nejprve vybereme počet hráčů. Minimum jsou dva, maximum pět hráčů. U každého hráče je možné zvolit jméno. Zaškrtneme-li u libovolného z hráčů políčko `Počítač`, objeví se nabídka dostupných umělých inteligencí. Přesný princip jejich fungování je popsán v kapitole 3. Chceme-li testovat naši vlastní umělou inteligenci (návod, jak ji přidat, je uveden v sekci 11.7), vybereme možnost *Vlastni UI* a do kolonky pro zadání jména napíšeme jméno třídy, která popisuje tuto implementaci. Jméno třídy je nutné z obou stran ohraničit znakem $. Další znaky se berou jako jméno hráče. V případě, že možnost `Počítač` není zaškrtnuta, tahy příslušného hráče provádí člověk.

Chceme-li vytvořit textový záznam ze hry, zaškrtneme možnost `Záznam ze hry`. Pro pořízení obrázků zaškrtneme políčko `Obrázky ze hry`. Hru zahájíme stisknutím tlačítka `Nová hra`. Stisknutím tlačítka `Konec` ukončíme aplikaci.

Na obrázku 10.2 vytváříme hru pro tři hráče. Hráč 1 je člověk, hráče 2 a 3 ovládá počítač. První počítačem ovládaný hráč implementuje strategii popsanou v kapitole 3.2. U hráče 2 vidíme demonstraci použití externě dodané umělé inteligence. V tomto příkladu vkládáme strategii *TestovaciUI* se jménem `cizí UI`. Ze hry chceme pořídit textový záznam.

10.3 Menu

V horní liště programu se nachází menu, které má následující funkce.

- **Panel Soubor**
  - **Začít hru** – vytvoří novou hru.
  - **Restartovat hru** – ukončí právě rozehranou hru a začne novou se stejným nastavením.
  - **Zpět na úvodní obrazovku** – ukončí právě rozehranou hru a vrátí se zpět na úvodní obrazovku.
  - **Konec** – ukončí program.
- **Panel Nápověda**
  - **Pravidla hry** – zobrazí soubor s oficiálními pravidly hry.

  - **O programu** – zobrazí základní informace o aplikaci.

- **Panel Nastavení**

  - **Vybrat složku pro ukládání záznamů** – umožňuje změnit úložiště textových záznamů ze hry.
  - **Vybrat složku pro ukládání obrázků** – umožňuje změnit úložiště obrázků ze hry.
  - **Mazat odebrané figurky z hracího plánu** – v případě zaškrtnutí této možnosti se figurky, za které hráč již získal body, smažou z hracího plánu.
  - **Animovat tahy počítače** – je-li tato možnost zaškrtnuta, dochází při tahu počítače ke grafickému znázornění přesunu kartičky na vybranou pozici.
  - **Obrázky** – při zaškrtnutí možnosti `Během celé hry` se po každém tahu vytvoří a uloží obrázek hracího plánu. Při volbě možnosti `Jen na konci` se z každé hry pořídí jen obrázek výsledné pozice. V tomto případě je vhodné vypnout možnost **Mazat odebrané figurky z hracího plánu**. Výběr je relevantní pouze v případě, že jsme na úvodní obrazovce vybrali možnost `Obrázky ze hry`.
  - **Upravit pravidla hry** – otevře nové okno, ve kterém je možné zapnout vybrané modifikace hry uvedené v kapitole 9.

10.4 Herní obrazovka

Po zahájení nové hry se objeví hrací plocha (viz obrázek 10.3). V horní části najdeme informace o hráčích. U každého hráče můžeme pozorovat tyto údaje:

- **Jméno** – zobrazuje se jméno, které bylo nastaveno na úvodní obrazovce.
- **Barva** – barevný kruh vedle jména určuje, jakou barvu mají hráčovy figurky.
- **Aktuální počet bodů** – aktualizuje se vždy po ukončení tahu každého z hráčů.
- **Člověk/Počítač** – pozná se dle ikony člověka/robota vedle ukazatele počtu bodů.
- **Počet volných figurek** – aktualizuje se ihned po položení nebo vrácení figurky.

Hráč, jehož jméno je zvýrazněno tučně, je právě na řadě.

Na pravé straně obrazovky se nachází doposud neotočené kartičky. Hlavní část zabírá hrací plocha s již odehranými kartičkami. Na obrázku 10.3 se zde nachází pouze startovní karta. Pomocí tlačítek s obrázkem lupy na levé straně můžeme přibližovat nebo oddalovat pohled na hrací plochu.

10.5 Jak hrát

Hráč, který je na řadě, postupuje dle následujících kroků.

1. **Výběr kartičky** – vybereme libovolnou kartičku z pravé strany obrazovky. Existuje-li přípustné umístění této kartičky, zobrazí se její náhled na levé straně obrazovky (viz obrázek 10.4). Není-li možné vybranou kartičku vyložit, objeví se upozornění a dojde k zahození dané kartičky. V tom případě pokračujeme vybráním jiné kartičky.

2. **Vyložení na hrací plochu** – vybranou kartičku musíme dle pravidel vložit do hry. Kartičku lze otáčet pomocí tlačítek `-90` a `+90`. Samotné vyložení provedeme přetažením kartičky z náhledu na herní plochu, případně kliknutím na zvolené místo herní plochy. Je-li umístění v souladu s pravidly, kartička zůstane na místě. V opačném případě je vrácena zpátky. Pomocí tlačítka `Zpět` se dá kartička vrátit a poté vyložit na jiné místo.

3. **Umístění figurky (volitelné)** – po vložení kartičky do hry máme možnost umístit figurku. Pro tento úkon se zpřístupní nová tlačítka (viz obrázek 10.5). Výběr figurky provedeme pomocí kliknutí na tlačítko s jejím obrázkem. Kliknutím na příslušnou část vyložené kartičky dojde k umístění figurky. Pokud je položení v souladu s pravidly, objeví se v dané části obrázek figurky. Pro větší přehlednost každá figurka obsahuje písmeno podle části kartičky, na které stojí. Používáme označení:

   - **L** – lupič
   - **R** – rytíř
   - **M** – mnich
   - **S** – sedlák

   Již umístěnou figurku můžeme vrátit zpět stisknutím tlačítka s přeškrtnutou figurkou.

4. **Ukončení tahu** – tah ukončíme stiskem tlačítka `Ukonči tah`. Poté dojde k aktualizaci bodů a ve hře bude pokračovat další hráč.

10.6 Barevné zvýraznění karet

Hra si pamatuje a umí zobrazit poslední vyloženou kartičku každého hráče. Tato karta je zvýrazněna stejnou barvou, jakou mají hráčovy figurky. Ve chvíli, kdy se hráč dostane na řadu, se na hrací ploše objeví barevně označené tahy jeho soupeřů.

Zvýraznění kartiček automaticky zmizí poté, co si hráč kliknutím vybere a otočí jednu z kartiček. Podbarvení poslední kartičky hráče lze rovněž zrušit kliknutím na barevný kruh vedle jeho jména. Pro opětovné ukázání stačí na barevném kruhu stisknout a podržet tlačítko myši.

10.7 Konec hry

Hra končí po tahu, ve kterém byla vyložena poslední kartička. Poté dojde k automatickému přičtení bodů za všechny zbylé figurky. V novém okně se zobrazí závěrečné výsledky (viz obrázek 10.6). Kromě celkového počtu bodů tabulka uvádí i body za cesty, města, kláštery a louky.

Pořadí hráčů se určuje podle počtu dosažených bodů. V případě rovnosti bodů je v konečném pořadí výše uveden hráč, který začínal později (tj. hráč s vyšším startovním číslem).

10.8 Textový záznam ze hry

Je-li uživatelem vybrána tato možnost, vytváří se ze hry záznam v podobě textového souboru. Jméno má podle data a času, ve kterém došlo k založení hry. Výchozí umístění záznamu je složka *logy* v místě, kde se nachází spustitelný program. Změnu složky lze v programu provést v panelu **Nastavení**.

10.8.1 Formát souboru

Soubor začíná hlavičkou. Je zde uveden počet hráčů a informace o každém hráči ve tvaru:

```
Hráč POŘADÍ: JMÉNO - UI.
```

Význam proměnných výše je následující:

- `POŘADÍ` – pořadí hráče ve hře – číslo od 1 do 5.
- `JMÉNO` – jméno hráče.
- `UI` – druh umělé inteligence, kterou hráč implementuje.

Například tedy:

```
Hráč 2: Michal - Hladova UI.
```

Dále následuje popis jednotlivých tahů. Každý tah je ve formátu:

```
Tah: ČÍSLO
HRÁČ:KARTA XSOUŘADNICE;YSOUŘADNICE ROTACE-FIGURKA
STAV.
```

Proměnné nám říkají:

- `ČÍSLO` – číslo tahu. Určuje, kolik kartiček již bylo odehráno. Tah 1 je položení startovní kartičky, hráči pak hrají tahy číslo 2 až 72.
- `HRÁČ` – pořadí hráče, který tah zahrál – číslo od 1 do 5.
- `KARTA` – číslo kartičky. Každá karta má jedinečný identifikátor.
- `XSOUŘADNICE` – souřadnice x hrací plochy, kam byla kartička umístěna.
- `YSOUŘADNICE` – souřadnice y hrací plochy, kam byla kartička umístěna.

- `ROTACE` – otočení kartičky ve hře vůči výchozímu natočení (uváděno ve stupních). Jako výchozí natočení kartičky uvažujeme to, ve kterém se kartička zobrazila v náhledu hned po vybrání.
- `FIGURKA` – umístění figurky na kartičce zapsané pomocí dvojice čísel od 0 do 2 (označujeme ji jako region kartičky). Závisí na otočení. Při tahu bez figurky je uvedeno slovo *null*.
- `STAV` – pro každého hráče obsahuje jeden řádek, ve kterém je uveden jeho počet bodů a volných figurek.

Zápis tahu v souboru tak může vypadat například takto:

```
Tah: 9
2:36 5;3 270-0,1
Hráč 1: 0 bodů 6 volných figurek
Hráč 2: 0 bodů 5 volných figurek
Hráč 3: 0 bodů 6 volných figurek
```

10.9 Obrázkový záznam ze hry

Pořizujeme-li obrázkový záznam ze hry, vytvoří se složka s názvem *Hra Datum_a_čas*. Ve výchozím nastavení se vytváří ve složce *screenshoty* na místě, kde se nachází program. Změnu umístění lze provést v panelu **Nastavení**.

Pořízené obrázky jsou v grafickém formátu GIF. Zachycují herní plochu s vyloženými kartičkami a aktuální informace o hráčích.

10.10 Konzolová aplikace

Pro testování umělých inteligencí je k dispozici také konzolová varianta hry Carcassonne. Je určena pouze pro hraní umělých inteligencí a umožňuje jim sehrát více her najednou.

Aplikace se spouští z příkazové řádky. Nastavení všech potřebných údajů o hře lze provést pomocí textového souboru, který předáme při spuštění jako parametr, nebo přímo v programu.

10.10.1 Nastavení údajů ze souboru

Program akceptuje textové soubory s touto strukturou:

```
POČET HRÁČŮ
INFORMACE
POČET HER
ZÁPISY
NÁHODNOST
"VÝSTUP"
[MODIFIKACE].
```

Každá z těchto proměnných musí být na samostatném řádku. Význam a omezení proměnných uvádíme níže.

- `POČET HRÁČŮ` – počet hráčů - číslo v rozmezí 2 až 5.
- `INFORMACE` – pro každého hráče musí být na samostatném řádku uvedena informace ve tvaru: `"TYP UI""JMÉNO"`.
  - `TYP UI` – název jedné z umělých inteligencí, shodný s názvem v aplikaci s grafickým rozhraním.
  - `JMÉNO` – jméno pro vybranou umělou inteligenci. Zvolíme-li `TYP UI` jako *Vlastni UI*, potom ve jméně uvedeme jako první název třídy, která danou strategii implementuje. Jméno třídy musí být z obou stran odděleno znakem $.
- `POČET HER` – počet her, které spolu počítače sehrají.
- `ZÁPISY` – můžeme zvolit, zda chceme z každé sehrané hry pořídit textový záznam. Možnost `ano "ÚLOŽIŠTĚ"`, kde `ÚLOŽIŠTĚ` obsahuje umístění na disku, vytvoří v tomto místě patřičné záznamy. V ostatních případech nebude záznam ze hry pořizován.
- `NÁHODNOST` – pokud chceme na začátku inicializovat generátor náhodných čísel (pro opakovatelnost testů), zvolíme možnost `ano SEED`, kde `SEED` uvádí číslo, kterým bude na začátku inicializován náhodný generátor. V případě jiné volby bude počáteční hodnota náhodná.
- `VÝSTUP` – název souboru (včetně cesty), kam chceme zapsat výsledky testu. Musí být ohraničen uvozovkami. Napíšeme-li místo názvu souboru pomlčku, vypíšou se výsledky na standardní výstup.
- `MODIFIKACE` – tvoří volitelnou část souboru. Umožňuje zapnout modifikace z kapitoly 9. Všechny vybrané modifikace se uvádí na jednom řádku a oddělují se středníkem. Rozlišujeme celkem tři druhy:
- `m2` – zapne počítání malých měst za dva body,
- `spravedl` – zapne spravedlivou hru,
- `figurky = X` – nastaví výchozí počet figurek u každého hráče na hodnotu `X`.

Níže uvádíme příklad textového souboru. Program používá pro čtení i zápis souborů kódování UTF-8.

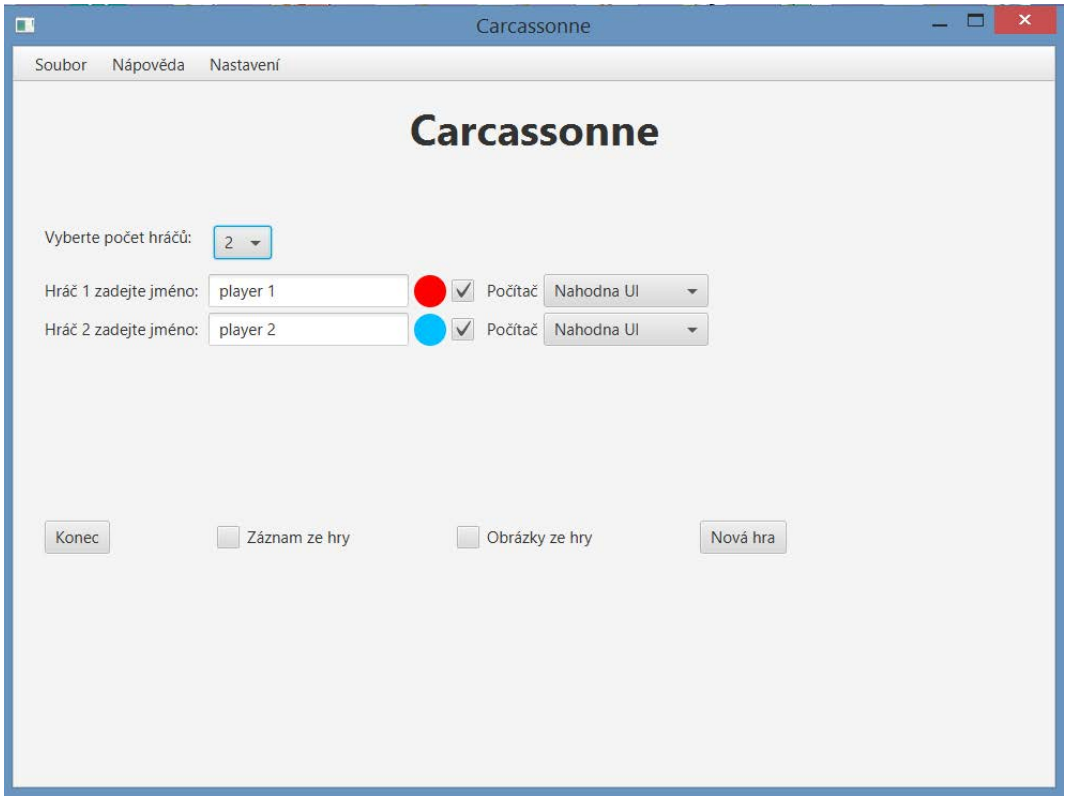
```
3
"Vazena UI" "PC 1"
"Hladova UI" "PC 2"
"Vlastni UI" "$TestovaciUI$PC s vlastní implementací"
10
ne
ano 1111
"logy/vysledky.txt"
```

10.10.2 Nastavení údajů v programu

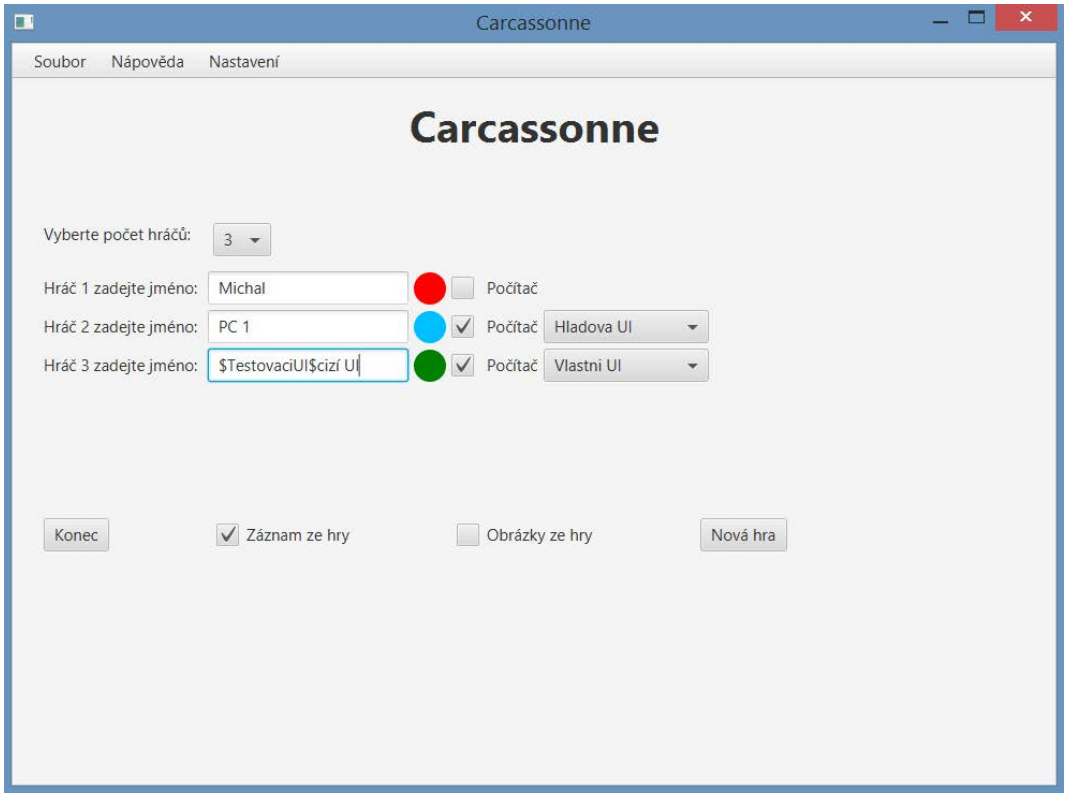
Chceme-li nastavení údajů provést za běhu programu, spouštíme ho bez parametrů. Dále doplňujeme potřebná data dle instrukcí, které aplikace vypisuje na obrazovku. Vyžadují se stejné údaje a platí i stejná omezení jako při nastavení ze souboru.

Níže uvádíme výpis z programu, pomocí kterého spustíme stejný test, jaký je uvedený v předchozí kapitole. Navíc ještě přidáme dvě modifikace pravidel. Řádky začínající znakem > doplňuje uživatel, ostatní jsou vypsány programem.

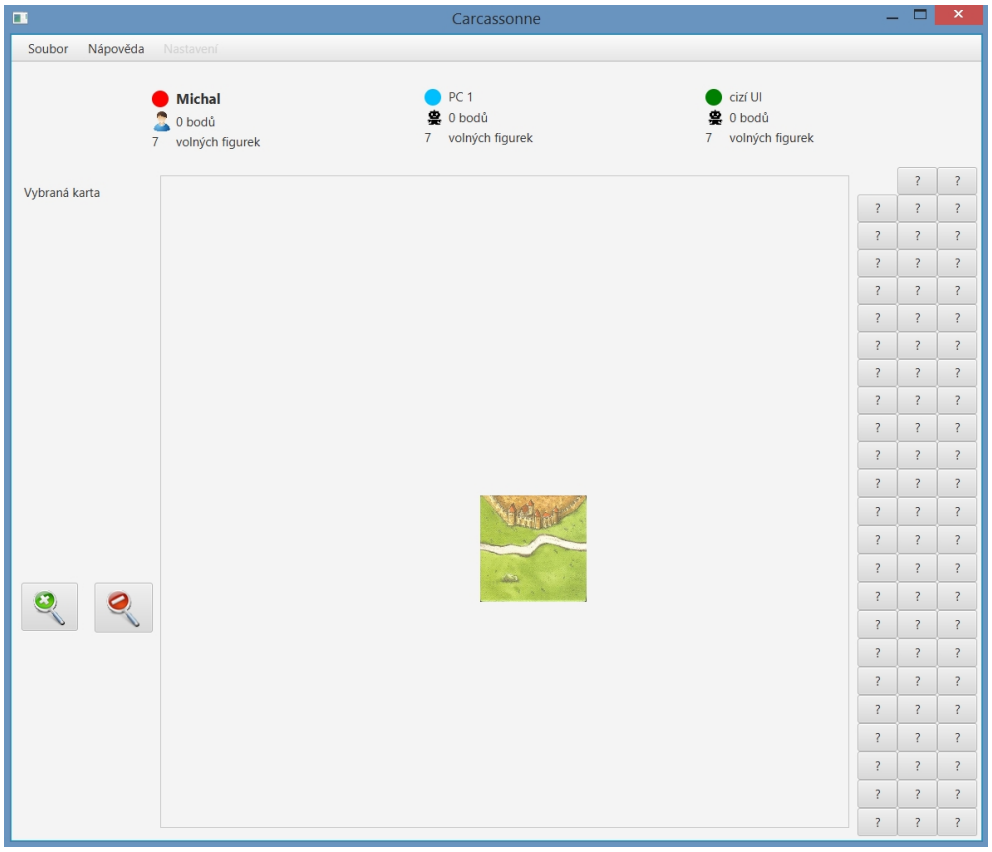
```
Zadejte počet hráčů:
>3
Výběr umělé inteligence - nápověda:
1 = Nahodna UI
2 = Hladova UI
3 = Optimalizacni UI
4 = Vazena UI
5 = UI 19/3
6 = UI 19/5 pr
7 = UI 19/3 pr
8 = Vlastni UI
----------------------------------------
Vyberte druh 1. hráče:
>4
Zadejte jméno 1. hráče:
>PC 1
Vyberte druh 2. hráče:
>2
Zadejte jméno 2. hráče:
>PC 2
Vyberte druh 3. hráče:
>8
Zadejte jméno 3. hráče:
>PC s vlastní implementací
Zadejte jméno třídy, která obsahuje implementaci
Vaší umělé inteligence.
>TestovaciUI
Zadejte počet her:
>10
Pořizovat zápis ze hry? ano/ne:
>ne
Inicializovat náhodný generátor použitý ve hře? ano/ne:
>ano
Zadejte nový seed generátoru - vckaždé další hře se
zvětší o 1:
>1111
Napište název souboru, do kterého mají být zapsány výsledky.
Případně zvolte "-" pro výpis na standardní výstup.
>logy/vysledky.txt
Zadejte případné modifikace pravidel, nebo pokračujte stiskem
klávesy Enter.
Nápověda:
m2          - zapne počítání malých měst za 2 body
spravedl    - zapne spravedlivou hru
figurky = X - změní počet figurek na začátku hry na X
(musí být od 3 do 11)
Chcete-li zadat více modifikací, oddělte je středníkem.
>m2 ; figurky = 5
```



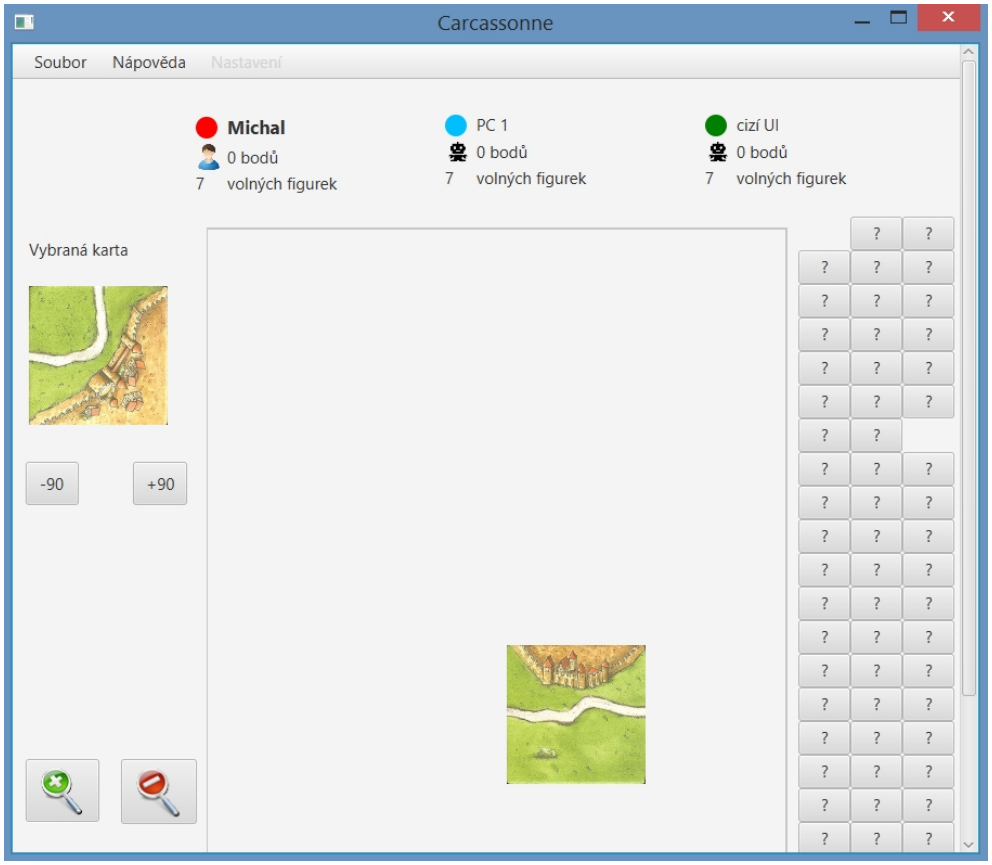
Obrázek 10.1: Úvodní obrazovka.



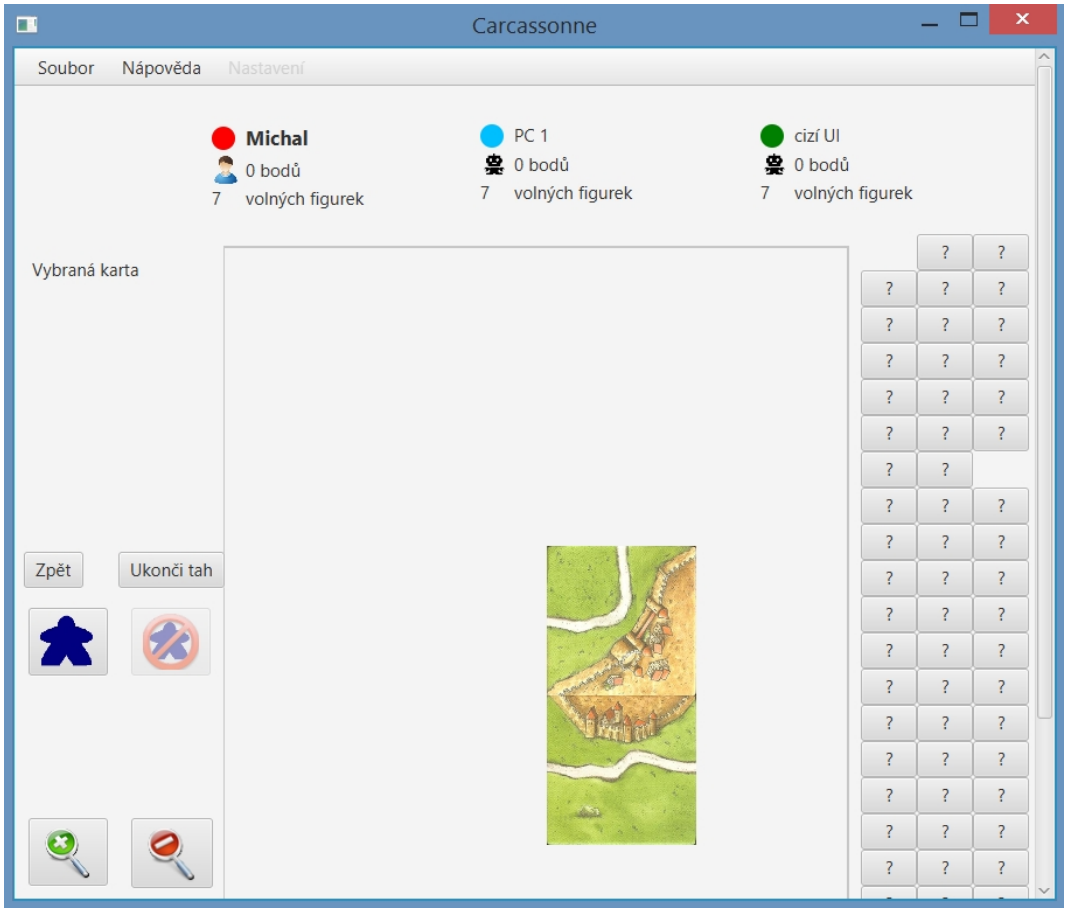
Obrázek 10.2: Nastavení nové hry.



Obrázek 10.3: Nová hra.



Obrázek 10.4: Náhled vybrané kartičky.



Obrázek 10.5: Možnost přiložení figurky.

Pořadí	Jméno hráče	Body za města	Body za kláštery	Body za cesty	Body za louky	Body celkem
1.	Michal	42	23	14	6	85
2.	PC 1	27	18	18	21	84
3.	cizí UI	4	0	6	12	22

Obrázek 10.6: Tabulka s výsledky.

11. Programátorská dokumentace

Aplikace je vytvořena v programovacím jazyce Java verze 8 [3]. Pro psaní kódu bylo využito vývojové prostředí Eclipse. Tvorba grafického uživatelského rozhraní probíhala pomocí nástroje JavaFX. Návrh základního formuláře hry byl vytvořen nástrojem JavaFX Scene Builder [4].

Níže se zaměříme pouze na základní princip fungování programu a přehled jeho nejdůležitějších částí. Zevrubný popis všech tříd a jejich metod obsahuje dokumentace vygenerovaná ze zdrojových kódů pomocí nástroje Javadoc. Dokumentaci nalezneme v elektronické příloze této práce B.

11.1 Členění programu

Celý program je rozdělen do několika balíčků (packages). Implementace hry Carcassonne se skládá z následujících balíčků.

- **Balíček** `carcassonne` – obsahuje třídy, které v sobě mají metodu `main()`.
- **Balíček** `carcassonne.herniProstredi` – obsahuje implementaci všech herních prvků, herní logiky a grafického prostředí.
- **Balíček** `carcassonne.umeleInteligence` – obsahuje implementaci umělých inteligencí.
- **Balíček** `carcassonne.obrazky` – zde se nachází obrázky použité v programu. Tento balíček neobsahuje žádné zdrojové kódy, nebudeme se mu tedy již dále věnovat.

Pro implementaci genetického algoritmu byla využita již existující knihovna M. Piláta [15]. V programu používáme třídy z níže uvedených balíčků.

- **Balíček** `evolution` – zahrnuje v sobě třídy implementující základní genetický algoritmus, metody nahrazení a výpočet fitness funkce.
- **Balíček** `evolution.individuals` – obsahuje třídy definující jedince v genetickém algoritmu.
- **Balíček** `evolution.operators` – zde se nachází rozhraní pro operátory v genetickém algoritmu.
- **Balíček** `evolution.selectors` – obsahuje rozhraní pro selektory v genetickém algoritmu

Základní evoluční algoritmus jsme dále vylepšili a přizpůsobili pro vývoj strategie ve hře Carcassonne.

- **Balíček** `evolution.vahyCarcassonne` – obsahuje třídy rozšiřující výše zmíněnou knihovnu.

11.2 Balíček `carcassonne`

- Třída `Hlavni` – vytváří nový formulář (tzv. scénu). Konkrétní vzhled formuláře se načítá z připraveného souboru s příponou FXML. Tento soubor je přítomen v balíčku `carcassonne.herniProstredi`.
- Třída `CarcassonneTestyUI` – slouží pro testování umělých inteligencí mezi sebou. Nejdůležitější je samotná metoda `main()`, která načte od uživatele potřebná data (z externího souboru nebo z konzole). Podle těchto dat sehraje daný počet her a připraví z nich požadovaný výstup.
- Třída `TestOptUI` – obsahuje implementaci experimentu pro nalezení vhodných konstant u inteligence *Optimalizacni UI* (viz. kapitola 4).

11.3 Balíček `carcassonne.herniProstredi`

- Třída `Hra` – objekt této třídy reprezentuje jednu konkrétní Carcassonne hru. Povinným parametrem pro každou hru je seznam hráčů, kteří ji budou hrát. Volitelně je možné přidat do hry objekt splňující rozhraní `Zaznam` k pořízení záznamu ze hry a údaje o změně pravidel popsané v kapitole 9.

  Třída má nastavitelný příznak `TestRezim`. Je-li zapnutý, přebírá třída řízení a po vytvoření konkrétní instance se ihned spustí smyčka, která sehraje celou hru. Toho využívá konzolová varianta aplikace pro testování umělých inteligencí. Při vypnutém příznaku se hra řídí odjinud pomocí volání příslušných metod. Toto řešení používá program s grafickým uživatelským rozhraním, kde průběh hry řídí třída `FXMLDocumentController`.

  Dále v sobě třída `Hra` obsahuje datové položky (odehrané kartičky, položené figurky, informace o hráčích) a spravuje také pomocné třídy, které mají na starost specifické úkony v rámci herní logiky.

  - `HraPravidla` posuzuje a rozhoduje o korektnosti umístění kartiček a figurek.
  - `HraVyhodnoceni` počítá body za jednotlivé herní objekty a přiděluje je příslušným hráčům.
  - `HraProhledavani` umí procházet herní plochu a poskytovat informace o velikosti či obsazenosti herních objektů.
  - `HraVyrobaKarticek` vytváří základní sadu 72 kartiček, které se v Carcassonne používají.
  - `HraZpracovaniTahuPC` se stará o komunikaci mezi logikou hry a umělou inteligencí. Nejprve má za úkol vybrat náhodnou kartičku a předat ji umělé inteligenci. Když potom umělá inteligence vrátí tah, který chce zahrát, tato třída zařídí jeho přidání do hry, nebo ohlásí chybu v případě, že tah je proti pravidlům (korektnost opět posuzuje třída `HraPravidla`).
  - `HraHodnoceniPozice` počítá data, která využívá při vyhodnocování našich napsaných pravidel ve třídách `SadaPravidel` a `SadaVylepseni`.

Jedná se o soubor různých datových struktur, jejichž obsah se při každém možném tahu přepočítává.

  - `TabulkaSkoreAFigurek` obsahuje datovou strukturu pro uchovávání aktuálního skóre hry a počtu bodů za jednotlivé herní objekty.

- Třída `Karticka` – kartičku reprezentujeme jako dvourozměrné pole znaků o rozměrech 3x3. Znaky udávají typ krajiny, který je na dané části kartičky zobrazen. Rozlišujeme typ *město (m)*, *cesta (c)*, *klášter (k)*, *louka (l)* a *nevyužito (n)*.

  Převod obrázku kartičky na pole znaků jsme provedli následovně. Obrázek jsme rozdělili na devět dílků (regionů). Z nich jsou čtyři rohové, čtyři krajové a jeden středový. Znak pro rohové a krajové části jsme vybrali podle typu krajiny, která se nachází na hraně, se kterou daná kartička sousedí s ostatními. U rohových regionů máme sousední hrany dvě, přičemž může dojít k tomu, že na jedné z nich je louka a na druhé město. Potom vybereme jeden z nich na základě toho, jakou figurku chceme umožnit v tomto regionu hráči položit (zda rytíře nebo sedláka). Při výběru znaku pro středový region bereme opět v úvahu druh figurky, který zde budeme pokládat.

  Zároveň při převodu do znakové reprezentace dbáme i na zachování souvislosti všech objektů z kartičky. Herní objekt je souvislý, pokud jsou každé dva jeho regiony propojeny pomocí regionů se stejným písmenem. Regiony jsou propojeny, pokud spolu sousedí hranou. Protože na některých kartičkách nejsme schopni zachytit souvislost luk, které vedou mezi cestou a městem, přidáváme znaky *M,L*. Dva oddělené regiony označené *L* tvoří na kartičce stejnou louku, pokud jsou propojené regiony označenými znakem *M* nebo *L* .

  `Karticka` v sobě také obsahuje údaj o pozici ve hře, rotaci ve hře a položené figurce. Každá instance má svůj číselný identifikátor a svůj grafický vzhled uložený v objektu implementujícím rozhraní `Obrazek`.

- Třída `Figurka` – tato třída popisuje datovou reprezentaci figurky (družiníka) v deskové hře Carcassonne. Při vzniku nové instance je pevně svázána s hráčem (majitelem) jednoznačně určeným pomocí výčtového typu `IdHrace`. Družiník se do hry umísťuje vždy na jeden z regionů vybrané kartičky. Podle regionu rozlišujeme různé druhy družiníků. Jsou to *rytíř (R)*, *mnich (M)*, *lupič (L)*, *sedlák (S)* a *nepřiřazený (X)*.

  Celkově je figurka v programu modelována pomocí návrhového vzoru *Model-View-Controller*, tato třída popisuje část *Model*.

- Třída `FigurkaView` – má na starost grafické zobrazení figurky v aplikaci. Figurku vykreslujeme jako barevný trojúhelník (barva závisí na `IdHrace`) s písmenem označujícím druh figurky (popsaným ve třídě `Figurka`).

- Třída `FigurkaController` – zajišťuje komunikaci mezi třídami `Figurka` a `FigurkaView`.

- Třída `GeneratorNahodnychCisel` – pseudonáhodný generátor, zajišťuje zamíchání kartiček a rozhodování náhodné umělé inteligence. Využívá návrhový vzor *Singleton*.

- Třída `Tah` – s instancemi třídy `Tah` pracují umělé inteligence. Objekty této třídy lze porovnávat pomocí atributu `ohodnoceni`.

- Třída `FXMLDocumentController` – v této třídě se nachází kód, který obsluhuje formulář definovaný v souboru `HraGui.fxml`. Zajišťuje tedy obsluhu všech událostí vyvolaných uživatelem a přizpůsobuje vzhled okna programu aktuální situaci. Přeposílá data od uživatelů (hráčů) do třídy `Hra`, kde dochází k jejich vyhodnocování a získané výsledky pak nazpět graficky zobrazuje hráčům v aplikaci . `FXMLDocumentController` využívá různé stavy hry, které definuje výčtový typ `Stav`.

- Třída `FXMLVysledkyController` – kód v této třídě obsluhuje formulář přítomný v souboru `VysledkyGui.fxml`. Jedná se nové okno, které na konci každé hry zobrazuje pořadí a body všech hráčů.

- Rozhraní `Hrac` – definuje základní sadu metod, kterou musí implementovat každý objekt, který se předává v konstruktoru třídy `Hra`. Nejdůležitější je metoda `Tah zahrajTah()`, ve které se nachází implementace algoritmu pro rozhodování dané umělé inteligence

- Třída `HracBezUI` – abstraktní třída, která jednoduchým způsobem implementuje metody rozhraní `Hrac`. Každý potomek této třídy musí předefinovat pouze metody `boolean isPC()` a `Tah zahrajTah()`. Všechny vytvořené umělé inteligence dědí od této třídy.

  Koncept hráče v deskové hře Carcassonne je také modelován pomocí vzoru *Model-View-Controller*. Jako *Model* uvažujeme právě libovolného potomka této třídy.

- Třída `HracView` – libovolný hráč má v aplikaci s uživatelským rozhraním i grafické zobrazení. V této třídě definujeme konkrétní podobu. Jedná se o panel zobrazující informace o jméně hráče, počtu bodů a volných figurek. Data o každém hráči se musí zobrazovat v reálném čase a automaticky se aktualizovat. O přenos aktuálních dat z *Modelu* do této třídy se stará třída `HracController`.

- Rozhraní `Zaznam` – obsahuje soubor metod, které můžeme využívat pro pořizování záznamů ze hry Carcassonne. Objekt implementující toto rozhraní je možné předat v konstruktoru třídy `Hra`.

- Třída `Zaznamnik` – základní implementace rozhraní `Zaznam`. V aplikaci se používá pro pořizování záznamu ve formě textového souboru. Shromažďuje informace na začátku hry, po každém položení kartičky a na konci hry.

- Třída `SadaPravidelUI` – abstraktní třída, poskytující šablonu pro psaní pravidel. Jako pravidlo můžeme označit libovolnou funkci, která vezme pozici ve hře Carcassonne a vrací číslo, které o dané pozici něco vyjadřuje. Tato pravidla využívají umělé inteligence založené na principu ohodnocující funkce. Každá z nich používá svoji sadu pravidel, ke které ještě přidává váhy určující důležitost konkrétního pravidla.

Každý potomek této třídy předefinovává metodu `ohodnot()`. Vstupem je kartička, tah a hráč v dané hře. Výstupem je, na základě posouzení všech pravidel vynásobených určenými váhami, číselné ohodnocení tahu.

- Třída `SadaPravidel` – potomek `SadaPravidelUI`. Implementuje všech 19 pravidel uvedených v tabulce 3.1.
- Třída `SadaVylepseni` – vylepšuje pravidla číslo 10 až 13 z tabulky 3.1. Na základě těchto pravidel se rozhoduje umělá inteligence popsaná v sekci 3.4.2.
- Rozhraní `Obrazek` – rozhraní s jedinou metodou `vykresli()`. Implementují ho třídy `HerniObrazek` a `KonzolovyObrazek`. Instance třídy `Obrazek` se předávají jako parametr při vytváření objektů třídy `Karticka`. Konkrétní implementace závisí na tom, zda se kartičky vytváří pro aplikaci s grafickým rozhraním nebo konzolovou aplikaci. Vyrábí se pomocí třídy `ObrazekFactory`.

11.4 Balíček `carcassonne.umeleInteligence`

Všechny umělé inteligence v tomto balíčku jsou potomky třídy `HracBezUI`. Všechny přepisují metodu `isPC()` tak, aby vracela `true`. V metodě `zahrajTah()` pak implementují svůj vlastní algoritmus na hledání nejlepšího možného tahu. Princip fungování je uveden v kapitole 3.

11.5 Balíček `evolution.vahyCarcassonne`

- Třída `NajdiVahy` – třída s metodou `main()`. Vytváří nový genetický algoritmus. Jeho nastavení lze měnit pomocí konfiguračního souboru, který se předá jako parametr při spuštění z příkazové řádky.
- Třída `CaTournamentSelector` – vytváří implementaci turnajové selekce (viz sekce 5.3).
- Třída `CaDiferencialniSelekce` – vytváří implementaci diferenciální selekce (viz sekce 5.3).
- Třída `CaUniformniKrizeni` – implementuje uniformní křížení (viz sekce 5.4).
- Třída `CaCrossover` – implementuje speciální n-bodového křížení (viz sekce 5.4).
- Třída `CaMutationClassic` – obsahuje implementaci klasické mutace (viz sekce 5.5).
- Třída `CaAdaGen` – obsahuje implementaci adaptivní mutace (viz sekce 5.5).
- Třída `CaFitnessVuciUI` – stará se o výpočet fitness funkce jedinců v genetickém algoritmu. Metoda výpočtu je uvedena v sekci 5.7.

- Třída `CaHraVeVlaknu` – instance této třídy vytvoří jedno nové vlákno, ve kterém sehraje s danými hráči předem určený počet her Carcassonne. Používá se při selekci a výpočtu fitness. Každý jedinec z populace je počítán ve svém vláknu, čímž dochází na vícejádrových procesorech k paralelizaci výpočtu a značné úspoře času.
- Třída `CaIndividual` – implementuje jedince představeného v sekci 5.2.
- Třída `CaLogger` – třída pro pořizování textových záznamů o průběhu evoluce.
- Třída `GeneratorMultiThread` – implementuje pseudonáhodný generátor používaný při evoluci. Obaluje třídu `ThreadLocalRandom` ze standardní knihovny. Tento typ generátoru je doporučován při programování ve více vláknech [16][17]. Nevýhodou ovšem je nemožnost opakování vygenerovaných čísel (nelze nastavit tzv. seed tohoto generátoru).

11.6 Komunikace objektů při průběhu hry

V aplikaci s grafickým rozhraním probíhá hra následovně. Uživateli se zobrazuje formulář s aktuální pozicí dané hry. Pomocí kliknutí myši nebo stiskem kláves generuje v tomto formuláři události. Ty obsluhuje třída `FXMLDocumentController`, která na základě jejich druhu rozpozná akci, kterou se chystá hráč provést.

Příkaz k provedení akce předá třída `FXMLDocumentController` herní logice (objektu třídy `Hra` a jejím pomocným třídám). Zde proběhne kontrola korektnosti a případně i zápis do datových struktur těchto tříd. `FXMLDocumentController` dostane informaci o úspěchu akce a vykreslí do formuláře aktuální podobu hry.

11.6.1 Příklad komunikace při pokládání figurky

Nyní uvedeme názorný příklad. Řekněme, že hráč, který je na tahu, již vybral a umístil do hry danou kartičku a právě se chystá položit figurku. Klikne tedy na určité místo ve formuláři. Ve třídě `FXMLDocumentController` se spustí vykonávání metody, která obsluhuje událost kliknutí na okno s kartičkami ve formuláři. Na základě stavu hry (uloženého v proměnné výčtového typu `Stav`) metoda ví, že hráč se chystá položit figurku.

Ze souřadnic místa kliknutí nejprve spočítá, zda hráč klikl na místo, kde leží poslední přidaná kartička (pouze na ni může být figurka umístěna). Když ano, převede souřadnice zvoleného místa (v pixelech) na relativní pozici kartičky (x, y) ve čtvercové mřížce a číslo regionu na kartičce (a, b).

Do objektu třídy `Hra` pošle požadavek na přidání figurky do regionu (a, b) kartičky na pozici (x, y). Z regionu `Hra` zjistí podklad kartičky a tedy i druh pokládané figurky. Otestuje korektnost v rámci pravidel a případně figurku přidá do hry. Zároveň herní logika nastaví figurce její druh a danému hráči také sníží počet jeho volných figurek o jedna. Do třídy `FXMLDocumentController` vrací informaci o úspěchu či neúspěchu. Skončí-li akce úspěchem, vytvoří třída `FXMLDocumentController` k figurce instanci třídy `FigurkaView` a instanci třídy `FigurkaController`. Následně si nechá figurku graficky vykreslit na komponentu `StackPane` a přidá ji do formuláře na místo kliknutí. Provede i aktualizaci panelu

s informacemi hráčů, kde se zobrazení nový počet zbývajících volných figurek vybraného hráče.

11.7 Přidání vlastní umělé inteligence

Obě varianty programu umožňují přidávat nové umělé inteligence. Pro přidání vlastní implementace stačí v balíčku `carcassonne.umeleInteligence` vytvořit třídu implementující rozhraní `Hrac` a naprogramovat do ní vybraný algoritmus. Přeloženou třídu (soubor `.class`) je dále nutné vložit do složky *bin/carcassonne/umeleInteligence* v místě, kde se nachází spustitelný soubor.

Závěr

V rámci bakalářské práce jsme vytvořili vlastní umělou inteligenci pro deskovou hru Carcassonne včetně implementace jejího herního prostředí. To umožňuje hru dvou až pěti hráčů na jednom počítači s možností nahradit libovolný počet z nich vybranou umělou inteligencí. Při implementaci jsme se soustředili na základní variantu hry bez rozšíření.

Práce prozkoumala různé možnosti návrhu umělé inteligence pro tuto hru. Na základě analýzy se pak věnovala především vývoji strategie posuzující pozice hry na základě pravidel, které tvořily ohodnocující funkci. Klíčovým úkolem bylo přiřadit těmto pravidlům vhodné váhy, což bylo vyřešeno za pomoci genetických algoritmů. Během evoluce bylo sehráno přes deset milionů her, ze kterých vzešlo nastavení vhodných vah. Implementovány byly i další umělé inteligence využívající jiné strategie.

Dále jsme v práci otestovali výkonnost všech těchto počítačem řízených hráčů. Zjistili jsme, že umělá inteligence vyšlechtěná pomocí genetického algoritmu dopadla z testovaných nejlépe. Její váhy jsme rozebrali z pohledu lidského hráče, díky čemuž máme představu o výhodné strategii pro tuto hru z pohledu počítače.

Naprogramovali jsme aplikaci s grafickým rozhraním a konzolovou aplikaci pro spouštění testů. Oba programy nabízí podporu pro přidání nových umělých inteligencí.

Další možnosti pokračování

V práci lze pokračovat především dalším vývojem umělých inteligencí. Inteligence vyvinutá pomocí genetických algoritmů nabízí snadnou možnost přidání nových pravidel. Prostor pro výzkum můžeme nalézt i v oblasti samotné evoluce, například vytvořením dalších operátorů nebo odlišným počítáním fitness funkce. V neposlední řadě se nabízí možnost implementace algoritmu založeného na jiných strategiích výběru tahu, než kterým se věnujeme v této práci.

Zajímavou myšlenkou může být přidání podpory pro další herní rozšíření, která pro Carcassonne vznikla. V tomto případě by však bylo nutné výrazně zasáhnout do samotné logiky hry, protože většina z vydaných rozšíření přináší nová pravidla, kartičky s novými motivy krajiny i nové figurky se speciálními významy. Zlepšit by se musela i umělá inteligence, aby byla schopna posuzovat nové situace vzniklé přidáním těchto rozšíření.

Literatura

[1] RUSSELL, Stuart J. a Peter. NORVIG. Artificial intelligence: a modern approach. 3rd ed. Upper Saddle River: Prentice Hall, c2010. ISBN 978-0136042594.

[2] MINDOK. Pravidla hry Carcassonne [online]. [cit. 2018-07-10]. Dostupné z: http://www.mindok.cz/userfiles/files/pravidla/8595558300105_50.pdf

[3] ORACLE. Java Platform Standard Edition 8 Documentation [online]. [cit. 2018-05-04]. Dostupné z: https://docs.oracle.com/javase/8/docs/.

[4] ORACLE. Building a JavaFX Application Using Scene Builder [online]. [cit. 2018-05-16]. Dostupné z: https://docs.oracle.com/javase/8/scene-builder-2/get-started-tutorial/jfxsb-get_started.htm#JSBGS101.

[5] Carcassonne - Tiles & Tactics [online]. [cit. 2018-06-05]. Dostupné z: https://store.steampowered.com/app/598810/Carcassonne_Tiles_Tactics/.

[6] JCloisterZone [online]. [cit. 2018-06-05]. Dostupné z: http://jcloisterzone.com/cs/.

[7] Carcassonne: Last Stend [online]. [cit. 2018-06-05]. Dostupné z: http://www.webgames.cz/carcassonne/7543-0/.

[8] Carcassonne [online]. [cit. 2018-06-05]. Dostupné z: http://marketplace.xbox.com/en-us/Product/Carcassonne/66acd000-77fe-1000-9115-d80258410840#.

[9] Carcassonne: Official Board Game -Tiles & Tactics [online]. [cit. 2018-06-05]. Dostupné z: https://play.google.com/store/apps/details?id=com.asmodeedigital.carcassonne.

[10] HEYDEN, Cathleen. IMPLEMENTING A COMPUTER PLAYER FOR CARCASSONNE [online]. 2009 [cit. 2018-04-25]. Dostupné z:https://project.dke.maastrichtuniversity.nl/games/files/msc/MasterThesisCarcassonne.pdf. Master Thesis. Maastricht University.

[11] MINDOK. Kompletní pravidla Carcassonne: základní hra & 9 rozšíření! [online]. [cit. 2018-07-02]. Dostupné z: https://www.mindok.cz/userfiles/files/pravidla/8595558301126_50.pdf.

[12] Graph 3D - Playground [online]. [cit. 2018-05-10]. Dostupné z: http://almende.github.io/chap-links-library/js/graph3d/playground/.

[13] MITCHELL, Melanie. An introduction to genetic algorithms. Cambridge, Mass.: MIT Press, c1996. ISBN 02-621-3316-4.

[14] FINCK, Steffen, Nikolaus HANSEN, Raymond ROS a Anne AUGER. Real-Parameter Black-Box Optimization Benchmarking 2010: Presentation of the Noiseless Functions [online]. [cit. 2018-07-05]. Dostupné z: http://coco.lri.fr/downloads/download15.02/bbobdocfunctions.pdf.

[15] PILÁT, Martin. A simple evolutionary algorithms framework used during the seminar on EAs [online]. [cit. 2018-07-04]. Dostupné z: https://github.com/martinpilat/evaTeaching.

[16] ORACLE. Class ThreadLocalRandom [online]. [cit. 2018-04-24]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadLocalRandom.html.

[17] VORONTSOV, Mikhail. Java Performance Tuning Guide [online]. [cit. 2018-04-24]. Dostupné z: http://java-performance.info/java-util-random-java-util-concurrent-threadlocalrandom.multithreaded-environments/.

Seznam obrázků

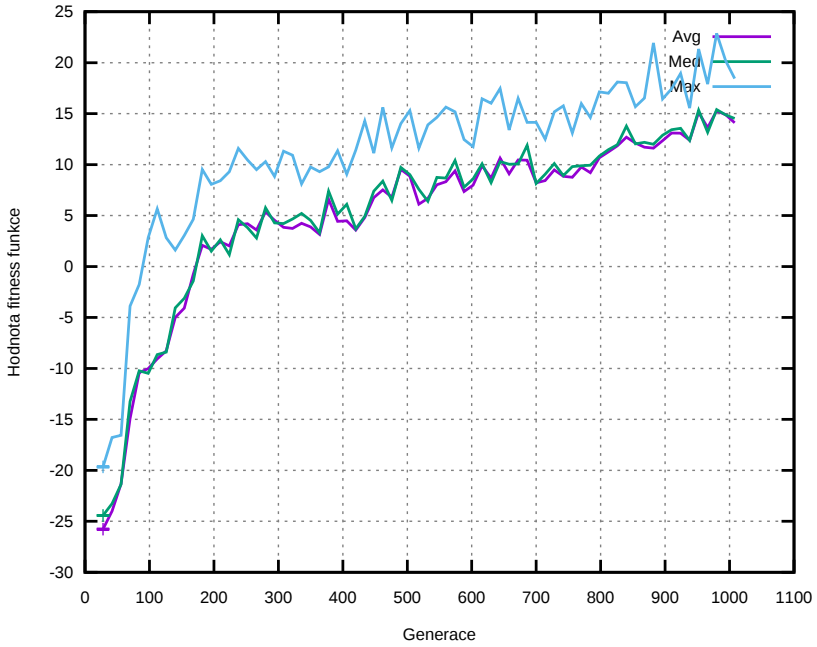
Seznam tabulek

Přílohy

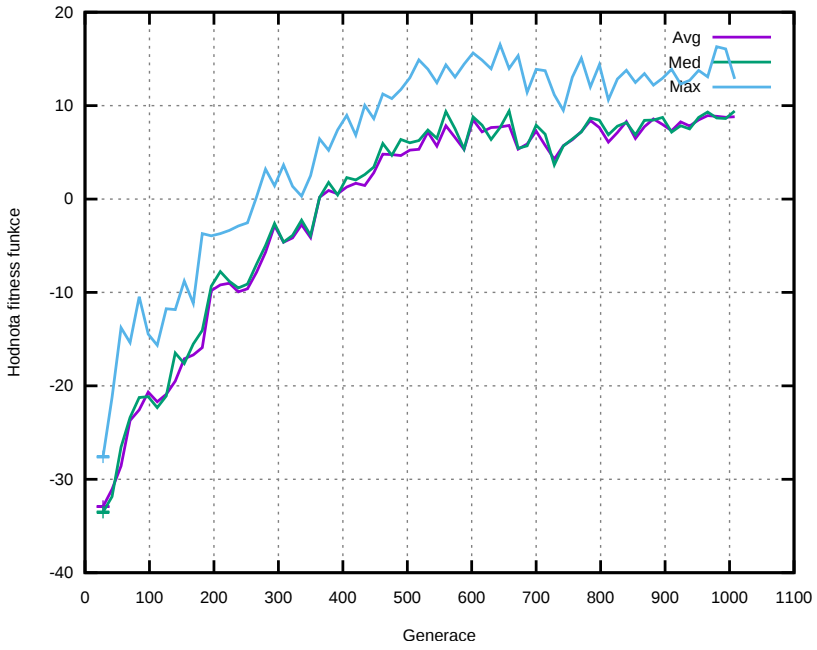
A. Výsledky evoluce – grafy

Zde uvádíme grafické výsledky běhů genetického algoritmu, které popisujeme v kapitole 6. Každý obrázek obsahuje graf se třemi křivkami. Křivka označená `Max` znázorňuje růst fitness funkce u nejlepšího jedince, křivka `Med` pak u prostředního jedince představujícího medián populace. Průměrná fitness populace je zachycena křivkou s označením `Avg`.

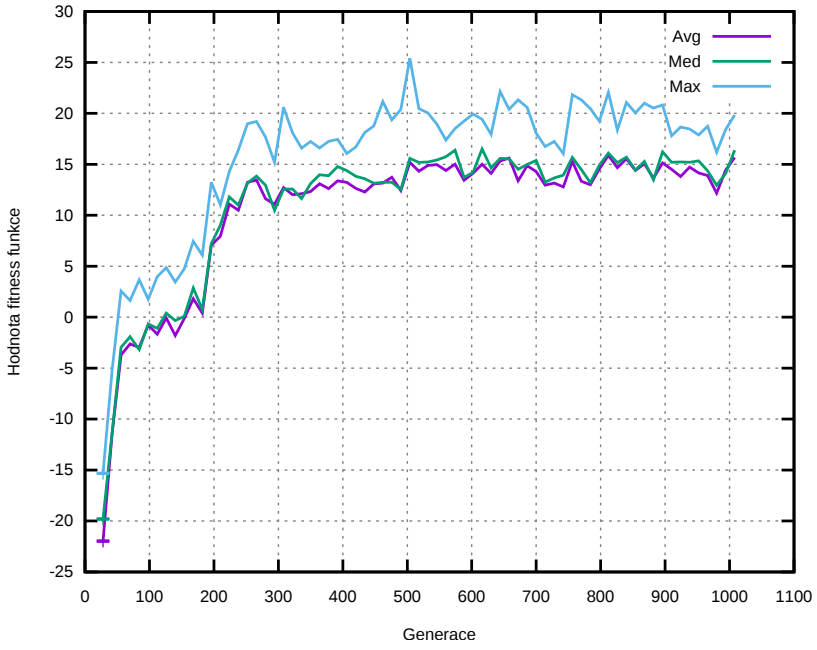
V popisku každého z obrázků je uveden seznam hodnot, které se změnily oproti výchozím hodnotám z tabulky 6.1.



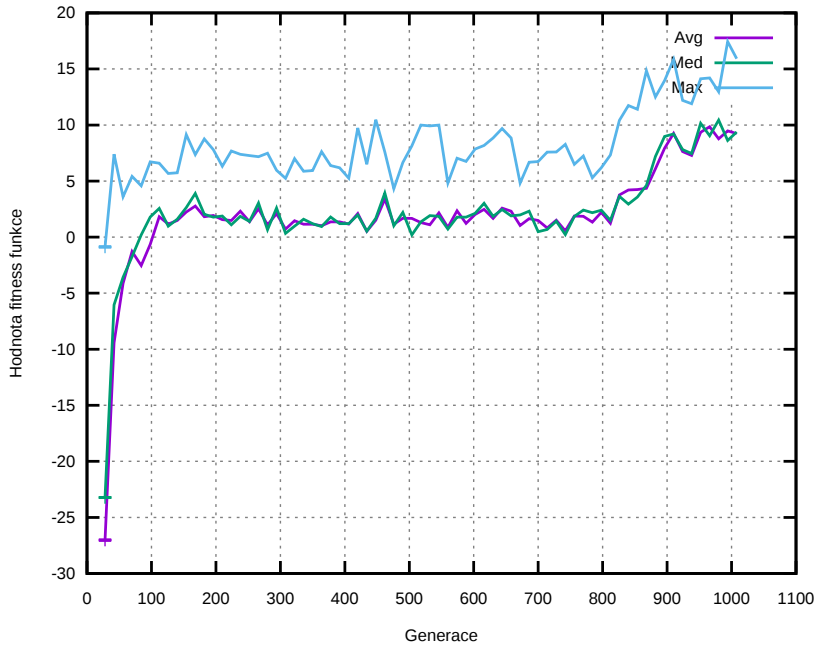
Obrázek A.1: Všechny hodnoty jsou stejné s výchozími.



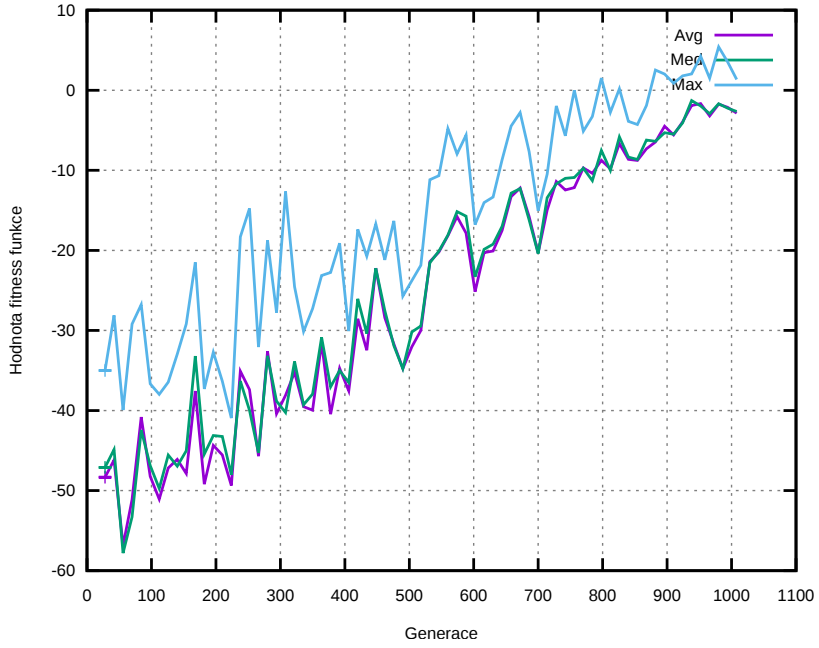
Obrázek A.2: faze = 5.



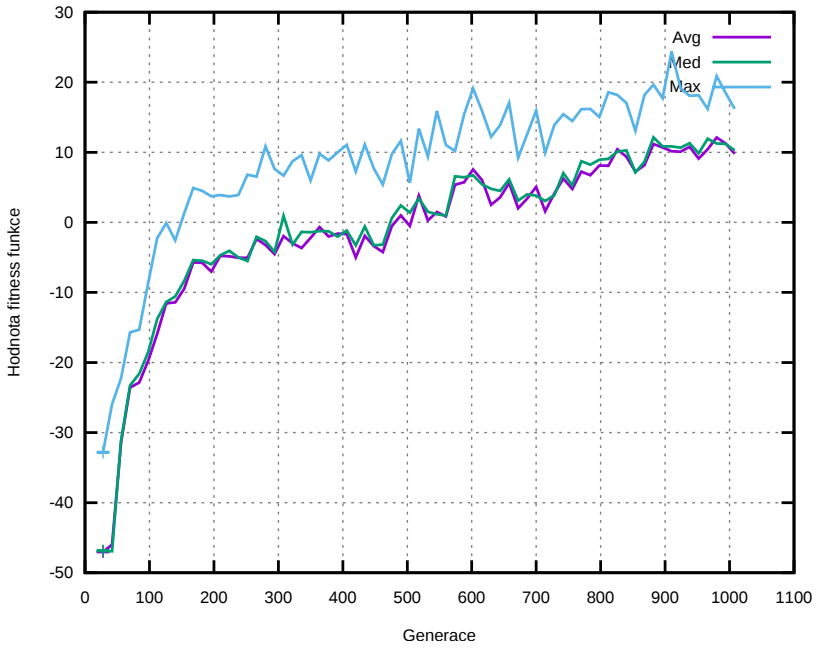
Obrázek A.3: popSize = 15, faze = 5; mutProbPerBit = 0,6.



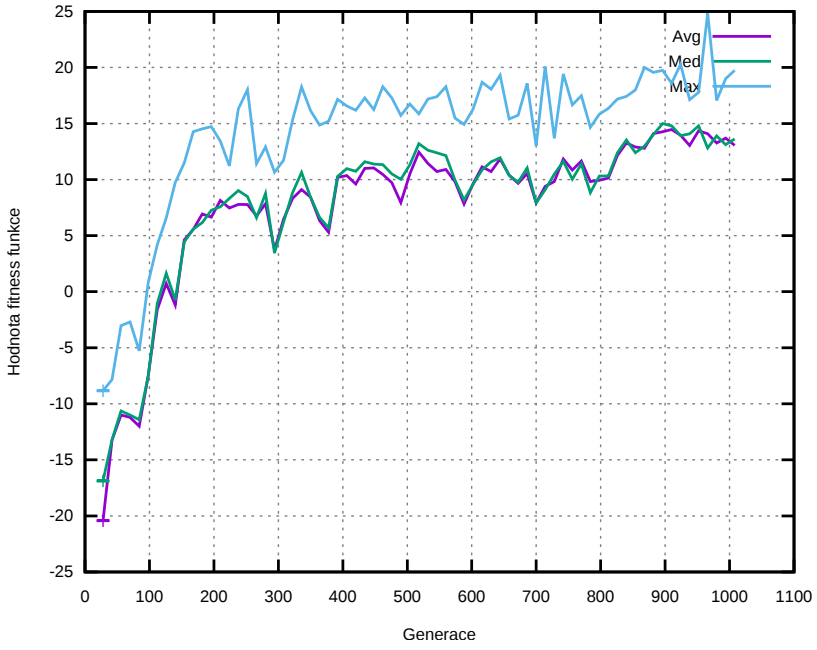
Obrázek A.4: mutProbPerBit = 0,5; matingGames = 6; environmentGames = 6; crossOp = uniformní.



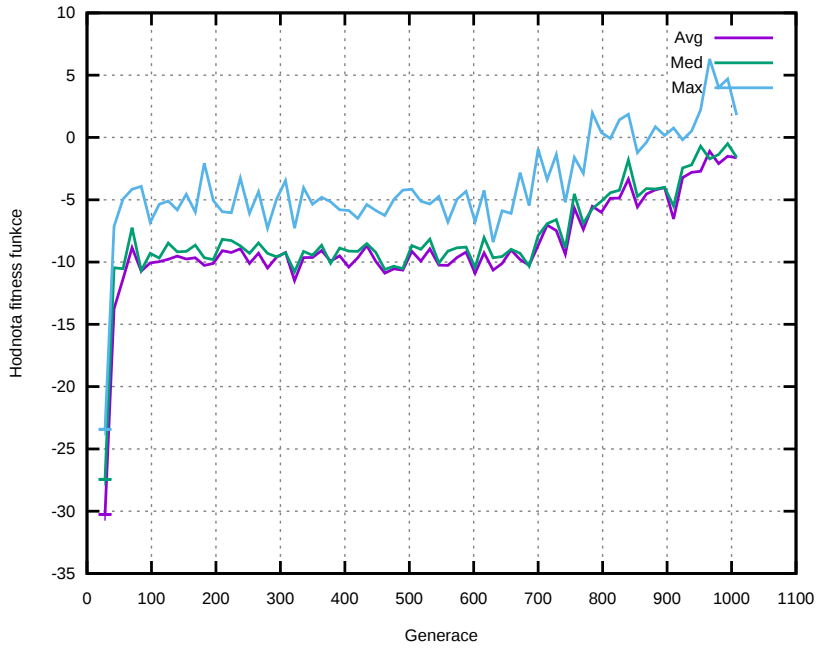
Obrázek A.5: xOverProb = 0.8; envSel = turnajová; crosOp = uniformní; replacement = jednoduché; vuciUI = UI 19/3 pr.



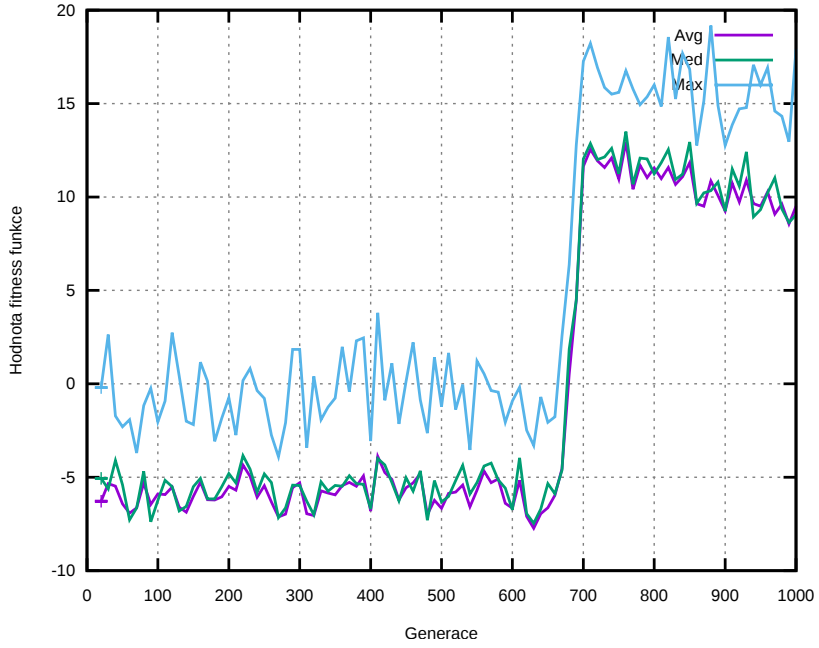
Obrázek A.6: players = 3.



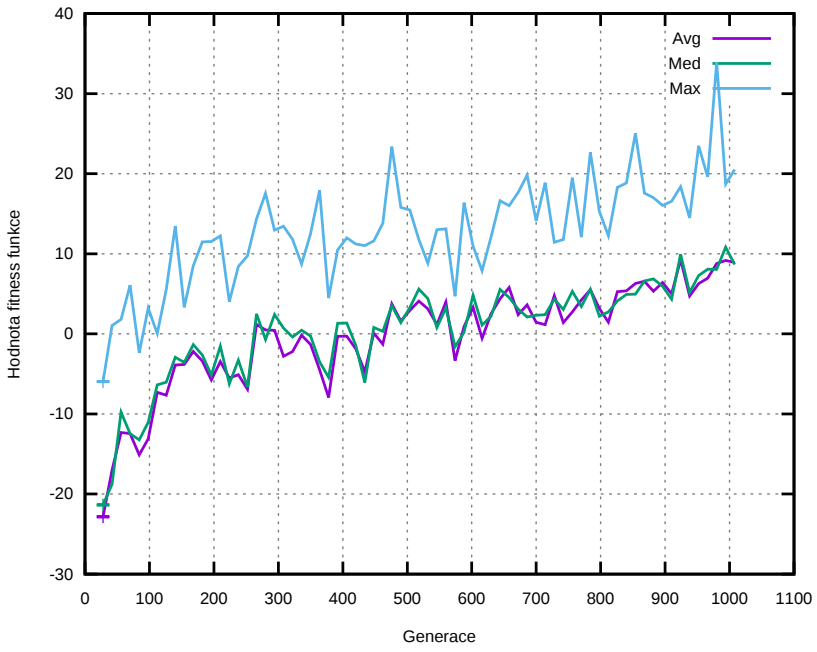
Obrázek A.7: vyvijenaUI = UIkFazi; crossOp = uniformní.



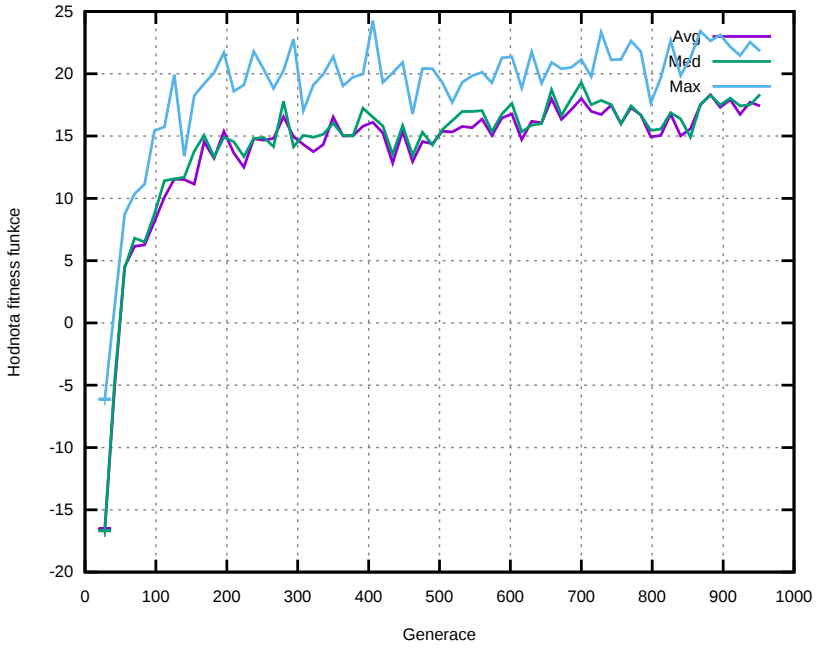
Obrázek A.8: popSize = 15; crossOp = uniformní; vuciUI = vůči nejlepší UI z předchozí konfigurace.



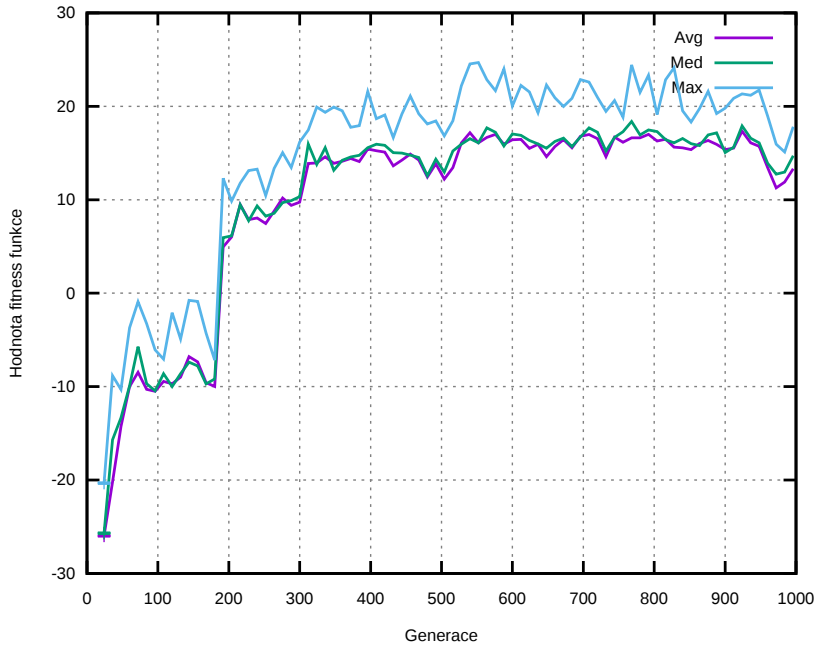
Obrázek A.9: vyvijenaUI = UIkFazi; popSize = 15; mutProb = 0,5; crossOp = uniformní.



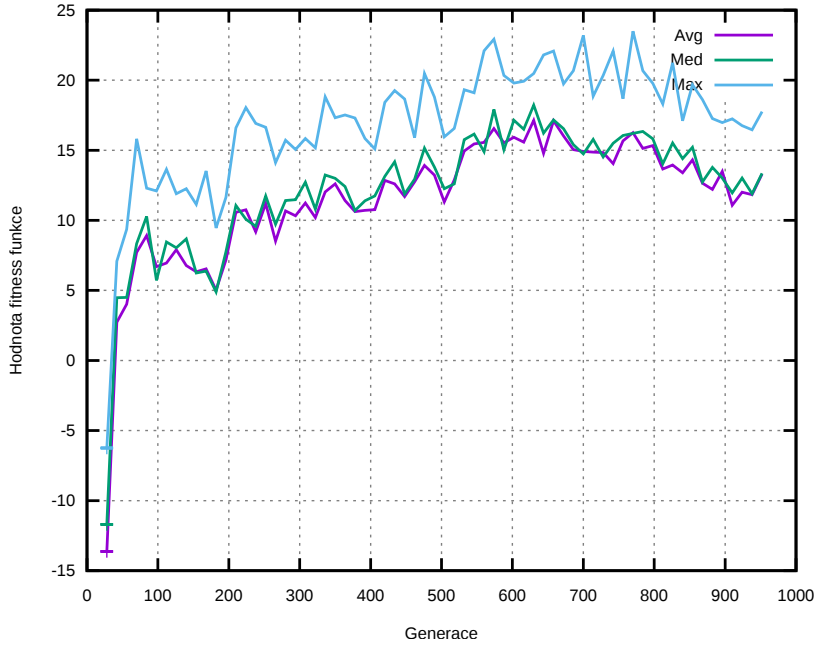
Obrázek A.10: players = 5.



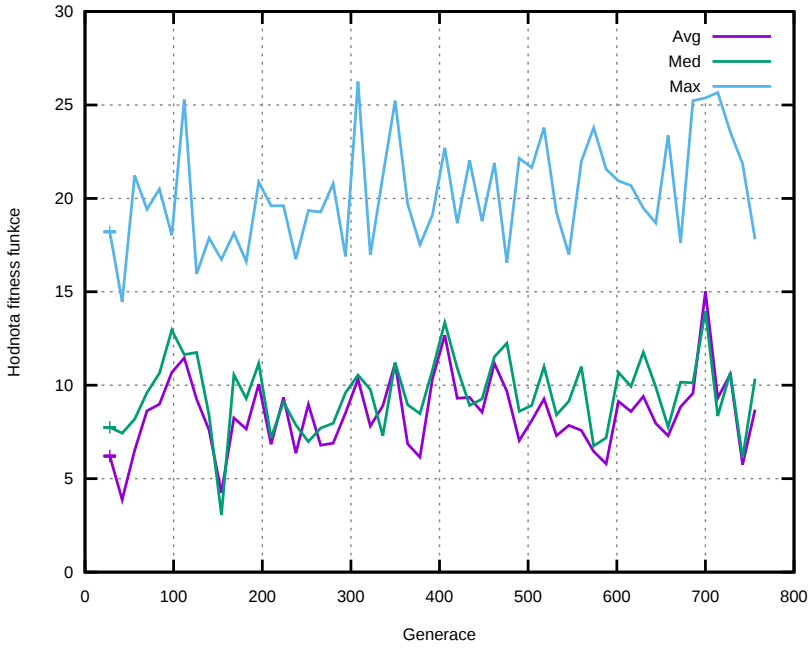
Obrázek A.11: popSize = 15; xOverProb = 0,9; mutProb = 0,9; mutProbPerBit = 0,1; matingGames = 10; environmentGames = 10; mutOp = klasická.



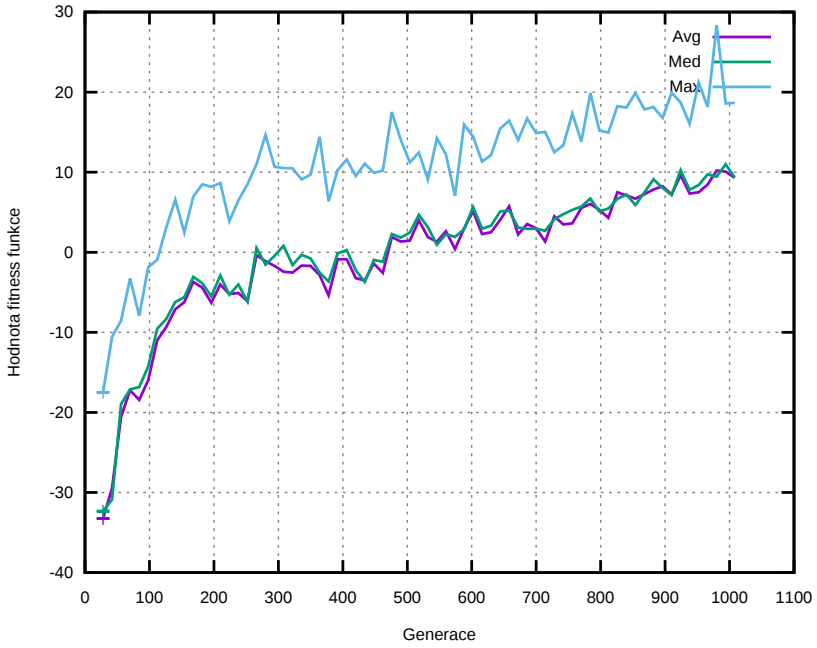
Obrázek A.12: popSize = 15; xOverProb = 0,65; mutProb = 0.5; mutProbPerBit = 0,8; environmentGames = 10.



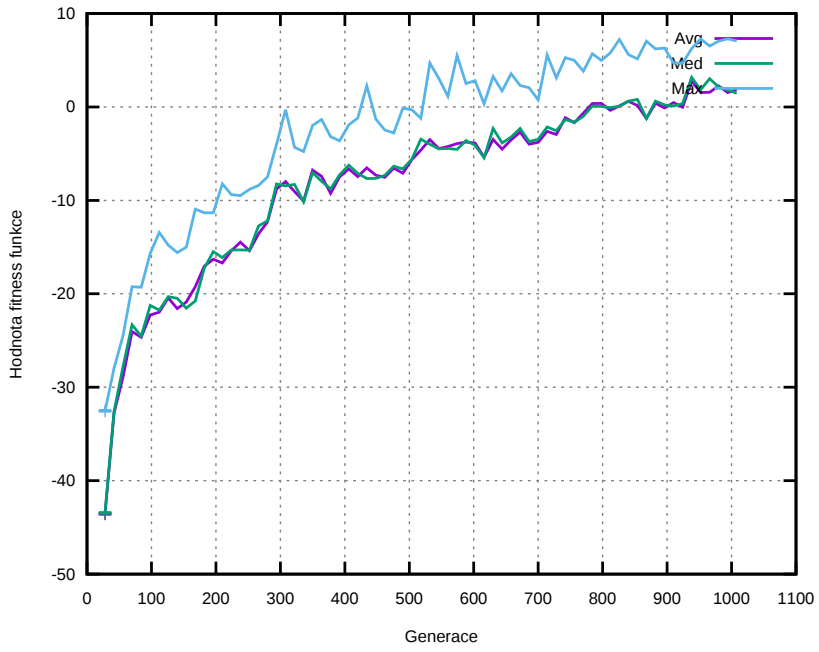
Obrázek A.13: popSize = 15; mutProbPerBit = 0,2; envSel = turnajová; mutOp = klasická; replacement = jednoduché; crossOp = uniformní.



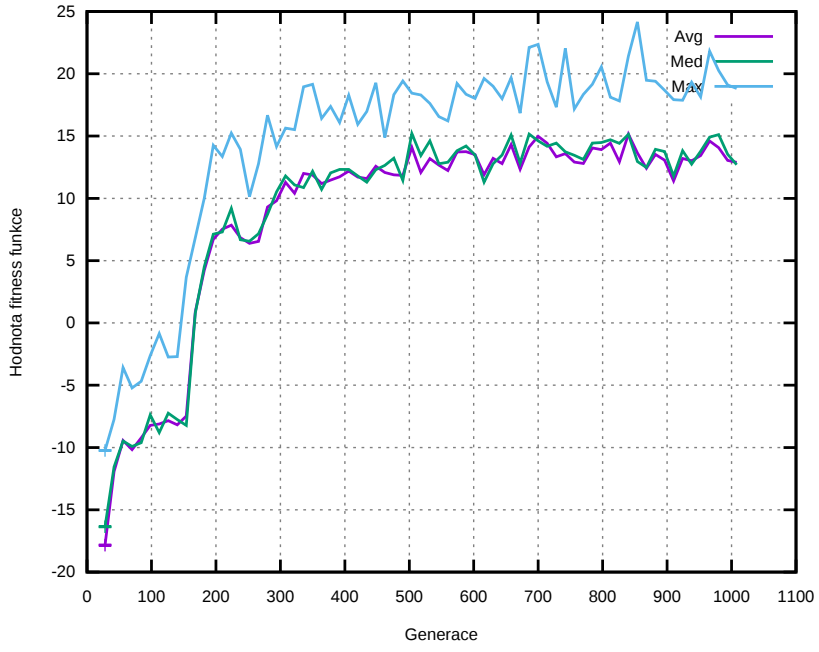
Obrázek A.14: maxGeneration = 750 – začínala s populací, kterou vytvořila předchozí konfigurace a rozšířila ji o pět jedinců.



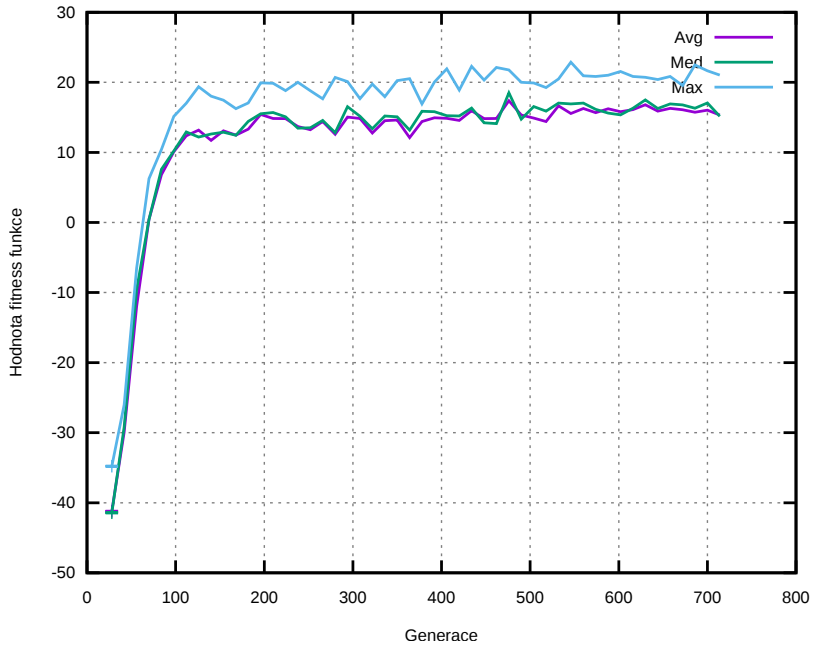
Obrázek A.15: players = 4



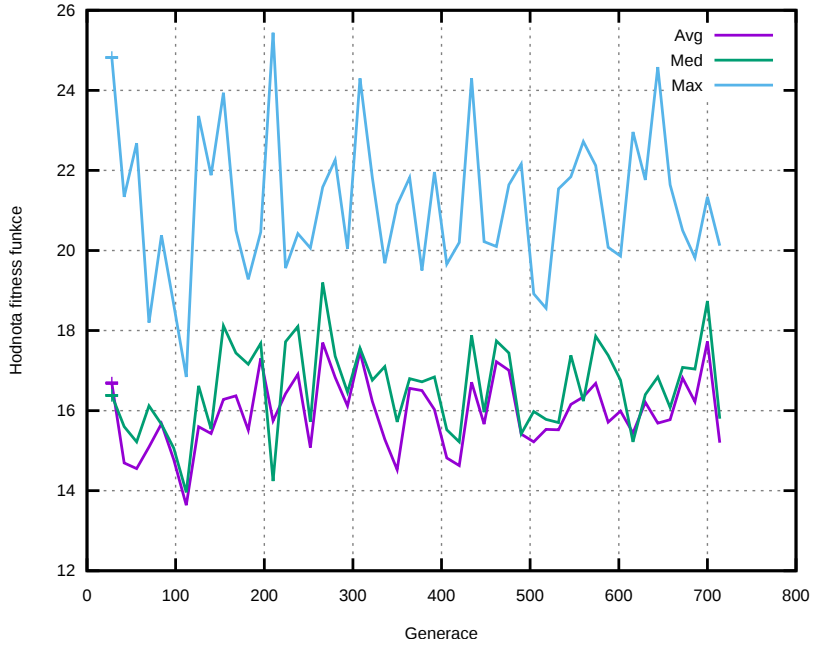
Obrázek A.16: faze = 5; mutProb = 0,5; mutProbPerBit = 0,2.



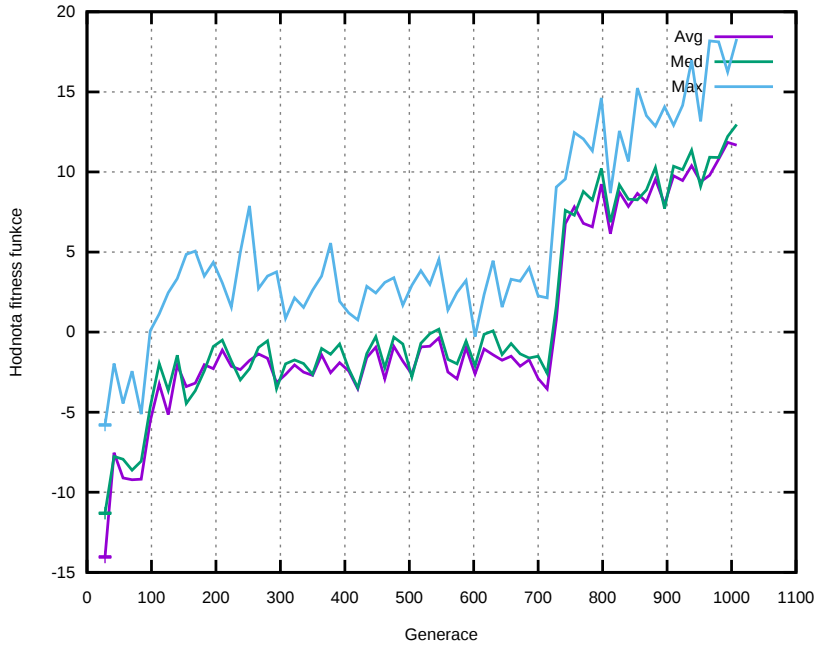
Obrázek A.17: mutProb = 0,9; mutProbPerBit = 0,1; xOverProb = 0,85; mutOp = klasická; crosOp = uniformní.



Obrázek A.18: popSize = 15; maxGeneration = 700; mutProbPerBit = 0,1; matingGames = 30; environmentGames = 24; mutOp = klasická.



Obrázek A.19: popSize = 15; maxGeneration = 700; mutProbPerBit = 0,1; mutOp = klasická – začínala s populací, kterou vytvořila předchozí konfigurace, bylo zvýšeno rozpětí jedince na interval $\langle -5, 30 \rangle$.



Obrázek A.20: popSize = 15; faze = 1; crossOp = uniformní.

B. Obsah elektronické přílohy

- **Složka `HraCarcassonne`** – obsahuje spustitelné soubory, zdrojové kódy a ukázkově vložené obrázkové i textové záznamy ze hry.
- **Složka `Dokumentace`** – obsahuje programátorskou dokumentaci vygenerovanou nástrojem Javadoc.
- **Složka `VýsledkyGA`** – obsahuje výsledky evoluce.
- **Složka `Testy`** – obsahuje soubory s nastavením a s výsledky všech testů umělých inteligencí provedených pomocí programu `CarcassonneTestyUI.jar`.
- **Složka `Ostatní`** – obsahuje zbývající podklady použité v této práci. Jsou rozdělené do složek s názvem kapitoly či sekce dle toho, kde byly využity.