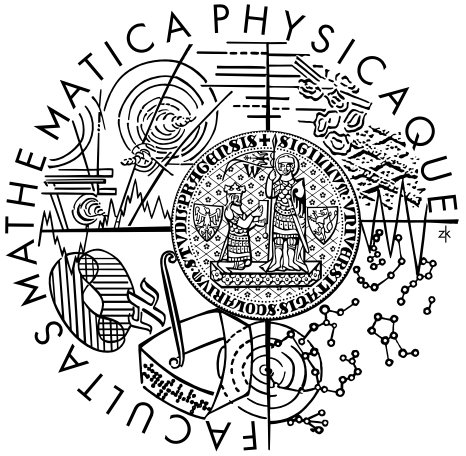
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jan Klůj

Alhambra

Katedra softwaru a výuky informatiky

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

Praha 2015

Rád bych na tomto místě poděkoval RNDr. Tomáši Holanovi, Ph.D. za všechny cenné rady a pomoc při vytváření této práce. Dále bych rád poděkoval své rodině za neustálou podporu.

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

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

V Praze dne 14. 5. 2015 Podpis autora

Název práce: Alhambra

Autor: Jan Klůj

Katedra: Katedra softwaru a výuky informatiky

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

Abstrakt: Bakalářská práce se zabývá implementací deskové hry Alhambra. Kromě implementace pravidel hry obsahuje program také grafické uživatelské prostředí. Hru může hrát dva až šest hráčů, kteří se střídají u jednoho počítače. Dále se v práci zabýváme umělou inteligencí, proti které je možné hrát. Logika rozhodování umělé inteligence je tvořena pomocí evolučního algoritmu a strojového učení.

Klíčová slova: Alhambra, hra, umělá inteligence, evoluční algoritmus

Title: Alhambra

Author: Jan Klůj

Department: Department of Software and Computer Science Education

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

Abstract: The bachelor thesis deals with the implementation of the board game Alhambra. Besides of the implementation of the game rules, program includes also a graphical user interface. The game can be played by two to six players who take turns at one computer. Further we deal with an artificial intelligence, against which we can play. Decision logic of the artificial intelligence is made by using the evolution algorithm and machine learning.

Keywords: Alhambra, game, artificial intelligence, evolution algorithm

Obsah

Úvod

Alhambra je desková hra, která vznikla v roce 2003, a jejím autorem je Dirk Henn [1].

Hráči postupně dobírají peněžní karty, za které následně mohou kupovat dlaždice budov a ty staví ve svém herním poli. Hry se může účastnit dva až šest hráčů a tito hráči se postupně střídají. Vyhrává ten, kdo postaví nejhonosnější Alhambru.

Rozhodli jsme se vytvořit vlastní softwarovou implementaci této hry podle její skutečné předlohy, která by byla hratelná pomocí grafického prostředí. Bakalářská práce se zároveň zabývá tvorbou umělé inteligence, proti které budeme moci tuto hru hrát. Pro tvorbu umělé inteligence použijeme genetický (evoluční) algoritmus a pomocí strojového učení se tato umělá inteligence „naučí“ hrát.

Práce je členěna do několika kapitol. V kapitole 1 se věnujeme pravidlům hry, jejich dodefinování a změnám oproti původní hře. Kapitola 2 popisuje implementaci hry a zejména komunikaci mezi komponentami. Zmíníme se zde i o nejdůležitějších třídách. Kapitola 3 se zabývá způsobem rozhodování umělé inteligence a uvádí výsledky evolučního algoritmu. Následuje kapitola 4, ve které se vrátíme k implementaci, nikoliv však hry jako takové, ale reprezentaci umělé inteligence a implementaci jejího rozhodování. Na závěr nesmí chybět kapitoly 5 a 6 popisující uživatelskou a programátorskou dokumentaci.

Implementace hry již existuje přímo od vydavatelů originální hry, a to například pro systémy Android [2] nebo iOS [3]. My vytvoříme novou nezávislou implementaci. Bakalářská práce nevlastní autorská práva na samotnou hru (pravidla a obrázky hry) a slouží pouze ke studijním účelům.

1. Hra

1.1 Originální pravidla hry, jejich neúplnost a úpravy

Originální pravidla hry jsou převzata ze skutečné deskové hry Alhambra. Tato originální pravidla hry obsahují ne zcela ošetřené situace, které mohou ve hře nastat. Pravidla nedefinují chování v takových situacích. Cílem následujících odstavců je tyto situace popsat.

Pokud hráči odeberou všechny karty a počet karet použitých k zaplacení budov je nulový, nelze provést tah sebrání karet a hráč musí zvolit jiný tah.

Nastane-li situace, kdy hráč nemůže odehrát tah (ani jeden ze tří typů tahu není možné provést), tak tento hráč tah ztrácí. Příkladem této situace je, když hráč po startu hry neustále bere nové peněžní karty. Až všechny peněžní karty budou vyčerpány, musí udělat jeden ze dvou zbývajících tahů – koupit budovu, nebo přestavět svou Alhambru. Druhá varianta není možná, neboť ještě daný hráč nepostavil žádné budovy. Zvolí tedy tah pro koupi budovy a zaplatí ji přesně. Situace se opakuje, neboť nově koupené dlaždice budovy lze umisťovat až po dokončení všech tahů daného hráče (v řadě). Hráč takto zaplatí přesně všechny čtyři budovy, které jsou k dispozici na trhu a získává pátý (a svůj v řadě poslední) tah. Hráč tento tah nemůže zahrát, tak jej ztrácí a pouze umístí koupené dlaždice budov z předchozích tahů.

Implementace rovněž obsahuje změny pravidel oproti originální hře:

- algoritmus výběru pořadí hráčů na začátku hry – hráči jsou sestupně setříděni třídícím algoritmem podle počtu karet, v případě stejného počtu podle součtu hodnot karet. Pokud i tento součet hodnot karet je u dvou (či více) hráčů stejný, při porovnávání dvou hráčů v třídícím algoritmu je jako menší označen první hráč z dvojice;
- skóre jednolivých hráčů není zobrazováno pomocí bodovací desky;
- na samotném konci hry jsou zbývající budovy na trhu přenechány hráčům podle jejich karet, které vlastní (dle pravidel). Tyto budovy nejsou však přenechány naráz, ale postupně v takovém pořadí, v jakém jsou na trhu umístěny. Po každém udělení budovy musí hráč nejprve budovu umístit a až pak se vyhodnocuje, komu z hráčů připadne další z budov na trhu.

2. Implementace hry

2.1 Zvolené technologie

Jako cílovou platformu naší aplikace jsme zvolili systém Windows. Následně pro implementaci byl zvolen programovací jazyk C#. Program využívá již předpřipravené knihovny .NET frameworku 4.0. Tato verze je i minimální nutná pro spuštění aplikace, neboť jsou použity třídy jako `Tuple` nebo `Task`. Pro grafické prostředí jsme použili technologii Windows Forms. Tvorba celé aplikace probíhala za pomoci vývojového nástroje Microsoft Visual Studio 2013. Program nezaručuje správný běh na ostatních platformách, nicméně je možné pro ostatní platformy (například Linux) využít projekt Mono. Při samotném programování byla použita dokumentace MSDN jazyka C# a platformy .NET [4].

2.2 Návrh

Aplikace využívá modifikaci architektonického vzoru model-view-controller. Ten se primárně používá pro webové aplikace, nicméně můžeme jeho modifikaci aplikovat i na naši hru:

- Model — představuje část jádra hry implementující pravidla hry jako takové. Zahrnuje reprezentaci aktuálního stavu hry (budovy, karty, ...), seskupuje data o hře a pro daný tah dokáže vyhodnotit, zda je v souladu s pravidly hry;
- Controller — logická jednotka, která má na starosti řízení hry a obsahuje hlavní smyčku hry. Společně s komponentou Model tvoří jádro hry, které je schopno fungovat i bez grafického prostředí. Nerozlišuje, zda tah, který zpracovává, pochází od reálného hráče, či umělé inteligence;
- View — grafické prostředí (GUI) zobrazuje aktuální stav hry a na základě činnosti uživatele komunikuje s jádrem hry. Tvoří „prostředníka“ v komunikaci mezi uživatelem a jádrem hry. Rovněž pomocí něj uživatel rozhoduje o počátečním nastavení hry.

Takový návrh jsme zvolili proto, abychom mohli vytvářet jednotlivé části na sobě co nejméně závislé. Příkladem může být situace, kdy potřebujeme, aby hra počítačových hráčů fungovala i bez jakéhokoliv GUI a mohli jsme hry spouštět v rámci evolučního algoritmu.

Dále v této kapitole rozebereme hlavní části implementace aplikace, včetně stručného popisu některých tříd. Detailnější popis technické části nalezneme v kapitole Programátorská dokumentace.

2.3 Jádro hry

Jádro hry obsahuje tyto nejdůležitější části:

- třída `Game` — obsahuje datové položky hry a poskytuje nejdůležitější metody pro práci s nimi;
- třída `Move` — abstraktní třída poskytující základ reprezentace každého specifického typu tahu. Jejími potomky jsou `TakeCardMove`, `BuyBuildingMove` a `RebuildMove` reprezentující každý specifický typ tahu;
- třída `MoveChecker` — poskytuje metody pro kontrolu daného tahu, zda je v souladu s pravidly hry;
- třída `Controller` — obsahuje hlavní smyčku hry a poskytuje metody pro počítání skóre, ukládání a načítání hry;
- třída `Player` — instance této třídy reprezentují jednotlivé hráče (buďto reálné hráče, nebo počítačové hráče);
- třída `Building` — reprezentuje jednotlivé budovy;
- struktura `Card` — seskupuje hodnotu a typ karty;

2.3.1 Ukládání a logy her

Hry je možné ukládat a ukládají se pomocí binární serializace. Serializuje se instance třídy `Game` (a její reference) kromě instance grafického rozhraní. Hra je uložena do souboru `<jmeno>.data`, kde `jmeno` zadá uživatel. Výchozí adresář pro uložení je `AppData/Alhambra/Save`, kde `AppData` adresář pro data aplikací.

V průběhu her se automaticky vytvářejí logy, které textově zobrazují průběh hry. Tyto logy jsou ukládány do souborů `YYMMDD_HHMMSS.log`, kde jméno představuje datum a čas. Adresář pro uložení je stejný jako pro ukládání her, koncová složka cesty je `Logs`.

2.4 Grafické uživatelské prostředí

Tvoří jej třídy z knihovny Windows Forms, které jsou doplněny o vlastní implementace tříd, které jsou potomky třídy `Control`. Takové třídy mají v názvu příponu `CC` (Custom Control), např. `BuildingCC`. Nejdůležitějšími částmi GUI jsou:

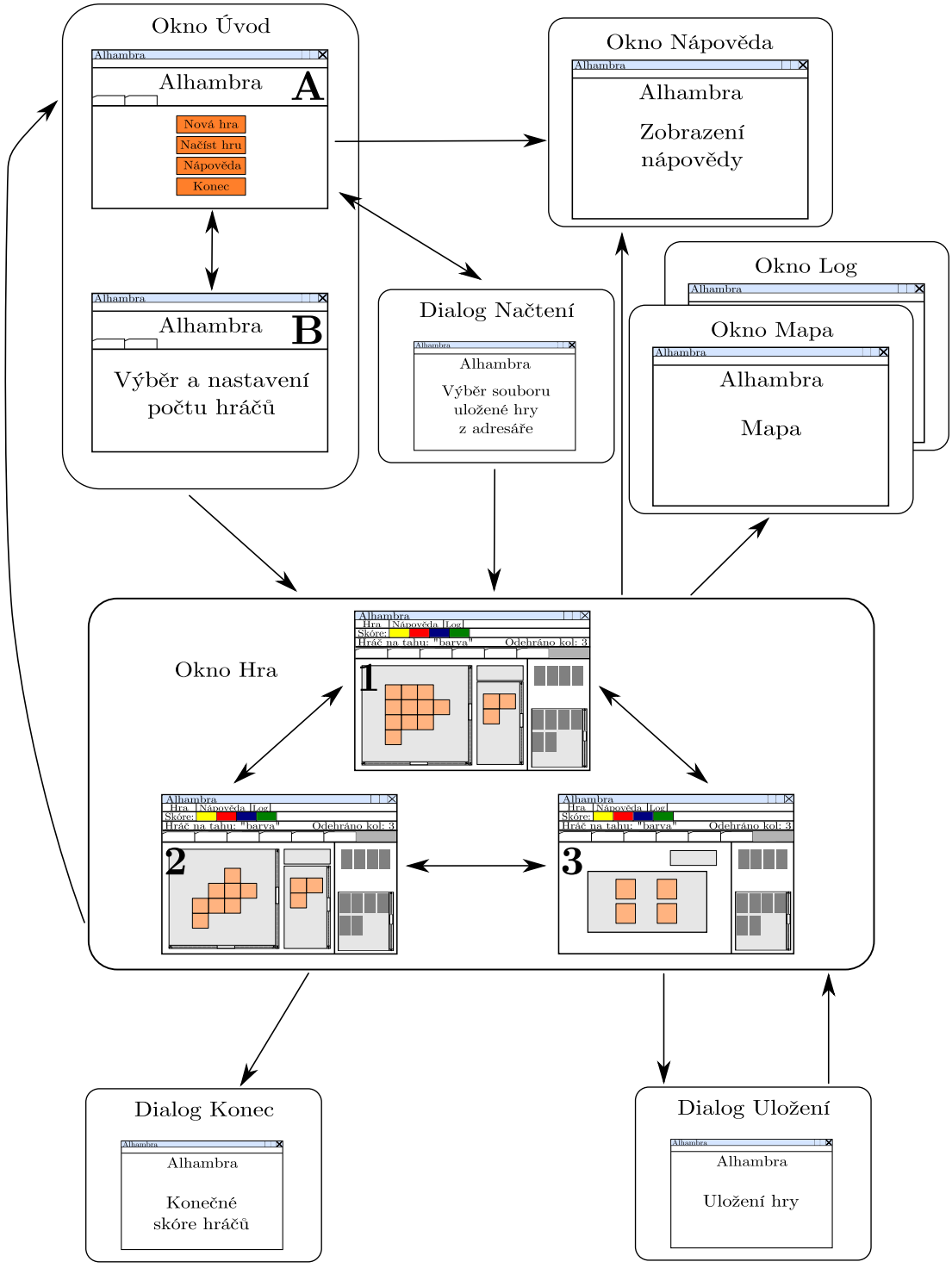
- třída `BuildingCC` — graficky reprezentuje jednotlivé budovy ve hře a poskytuje pro ně obsluhu uživatelských událostí (`MouseDown`, `MouseClick`, atp.);
- třída `CardCC` — analogie k `BuildingCC` pro karty;
- třída `IntroductionForm` — tvoří úvodní okno aplikace. Obsahuje obsluhu tlačítek a rozbalovacích seznamů, pomocí kterých uživatel vytvoří počáteční nastavení hry;
- třída `GameForm` — okno ve kterém probíhá samotná hra. Obsahuje proměnné, ve kterých se udržují jednotlivé součásti tohoto okna;

- třída `MapCC` — tvoří mapu pro uživatele, do které bude umisťovat své dlaždice budov;
- třída `PlayersCardsCC` — ovládací prvek, pomocí kterého uživatel pracuje se svými kartami;
- třída `PostponedBuildingsCC` — tvoří odkládací plochu pro dlaždice budov;
- třída `ScoringCardInfoForm` — poskytuje informační okno zobrazující počet získaných bodů hráčů. Je zobrazováno po každém sčítání bodů.

2.4.1 Diagram stavů grafického prostředí

Diagram 2.1 popisuje, v jakých stavech se může aplikace nacházet. Jednotlivé stavy jsou tvořeny následujícím způsobem:

- *Okno Úvod* — instance třídy `IntroductionForm`, která je potomkem `Form`. Je prvním zobrazeným oknem po spuštění aplikace;
- *Dialog Načtení* a *Dialog Uložení* — pro tyto dialogy jsou použity třídy `OpenFileDialog` a `SaveFileDialog` ze jmenného prostoru `System.Windows.Forms`;
- *Okno Nápověda* — je reprezentováno oknem výchozího internetového prohlížeče. Samotná nápověda je napsána v jazyce HTML;
- *Okno Log* — instance třídy `LogForm`. Textově reprezentuje dosavadně odehrané tahy;
- *Okno Mapa* — instance třídy `Form`, do které je vložena vlastní control `MapCC` reprezentující mapu s postavenými dlaždicemi budov daného hráče;
- *Okno Hra* — instance třídy `GameForm`. Okno se může nacházet ve třech různých stavech:
  1) zobrazení vlastní mapy s postavenými budovami;
  2) zobrazení mapy cizího hráče;
  3) zobrazení herního trhu;
- *Dialog Konec* — pouze `MessageBox` zobrazující informace o počtu bodů a konci hry.



Obrázek 2.1: Diagram stavů grafického prostředí

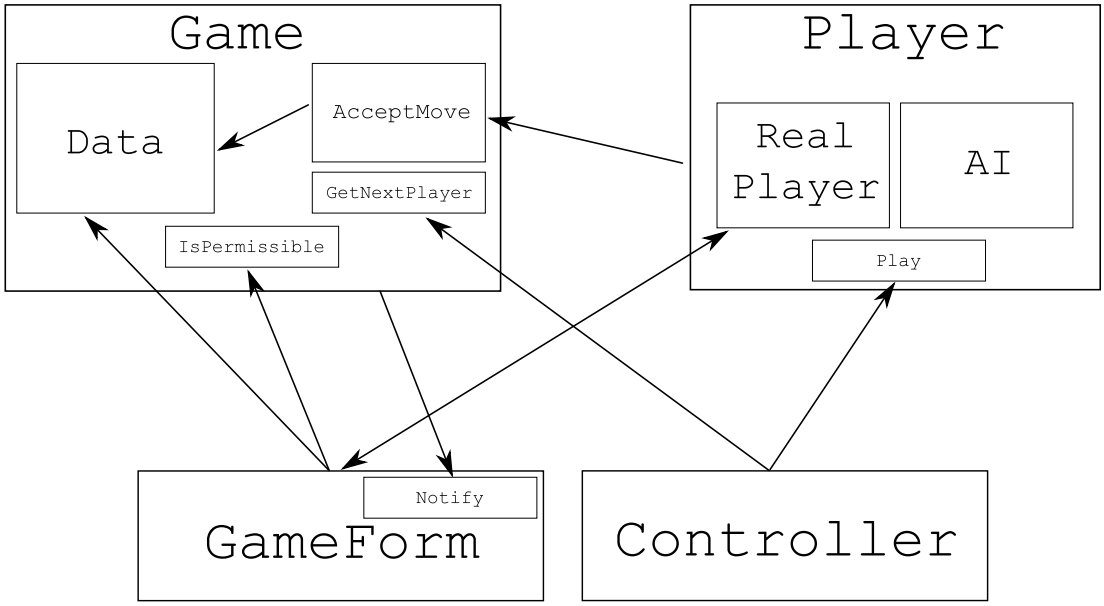
2.5 Komunikace mezi komponentami

`Controller` obsahuje hlavní cyklus hry. V tomto cyklu zkontroluje, zda byl dokončen tah hráče. Pokud ano, tak následně rozhodne, kdo z hráčů je další na tahu prostřednictvím metody `GetNextPlayer`, a zjistí, zda je daný hráč tvořen umělou inteligencí, nebo se jedná o reálného hráče. Pokud se jedná o umělou inteligenci, zavolá na instanci zjištěného hráče metodu `Play`, pomocí které bude odehrán patřičný tah. V opačném případě neprovede žádnou akci a aplikace je čistě řízena událostmi GUI.

Každý hráč předá informaci o svém právě provedeném tahu ke zpracování třídě `Game`, a to zavoláním metody `AcceptMove`. Tato metoda vykoná samotnou změnu dat na základě hráčem předaného tahu. Jakmile bude dokončen tah hráče, herní okno `GameForm`, pokud existuje, je notifikováno o změně datových položek hry prostřednictvím metody `Notify`. Herní okno se na základě těchto změn překreslí.

Před každým provedeným tahem probíhá kontrola, zda je daný tah v souladu s pravidly hry. Z GUI tato kontrola probíhá prostřednictvím volání metody `IsPermissible`.

Obrázek 2.2 znázorňuje výše popsanou komunikaci.



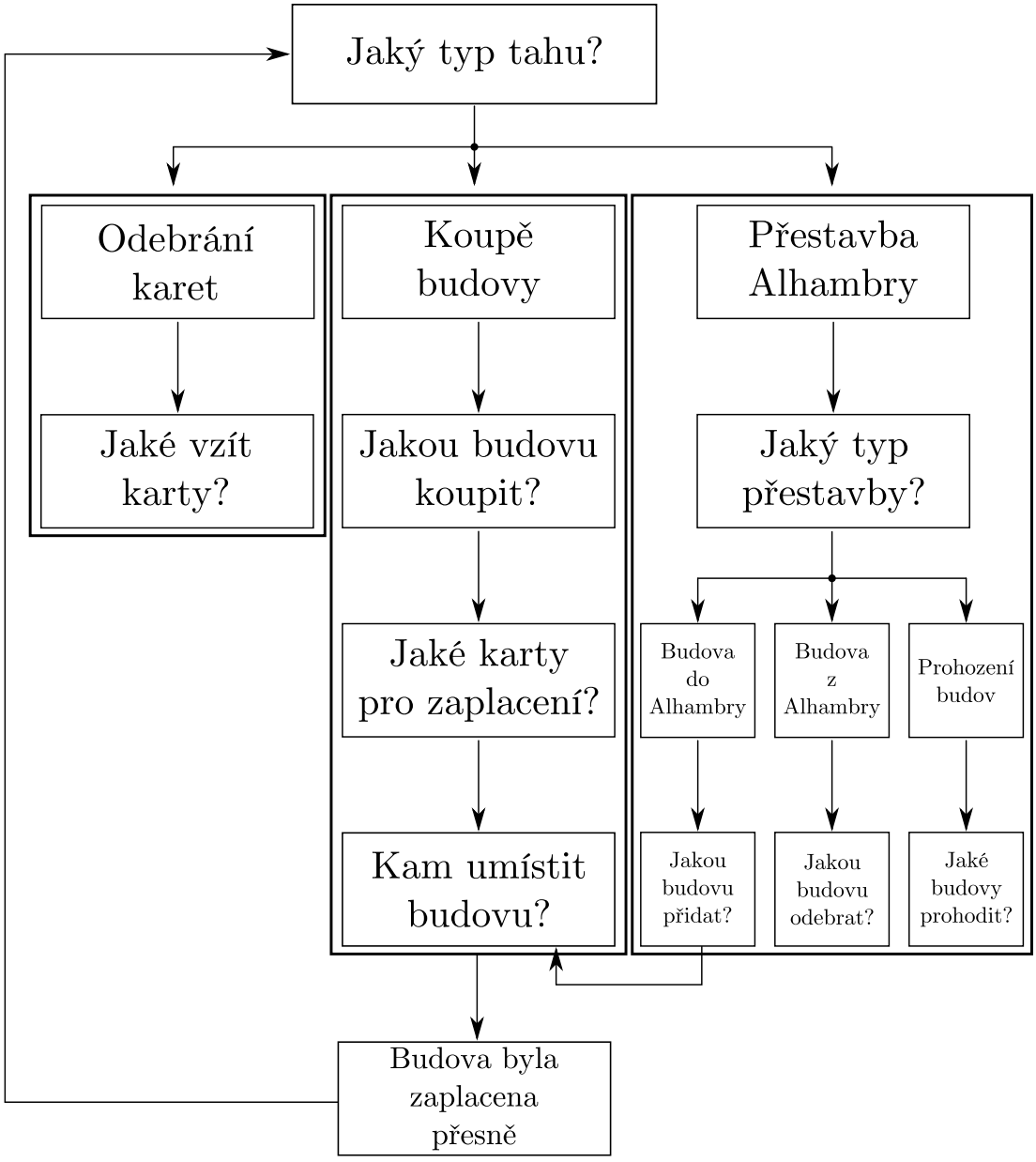
Obrázek 2.2: Komunikace mezi komponentami

3. Umělá inteligence

V této kapitole rozebereme chování a rozhodování počítačového hráče a vývoj pomocí evolučního algoritmu.

3.1 Rozhodování

Výstup umělé inteligence tvoří vybraný tah, který v dané situaci má zahrát počítačový hráč. Rozhodování o tomto tahu probíhá v několika krocích, které můžeme reprezentovat tzv. diagramem rozhodování:



Obrázek 3.1: Diagram rozhodování umělé inteligence

Vidíme, že výsledný tah závisí na dílčích, lokálních rozhodnutích. Pro každé lokální rozhodování máme možnosti, mezi kterými můžeme vybírat. Například

pro rozhodování *jaké vzít karty* máme šestnáct podmnožin karet na trhu, které můžeme vzít (některé z podmnožin samozřejmě nejsou v souladu s pravidly hry).

3.1.1 Lokální rozhodování

Pro lokální rozhodování vždy uvažujeme všechny možnosti, mezi kterými lze danou situaci vyřešit:

- *jaký typ tahu* — máme pouze tři možnosti;
- *jaké vzít karty* (tah odebrání karet) — na trhu jsou celkem čtyři karty, máme tedy šestnáct podmnožin karet, které teoreticky můžeme odebrat. Z toho je určitý počet podmnožin proti pravidlům (odebírání nula karet atd…);
- *jakou budovu koupit* (tah koupení budovy) — máme až čtyři budovy na trhu, to nám tedy dá i čtyři možnosti pro tento výběr;
- *jaké karty pro zaplacení* (tah koupení budovy) — zde můžeme mít až několikanásobně více možností, mezi kterými se rozhodujeme, protože uvážíme všechny podmnožiny karet, které daný hráč vlastní (opět uvažujeme pouze ty podmnožiny, které jsou v souladu s pravidly hry);
- *kam umístit budovu* (tah koupení budovy) — uvažujeme pozice, kam lze danou budovu umístit (dle pravidel). Počet takových pozic záleží na dosavadní velikosti postavené Alhambry;
- *jaký typ přestavby* (tah přestavba Alhambry) — tři možnosti;
- *jakou budovu přidat* (tah přestavby – budova do Alhambry) — tolik možností, kolik má daný hráč odložených budov;
- *jakou budovu odebrat* (tak přestavby – budova z Alhambry) — počet možností odpovídá počtu budov, které lze odebrat z herní mapy;
- *jaké budovy prohodit* (tah přestavby – prohození budov) — počet možností odpovídá počtu všech dvojic budov, které lze prohodit.

Tato lokální rozhodování můžeme vidět výše uvedeném obrázku 3.1.

3.1.2 Kvalita a váhy

Kritériem lokálního rozhodování nazveme jev, u kterého určujeme, zda pro danou možnost v lokálním rozhodování nastal či nikoliv. Každé kritérium má svou **váhu**, což je hodnota z intervalu $(0, 1)$. Pro různá lokální rozhodování máme různá kritéria.

Kvalitou rozumíme součet vah kritérií lokálního rozhodování, které pro danou možnost v lokálním rozhodování nastaly.

Výsledek daného lokálního rozhodování určíme tak, že pro všechny možnosti, ze kterých vybíráme, určíme kvalitu, a následně vybereme tu možnost, která má svou kvalitu nejvyšší.

Příklad kritéria. Uvažme situaci, kdy potřebujeme odpovědět na otázku lokálního rozhodování *kam umístit budovu*. Uvážíme všechny pozice, kam můžeme budovu umístit. Pro každou možnost vyhodnotíme daná kritéria, mezi která například patří „Zvýší se délka nejdelší stěny okolo Alhambry o dvě?“. Pokud je odpověď ano, pak se váha tohoto kritéria započítá do celkové kvality dané možnosti. Pokud je odpověď ne, váha se nezapočítá.

3.2 Evoluční algoritmus

Výsledného způsobu, jak hraje počítačový hráč, jsme dosáhli pomocí evolučního (genetického) algoritmu, což je jeden ze způsobů řešení lokálního prohledávání podle knihy Artificial Intelligence [5]. Jelikož můžeme hráče reprezentovat váhami pro kritéria lokálních rozhodování, tj. hodnotami z intervalu $(0, 1)$, budeme tyto hodnoty chápat jako gen hráče pro evoluční algoritmus.

3.2.1 Princip fungování

Hlavní složkou evolučního algoritmu jsou samotné instance hráčů. Mějme dvě skupiny hráčů, které tvoří *populaci*. Jedné z nich budeme říkat *učící se* hráči a druhé *statičtí* hráči. Na počátku mějme náhodnou populaci hráčů, kterou nazveme *generace* nula. Pojem *fitness* bude označovat číselnou hodnotu, kterou vyjadřujeme, jak moc daný hráč hraje „dobře“ a určíme jej tak, že spočteme průměrné skóre daného hráče v generaci. Algoritmus je následující:

1. začínáme s hráči generace nula, jejich geny jsou náhodné;
2. odehrajeme určitý počet her. Hráči jsou do her vybíráni tak, aby vždy v každé hře byl právě jeden učící se hráč a zbytek byli statičtí hráči. Tím docílíme toho, že hráči, kteří se učí, hrají vždy proti statickým hráčům, takže nejsou jejich výsledky navzájem ovlivněny mezi sebou. Zároveň výsledky hráčů tak můžeme porovnávat vůči pořád stejné skupině statických hráčů a vidět tak, jak moc se zlepšují;
3. každému hráči spočítáme jeho fitness;
4. vytvoříme novou generaci hráčů. Ze skupiny učících se vybereme jen ty nejlepší (ty, kteří mají největší fitness) a stvoříme nové hráče, potomky těch nejlepších. Tak doplníme skupinu učících se hráčů na původní počet. Statičtí hráči se nemění;
5. opakujeme od kroku dva.

Tímto algoritmem tedy můžeme „naučit“ hrát vstupní skupinu hráčů. Nevýhodou však může být to, že vstupní skupina učících se hráčů se vyvíjí pouze proti stále stejné skupině statických hráčů. Proto uvážíme následující modifikaci:

- po určitém počtu nových generací provedeme výměnu hráčů a to tak, že učící se hráči budou nyní statičtí;
- jako nové učící se hráče uvážíme opět náhodné;
- znovu spustíme původní algoritmus.

Každé poslední generaci před takovou výměnou hráčů budeme říkat *hlavní generace*. Hlavní generace 0 reprezentuje počáteční statické hráče (náhodné). Zároveň tak můžeme porovnat, jakých výsledků dosahují hráči navzájem mezi hlavními generacemi, tzn. pokud spočítáme hlavní generaci 1, tak necháme sehrát zápasy tyto hráče navíc s hráči hlavní generace 0. Stejně tak pokud dopočítáme hlavní generaci 2, tak ji necháme navíc sehrát zápasy s hráči hlavní generace 1 a 0, atd. Tím tedy dostaneme jakési „mezigenerační“ srovnání.

3.2.2 Proces křížení a mutace

Mějme dva hráče a potřebujeme vytvořit dalšího, jejich potomka. Hráči jsou reprezentováni pomocí vah pro kritéria lokálních rozhodování. Na tyto váhy se rovněž můžeme dívat jako na algoritmy, podle kterých hráči hrají.

Vytvoříme nové pole vah pro kritéria lokálních rozhodování tak, že tyto váhy budou obsahovat kombinaci vah z původních dvou rodičů, které potřebujeme zkřížit. Pro každou jednotlivou váhu

- se náhodně rozhodneme, zda použijeme váhu z prvního, či z druhého rodiče;
- rozhodneme, jestli provedeme mutaci této váhy (jen v malém procentu případů).

Nyní popíšeme, jakým způsobem provádíme mutace. Nechť máme váhu, kterou chceme pozměnit – mutovat. To nám zajistí následující algoritmus:

1. Vstup: hodnota $x \in (0, 1)$, pro kterou chceme provést mutaci;
2. nechť $d \in (-\frac{y}{2}, \frac{y}{2})$ náhodná hodnota, kde y je parametr evoluce `MutationInterval`;
3. dokud $(x + d) \notin (0, 1)$:

   $d := \frac{d}{2}$;
4. Výstup: $(x + d)$, nová zmutovaná váha.

Zároveň zajistíme, že nová zmutovaná váha nebude jen náhodná hodnota z celého intervalu $(0, 1)$, ale bude ovlivněna stávající hodnotou.

3.2.3 Parametry evoluce

Pro samotnou evoluci vždy první nastavíme několik parametrů:

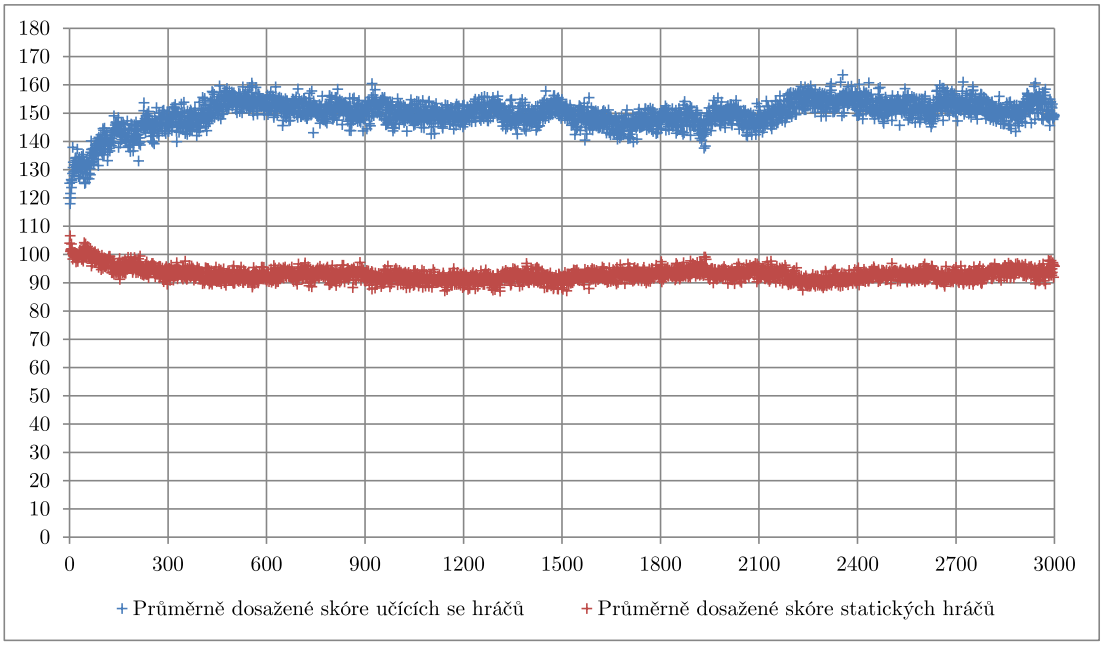
- `LearnableAlgorithms` — počet učících se hráčů;
- `StableAlgorithms` — počet statických hráčů;
- `NumberOfPlayers` — počet hráčů, kteří se účastní jedné hry;
- `Selecting` — každému učícímu se hráči vybíráme protivníky, se kterými se utká. Tento parametr udává počet, kolikrát tyto protivníky vybíráme;
- `IterationsForEach` — pro již vybrané hráče daných her udává počet jejich opakování;
- `SubGenerations` — počet generací, které budou uvnitř jedné hlavní generace;
- `Generations` — počet hlavních generací;
- `TopPlayersSelection` — počet nejlepších hráčů, kteří „přežijí“ do další generace;
- `Mutation` — šance na mutaci (desetinný zápis);
- `MutationInterval` — velikost intervalu, ve kterém se modifikuje mutovaná hodnota.

3.3 Výsledky evoluce

V této části si ukážeme, jakých výsledků evoluce dosahuje pro nastavení různých parametrů. Každý graf, který zde ukazujeme, zobrazuje výsledky stejným způsobem. Pro všechny hráče populace máme jejich fitness, tj. průměrné skóre, jaké v dané generaci nahráli. Graf zobrazuje průměrné skóre učících se hráčů oproti statickým. Jedná se tedy o průměr průměrných skóre. Osa x představuje počet generací, osa y ono průměrné skóre. Všechna data a výsledky evoluce jsou rovněž uvedena na přiloženém DVD.

Vzhledem k časové náročnosti výpočtů musíme jako první rozhodnout, kolik generací budeme vůbec počítat. Uvažme určité nastavení parametrů pro evoluci, kdy spočítáme 3000 generací:

```
LearnableAlgorithms=5          SubGenerations=3000
StableAlgorithms=5             TopPlayersSelection=2
Selecting=10                   Mutation=0,025
IterationsForEach=1            MutationInterval=0,5
```



Obrázek 3.2: Evoluce hry tří hráčů během 3000 generací

Z grafu 3.2 je patrné, že za hranicí 600. generace se výkony hráčů stabilizují a příliš se nezlepšují. Rovněž můžeme vidět, že mezi generacemi například 1200 a 2100 se vyskytuje jistý dlouhodobější pokles ve skóre učících se hráčů i přesto, že do každé nové generace vybíráme jen ty nejlepší jedince z generace předchozí. To může být způsobeno například:

- příliš velkou mutací;
- malým počtem odehraných her v jedné generaci;
- příliš mnoha generacemi – jedinci se už nedokáží zlepšovat.

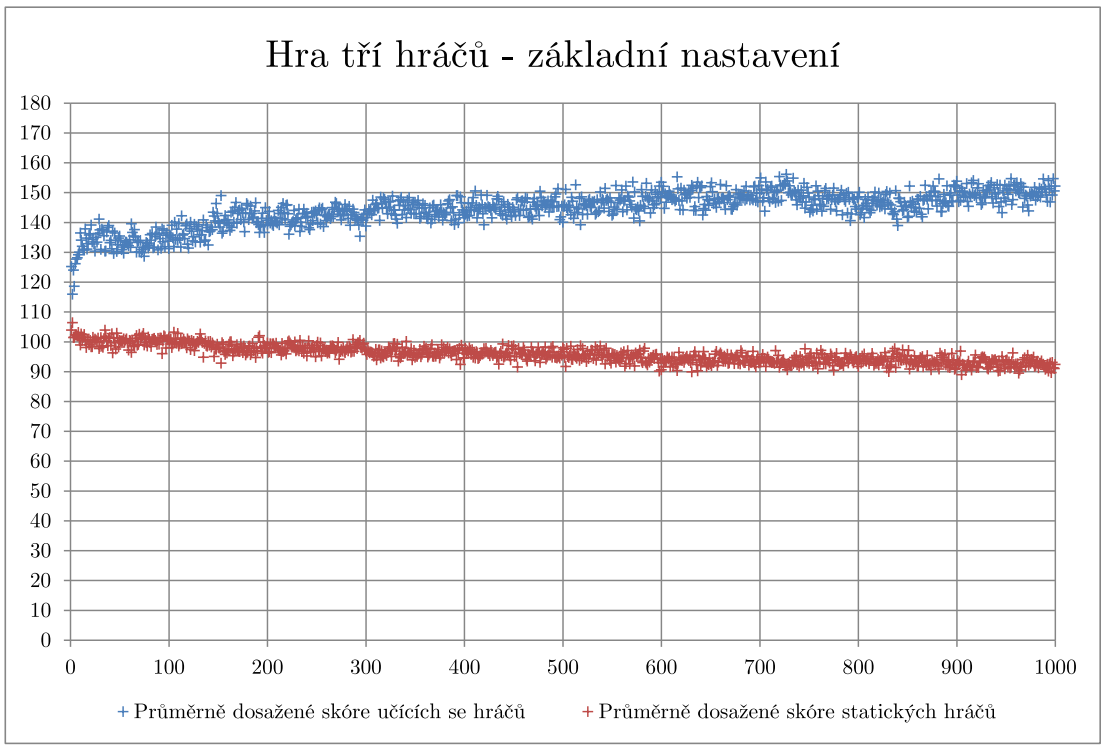
Rozhodli jsme se proto, že 1000 generací pro jednu hlavní generaci bude dostačujícím počtem.

3.3.1 Základní nastavení

Nyní je třeba zjistit, jak naše evoluce závisí na svých parametrech. Proto vytvoříme jisté *základní* nastavení parametrů, které budeme později modifikovat:

```
LearnableAlgorithms=5          SubGenerations=1000
StableAlgorithms=5             TopPlayersSelection=2
Selecting=10                   Mutation=0,01
IterationsForEach=1            MutationInterval=0,5
```

Výsledky a analýzu si ukážeme na hře o třech hráčích. Pro základní nastavení hry o třech hráčích máme tedy následující výsledky hlavní generace 1:



Obrázek 3.3: Evoluce hry tří hráčů základního nastavení – hlavní generace 1

Zároveň uvedeme i srovnání výsledné hlavní generace 1 se statickými hráči, tj. necháme spolu sehrát hry mezi hlavní generací 0 a 1. Výsledky dostáváme následující:

Hlavní generace	Fitness
0 vs. 1	103,04 vs. 140,50

Vidíme, že posun ve fitness je značný. Dále pozměníme parametry evoluce, abychom viděli, jaký mají vliv na její vývoj. Výsledky reprezentované grafy, pomocí kterých provádíme tuto analýzu, uvádíme v příloze A.

3.3.2 Analýza a výběr nejlepších parametrů

Zde si zanalyzujeme výsledky evoluce s pozměněnými parametry a poskládáme z nich „ideální“ nastavení těchto parametrů, pomocí kterého dostaneme další, finální výsledky.

Mutace

Zjistili jsme, že 0,5% mutace je příliš málo na to, aby se hráči zlepšovali a evoluce tak spíše stagnuje. Učící se hráči hned po několika generacích dosáhli své jisté úrovně, a pak se již nemění. Mutace ve výši 3% se zase ukázala jako velká – výsledky evoluce obsahují příliš velké výkyvy. Rovněž jsme vyhodnotili evoluci i s mutací 2% a spolu s 1% tyto dvě mutace podávají již lepší výsledky. Mutace 2% je lehce více nestabilní oproti mutaci 1%.

Také jsme zkusili změnit interval, ve kterém provádíme mutace hodnot. Při velikosti intervalu 0,5 v průběhu evoluce dosahujeme hodnot průměrného skóre mezi 150 a 160, nicméně při velikosti intervalu 0,66 se pohybujeme na hodnotách do 150. Stejně tomu tak je i při velikosti intervalu mutace 0,33. Do výsledného nastavení jsme se tedy rozhodli zvolit jako „ideální“ mutaci hodnotu 1% a velikost intervalu mutace 0,5.

Počet odehraných her v generaci

Dále jsme analyzovali, jaký vliv má počet odehraných her na výsledky. Zvýšením počtu vybíraných protivníků pro každého učícího se hráče (parametr `Selecting`) jsme dosáhli přesnějších výsledků, neboť tak máme větší vzorek odehraných her. Obecně platí, že čím více her odehrajeme, tím přesnější budeme mít data. Proto do výsledného nastavení zvolíme hodnotu 30. Zvýšením počtu iterací každé hry dostaneme další jisté zpřesnění, nicméně toto zvýšení omezuje „pestrost“ výsledků, pokud jej zvolíme na úkor parametru `Selecting`. Proto radši zvolíme vyšší hodnotu parametru `Selecting` a `IterationsForEach` ponecháme na hodnotě jedna.

Populace a výběr nejlepších hráčů

Porovnáním základního nastavení a nastavení s dvojnásobnou populací zjistíme, že tato změna populace výsledky příliš neovlivní.

Rovněž jsme zkusili pozměnit parametr `TopPlayersSelection`, který udává, kolik hráčů z populace bude vybráno do další generace jako „přeživší“. Naše populace hráčů je obecně poměrně malá a zvýšení tohoto počtu pro tuto populaci nemá význam.

Shrnutí

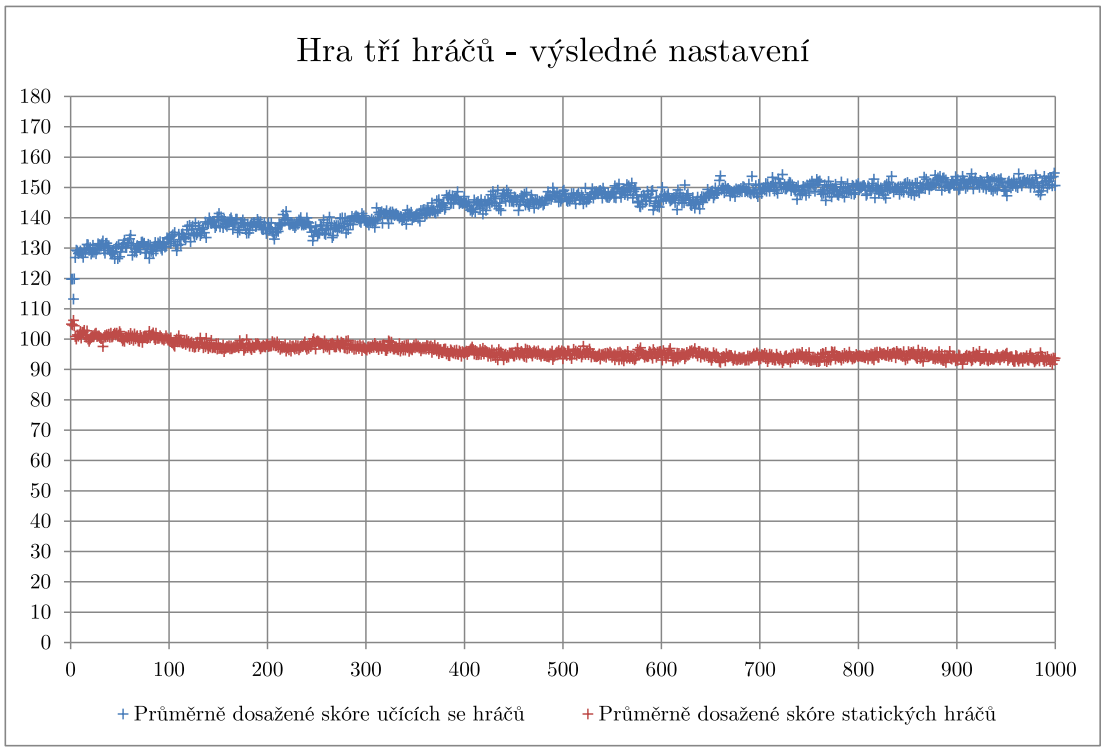
Na základě naší analýzy parametrů jsme tedy rozhodli o výsledném nastavení parametrů:

```
LearnableAlgorithms=5          SubGenerations=1000
StableAlgorithms=5             TopPlayersSelection=2
Selecting=30                   Mutation=0,01
IterationsForEach=1            MutationInterval=0,5
```

3.3.3 Výsledky finálního nastavení evoluce

Máme výsledné nastavení parametrů, tak pro něj spočítáme několik hlavních generací. Jelikož jsme si analýzu parametrů ukazovali na hře třech hráčů, i zde uvedeme výsledky pro hru tří hráčů.

Hlavní generace 1

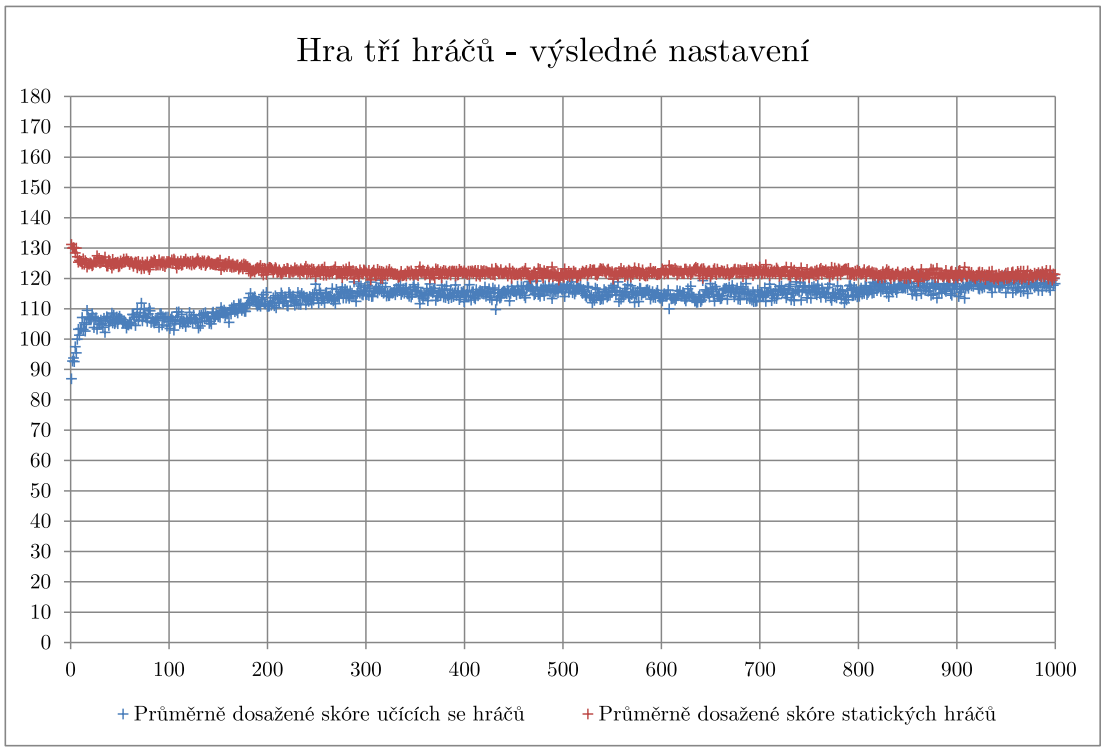


Obrázek 3.4: Evoluce hry tří hráčů s výsledným nastavením – hlavní generace 1

Srovnání hlavní generace 1 s počátečními náhodnými hráči (hlavní generace 0):

Hlavní generace	Fitness
0 vs. 1	100,66 vs. 146,65

Hlavní generace 2



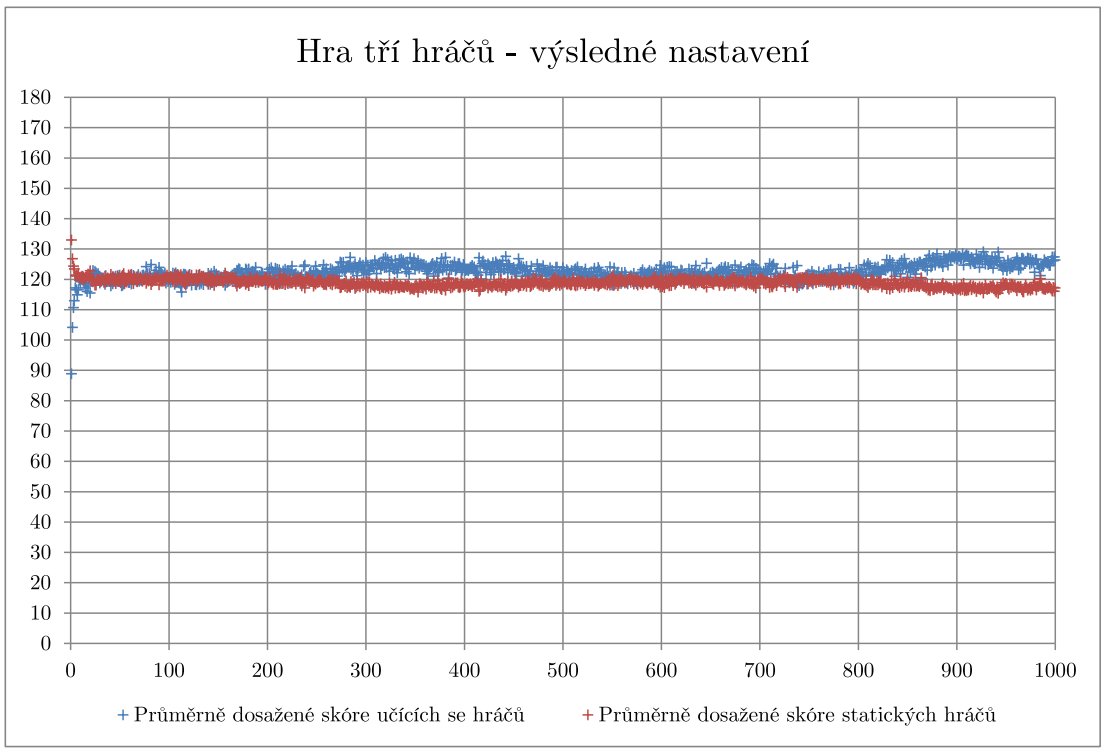
Obrázek 3.5: Evoluce hry tří hráčů s výsledným nastavením – hlavní generace 2

A opět, srovnání hlavních generací:

Hlavní generace	Fitness
0 vs. 2	101,49 vs. 144,64
1 vs. 2	121,29 vs. 118,02

Vidíme, že hlavní generace 2 je horší než generace předchozí. Rovněž můžeme vidět, že rozdíl skóre hlavní generace 0 a 2 je menší, než rozdíl hlavní generace 0 a 1.

Hlavní generace 3



Obrázek 3.6: Evoluce hry tří hráčů s výsledným nastavením – hlavní generace 3

Srovnání hlavních generací:

Hlavní generace	Fitness
0 vs. 3	99,92 vs. 146,71
1 vs. 3	118,56 vs. 120,51
2 vs. 3	117,62 vs. 125,32

Podíváme-li se na srovnání hlavních generací, vidíme, že hlavní generace 3 je nejúspěšnější. Rozdíl hlavní generace 3 a hlavní generace 0 (počáteční náhodní hráči) je vetší než v případě srovnání hlavní generace 1 a 0 a rozdíl hlavní generace 3 a 1 je větší než rozdíl hlavních generací 2 a 1. Hlavní generace 3 je tedy lepší než všechny předchozí hlavní generace.

3.3.4 Reprezentace výsledků

Výsledky evoluce – zejména konkrétní váhy, kterých jsme dosáhli, můžeme převést zpátky na jejich skutečnou „lidskou“ reprezentaci. Zde vybereme a srovnáme ty nejzajímavější z nich.

Data, která zde zobrazujeme, jsou data jednoho jedince z populace učících se hráčů z patřičné hlavní generace. Na tom, kterého jedince vybereme, až tak nezáleží, protože populace v hlavní generaci, tedy populace, která prošla 1000 generacemi, je téměř shodná. Hlavní generaci vybíráme podle toho, jestli je lepší než ty ostatní, a to poznáme z jejich vzájemného srovnání. Zde si zobrazíme data opět pro hru tří hráčů. Na základě výše uvedených srovnání hlavních generací pro hru tří hráčů jsme jako nejlepší generaci vybrali hlavní generaci 3.

Všechna data v tabulkách zobrazujeme na prvních pět desetinných míst. Kompletní seznam všech kritérií lokálních rozhodování nalezneme v příloze B.

Fází hry rozumíme dobu hry mezi jednotlivými sčítacími koly. Například fáze dva je část hry mezi prvním a druhým sčítacím kolem. Celkem jsou tedy tři fáze hry.

Kritérium	Fáze hry	Hodnota
Jedná se o tah odebrání karet	1	0,63510
	2	0,17937
	3	0,31306
Jedná se o tah koupení budovy	1	0,99319
	2	0,92522
	3	0,65624
Jedná se o tah přestavby Alhambry	1	0,18122
	2	0,46883
	3	0,17548
Odebírám barvu karty, které mám nejvíce	1	0,94984
	2	0,67477
	3	0,41985
Budovu přeplácím o tři	1	0,77217
	2	0,22828
	3	0,99996
Zvýší se délka nejdelší stěny o dvě	1	0,06225
	2	0,13880
	3	0,64316
Zvýší se délka nejdelší stěny o čtyři	1	0,74215
	2	0,96537
	3	0,45194

Tabulka 3.1: Váhy některých kritérií pro hru tří hráčů

Z tabulky 3.1 můžeme usuzovat následující:

- odebírat karty je výhodné především v první fázi hry;
- v první fázi hry je nejlepší odebírat karty pořád stejné barvy (například abychom mohli koupit alespoň nějaké budovy, než přijde první sčítací kolo);
- kupovat budovy se vyplácí po celou hru, nejlépe pokud ji platíme přesně;
- přeplácet budovy je výhodné v první a poslední fázi hry, pokud danou budovu nemůžeme zaplatit přesně;
- pokud umisťujeme budovu, která může prodloužit délku nejdelší stěny o dvě, zejména v první fázi hry může být lepší ji umístit na nějakou „lepší“ pozici (například proto, abychom v dalších kolech mohli tyto „kousky“ stěn propojit a vytvořit tak delší stěnu). Pokud danou budovou můžeme prodloužit nejdelší stěnu například o čtyři, tak je to velmi výhodné ve všech fázích hry.

Váhy výsledných hráčů evoluce poskytují však více informací, než jsme zde v tabulce 3.1 zobrazili. Uveďme ještě několik dalších poznatků:

- kupovat budovy s menšími hodnotami (modré, červené) se vyplácí především v první fázi hry. Ostatní budovy se naopak vyplácí kupovat ve zbylých dvou fázích. Tím můžeme docílit například toho, že v prvním sčítacím kole budeme mít více budov, protože jejich ceny jsou menší. Toto sčítací kolo je obecně poměrně brzy, proto se vyplácí kupovat i méně hodnotné budovy;

- v první fázi hry není výhodné kupovat takové barvy budov, kterými srovnáme počet budov dané barvy s jiným hráčem. Lepší může být kupovat budovy, které ještě nikdo z hráčů nemá. Tento způsob kupování budov v první fázi hry může být výhodnější proto, že v prvním sčítacím kole se udělují budovy pouze hráčům s největším počtem budov dané barvy a pokud má více hráčů stejný počet, tak se skóre dělí. Protože se v tomto skórovacím kole uděluje velmi málo bodů, tak každý z těchto hráčů nedostane téměř nic.

3.3.5 Rozšíření pro ostatní počty hráčů

Výše popsaná rozhodnutí, cesta k výsledným parametrům pro evoluci, analýza výsledků a jejich reprezentace se týkala vždy her se třemi hráči. Jelikož hry s jiným počtem hráčů, konkrétně hry pro čtyři, pět a šest hráčů, jsou stejným režimem hry, spustili jsme pro tyto počty hráčů evoluci se stejnými finálními parametry, jako pro hru tří hráčů.

Protože hry, kterých se účastní pouze dva hráči, mají jiný režim (obsahují imaginárního hráče), tak jsme celý postup zjišťování „ideálního“ nastavení parametrů pro evoluci opakovali. Tyto výsledky (vzhledem k jejich rozsahu) uvádíme na přiloženém DVD. Výsledky konečného nastavení evoluce pro všechny počty hráčů jsou na tomto disku rovněž k nalezení.

Nejlepší výsledky, kterých jsme pomocí evoluce dosáhli, jsou použity ve finální verzi programu hry.

3.3.6 Realizace spuštění evoluce

Jelikož spočíst tyto výsledky evoluce je poměrně časově náročné (řádově dny), využili jsme více počítačů. Evoluci s různými parametry pro hru dvou a tří hráčů jsme spustili na dvaceti počítačích, na každém z nich separátní instanci s určitým nastavením. Programy jsme nechali běžet cca 70 hodin. Po zpracování a vyhodnocení výsledků jsme spustili (už jen na jednom počítači, postupně) finální nastavení parametrů pro hry se všemi počty hráčů. Celkem bylo odehráno více než 6 milionů her.

4. Implementace umělé inteligence

Každého počítačového hráče tvoří instance třídy, která implementuje interface `IArtificialIntelligence`. Tento interface má následující definici:

```
interface IArtificialIntelligence
{
    Player RepresentedPlayer
    {
        get;
        set;
    }
    void Initialize();
    Move GetSolution();
    Position GetPositionForFreeBuilding(Building toBeAccepted);
}
```

- Vlastnost `RepresentedPlayer` — instance hráče, který je reprezentován patřičnou umělou inteligencí;
- metoda `Initialize` — slouží pro inicializaci položek a částí dané instance třídy, která tento interface implementuje;
- metoda `GetSolution` — vrací instanci potomka třídy `Move` (protože tato třída je abstraktní), která reprezentuje tah vybraný danou umělou inteligencí;
- metoda `GetPositionForFreeBuilding` — hráč může na konci hry dostat dlaždici budovy navíc. Metoda vrací pozici, na kterou bude umístěna tato budova.

4.1 Rozšiřitelnost implementace AI

Běh samotné hry není nijak závislý na konkrétních třídách implementující `IArtificialIntelligence`. Při vytváření nové hry (z GUI) je vytvořena instance třídy `AIEnumerator` implementující `IEnumerator`. Pomocí metody `MoveNext` a vlastnosti `Current` je hra doplněna o patřičný počet počítačových hráčů. Implementace hry jako takové přistupuje k dané umělé inteligenci skrze interface `IArtificialIntelligence`. Je tedy možné naprogramovat vlastní třídu, která implementuje tento interface, a nastavit daný `AIEnumerator` tak, aby vracel instance tříd jiných vlastních implementací umělé inteligence pro tuto hru.

4.2 Implementace rozhodování

Rozhodování o výběru tahu je obsaženo ve třídě `AIWeighedMoves`. Tato třída implementuje `IArtificialIntelligence` a poskytuje tedy metodu `GetSolution`, která rozhodne, jaký tah zahrát. Tato metoda projde diagramem rozhodování. Podrobnosti o tomto diagramu nalezneme v kapitole Umělá inteligence.

4.2.1 Reprezentace kritérií a jejich vah

Jednotlivá kritéria lokálních rozhodování jsou reprezentována pomocí metod. Všechny tyto metody mají stejnou hlavičku, kde `identificator` je jméno dané metody:

```
bool identificator(object parameter) { ... }
```

Každá taková metoda odpovídá na otázku, zda platí dané kritérium lokálního rozhodování pro možnost (obvykle) předanou parametrem. Pokud tedy uvážíme (již výše zmíněný) příklad, kde rozhodujeme o kritériu „Zvýší se délka nejdelší stěny okolo Alhambry o dvě?“, je zavolána metoda `IncreaseWallBy_2(position)`, kde jako parametr předáváme konkrétní pozici.

Všechny metody určující kritéria lokálního rozhodování jsou uloženy v poli delegátů. Toto pole má tvar `Func<object, bool>[]`. Váhy k jednotlivým kritériím lokálního rozhodování (metodám) jsou uloženy v poli hodnot `double`. Tyto váhy jsou však uloženy třikrát – to znamená zvlášť pro každé sčítací kolo hry, tedy jednou pro vyhodnocování kritérií před prvním sčítacím kolem, jednou před druhým a jednou před třetím sčítacím kolem. Tím docílíme toho, že stejná kritéria lokálního rozhodování mohou mít různou váhu v různých částech (fázích) hry. Pole vah je tedy třikrát větší, přesněji (počet-sčítacích-kol)-krát větší, než je celkový počet kritérií. První třetina pole vah a pole delegátů na metody určující kritéria lokálních rozhodování si navzájem odpovídají pomocí indexů.

Protože některá kritéria lokálního rozhodování spolu mohou sémanticky souviset (vztahují se ke stejnému lokálnímu rozhodování), je pro každé lokální rozhodování vytvořeno pole indexů, které indikují patřičná kritéria.

Rozhodování počítačového hráče tedy závisí čistě na zmíněném poli vah pro jednotlivá kritéria lokálních rozhodování. Pak tedy můžeme umělou inteligenci – počítačového hráče reprezentovat tímto polem. Celkový počet různých kritérií lokálních rozhodování je 76 a uvádíme je v příloze B.

Při vytváření instancí počítačových hráčů pro hru s grafickým prostředím je třeba inicializovat váhy pro kritéria lokálních rozhodování. Váhy pro jednotlivé hráče jsou načítány z textových souborů v adresáři `Weights/<Number>Players`, který je v adresáři s instalací aplikace. Místo `Number` je vždy patřičný počet hráčů (slovy – anglicky). Jméno souboru musí vždy být ve tvaru `<Color>_<number of players>`, kde `Color` je barva hráče (rovněž slovy) a `number of players` je číslovka pro daný počet hráčů. Příklad takového souboru může být

`Weights/ThreePlayer/Red_3.txt`,

který definuje váhy pro kritéria lokálních rozhodování červeného hráče hry tří hráčů. V případě nesprávného formátu těchto souborů použijeme třídu `WeightsStorage`, ve které jsou uloženy váhy pro různé počty hráčů ve hře. Tyto váhy jsou ty nejlepší, kterých jsme v evoluci dosáhli. Tím jsme zajistili to, že umělá inteligence bude hrát podle uživatelsky definovaných vah.

4.2.2 Výjimky v rozhodování

Jelikož váhy pro kritéria lokálních rozhodování se pozměňují (mutují) na základě jisté náhody, může nastat situace, kdy tyto váhy zmutují tak, že největší váhu bude mít tah přestavby Alhambry. Tím pádem tedy může nastat situace, kdy všichni hráči ve hře budou prioritně přestavovat Alhambru. To následně zapříčiní, že se tato hra zacyklí – všichni hráči budou neustále dělat tah přestavby. Takoví hráči, kteří pořád přestavují Alhambru, nenahrají mnoho bodů a v evoluci se příliš daleko nedostanou. Těchto cyklických her není mnoho, nicméně je třeba tyto situace ošetřit.

U každého hráče (řízeného umělou inteligencí) se počítá, kolik tahů přestavby Alhambry udělal v řadě. Pokud počet těchto tahů přestavby přesáhne jistou konstantu (v našem případě 10), tak se váha pro tah přestavby Alhambry nastaví na nula. Tím docílíme toho, že budou mít přednost tahy odebrání karet a koupení budovy. To ale neznamená, že výsledný tah nemůže být tah přestavby (ostatní tahy nemusí být v dané situaci proveditelné). Pokud počet tahů přestavby dosáhne dvojnásobné hodnoty přednastavené konstanty (tedy celkem 20), bude tato hra ukončena a spuštěna znovu. Toto ukončení hry se týká pouze her, které běží v rámci evoluce.

Další změna oproti obecným pravidlům, podle kterých se AI rozhoduje, se týká tahu odebírání karet. Pokud se umělá inteligence rozhoduje, jaký tah udělat, zkontroluje počet karet, které vlastní. Pokud tento počet přesáhne stanovenou konstantu (napevno 20), tak se váha pro tah odebrání karet nastaví na nula a opět, jako ve výše zmíněném případě s přestavbou Alhambry, neznamená to, že výsledný tah nemůže být tah odebrání karet. Důvodem je urychlení výpočtu, neboť při rozhodování, jakými kartami zaplatíme budovu, uvažujeme všechny podmnožiny karet dané barvy, které vlastníme.

4.3 Implementace evolučního algoritmu

Možnost spouštění dávkových her bez grafického prostředí poskytuje konzolová aplikace `AlhambraBatchRunner`. Evoluční algoritmus potřebuje nějaké počáteční nastavení a parametry, podle kterých bude běžet. Toto nastavení parametrů se načítá ze souboru `EvolutionConfig.txt`, který musí být umístěn ve stejné složce, jako aplikace pro běh evoluce.

4.3.1 Paralelní běh

Samotné hry, které potřebujeme odehrát v evolučním algoritmu, běží paralelně. K tomu nám slouží instance třídy `System.Threading.Tasks.Task`, ve kterých se hry odehrávají. Pro každého učícího se hráče několikrát vybíráme soupeře, proti

kterým bude hrát. Pro každý tento výběr vytvoříme jeden `Task`. Celkem je tedy v každé generaci spuštěno

$$\texttt{LearnableAlgorithms} * \texttt{Selecting}$$

instancí třídy `Task`, kde `LearnableAlgorithms` a `Selecting` jsou parametry evoluce, které jsme popsali v kapitole Umělá inteligence v sekci 3.2.3.

Počítání samotných nových generací probíhá vždy v původním vlákně, ve kterém byl program evoluce spuštěn.

4.3.2 Opakovatelnost výpočtů

Je žádoucí, aby naše výsledky experimentů s evolučním algoritmem byly opakovatelné. K tomu nám slouží třída `AlhambraRandom`, která obsahuje instanci třídy `System.Random`, říkejme jí *hlavní generátor*, které jsme nastavili počáteční `Seed` hodnotu na `150693`. Tento hlavní generátor náhodných čísel je použit při počítání nových generací (mutace, ...). Jelikož samotné hry běží mnohdy paralelně, pak i například „rozdávání“ karet v těchto hrách se může dít paralelně. Pro tyto účely není možné použít zmíněný hlavní generátor čísel, neboť instance tříd `Random` nejsou thread-safe. Pokud bychom zajistili pomocí synchronizačních primitiv vzájemné vyloučení pro tento generátor náhodných čísel, výpočet stejně nebude opakovatelný, protože dané instance třídy `Task` se mohou a typicky naplánují na různá vlákna v různém pořadí.

Řešením toho problému je umožnit, aby každý `Task` měl svou vlastní instaci třídy `Random`. Při vytváření nových instancí třídy `Task` vytvoříme i novou instanci třídy `Random`, kterou daný `Task` dostane a kterou bude používat. `Seed` použitý pro každou instanci je náhodné číslo, vygenerované hlavním generátorem. Vytváření těchto instancí probíhá vždy před spuštěním běhu her v generaci. Tím zaručíme, že každé spuštění evoluce se stejnými počátečními parametry vrátí stejné výsledky.

5. Uživatelská dokumentace

5.1 Instalace a minimální požadavky na systém

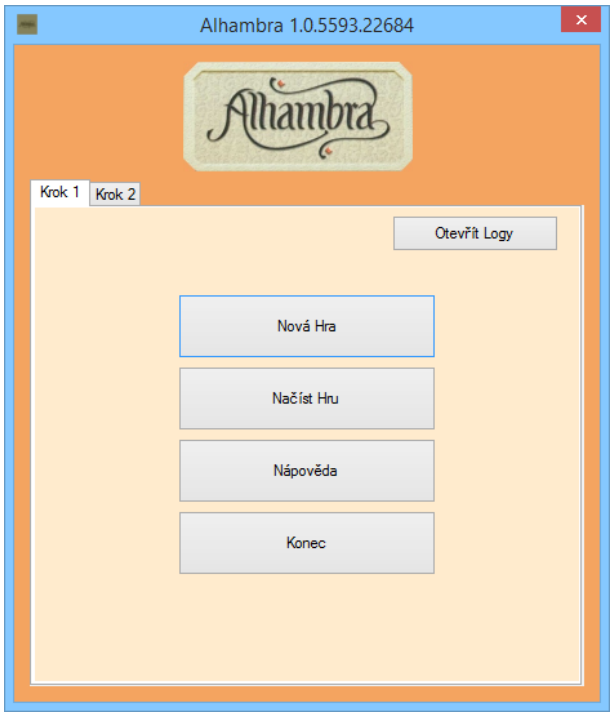
Pro správný běh aplikace je nutné mít operační systém Windows 7 (nebo novější) a nainstalovaný Microsoft .NET framework verze alespoň 4.0.

Aplikaci instalujeme spuštěním instalačního balíčku `AlhambraSetup.msi`.

5.2 Jak začít hrát

Máme-li operační systém Windows 7, aplikaci můžeme spustit z nabídky start ze složky *Alhambra*, program `Alhambra.exe`. Na systému Windows 8 můžeme použít zástupce *Alhambra* v prostředí Metro. V obou případech můžeme rovněž pro spuštění použít zástupce na ploše.

Po spuštění programu se objeví úvodní okno se základní nabídkou hry. V hlavičce programu vidíme aktuální verzi hry.



Obrázek 5.1: Okno Úvod

Kliknutím na tlačítko *Nová Hra* se zobrazí nabídka nastavení hráčů pro danou hru. Tuto nabídku můžeme vidět na obrázku 5.2. Tlačítkem *Načíst Hru* se zobrazí dialogové okno pro výběr souboru s uloženou hrou. Kliknutím na tlačítko *Nápověda* se otevře nové okno prohlížeče, kde se zobrazí nápověda hry. Nápověda kromě popisu ovládání obsahuje i popis pravidel hry. Ten byl převzat z portálu *deskovehry.cz* [6]. Tlačítkem *Otevřít Logy* otevřeme složku, kde jsou uloženy logy odehraných her. Pomocí tlačítka *Konec* hru ukončíme.

Pomocí druhé části úvodního okna nastavujeme příslušnou hru. Kliknutím na tlačítko *Přidat Hráče* se zobrazí další panel s hráčem navíc. Analogicky pro tlačítko *Odebrat Hráče*. Pomocí rozbalovacích seznamů nastavíme barvu hráče a také to, zda je daný hráč řízený počítačem, nebo člověkem. Pro urychlení tohoto nastavování můžeme použít tlačítko *Naplnit*, které tuto práci udělá za nás (výchozí nastavení – počítačoví hráči). Tlačítkem *Zpět* se dostaneme do hlavní nabídky úvodního okna, tlačítkem *Start Hry* zahájíme samotnou hru.

Obrázek 5.2: Okno Úvod – nastavení hráčů

5.3 Herní okno

Obrázek 5.3: Okno Hra – trh

Obrázek 5.4: Okno Hra – mapa

Okno, kde probíhá samotná hra, obsahuje tyto části:

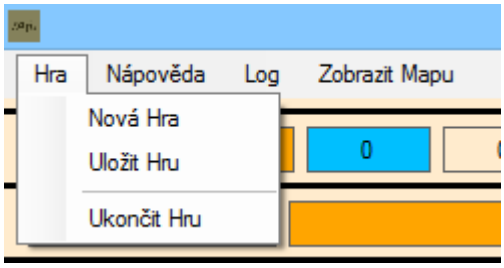
1) hlavní nabídka hry;
2) skóre jednotlivých hráčů, v takovém pořadí, jak mají hrát;
3) barva hráče, který je aktuálně na tahu;
4) záložky pro trh a mapy jednotlivých hráčů
5) informace o hře;
6) herní trh;
7) zobrazení počtu budov dané barvy pro jednotlivé hráče;
8) pole pro zobrazení koupených, ale ještě neumístěných budov;
9) karty na trhu k dobírání;
10) karty hráče, který je aktuálně na tahu;
11) mapa vybraného hráče;
12) odkládací plocha vybraného hráče.

Po spuštění hry se zároveň zobrazí okno ukazující počáteční karty všech hráčů.

Obrázek 5.5: Okno s počátečními kartami hráčů

5.4 Ovládání hry

Kliknutím na položku *Hra* v hlavní nabídce hry, budou zobrazeny možnosti hry, a to *Nová Hra*, *Uložit Hru* a *Ukončit Hru*. Vybráním položky *Nová hra* bude bude zobrazeno úvodní okno pro nastavení nové hry. Zvolením položky *Uložit Hru* bude zobrazeno okno pro výběr adresáře pro uložení hry. Kliknutím na položku *Ukončit Hru* bude hra ukončena. Avšak vždy před ukončením stávající hry bude zobrazeno dialogové okno pro potvrzení konce hry.

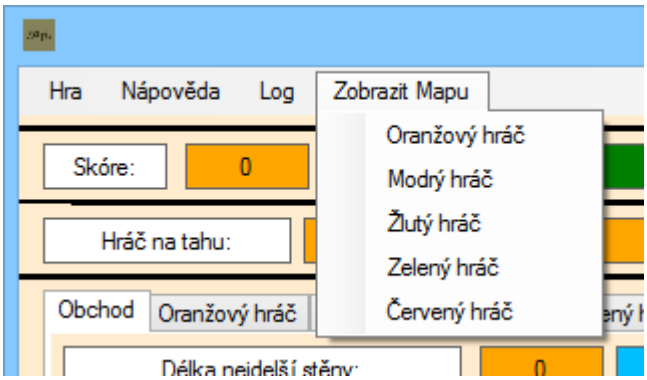


Obrázek 5.6: Hlavní nabídka hry

Kliknutím na položku *Nápověda/Zobrazit Nápovědu* bude otevřeno nové okno prohlížeče obsahující nápovědu ke hře.

Kliknutím na položku *Log/Zobrazit Log* bude otevřeno nové okno s textovým popisem stávající rozehrané hry.

Kliknutím na položku *Zobrazit Mapu* a následným vybráním hráče bude zobrazena mapa vybraného hráče v novém okně.

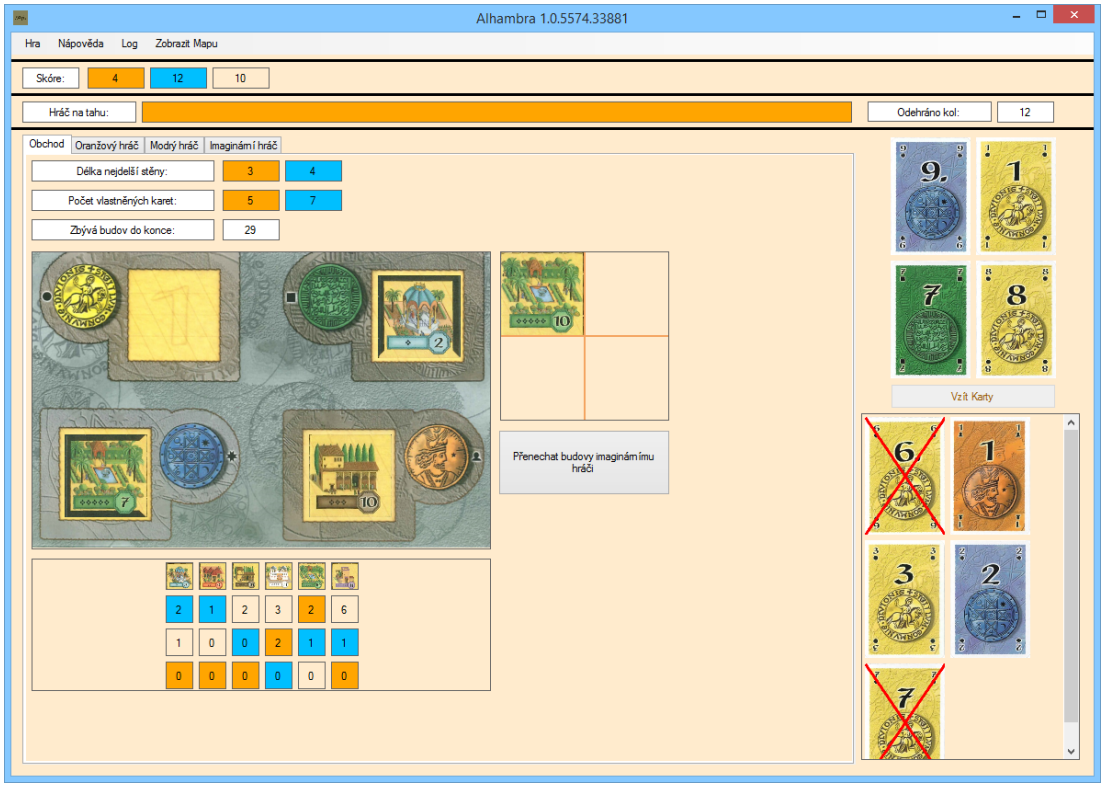


Obrázek 5.7: Výběr zobrazení mapy hráče

Chceme-li udělat tah odebrání karet, levým tlačítkem myši provedeme daný výběr. Pravým tlačítkem myši je daný výběr zrušen. Vybrané karty, které chceme odebrat, potvrdíme stiskem tlačítka *Vzít Karty*.

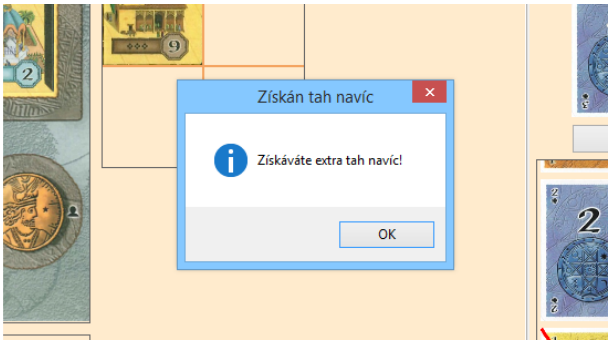
Obrázek 5.8: Výber karet pro odebrání

Pokud chceme koupit budovu, kliknutím levého tlačítka ji vybereme. Pravým tlačítkem je opět výběr zrušen. Následně vybereme karty, kterými danou budovu zaplatíme. Výběr karet hráče se provádí stejně, jako výběr karet na trhu. Jakmile bude cena karet dostatečná, bude dlaždice budovy koupena a zobrazena v místě pro koupené budovy, tak jak zobrazuje obrázek 5.9. Navíc, je-li hra v režimu dvou hráčů, zobrazí se tlačítko pro přenechání koupených budov imaginárnímu hráči.



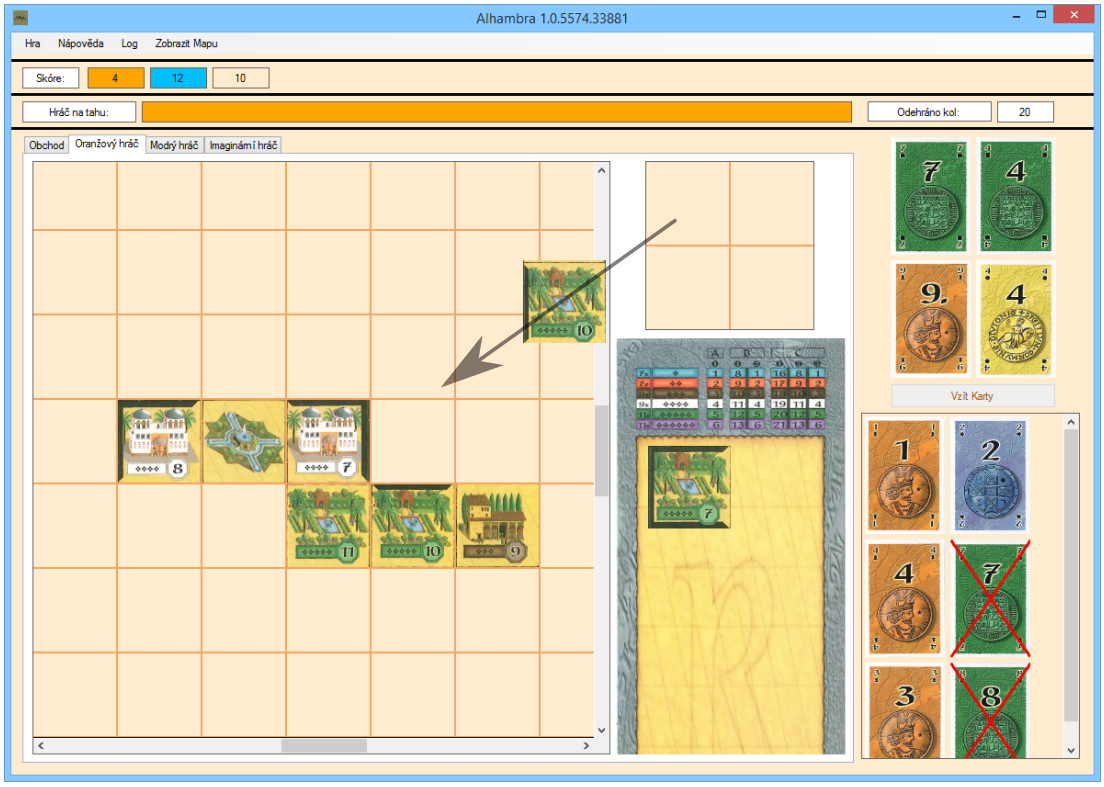
Obrázek 5.9: Koupě budovy

Pokud zaplatíme budovu přesně, budeme upozorněni dialogem na to, že získáváme tah navíc; v takovém případě na nově koupenou dlaždici budovy nelze klikat, ani ji nijak posunovat, neboť je nejprve třeba dokončit nově získaný tah.



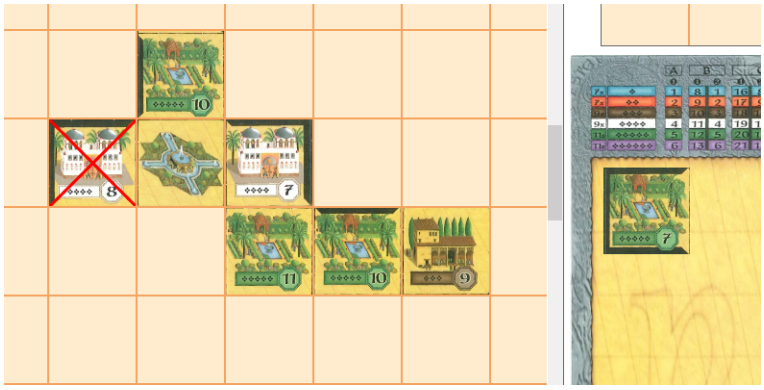
Obrázek 5.10: Dialog o získání tahu navíc

Umístění dlaždice budovy do herní mapy provedeme přetáhnutím myši. Pokud budovu budeme chtít umístit na pozici, která není v souladu s pravidly hry, dlaždice budovy se do herní mapy neumístí. Pokud budovu přetáhneme nad odkládací plochu, bude tato budova odložena.



Obrázek 5.11: Umístění budovy do herní mapy

Pro tah přestavby Alhambry můžeme buďto přetáhnout budovu z odkládací plochy do herní mapy, nebo obráceně z herní mapy do odkládací plochy. Pokud chceme vyměnit dvě budovy, vybereme je pravým tlačítkem myši. Pokud obě budovy lze dle pravidel prohodit, ihned se tak stane.

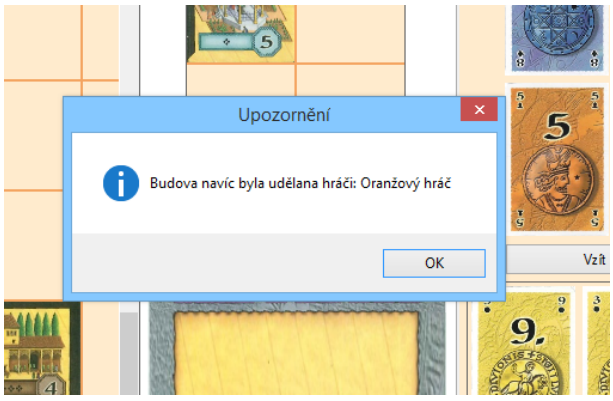


Obrázek 5.12: Výběr budovy pro tah přestavby

Pokud nastane skórovací kolo, je zobrazena tabulka s nově získanými body pro jednotlivé hráče a jejich pořadí v počtech budov jednotlivých barev.

Obrázek 5.13: Dialog skórovacího kola

Pokud je hra dohrána až do konce, dle pravidel mají být zbylé dlaždice budov, které jsou na trhu, dány některým hráčům. Pokud je taková budova dána nějakému hráči, bude upozorněn dialogem 5.14. Tuto dlaždici budovy hráč umisťuje do herní mapy standardním způsobem v souladu s pravidly hry.



Obrázek 5.14: Dialog o získání budovy navíc

Ve všech částech herního okna, kde je možné rolovat, můžeme pomocí kolečka myši. Pokud chceme rolovat horizontálně, musíme navíc podržet klávesu `shift`.

Platí, že před výběrem nějaké karty, nebo dlaždice budovy je zakázáno mít vybraný jiný tah. Například, pokud hráč chce zvolit budovu ke koupi, nesmí mít označené nějaké karty k odebrání (před výběrem dané budovy musí vybrané karty k odebrání zrušit).

5.5 Vlastní nastavení umělé inteligence

Hra poskytuje uživateli možnost nastavit umělou inteligenci dle vlastního uvážení. Umělá inteligence se rozhoduje na základě kritérií lokálních rozhodování a jejich vah. Přehled těchto kritérií nalezneme v příloze B. Jednotlivé váhy jsou uloženy v adresáři s instalací aplikace v následujících podadresářích:

- `Weights/TwoPlayers`
- `Weights/ThreePlayers`
- `Weights/FourPlayers`
- `Weights/FivePlayers`
- `Weights/SixPlayers`

V každém z těchto adresářů je šest textových souborů, kde každý z nich odpovídá barvě jednoho hráče. Po instalaci jsou v těchto souborech uloženy váhy těch nejlepších hráčů, kterých jsme pomocí evolučního algoritmu dosáhli. Tyto soubory je možné upravovat a uživatel si sám může vymyslet své vlastní váhy, podle kterých bude následně daný počítačový hráč hrát. Podmínkou je však to, že soubory musí být pořád stejně pojmenované v patřičných adresářích a musí mít zachovaný formát. Je možné, že pro přepsání nebo úpravu těchto souborů je potřeba oprávnění správce. Pokud některý ze souborů nebude k nalezení, nebo bude mít špatný formát, bude uživatel upozorněn dialogem před začátkem hry. V takovém případě se použijí výsledné váhy z evolučního algoritmu.

6. Programátorská dokumentace

Aplikace byla napsána v jazyce C# pro platformu Windows. Program využívá .NET framework verze 4.0. Tvorba aplikace probíhala za pomoci vývojového nástroje Microsoft Visual Studio 2013 Ultimate. Dále si zde ukážeme, jak jsou jednotlivé části aplikace tvořeny z technického hlediska. Tento popis není úplný a pro jeho doplnění slouží dokumentace vygenerovaná ze zdrojových kódů pomocí softwaru Doxygen. Tato dokumentace je k dispozici na přiloženém DVD v adresáři `Dokumentace`.

6.1 Členění zdrojových kódů řešení

Řešení (solution) je rozděleno na následující celky, které jsou samostatnými Visual Studio projekty:

- `Alhambra` — obsahuje samotnou implementaci hry, umělou inteligenci i grafické prostředí (okenní aplikace);
- `AlhambraBatchRunner` — umožňuje dávkové spuštění her počítačových hráčů, pod různým nastavením parametrů (konzolová aplikace);
- `AlhambraUnitTests` — testovací třídy a metody pro projekt;
- `AlhambraSetup` — umožňuje vytvoření instalačního balíčku.

6.2 Projekt Alhambra

Tento projekt implementuje celou hru Alhambra, včetně počítačového hráče a grafického prostředí.

6.2.1 Implementace jádra hry

- třída `Game` — obsahuje datové položky hry (karty a budovy jednotlivých hráčů, aktuální skóre, pořadí hráčů, ...) a poskytuje nejdůležitější metody pro práci s nimi. Nejdůležitější metodou je metoda `AcceptMove` (a její přetížení), která na základě předaného tahu hráče aktualizuje datové položky hry. Další významné metody této třídy zahrnují například rozdání karet na počátku hry, nebo spočtení délky nejdelší stěny okolo postavené Alhambry daného hráče, aktualizaci datových položek hry na základě předaného tahu hráče;
- třída `Move` — abstraktní třída poskytující základ reprezentace každého specifického typu tahu. Jejími potomky jsou `TakeCardMove`, `BuyBuildingMove` a `RebuildMove` reprezentující každý specifický typ tahu. Každá z těchto tříd obsahuje veškeré informace o daném tahu;
- třída `MoveChecker` — poskytuje metody pro kontrolu daného tahu, zda je v souladu s pravidly hry. Nejdůležitější z nich je metoda `IsMoveAllowed`.

Tato metoda dokáže pro předaný tah odpovědět, zda je v souladu s pravidly hry jako celek. Rovněž tato třída poskytuje metody i pro dílčí „otázky“, například metoda `CanAddBuilding` resp. `CanRemoveBuilding` odpoví, zda je možné zadanou budovu umístit do resp. odebrat z mapy určitého hráče;

- třída `Controller` — obsahuje hlavní smyčku hry, která je spuštěna zavoláním metody `ExecuteNewMove`. Dále poskytuje metody pro počítání skóre (`EvaluatePoints`), ukládání (`SaveGame`) a načítání hry (`LoadGame`). Tato třída má obecně na starosti řízení hry „shora“;
- třída `Building` — reprezentuje jednotlivé budovy (obsahuje proměnné, které udávají vlastnosti dané budovy);
- struktura `Card` — seskupuje hodnotu a typ karty;
- struktura `Position` — udržuje dvě `int` hodnoty reprezentující řádek a sloupec v herní mapě. Pro tuto strukturu rovněž definujeme operátory `==` a `!=` pro snadné porovnávání;
- třída `CodeStorage` — vytváří jisté podklady pro hru. Obsahuje definice všech instancí budov, které se ve hře nacházejí. Zároveň, pokud je to žádoucí, dokáže tyto jednotlivé instance namapovat na patřičné obrázky pro grafickou reprezentaci. Rovněž tato třída obsahuje metody, které pro zadanou budovu či kartu vrátí obrázek dané budovy či karty;
- třída `AlhambraRandom` — poskytuje (pseudo-)náhodná čísla využívaná v aplikaci.

6.2.2 Implementace hráčů

- třída `Player` — instance této třídy reprezentují jednotlivé hráče (buďto reálné hráče, nebo počítačové hráče); každá instance si udržuje odkazy na jednotlivé seznamy karet a budov vlastněných daným hráčem;
- interface `IArtificialIntelligence` — definuje základní metody pro umělou inteligenci ve hře. Každá AI musí tento interface implementovat.
- třída `AIWeighedMoves` — tvoří naši umělou inteligenci, která implementuje `IArtificialIntelligence`. Poskytuje metody pro procházení diagramu rozhodování a implementuje všechna kritéria pro lokální rozhodování;
- třída `WeightsStorage` — tato třída pouze sdružuje pole vah pro kritéria lokálních rozhodování. Tyto váhy jsou ty nejlepší, kterých jsme v evoluci dosáhli;
- třída `AIEnumerator` — napomáhá aplikaci o rozšiřitelnost dalších implementací umělé inteligence.

6.2.3 Implementace grafického uživatelského prostředí

- třída `BuildingCC` — graficky reprezentuje jednotlivé budovy ve hře a poskytuje pro ně obsluhu uživatelských událostí (`MouseDown`, `MouseClick`, ...). Rovněž zajišťuje změny vzhledu spojené s výběrem dané budovy hráčem;
- třída `CardCC` — analogie k `BuildingCC` pro karty;
- třída `IntroductionForm` — tvoří úvodní okno aplikace. Obsahuje obsluhu tlačítek a rozbalovacích seznamů, pomocí kterých uživatel vytvoří počáteční nastavení hry;
- třída `GameForm` — okno, ve kterém probíhá samotná hra. Obsahuje proměnné, ve kterých se udržují jednotlivé součásti tohoto okna. Podle aktuálního stavu okna (vybrané karty, budovy) dokáže sestavit tah, tj. instanci nějakého potomka třídy `Move`, a následně tento tah předat třídě `Game` ke zpracování;
- třída `MapCC` — tvoří mapu pro uživatele, do které bude umisťovat své dlaždice budov;
- třída `PlayersCardsCC` — ovládací prvek, pomocí kterého uživatel pracuje se svými kartami;
- třída `PostponedBuildingsCC` — tvoří odkládací plochu pro dlaždice budov;
- třída `ScoringCardInfoForm` — poskytuje informační okno zobrazující počet získaných bodů hráčů. Je zobrazováno po každém sčítání bodů.

Třídy `MapCC`, `PlayersCardsCC` a `PostponedBuildingsCC` obsahují jejich nejdůležitější metodu `RefreshData`, pomocí které dokáží obnovit svůj vizuální vzhled na základě dat hry.

Instance herního okna grafického prostředí se vytváří z kódu úvodního okna. Zároveň se vždy vytvoří jedno vlákno navíc, ve kterém běží hlavní smyčka hry.

6.2.4 Ukládání a logy her

Stav hry se ukládá pomocí třídy `BinaryFormatter` ze jmenného prostoru `System.Runtime.Serialization.Formatters.Binary` a jedná se tedy o binární serializaci. Serializuje se instance třídy `Game` (a tedy i graf referencí z této třídy), kromě instance grafického rozhraní, která má atribut `[NonSerialized]`. Hra je uložena do souboru `<jmeno>.data`, kde `jmeno` zadá uživatel. Výchozí adresář pro uložení je `AppData/Alhambra/Save`, kde `AppData` je adresář pro data aplikací. Získáno pomocí `Environment.SpecialFolder.ApplicationData`.

V průběhu her se automaticky vytvářejí logy, které textově zobrazují průběh hry. Tyto logy jsou ukládány do souborů `YYMMDD_HHMMSS.log`, kde jméno představuje datum a čas. Tyto soubory mají textový formát. Adresář pro uložení je stejný jako pro ukládání her, koncová složka cesty je `Logs`.

6.2.5 Zdroje

Aplikace využívá konceptu zdrojů, tzv. resources.

Texty

Všechny textové popisky, které uživatel může vidět, se v kódu vždy odkazují na patřičný zdroj. K tomu slouží zdroje:

- `StringsGame.resx`/`.cs.resx` — obsahuje řetězcové konstanty související se samotnou hrou;
- `StringsGameForm.resx`/`.cs.resx` — řetězce zobrazované v herním okně;
- `StringsIntroductionForm.resx`/`.cs.resx` — řetězce zobrazované v úvodním okně;

Výchozí jazyk hry je jazyk anglický, hra je ale přeložena do českého jazyka. Tato „čeština“ je uložena jako satelite assembly `Alhambra.resources.dll` ve složce `cs`. Program si svůj jazyk vybírá sám podle aktuálního nastavení, získaného pomocí `System.Globalization.CultureInfo.CurrentUICulture`.

Obrázky

Obrázky částí grafického prostředí jsou s programem rovněž provázány pomocí odkazů na zdroje, konkrétně pomocí `Pictures.resx`.

6.3 Projekt AlhambraBatchRunner

Projekt, který umožňuje spouštět hry bez grafického prostředí pro evoluční algoritmus. Obsahuje dvě třídy:

- `Algorithm` — reprezentuje jeden algoritmus, dle kterého může hrát umělá inteligence. Obsahuje pole vah pro vyhodnocování kritérií lokálních rozhodování;
- `BatchRunner` — implementuje samotné spuštění her mezi různými hráči.

6.4 Projekt AlhambraUnitTests

Obsahuje implementaci unit-testování aplikace. Projekt je členěn do několika tříd, které testují nějak sémanticky podobné části programu. Nejdůležitější z nich jsou:

- `AcceptMoveTests` — testuje část herního jádra, která je zodpovědná za změny datových položek na základě tahů hráčů (primárně prostřednictvím metody `AcceptMove`);
- `EvaluatePointsTests` — testuje části aplikace zodpovědné za počítání skóre hry;
- `MoveCheckerTests` — testuje třídu `MoveChecker`, která je zodpovědná za testování jednotlivých tahů, zda jsou v souladu s pravidly hry;
- `AIWeighedMovesTests` — testuje třídu `AIWeighedMoves` a její rozhodování.

Celkový počet jednotlivých unit-testovacích metod je 74.

6.5 Projekt AlhambraSetup

Pomocí tohoto projektu jsme vytvořili instalační balíček `AlhambraSetup.msi`, který nainstaluje hru na cílový počítač.

Pro projekt `AlhambraSetup` je použit `Setup Project`, který je typu `Visual Studio Installer`. Tento typ projektu není nainstalován spolu s instalací Visual Studia a je potřeba jej doinstalovat. V opačném případě nemusí být možné aplikaci v již zmíněném Visual Studiu otevřít. Tento projekt rovněž nemusí být možné doinstalovat do všech verzí vývojového prostředí.

6.6 Známé chyby

Program obsahuje jednu známou chybu, kterou se nepodařilo odladit. Tato chyba se může vyskytnout v situaci, kdy hráč má několik budov z předchozího tahu a jako poslední tah je vybrán tah přestavby Alhambry, a to konkrétně odložení budovy z herní mapy na odkládací plochu a pozice odkládané budovy a pozice budov nových jsou incidentní. V takovém případě může dojít (ale ne vždy dochází) k porušení pravidel hry (incidence stěn nebo vytvoření díry v herní mapě). Tato chyba nebyla zaznamenána během her s uživatelským prostředím, ale pouze během evoluce. Tato situace ve hře není příliš běžná, protože musí dojít zároveň k více specifickým situacím. Případ byl zaznamenán během evoluce nejspíše jen proto, protože jsme odehráli dostatečné množství her a tak se mohla objevit i tato málo pravděpodobná situace.

Každá hra, ve které se objeví tato chyba, je ukončena a v evoluci spuštěna znovu. Do výsledků evoluce se tedy nezapočítává. Celkem se takových her objevilo cca 100 z celkového počtu cca 6 200 000, což odpovídá méně než 0,002% případů.

Závěr

Bakalářská práce poskytuje vlastní implementaci deskové hry Alhambra, kterou je možné hrát pomocí grafického uživatelského prostředí. Hru mohou hrát lidé, kteří se střídají u jednoho počítače, nebo kteréhokoliv hráče může zastoupit počítačový hráč. Dalším hlavním cílem práce, který jsme si předsevzali, bylo vytvořit netriviální umělou inteligenci, a to pomocí genetických algoritmů a strojového učení.

Umělá inteligence provádí své tahy na základě lokálních rozhodnutí. K tomu slouží tzv. kritéria lokálních rozhodování a jejich váhy. Hodnoty těchto vah můžeme chápat i jako jistý výstup práce. Naimplementovali jsme genetický algoritmus, kterým jsme tyto váhy získali, a v kapitole Umělá inteligence jsme vybrali některé z nich a převedli je zpátky na „lidskou“ reprezentaci. Tím dáváme jistý návod, jak hru hrát. Protože data, která jsme získali, jsou poměrně obsáhlá, ponecháme na čtenáři, aby si sám z těchto dat odvodil, jaké jsou dobré tahy pro různé počty hráčů.

Aplikace není bezchybná, o čemž pojednávala sekce 6.6, ale i přesto myslíme, že se předsevzaté cíle podařilo splnit.

Budoucnost

Projekt implementuje standardní pravidla hry, to ale neznamená, že by se na něm v budoucnosti nedalo dále pracovat. Hlavní částí, která se dá takřka neomezeně rozšiřovat, jsou kritéria lokálních rozhodování, která se dají vymýšlet další a další.

Rovněž by mohlo existovat nějaké rozšíření, které by dovolilo umělé inteligenci „podvádět“, například tím, že by bylo možné vidět do hráčových karet. Dále by se mohlo zobrazovat výsledné skóre procentuálně (tj. počet bodů hráče vůči součtu bodů udělených celkem), nebo jinak zvýhodňovat umělou inteligenci (násobení skóre nějakým koeficientem).

Ve hře by se navíc mohla objevit možnost krokovat hry, které hrají pouze počítačoví hráči. Tyto možnosti ponecháváme jako tipy pro další vývoj aplikace.

Seznam použité literatury

[1] Henn, Dirk. 2003. *Alhambra*. Corfix. ISBN 0-9740913-7-5.

[2] Queen Digitals. *Alhambra Game pro Android*. [online]. [cit. 2015-05-14]. Dostupné z: `https://play.google.com/store/apps/details?id=de.brettspielwelt.alhambra`.

[3] Queen Games. *Alhambra Game pro iOS*. [online]. [cit. 2015-05-14]. Dostupné z: `https://itunes.apple.com/us/app/alhambra-game/id724408296?mt=8`.

[4] Microsoft. *MSDN dokumentace jazyka C# a platformy .NET*. [online]. [cit. 2015-04-06]. Dostupné z: `https://msdn.microsoft.com/en-US/library/618ayhy6.aspx`.

[5] Russell, Stuart a Peter Norvig. 2010. *Artificial Intelligence: A Modern Approach*. 3rd ed. Upper Saddle River: Prentice Hall, xviii, 1132 s. Prentice Hall series in artificial intelligence. ISBN 978-0-13-604259-4.

[6] *Popis pravidel hry Alhambra*. [online]. [cit. 2015-04-06]. Dostupné z: `http://www.deskovehry.cz/index.php/Alhambra`.

Seznam obrázků

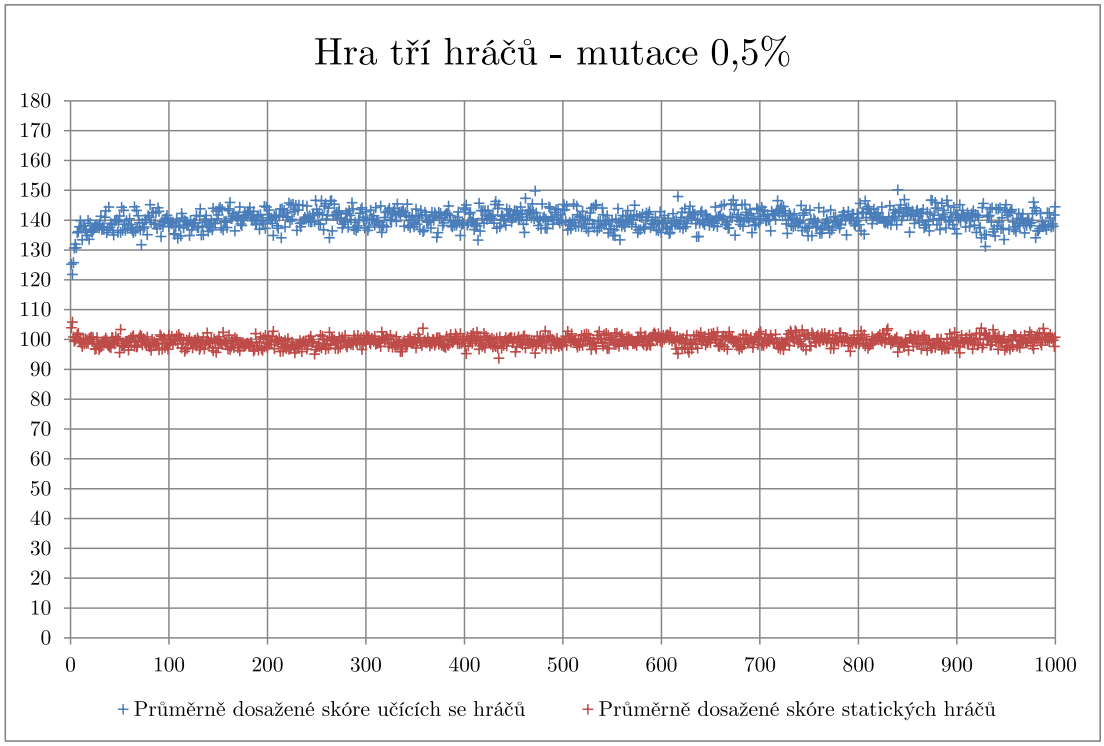
Seznam tabulek

Přílohy

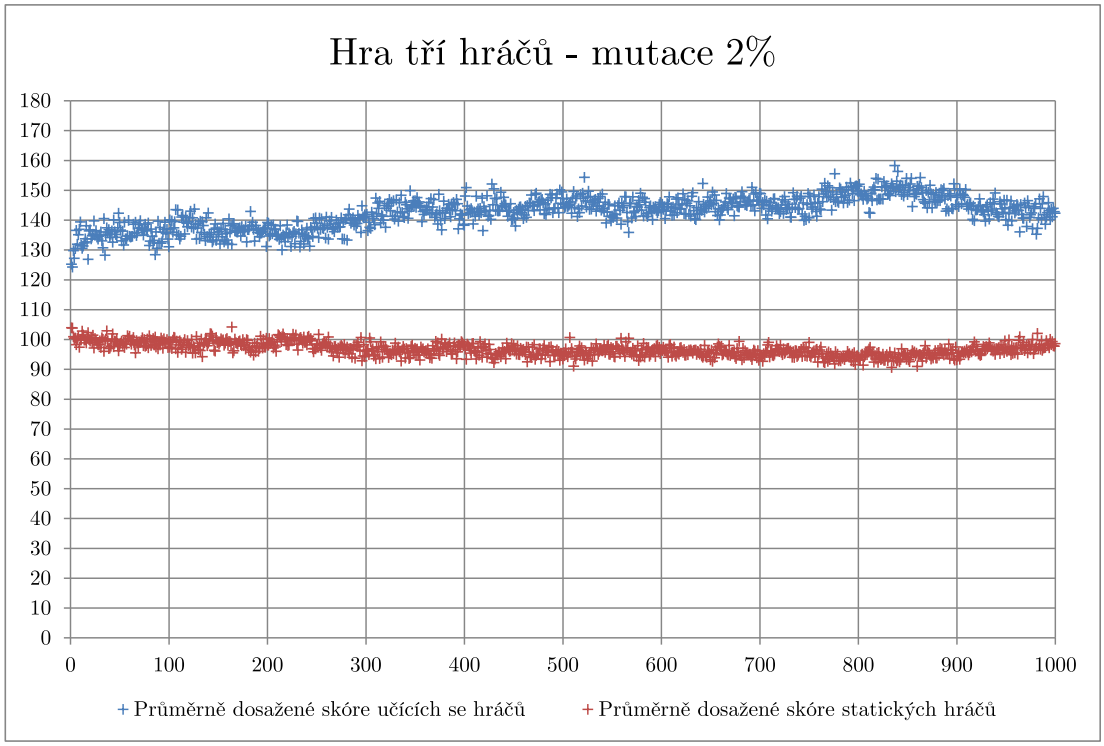
A. Výsledky evoluce – data

V kapitole 3 v sekci 3.3.2 jsme provedli analýzu toho, jak se změní výsledky evoluce, pokud změníme její parametry. Zde uvádíme podklady, ze kterých čerpáme.

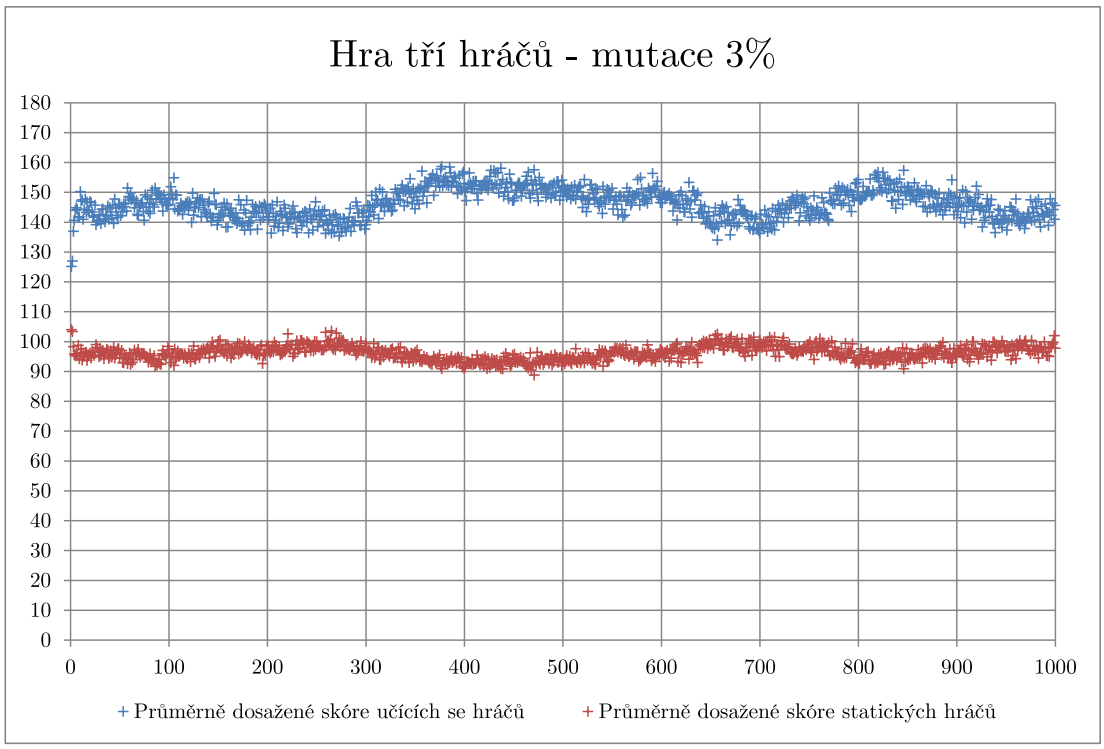
Změny mutace



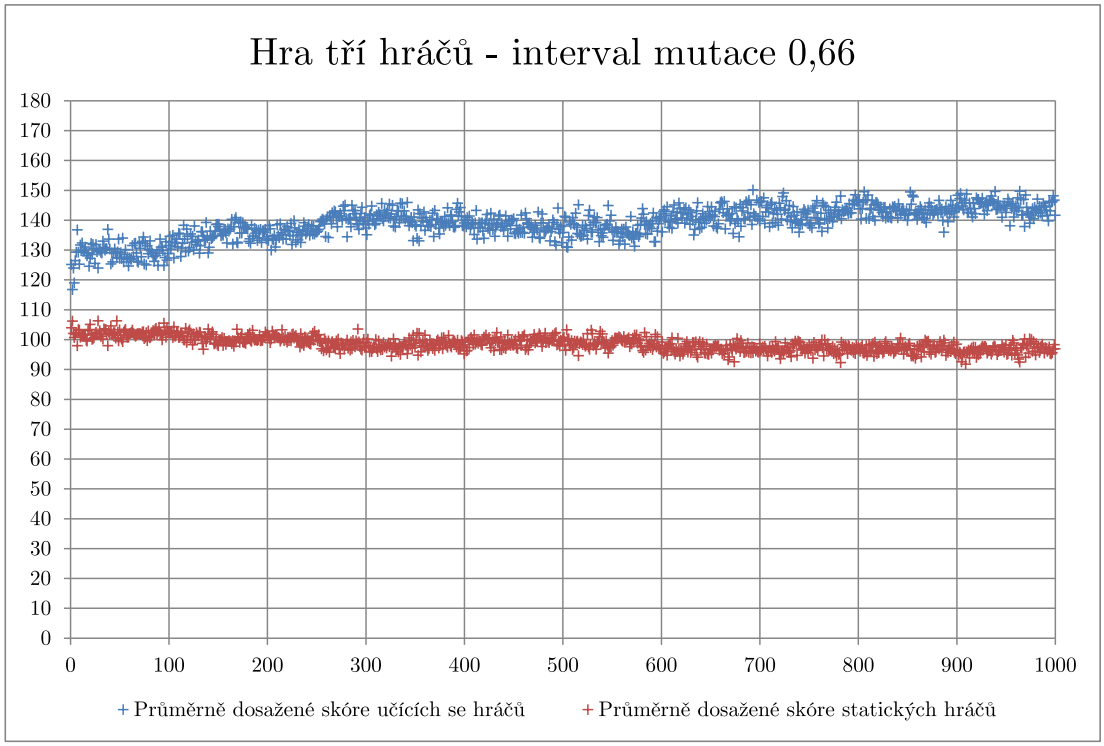
Obrázek A.1: Evoluce hry tří hráčů s mutací 0,5%



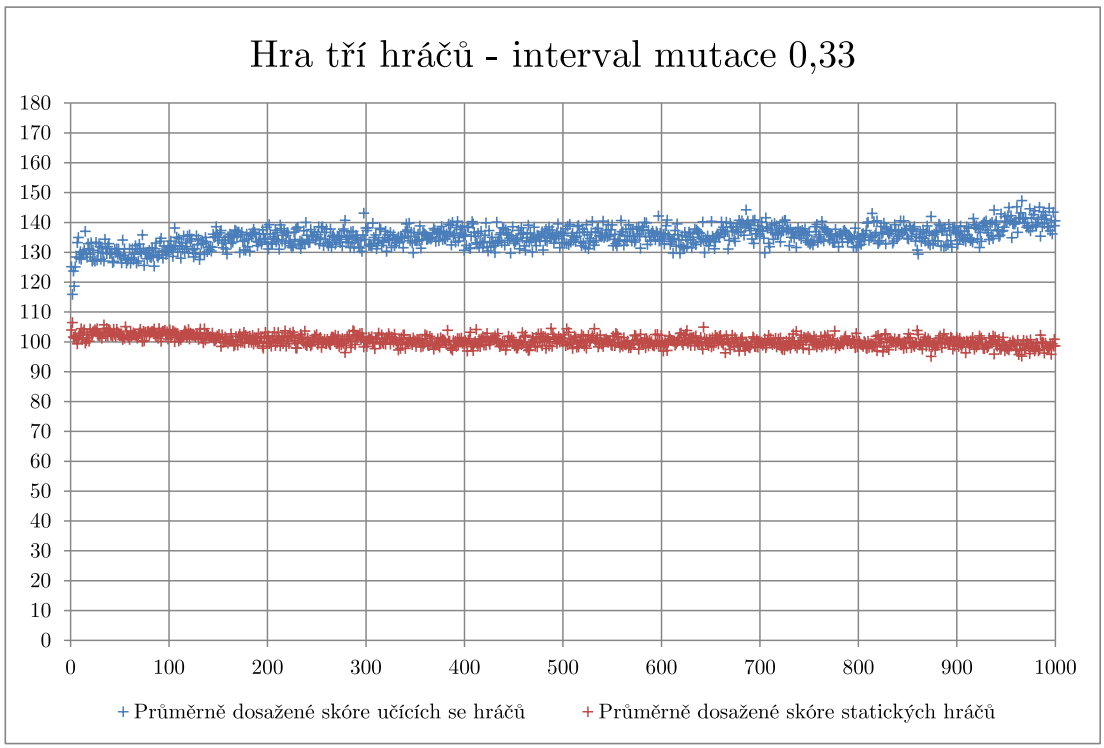
Obrázek A.2: Evoluce hry tří hráčů s mutací 2%



Obrázek A.3: Evoluce hry tří hráčů s mutací 3%

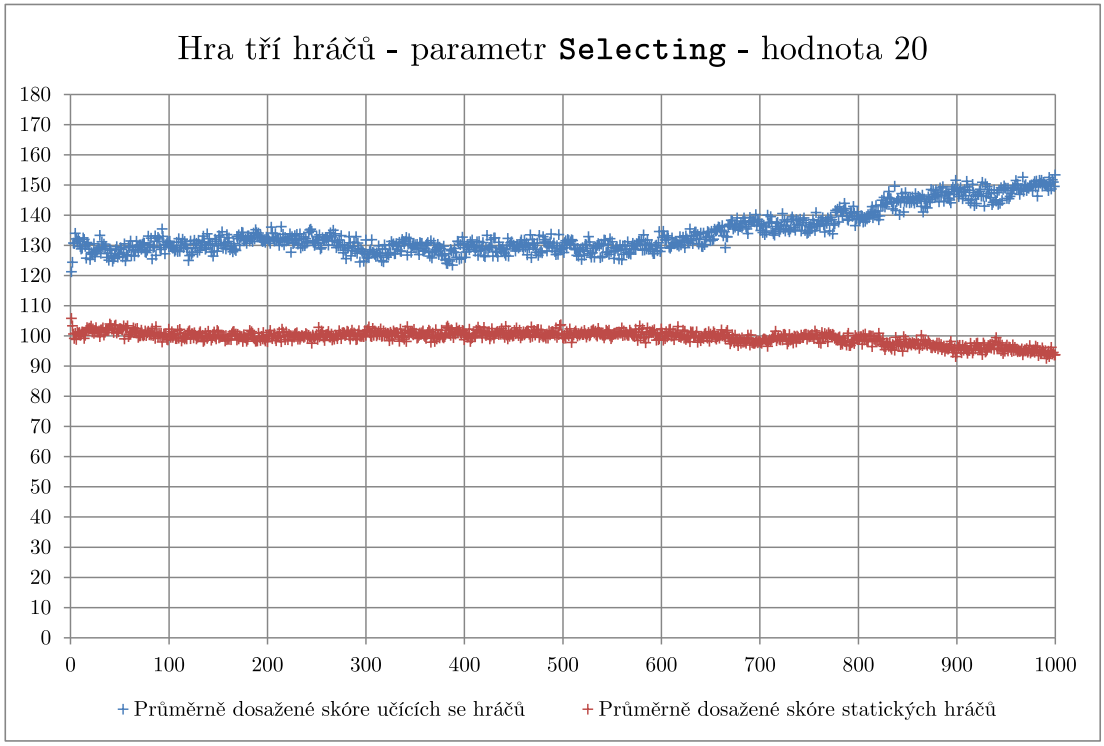


Obrázek A.4: Evoluce hry tří hráčů se zvětšeným intervalem mutace

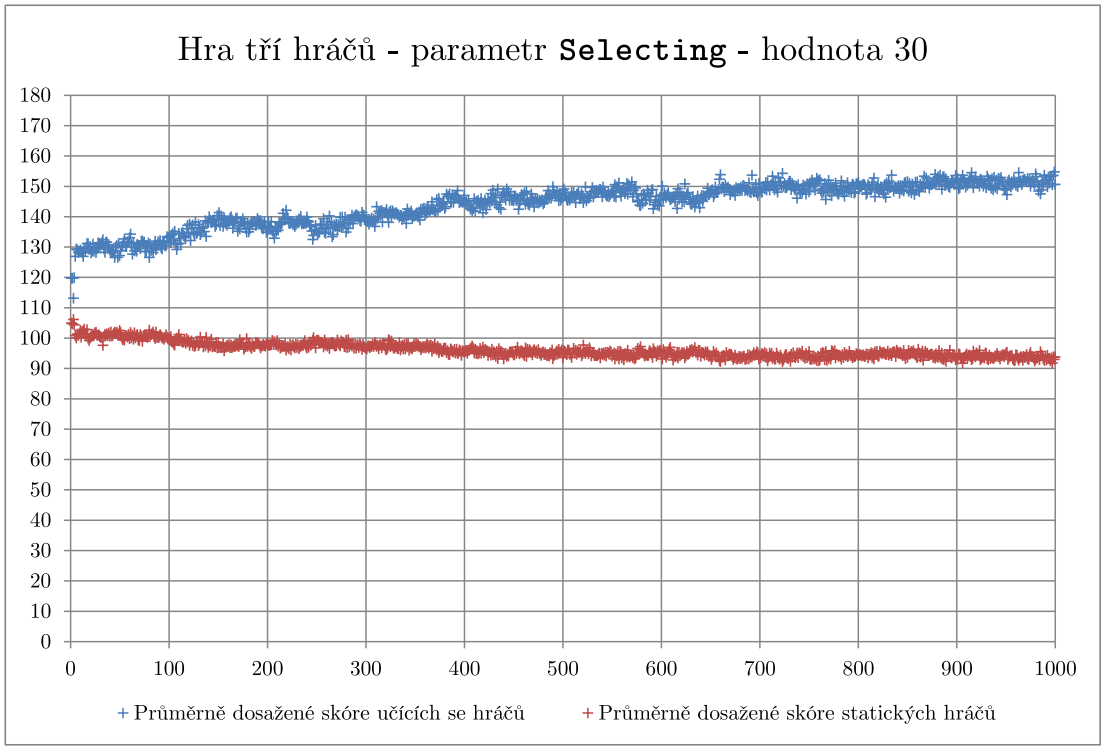


Obrázek A.5: Evoluce hry tří hráčů se zmenšeným intervalem mutace

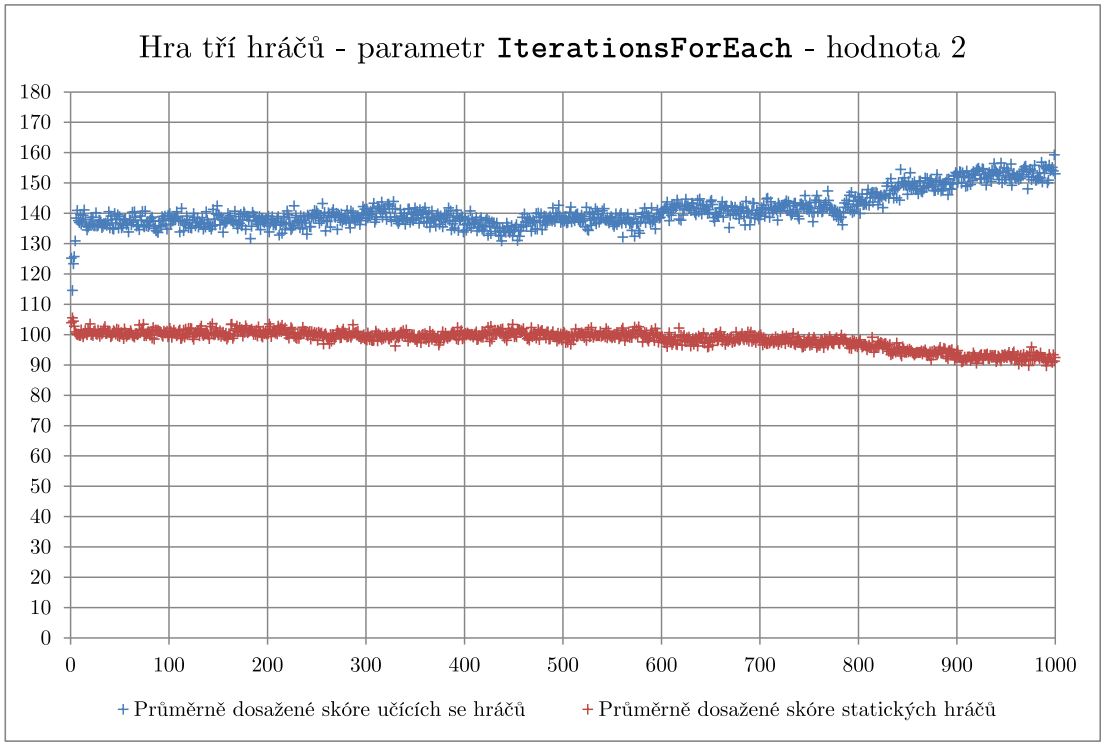
Změny počtu odehraných her v jedné generaci



Obrázek A.6: Evoluce hry tří hráčů s dvakrát větším počtem odehraných her v generaci

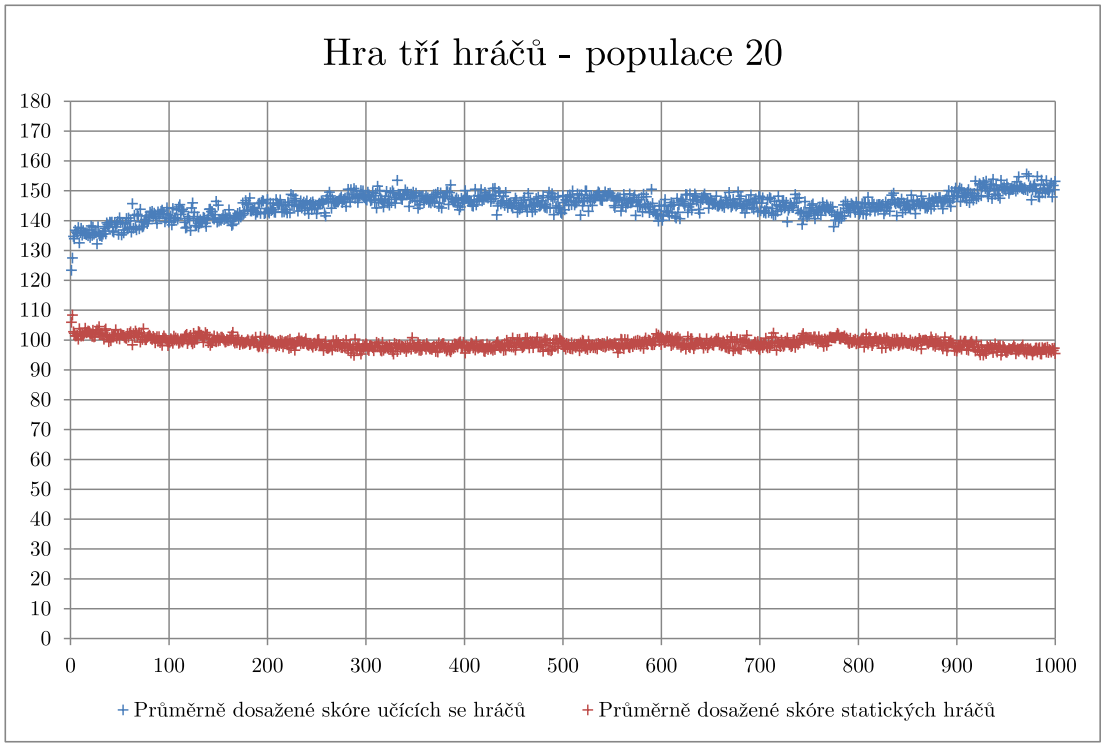


Obrázek A.7: Evoluce hry tří hráčů s třikrát větším počtem odehraných her v generaci

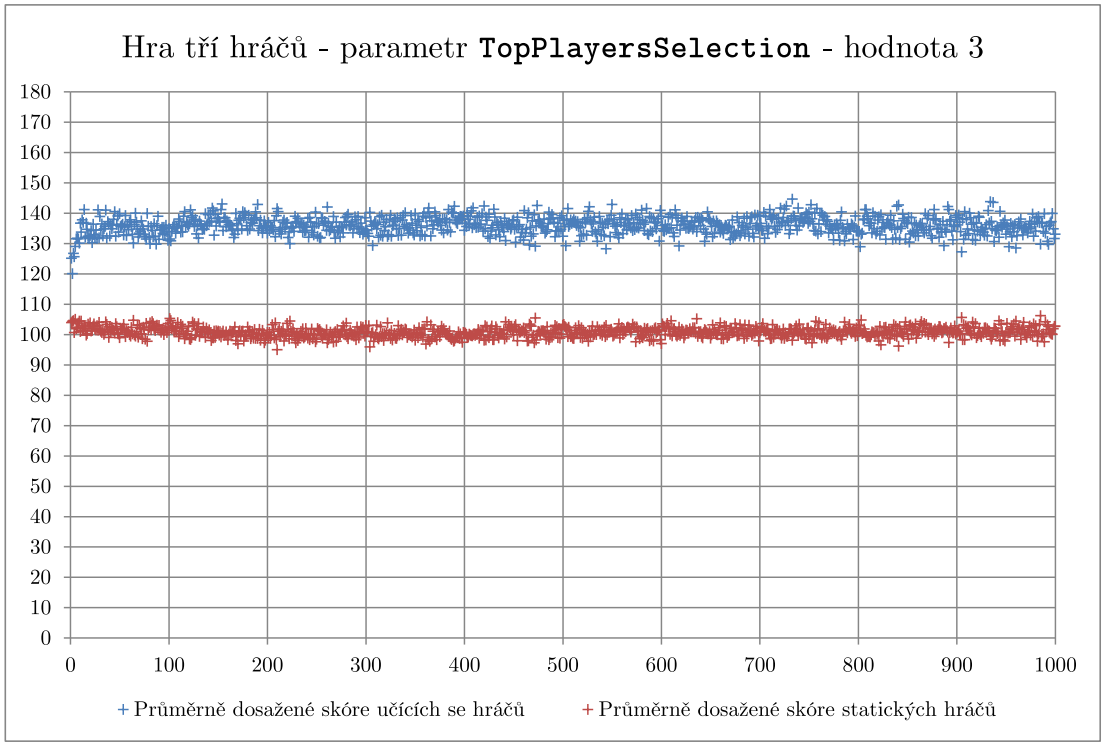


Obrázek A.8: Evoluce hry tří hráčů s dvojnásobnou iterací každé hry

Změna velikosti populace a výběr nejlepších hráčů



Obrázek A.9: Evoluce hry tří hráčů s populací 20 (10-10)



Obrázek A.10: Evoluce hry tří hráčů s větším počtem nejlepších hráčů, kteří přežijí do další generace

B. Kritéria lokálních rozhodování

Následující tabulka zobrazuje všechna kritéria pro lokální rozhodování, která naše umělá inteligence používá. Tato kritéria jsou seřazena tak, jak je můžeme nalézt ve výpisech vah jednotlivých generací (výpis vah obsahuje tyto váhy třikrát za sebou, každé pro určitou fázi hry).

Pokud chceme spočítat výsledek lokálního rozhodování, tak pro všechny možnosti, mezi kterými se rozhodujeme, vyhodnotíme všechna patřičná kritéria a ta z kritérií, která jsou splněna, přispějí svou vahou do kvality dané možnosti.

Jaký typ tahu?	
1.	Jedná se o tah odebrání karet
2.	Mohu vzít dost karet, abych koupil budovu
3.	Mohu vzít dost karet, abych koupil budovu přesně
4.	Mohu vzít kartu s hodnotou 1
5.	Mohu vzít kartu s hodnotou 2
6.	Mohu vzít kartu s hodnotou 3
7.	Mohu vzít kartu s hodnotou 4
8.	Mohu vzít kartu s hodnotou 5
9.	Mohu vzít kartu s hodnotou 6
10.	Mohu vzít kartu s hodnotou 7
11.	Mohu vzít kartu s hodnotou 8
12.	Mohu vzít kartu s hodnotou 9
13.	Jedná se o tah koupení budovy
14.	Mohu koupit budovu přesně
15.	Mohu koupit modrou budovu
16.	Mohu koupit červenou budovu
17.	Mohu koupit hnědou budovu
18.	Mohu koupit bílou budovu
19.	Mohu koupit zelenou budovu
20.	Mohu koupit fialovou budovu
21.	Jedná se o tah přestavba Alhambry
22.	Je má Alhambra celá obestavěná zdmi
23.	Mám odloženou budovu

Tabulka B.1: Seznam kritérií pro lokální rozhodování *Jaký typ tahu?*

Kritéria 1 až 12 se vyhodnocují pro možnost tahu odebrání karet, kritéria 13 až 20 pro koupení budovy a kritéria 21 až 23 pro tah přestavby Alhambry.

Jaké vzít karty?	
24.	Odebírám kartu s hodnotou 1
25.	Odebírám kartu s hodnotou 2
26.	Odebírám kartu s hodnotou 3
27.	Odebírám kartu s hodnotou 4
28.	Odebírám kartu s hodnotou 5
29.	Odebírám kartu s hodnotou 6
30.	Odebírám kartu s hodnotou 7
31.	Odebírám kartu s hodnotou 8
32.	Odebírám kartu s hodnotou 9
33.	Odebírám karty, se kterými budu moci koupit budovu
34.	Odebírám karty, se kterými budu moci koupit budovu přesně
35.	Odebírám kartu barvy, které mám nejvíce

Tabulka B.2: Seznam kritérií pro lokální rozhodování *Jaké vzít karty?*

Jakou budovu koupit? / Jakou budovu do Alhambry zvolit?	
36.	Volím modrou budovu
37.	Volím červenou budovu
38.	Volím hnědou budovu
39.	Volím bílou budovu
40.	Volím zelenou budovu
41.	Volím fialovou budovu
42.	Danou budovou srovnám počet budov dané barvy s nějakým hráčem
43.	Danou budovou zvýším náskok v počtu budov dané barvy na jedna
44.	Danou budovou zvýším náskok v počtu budov dané barvy na dva

Tabulka B.3: Seznam kritérií pro lokální rozhodování *Jakou budovu koupit? / Jakou budovu do Alhambry zvolit?*

Jaké karty pro zaplacení?	
45.	K zaplacení používám jednu kartu
46.	K zaplacení používám dvě karty
47.	K zaplacení používám tři karty
48.	K zaplacení používám více než tři karty
49.	Budovu přeplácím o jedna
50.	Budovu přeplácím o dvě
51.	Budovu přeplácím o tři
52.	Budovu přeplácím o čtyři
53.	Budovu přeplácím o pět
54.	Budovu přeplácím o více než pět
55.	Nepřeplácím budovu

Tabulka B.4: Seznam kritérií pro lokální rozhodování *Jaké karty pro zaplacení?*

Kam umístit budovu?	
56.	Zvýší se délka nejdelší stěny o jedna
57.	Zvýší se délka nejdelší stěny o dvě
58.	Zvýší se délka nejdelší stěny o tři
59.	Zvýší se délka nejdelší stěny o čtyři
60.	Zvýší se délka nejdelší stěny o více než čtyři
61.	Alhambra nebude celá obestavěná zdmi
62.	Přenechání budovy imaginárnímu hráči [1]

Tabulka B.5: Seznam kritérií pro lokální rozhodování *Kam umístit budovu?*

Jaký typ přestavby? [2]	
63.	Je celá Alhambra obestavěná zdmi a mám odloženou budovu

Tabulka B.6: Seznam kritérií pro lokální rozhodování *Jaký typ přestavby?*

Jakou budovu odebrat?	
64.	Nesníží se délka nejdelší stěny
65.	Sníží se délka nejdelší stěny o jedna
66.	Sníží se délka nejdelší stěny o dva
67.	Sníží se délka nejdelší stěny o tři
68.	Sníží se délka nejdelší stěny o více než tři
69.	Budova nemá žádnou stěnu
70.	Budova má jednu stěnu
71.	Budova má dvě stěny
72.	Budova má tři stěny

Tabulka B.7: Seznam kritérií pro lokální rozhodování *Jakou budovu odebrat?*

Jaké budovy prohodit?	
73.	Prohození budov zvýší nejdelší stěnu o jedna
74.	Prohození budov zvýší nejdelší stěnu o dva
75.	Prohození budov zvýší nejdelší stěnu o tři
76.	Prohození budov zvýší nejdelší stěnu o více než tři

Tabulka B.8: Seznam kritérií pro lokální rozhodování *Jaké budovy prohodit?*

[1] Přesná definice tohoto kritéria: Je-li před koncem hry (tj. zbývá poslední budova plus budovy na trhu) a je režim dvou hráčů, tak přenechání budovy imaginárnímu hráči způsobí, že počet budov patřičné barvy imaginárního hráče se srovná s tímto počtem soupeře.

[2] Zde používáme navíc již dvě použitá kritéria – „Je má Alhambra celá obestavěná zdmi?“ a „Mám odloženou budovu?“.

C. Obsah přiloženého DVD

- `AlhambraPrace.pdf` — tato práce v elektronické podobě;
- `AlhambraSetup.msi` — instalační balíček;
- `Projekt` — složka obsahující celý projekt pro Microsoft Visual Studio 2013, včetně všech zdrojových kódů;
- `Evoluce` — složka obsahující výsledky evoluce;
- `Dokumentace` — složka obsahující vygenerovanou dokumentaci pro doplnění technické části práce.