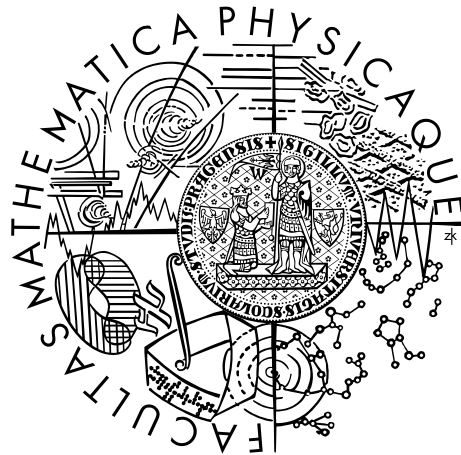


Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Pavel Kratochvíl

Umělá inteligence pro osadníky

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Filip Zavoral, Ph.D.

Studijní program: Informatika

Studijní obor: Obecná informatika

Praha 2016

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 v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V dne

Podpis autora

Název práce: Umělá inteligence pro osadníky

Autor: Pavel Kratochvíl

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Filip Zavoral, Ph.D., Katedra softwarového inženýrství

Abstrakt: Předmětem bakalářské práce je implementace hry Osadníci z katanu ve variantě pro dva až čtyři hráče. Dále se v práci zabýváme vývojem umělé inteligence do hry. Při návrhu implementace umělé inteligence bylo využito genetických algoritmů tak, aby počítačový hráč byl schopen porazit průměrného hráče této hry.

Klíčová slova: Umělá inteligence, Genetické algoritmy, Osadníci

Title: Artificial Intelligence for Settlers

Author: Pavel Kratochvíl

Department: Department of Software Engineering

Supervisor: RNDr. Filip Zavoral, Ph.D., Department of Software Engineering

Abstract: The subject of the thesis is implementation of the game Settlers of Catan for two to four players. Artificial intelligence for this game was also developed in this thesis. Moreover, genetic algorithms were used in the implementation of the artificial intelligence, so that a computer player was able to beat the average player of this game.

Keywords: Artificial intelligence, Genetic algorithms, Settlers

Na tomto místě bych rád poděkoval RNDr. Filipu Zavoralovi, Ph.D za cenné připomínky a rady při psaní práce. Dále bych chtěl poděkovat rodině za podporu při studiu.

Obsah

| | |
|--|-----------|
| Úvod | 3 |
| 0.1 Cíle bakalářské práce | 3 |
| 0.2 Popis hry | 3 |
| 0.3 Struktura práce | 3 |
| 1 Popis strategie | 4 |
| 1.1 Popis tahu | 4 |
| 1.2 Výběr tahu | 4 |
| 1.3 Reprezentace jedince | 5 |
| 1.4 Ohodnocení tahu | 5 |
| 1.4.1 Názvosloví | 5 |
| 1.4.2 První dva tahy hry - postavení cesty a vesnice | 5 |
| 1.4.3 Postavení cesty | 6 |
| 1.4.4 Postavení vesnice | 6 |
| 1.4.5 Postavení města | 7 |
| 1.4.6 Přemístění zloděje | 7 |
| 1.4.7 Koupě akční karty | 8 |
| 1.4.8 Použití akční karty | 8 |
| 2 Evoluční algoritmy | 10 |
| 2.1 Princip evolučního algoritmu | 10 |
| 2.2 Operátory evolučního algoritmu | 10 |
| 2.2.1 Selektce | 10 |
| 2.2.2 Křížení | 10 |
| 2.2.3 Mutace | 11 |
| 2.3 Ohodnocovací funkce | 11 |
| 2.3.1 Pevný protihráč | 11 |
| 2.3.2 Všichni proti všem | 11 |
| 2.3.3 Učící se pevný protihráč | 12 |
| 2.4 Parametry evolučního algoritmu | 12 |
| 3 Výsledky evoluce | 14 |
| 3.1 Nastavení parametrů | 14 |
| 3.1.1 Počet generací | 14 |
| 3.1.2 Velikost populace a počet her při ohodnocení | 14 |
| 3.1.3 Mutace a křížení | 15 |
| 3.1.4 Počet hráčů | 15 |
| 3.2 Způsob šlechtění | 15 |
| 3.3 Interpretace výsledků | 17 |
| 3.3.1 Vliv mutace a křížení | 17 |
| 3.3.2 Vyhodnocení závěrečného turnaje | 18 |
| 3.3.3 Styl hry strategie | 19 |

| | |
|--|-----------|
| 4 Implementace hry | 21 |
| 4.1 Technologie | 21 |
| 4.2 Návrh | 21 |
| 4.3 Jádro hry | 22 |
| 4.3.1 Třída Game | 22 |
| 4.3.2 Třída GameProperties | 23 |
| 4.3.3 Třída Player | 23 |
| 4.3.4 Třída PlayerProperties | 23 |
| 4.3.5 Třída Move | 23 |
| 4.3.6 Třída GameBorder | 24 |
| 4.4 Grafické rozhraní | 25 |
| 4.4.1 Třída MainWindow | 25 |
| 4.4.2 Třída GameWindow | 25 |
| 4.4.3 Třída Draw | 26 |
| 5 Implementace AI | 27 |
| 5.1 Simulátor | 27 |
| 5.1.1 Implementace simulátoru | 27 |
| 5.1.2 Ovládání | 28 |
| 5.2 Evoluce | 29 |
| 5.2.1 Implementace evoluce | 29 |
| 5.2.2 Nastavení a spuštění evoluce | 29 |
| 5.3 Osadníci z katanu AI | 30 |
| 6 Uživatelská dokumentace | 31 |
| 6.1 Nastavení nové hry | 31 |
| 6.2 Herní okno | 31 |
| 6.3 Ovládání hry | 32 |
| 6.3.1 První dva tahy hry | 32 |
| 6.3.2 Postavení budovy | 33 |
| 6.3.3 Přemístění zloděje | 33 |
| 6.3.4 Použití akční karty | 33 |
| 6.3.5 Výměna surovin | 34 |
| 6.3.6 Konec hry | 35 |
| 7 Závěr | 36 |
| 7.1 Zhodnocení výsledků | 36 |
| 7.2 Další vývoj | 36 |
| Seznam použité literatury | 38 |
| Příloha A - výsledky evoluce | 40 |
| Příloha B - parametry jedince | 45 |
| Příloha C - xml soubory | 46 |
| Příloha D - struktura naměřených dat | 47 |
| Příloha E - obsah přiloženého DVD | 48 |

Úvod

0.1 Cíle bakalářské práce

Tato bakalářské práce má dva cíle. Prvním cílem je naprogramovat implementaci hry Osadníci z Katanu ve verzi pro dva až čtyři hráče. Ve hře je dále možno generovat náhodný hrací plán nebo využít předpřipravený hrací plán. Druhým cílem je naprogramovat strategii pro tuto hru, která bude schopna porážet průměrného hráče této hry. Strategie je reprezentována vektorem čísel, podle kterého se ohodnotí jednotlivé možné tahy hráče. Za pomoci nástrojů evolučních algoritmů vyšlechtíme strategii na požadované úrovni obtížnosti.

0.2 Popis hry

Hru Osadníci z Katanu vytvořil v roce 1995 německý designer deskových her Klaus Teuber [1]. Hra se skládá z balíčku karet se surovinami, balíčku akčních karet, hrací desky a figurek měst, vesnic a cest. Princip hry je stavět města, vesnice a cesty na hracím plánu. Budovy následně produkují suroviny, které přísluší políčku, kde budova stojí. Za tyto suroviny je možné nakupovat akční karty nebo stavět další budovy. Hra končí v okamžiku, kdy nějaký hráč získá deset a více bodů. Podrobná pravidla lze najít například zde¹. Naimplementovaná hra obsahuje některé odlišnosti od původních pravidel. První se týká akčních karet. V původních pravidlech za určitých podmínek lze použít akční kartu ihned po zakoupení, a za jiných podmínek nikoliv. V rámci jednoduchosti jsem se rozhodl, že v imlementované hře bude vždy možné použít akční kartu ihned po jejím zakoupení. Dále také nemohou v naimplementované hře hráči měnit suroviny mezi sebou. Velmi by to zkomplikovalo uživatelské rozhraní hry, protože hra neimplementuje síťové rozhraní a tedy lze hrát pouze na jednom počítači.

0.3 Struktura práce

V první kapitole je popsán princip strategie a způsob reprezentace strategie jako vektoru čísel. V další kapitole je popsán princip evolučních algoritmů a způsob šlechtění strategií. V kapitole následující jsou zobrazeny a zhodnoceny jednotlivé varianty šlechtění. Další dvě kapitoly popisují implementaci hry a umělé inteligence. Na to navazuje i kapitola další, ve které je uživatelská dokumentace hry Osadníci z Katanu. Nakonec v závěru práce jsou zhodnoceny výsledky a možná rozšíření programu.

¹<http://www.albi.cz/gallery/download/2925/>

1. Popis strategie

Strategií budeme rozumět obecný popis chování a rozhodování počítačového hráče. Strategie je parametrizována vektorem čísel. Údaje v tomto vektoru udávají váhu jednotlivým dílčím rozhodnutím. Podrobně budou popsány v sekci 1.3. Tento vektor, který reprezentuje jednu konkrétní strategii, budeme nazývat jedinec.

Strategie při generování tahu nejdříve vygeneruje všechny tahy, které daný hráč může zahrát. Poté každý tah ohodnotí podle zvolených pravidel. Tomuto ohodnocení říkáme fitness. Pravidla pro ohodnocení jsou popsány v sekci 1.4. V poslední fázi vybere strategie tah s největším ohodnocením a ten se provede.

1.1 Popis tahu

Tah ve hře se skládá z několika částí. Těmto částem budeme říkat podtah. Existují tyto typy podtahů:

- Výměna surovin. Tento podtah může hráč zahrát v kterékoli části tahu kolikrát chce.
- Stavba. Hráč může postavit město, vesnici, cestu nebo akční kartu. Tento podtah může hráč zahrát pouze jednou za tah.
- Použití akční karty. Ve hře je pět druhů akčních karet. Jsou to - kupón, rytíř, dva materiály zadarmo, dvě cesty zadarmo a surovina od hráčů. Opět tento podtah může hráč zahrát pouze jednou za tah.
- Přemístění zloděje. Speciální podtah, hráč ho může (a zároveň musí) zahrát pouze v případě, že na kostkách padla sedmička nebo použil akční kartu rytíř.
- První dva tahy hry. Další speciální podtah. Je reprezentován jako jeden podtah, i když obsahuje dvě události. Stavbu vesnice a stavbu cesty.

Podtah výměna surovin je speciální, protože ho hráč může provést kolikrát chce. V typickém případě hráč vymění surovinu, aby mohl provést nějaký další podtah. Proto je ve strategii podtah reprezentován jako dvojice. První část reprezentuje výměnu surovin a druhá část reprezentuje stavbu, použití akční karty, přemístění zloděje nebo první dva tahy hry. Část reprezentující výměnu surovin může být prázdná.

1.2 Výběr tahu

V případě, že je strategie na tahu, vygeneruje všechny přípustné podtahy. Z těchto podtahů vždy vybere ten s nejlepším fitness. Toto opakuje, dokud existují nějaké podtahy, které může strategie provést. Je zde vidět určitá nevýhoda. Pokud má strategie alespoň nějaký možný podtah na výběr, tak ho provede. Občas je výhodné podtah neprovést a pošetřit si suroviny na příští kolo. Alternativa k tomu by mohla být strategie, která provede podtah jenom pokud jeho

fitness dosahuje určité hodnoty. Ovšem museli bychom dořešit výjimky například pro podtah na přemístění zloděje. Pokud padne na kostkách sedmička, tak hráč musí přemístit zloděje. Strategie vygeneruje všechny možné varianty tohoto tahu. Pokud by žádný z těchto podtahů nepřekročil minimální hranici ohodnocení, tak by se podtah přemístění zloděje neprovedl a strategie by porušila pravidla hry.

1.3 Reprezentace jedince

Jedinec obsahuje údaje, které udávají váhu dílčím rozhodnutím. V tabulce 7.2 v příloze B je uvedený název parametru a jeho zkratka, pod jakou se na něj budeme odkazovat v sekci 1.4. Název parametru je pak použit v xml reprezentaci jedince. Příklad konkrétního jedince v xml formátu je uveden v příloze C.

1.4 Ohodnocení tahu

V této sekci rozebereme popis ohodnocení jednotlivých podtahů. Ve vzorcích uvedených u každého z podtahů se budeme odkazovat na parametry jedince uvedené v tabulce 7.2 v příloze B.

1.4.1 Názvosloví

K popisu ohodnocení jednotlivých tahů je třeba zavést názvosloví hracího plánu. Hrací plán se skládá ze šestiúhelníků se surovinami. Těmto šestiúhelníkům budeme říkat stěny. Místa pro postavení vesnic a měst nazveme vrcholy, místa na postavení cest nazveme hrany. Každý vrchol tedy sousedí maximálně se třemi stěnami, každá hrana má dva vrcholy, a každá stěna sousedí se šesti vrcholy. Dále budeme často popisovat vlastnosti hráče na tahu. Takovému hráči budeme říkat hrající hráč, obvykle ve smyslu hráč řízený strategií. Oproti tomu všechny ostatní hráče budeme nazývat soupeř.

1.4.2 První dva tahy hry - postavení cesty a vesnice

Tyto dva podtahy jsou zásadní pro průběh celé hry. Na druhu surovin, ke kterým má hráč na začátku přístup, závisí styl hry hráče. Například pokud má hráč na začátku přístup ke kameni, vlně a obilí dá se předpokládat, že bude kupovat akční karty. Naopak pokud bude mít na začátku přístup k dřevu a cihle, bude se pravděpodobně snažit vytvořit nejdelší cestu a získat za ní dva body.

Tento podtah je určený vrcholem a hranou. Tedy strategie generuje všechny možné volné dvojice vrchol a hrana, kde hrana sousedí s vrcholem. Volné znamená, že na vrcholu může hrající hráč postavit vesnici (na vrcholu ani v jeho bezprostředním okolí neleží jiná vesnice) a na hraně může postavit cestu (neleží tam jiná cesta). Poté ohodnotí všechny tyto podtahy a vybere ten s největším fitness.

Nyní předpokládejme, že chceme ohodnotit podtah, kde vesnici postavil hrající hráč na vrchol a a cestu postavil na hranu b . Při ohodnocení tohoto tahu bychom měli zohlednit typ suroviny a pravděpodobnost produkce této suroviny na sousedních stěnách vrcholu a . Dále bychom měli vzít v úvahu, jestli na vrcholu

a leží nějaký přístav a počet hran, se kterými sousedí vrchol a (kvůli možnosti dále expandovat). Tyto faktory jsme zohlednili ve vzorci na výpočet fitness tahu. Je zde možnost rozšíření tohoto vzorce o další faktory zohledňující například větší okolí vrcholu a . Možnosti dalšího vývoje budeme diskutovat v závěru práce. Pro vypočítání fitness tahu jsem zvolil následující vzorec:

$$fitness = \sum_{\text{sousední stěny s } a} weightFace_{start} + ports + edgeGen \cdot edgeCount$$

kde $ports$ má hodnotu $villagePortTwo$, pokud vesnice leží na přístavu dvě ku jedné, $ports$ je $villagePortThree$, pokud vesnice leží na přístavu tři ku jedné a $ports$ je nula, pokud vesnice neleží na přístavu. Dále $edgeCount$ je počet hran, se kterými vrchol a sousedí. A nakonec zbývá uvést způsob výpočtu $weightFace_{start}$ pro jednotlivé stěny:

$$weightFace_{start} = goodNum \cdot curMat \cdot probFace + ctMiss \cdot missMat$$

kde $curMat$ je materiál na stěně (respektive parametr ze sekce 1.3 s tímto názvem), $ctMiss$ je počet stěn se surovinami, ke kterým zatím nemáme přístup (nemáme postavenou vesnici u stěny s tímto typem suroviny). Poslední parametr $probFace$ pak udává pravděpodobnost, s jakou padne na kostkách číslo, které je uvedené na stěně.

Cílem tohoto ohodnocení je seřadit vrcholy podle lukrativnosti. Největší váha by měla být na typu surovin v okolí a zejména na jejich pravděpodobnosti produkce.

1.4.3 Postavení cesty

Uvažme podtah, kdy hrající hráč postaví cestu na hraně e . Při ohodnocení tohoto podtahu zohledňujeme, zda postavením této cesty vznikla nová nejdelší cesta, zda se prodloužila nejdelší cesta hrajícího hráče, a také jestli vzniklo nové místo pro postavení vesnice. Opět je zde možnost v podstatě libovolně rozšiřovat vzorec pro výpočet o další aspekty, které by například zohledňovaly širší okolí při stavbě cesty. Možnosti rozšíření budou diskutované v závěru práce. Jako vzorec pro výpočet fitness jsem zvolil součet výše uvedených aspektů. Tedy vzorec vypadá takto:

$$fitness = roadGen + extRoad + longRoad + villageSpace$$

1.4.4 Postavení vesnice

Uvažme podtah hrajícího hráče, který postaví vesnici na vrcholu a . Ohodnocení tohoto podtahu je podobné jako ohodnocení prvních dvou tahů hry. Opět budeme nejvíce zohledňovat typ suroviny na stěnách sousedících s vrcholem a , pravděpodobnost generování suroviny a také, zda na vrcholu a leží přístav. Výsledný vzorec jsem zvolil následující:

$$fitness = villageGen + \sum_{\text{sousední stěny s } a} weightFace_{village} + ports$$

kde

$$weightFace_{village} = villageGoodNum \cdot probFace$$

je ohodnocení stěny, se kterou vesnice sousedí. $probFace$ je potom pravděpodobnost, že na kostkách padne číslo, které je na stěně. Hodnota parametru $ports$ se vypočítá stejně, jako hodnota tohoto parametru v prvních dvou tazích hry.

1.4.5 Postavení města

Tento podtah je hodnocen podobně, jako podtah postavení vesnice. Uvažme tedy podtah, kerý postaví město na vrcholu a . Pro ohodnocení jsem zvolil vzorec:

$$fitnes = townGen + \sum_{\text{sousední stěny s a}} weightFace_{town}$$

kde

$$weightFace_{town} = townGoodNum \cdot probFace$$

Tedy oproti výpočtu fitnes u vesnice je rozdíl v parametrech. Dále u výpočtu nebereme v úvahu přístavy. Zde při postavení města nenastává žádná změna. Hrající hráč má stále stejnou možnost výhodné výměny surovin a není tedy žádný důvod k započítání tohoto faktu do ohodnocení podtahu.

1.4.6 Přemístění zloděje

Podtah přemístění zloděje je definován stěnou, kam hrající hráč přemístí zloděje, a hráčem, kterého zloděj okrade. V popsaném výpočtu fitnes nebereme v úvahu, koho hrající hráč okrade. Tedy strategie pouze ohodnotí stěny podle podmínek popsaných v následujícím odstavci.

Dejme tomu, že na kostkách padla sedmička a hrající hráč přesunul zloděje na stěnu s . Ohodnocení tohoto podtahu by mělo záviset především na budovách, které jsou na vrcholech sousedících se stěnou s . Přičemž čím více budov soupeře, tím větší ohodnocení, naopak čím více budov hrajícího hráče, tím menší. Zde si uvědomme, že i pokud je na vrcholu u stěny s budova hrajícího hráče, tak může být výhodné sem umístit zloděje. Typický příklad může vypadat tak, že na stěně s budou dvě města soupeře a jedna vesnice hrajícího hráče. Na druhou stranu pokud se na stěně s nenachází žádné soupeřovy budovy, nemá většinou smysl přemísťovat zloděje zrovna sem. Tedy pokud na stěně s není žádná soupeřova budova, pak má podtah hodnotu parametru s názvem $thiefMove$. V opačném případě jsem zvolil vzorec, který přímo závisí na počtu soupeřových budov, a nepřímo závisí na počtu budov hrajícího hráče na stěně s . Tedy jako základ vzorce použijeme podíl počtu soupeřových budov a budov hrajícího hráče. Počet budov hrajícího hráče bude navíc navýšen o jedna, aby se zamezilo dělení nulou, pokud by na stěně neměl hrající hráč žádnou budovu. Dále je počet budov soupeře, i počet budov hrajícího hráče, vynásoben koeficientem. Tedy na jedinci závisí, jakou dá váhu každému z těchto dvou údajů. Nakonec celý podíl vynásobíme pravděpodobností produkce suroviny stěny s . Tato pravděpodobnost je opět opatřena koeficientem, který je zároveň parametr v jedinci. To zajistí, že váha této pravděpodobnosti je také zavislá na jedinci. Výsledný vzorec vypadá takto:

$$fitnes = \frac{thiefMove + ctOthBd \cdot thiefOthBd}{ctMyBd \cdot thiefMyBd + 1} \cdot prob \cdot thiefProb$$

kde $ctOthBd$ je počet budov soupeře (měst a vesnic) na stěně s . Oproti tomu $ctMyBd$ je počet budov hrajícího hráče na stěně s . Nakonec $prob$ je pravděpodobnost, že na kostkách padne číslo, které je uvedeno na stěně, kam se přemísťuje zloděj.

1.4.7 Koupě akční karty

Fitnes tohoto podtahu je rovno přímo hodnotě parametru s názvem *weight-BuyActionCardGeneral*. Opět je zde možnost rozšíření a zahrnutí více faktorů. Například pokud mám dva rytíře, tak si spíš koupím akční kartu, protože rytířů je v balíčku akčních karet nejvíc a je velká pravděpodobnost, že si ho vytáhnu. To je již však nad rámec této bakalářské práce.

1.4.8 Použití akční karty

Použití akční karty kupón Fitnes je rovno parametru s názvem *weightUseActionCardGeneral*. V průběhu hry není úplně zásadní, kdy se kupón použije. Je akorát hlavní aby se hráči nehromadily tyto akční karty. Protože pokud by hráč měl v záloze dvě akční karty kupón a zároveň by už měl celkově 8 bodů, tak by musel čekat další kolo na vítězství (hráč nemůže použít dvě akční karty za tah). Mezi tím by mohl vyhrát soupeř. Tedy tato akční karta se použije, pokud hráč nemá v rukávu žádnou další akční kartu, jejíž použití by mu přineslo více užitku.

Použití akční karty - rytíř Použití této akční karty je definované stěnou, kam hrající hráč přemístí zloděje. Tedy pokud hrající hráč vlastní akční kartu rytíř, strategie vygeneruje možné tahy pro přemístění zloděje. Je třeba ohodnotit jednotlivé stěny podle vhodnosti umístění zloděje. Takové ohodnocení jsme již jednou prováděli v sekci 1.4.6. Fitnes tohoto tahu se tedy počítá stejně, jako fitnes pro přemístění zloděje. Jako možné rozšíření se nabízí zohlednit pozici zloděje nebo počet již vyložených rytířů.

Použití akční karty - dva materiály zadarmo Tento podtah je určený dvěma surovinami, které hráč na tahu vybere. Strategie ovšem negeneruje všechny možné dvojice surovin, protože tato akční karta má smysl použít pouze pokud ještě ten tah tyto suroviny hrající hráč použije. V opačném případě pouze riskuje, že mu je někdo ukradne. Tedy jako možné tahy se generují pouze ty, které splňují tuto podmínku. Nejdříve zjistíme, zda byla tento tah již postavena nějaká stavba. Pokud zatím žádná stavba postavena nebyla, tak strategie zjistí, jestli nezbývá jedna nebo dvě suroviny k postavení po řadě cesty, vesnice, města nebo akční karty. Nakonec se vygenerují pouze tyto podtahy. Ohodnocení podtahů bude záviset na stavbě, kterou bude možné poté postavit. Fitnes se tedy rovná parametru $actTwoMat_{stavba}$, kde stavba je *road*, *village*, *town* nebo *act* po řadě pokud budu moct postavit cestu, vesnici, město nebo akční kartu.

Použití akční karty - dvě cesty zadarmo Mějme podtah, kdy hráč postaví cestu na hraně e a f . Fitnes pak záleží na umístění těchto cest. Ohodnocení cesty je popsáno v sekci 1.4.3. Použijeme tedy toto ohodnocení cesty pro každou z těchto

dvou cest zvlášť. Tyto dvě hodnoty sečteme a výsledek vynásobíme koeficientem, který je v parametru s názvem *weightUseTwoRoadGeneral*.

Použití akční karty - surovina od hráčů Při výpočtu fitness tohoto podtahu vycházíme z toho, že hrající hráč potřebuje víc tu surovinu, které má málo. Pokud hráč zahraje tuto akční kartu s tím, že vybere surovinu s , tak fitness vypočítáme jako:

$$fitness = \frac{weightUseMatFromPlGeneral}{count(s) + 1}$$

kde $count(s)$ je počet suroviny typu s hrajícího hráče. Opět je zde více možností pro ohodnocení. Například můžeme zohlednit, jestli hrajícímu hráči nezbyvá jedna surovina k provedení akce (postavení budovy nebo cesty). Ovšem při použití této akční karty není jisté, že nějakou surovinu získáme. Proto jsem zvolil tento vzorec.

2. Evoluční algoritmy

Strategie, která byla popsána v předchozí kapitole, je definována vektorem čísel. Tedy k nalezení chytré AI budeme potřebovat nástroj na prohledávání vektorového prostoru. Pod pojmem vektorový prostor si lze představit množinu všech možných jedinců. K tomuto účelu se ideálně hodí evoluční algoritmy [2]. V této kapitole budou popsány základní principy evolučních algoritmů použitých v této práci.

2.1 Princip evolučního algoritmu

Evoluční algoritmy se snaží simulovat evoluci, která probíhá v běžném životě. Nejprve zavedeme základní terminologii. Jedincem budeme rozumět vektor celých čísel. Skupina jedinců se nazývá populace. Dále se pak množina jedinců se stejným datem vzniku nazývá generace.

Evoluční algoritmus začíná vytvořením nulté generace, která je tvořena náhodně vygenerovanými jedinci (jedinci s náhodně vygenerovanými čísly). Poté se provede ohodnocení jednotlivých jedinců pomocí ohodnocovací funkce. Dále jedince v populaci spárujeme, podle pravidel popsaných v sekci 2.2.1. Spárování jedinci se následně zkříží a provede se mutace. Z nově vzniklých jedinců vytvoříme další generace. Tento postup opakujeme po zvolený počet generací.

2.2 Operátory evolučního algoritmu

Při generování následující generace se na všechny jedince stávající generace použijí operátory. V této sekci jsou popsány základní použité operátory.

2.2.1 Selektce

Operátor selektce má na starosti vybrat každému jedinci partnera pro křížení. V práci používáme ruletovou selektci. Tuto selektci si lze představit jako ruletové kolo, na kterém je tolik výsečí, kolik je jedinců v generaci. Každému jedinci přísluší právě jedna výseč. Velikost výseče přímo uměrně závisí na hodnotě fitnes jedince. Spárování pak probíhá pomocí točení rulety. Tedy jedinci s největší výsečí mají největší pravděpodobnost na spárování mezi sebou.

2.2.2 Křížení

Úkolem křížení je vytvoření nových jedinců ze dvou stávajících. Obvykle se vytvoří dva noví jedinci, v zájmu zachování konstatní velikosti populace. Používáme jednobodové, dvoubodové a uniformní křížení. Jednobodové křížení zvolí náhodně jeden bod v jedinci, a v tomto bodě se oba jedinci prohodí. Dvoubodové křížení zvolí takové body dva. A nakonec uniformní křížení u každého bodu zvolí, z jakého jedince se použije hodnota.

2.2.3 Mutace

Mutace je důležitá součást evolučního algoritmu, zabraňuje uvíznutí v lokálním maximu. Je ovšem důležité nastavit správnou hodnotu pravděpodobnosti mutace. Příliš častá mutace vede k nestabilitě výsledků evolučního algoritmu. Na druhou stranu příliš malá pravděpodobnost mutace vede ke ztrátě významu mutace. Mutace, kterou používáme, má dva parametry. První je pravděpodobnost mutace jedince. Pokud je rozhodnuto o mutaci jedince, pak se jedinec celý prochází a pro každou hodnotu se rozhoduje, zda se zmutuje nebo ne. Tuto pravděpodobnost reprezentuje druhý parametr. Mutace hodnoty probíhá jejím nahrazením za náhodnou hodnotu (z určitého intervalu). Zde se nabízí alternativa, že by velikost mutace závisela na hodnotě parametru.

2.3 Ohodnocovací funkce

Při provádění selekce je třeba každého jedince ohodnotit. K tomu se používá ohodnocovací funkce. Na volbě vhodné ohodnocovací funkce závisí úspěšnost celého evolučního algoritmu. Proto v práci používáme tři různé ohodnocovací funkce. V další kapitole pak testujeme úspěšnost jednotlivých typů této funkce. Zde jsou stručně popsány jednotlivé principy pro ohodnocení jedince.

2.3.1 Pevný protihráč

Toto je nejjednodušší a zároveň nejpřímočařejší implementace ohodnocovací funkce. Na vstupu vyžaduje již existující jedince, proti kterým se budou ostatní testovat. Při spuštění ohodnocovací funkce se pak proti každému jedinci v populaci sehraje několik her proti zadaným pevným protihráčům (jedincům). Fitness jedince je pak počet výher proti těmto protihráčům. Otázka je, jak vybrat pevné protihráče. Vhodné by pravděpodobně bylo zvolit tři protihráče, kteří hrají každý jiným způsobem. Já jsem zvolil jedince vyšlechtěné pomocí následujících dvou ohodnocovacích funkcí.

Hlavní výhoda tohoto přístupu je především možnost porovnávat fitness jedinců mezigeneračně a sledovat jejich progres. Na druhou stranu se jedinci šlechtí vůči několika zadaným protihráčům, což nemusí být ideální. Takový jedinec pak může s přehledem vyhrávat proti těmto protihráčům, ale oproti jiným může propadnout.

2.3.2 Všichni proti všem

Ohodnocovací funkce všichni proti všem testuje každého hráče v populaci s každým. Jako fitness se opět použije celkový počet výher daného jedince. Jsou naimplementovány tři varianty tohoto přístupu. U prvního se testují všechny dvojice proti sobě. U druhého všechny trojice, a u třetího všechny čtveřice proti sobě. V rámci zachování spravedlivosti u testování závisí na pořadí. Tedy pokud se testují dvojice, pak každá dvojice jedinců proti sobě sehraje dva zápasy. V jedné hře začíná jeden a ve druhé druhý.

Hlavní výhodou tohoto přístupu je absence pevných protihráčů. Naopak se každý jedinec testuje proti všem ostatním, tedy proti několika různým jedincům.

To zajišťuje určitou variabilitu při hrách. Proto musí být jedinec více univerzální aby uspěl. Musí umět vyhrát proti množství jiných jedinců (a ne pouze proti třem jako v předchozím případě). Tento přístup má ale také své nevýhody. Hlavní je velký počet her, které je třeba sehrát. Například pokud máme populaci velikosti 20 je třeba při testování všech dvojic provést 380 her. Při testování všech trojic 6840 her, a při testování všech čveřic dokonce 116280 her. Druhá nevýhoda je nemožnost porovnávání fitness jedinců mezigeneračně. Je to proto, že v každé generaci se jedinci testují proti jiné populaci, než proti které se testovali v generaci přechozí.

2.3.3 Učící se pevný protihráč

Tento přístup je kompromis mezi dvěma předchozími přístupy. V první generaci sehrají všichni jedinci zápas proti prvnímu (nebo prvním dvěma nebo třema, podle toho, kolik je nastaveno hráčů ve hře) jedinci v populaci. Tím se provede první ohodnocení a vyberou se tři (dva nebo jeden) nejlepší jedinci, proti kterým se bude několik následujících generací testovat. Po uplynutí několika následujících generací se provede aktualizace těchto protihráčů. Opět se vyberou tři (dva nebo jeden) noví nejlepší protihráči v generaci. Takto se dále postupuje, dokud není dosažen požadovaný počet generací.

V tomto přístupu je možné srovnání vždy určitý úsek při testování proti jedné sadě protivníků. Navíc zde nejsou žádné vstupní protihráči.

Využívá se zde výhody prvního přístupu pro možnost srovnání, ale zároveň se eliminují pevní protihráči.

2.4 Parametry evolučního algoritmu

Implementace evolučního algoritmu má na vstupu několik parametrů:

- velikost populace - počet jedinců v první generaci. V každé následující je tento počet stejný.
- počet generací.
- typ křížení - lze vybrat mezi jednobodovým, dvoubodovým a uniformním křížením.
- pravděpodobnost křížení - pravděpodobnost, s jakou se dva spárování jedinci skříží. V opačném případě postupují do další generace beze změny.
- pravděpodobnost mutace - zde jsou uvedeny dvě hodnoty. První určuje pravděpodobnost mutace jedince a druhá pravděpodobnost mutace jednotlivých parametrů jedince, v případě, že byl jedinec vybrán pro mutaci.
- ohodnocovací funkce - zde je na výběr ze tří funkcí popsaných v sekci 2.3. Funkce s názvem pevný protihráč pak má ještě atribut počet hráčů a seznam pevných protihráčů. Také má uvedeno, kolik her se má sehrát pro ohodnocení jednoho jedince. U funkce všichni proti všem lze nastavit počet hráčů v ohodnocovacích hrách. A konečně u funkce učící se pevný protihráč lze nastavit počet her potřebných pro ohodnocení jednoho jedince, počet

hráčů v ohodnovacích hrách a počet generací, po kterých se protivníci vymění za nové nejlepší jedince.

3. Výsledky evoluce

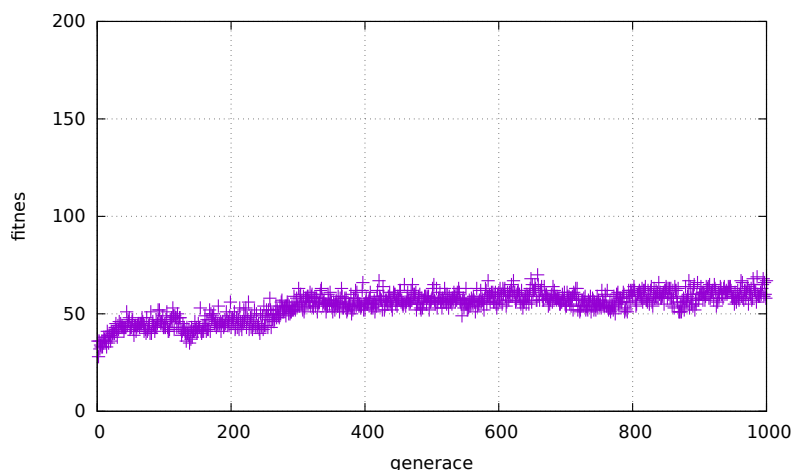
V této kapitole jsou uvedeny výsledky evoluce. V sekci 3.1 budeme diskutovat nastavení jednotlivých parametrů evoluce, v sekci 3.2 popíšeme systém, který použijeme k vyšlechtění nejlepšího jedince a v sekci 3.3 pak uvedeme reprezentaci naměřených dat.

3.1 Nastavení parametrů

Pokud bychom chtěli vyzkoušet všechny možné nastavení parametrů evoluce, tak bychom museli sehrát obrovské množství her. Musíme se proto rozhodnout, jaké parametry ponecháme stejné pro všechny evoluční algoritmy, a jaké budeme měnit.

3.1.1 Počet generací

Prvním parametrem evoluce je počet generací. Společným znakem evolučních algoritmů je, že v prvních generacích je vidět velké zlepšení jedinců. Toto zlepšení se postupem času zmenšuje. Je to způsobené tím, že v generaci je konstantní počet jedinců a do dalších generací jsou vybíráni ke křížení ti nejlepší. Tedy se postupem času dostane evoluce do lokálního maxima. Jediná šance na vývoj je pak mutace. Ovšem vývoj evoluce za pomoci pouze mutace je velmi pomalý. Spustili jsme tedy evoluci s 1000 generacemi. Vývoj fitness nejlepších jedinců v každé generaci je znázorněn na obrázku 3.1. Na obrázku je patrné, že od 300 generace se fitness zlepšuje jen minimálně a převážně stagnuje. Z toho důvodu v následující části spouštíme evoluční algoritmy pouze s 300 generacemi.



Obrázek 3.1: Evoluce s ohodnocovací funkcí pevný protihráč

3.1.2 Velikost populace a počet her při ohodnocení

Na velikosti populace závisí doba běhu evolučního algoritmu. Při určování velikosti populace budeme vycházet z toho, že chceme, aby jeden evoluční algoritmus doběhl do 24 hodin. Na počítači, kde spouštíme evoluci, trvá simulátoru

odehrát 20000 her zhruba za tři minuty. Tedy se odehraje přibližně jedna hra za 9 milisekund. Pokud nastavíme velikost populace na 40, a počet her na ohodnocení jednoho jedince nastavíme na 400, tak se každou generaci sehraje $40 \cdot 400 = 16000$ her. Tedy evoluce bude trvat přibližně $300 \cdot 2.4 = 720$ minut. V rámci srovnatelnosti podmínek nastavíme u ohodnocovacích funkcí pevný protihráč a učící se pevný protihráč velikost populace na 40 a počet her na ohodnocení na 400. Při použití ohodnocovací funkce všichni proti všem již nemůžeme určovat počet her pro ohodnocení jedince. Tento počet je určen velikostí populace. Velikost populace zvolíme takovou, aby evoluce doběhla do 24 hodin. Při velikosti populace 16 se musí sehrát každou generaci celkem 43680 her. To by již trvalo více jak 24 hodin. Proto zvolíme velikost populace 15, kde se celkem sehraje v jedné generaci $15 \cdot 14 \cdot 13 \cdot 12 = 32760$ her.

3.1.3 Mutace a křížení

Pravděpodobnost mutace ovlivňuje evoluční algoritmus. Příliš velká mutace způsobuje nestabilitu evolučního algoritmu a příliš malá mutace ztrácí na významu. Proto budeme tento parametr evoluce v jednotlivých evolučních algoritmech měnit a budeme sledovat výsledky jednotlivých nastavení. Mutaci jedince nastavíme v intervalu od 0.08 do 0.16 a mutaci jednoho bitu nastavíme v intervalu od 0.12 do 0.24. Hodnota 0.24 je již hraniční případ, protože se změní téměř čtvrtina celého jedince.

Pravděpodobnost křížení by měla být velká, aby se většina jedinců zkřížila, ale zase ne stoprocentní, aby byla možnost, že některý jedinec postoupí do další generace nezměněný. Provedli jsme devět evolucí s pravděpodobností křížení 0.9, devět s pravděpodobností křížení 0.8 a devět evolucí s pravděpodobností křížení 0.7. Ostatní nastavení evolučních algoritmů bylo stejné. S nejlepšími jedinci z poslední generace těchto evolučních algoritmů jsme sehráli turnaj, jehož výsledky jsou uvedeny v tabulce 7.1 v příloze A. Jedinci vyšlechtění s pravděpodobností mutace 0.7 dosáhli průměrně skóre $\frac{233348}{9} \doteq 25927$, jedinci vyšlechtění s pravděpodobností křížení 0.8 dosáhli průměrně $\frac{227756}{9} \doteq 25306$ bodů a jedinci vyšlechtění s pravděpodobností křížení 0.9 dosáhli průměrně skóre $\frac{240896}{9} \doteq 26766$. V průměru tedy dosahovali jedinci vyšlechtění s pravděpodobností křížení 0.9 lepších výsledků než ostatní jedinci. Proto jsme pravděpodobnost křížení ponechali pro všechny evoluční algoritmy stejnou, a to sice na hodnotě 0.9. Měnili jsme ale způsob křížení. V projektu používáme jednobodové, dvoubodové a uniformní křížení.

3.1.4 Počet hráčů

V následujících evolučních algoritmech používáme vždy čtyři hráče ve hře. Jde však o obecný postup a lze aplikovat i na jiný počet hráčů. To však již není předmětem bakalářské práce.

3.2 Způsob šlechtění

Používáme tři ohodnocovací funkce, tři druhy křížení a budeme používat tři různé nastavení mutace. Celkem tedy spustíme $3 \cdot 3 \cdot 3 = 27$ evolučních algoritmů.

Poté vybereme nejlepšího jedince z poslední generace každého spuštěného evolučního algoritmu a sehraje mezi nimi turnaj. Vítěze z tohoto turnaje nastavíme jako defaultního protihráče v implementované hře osadníci z katanu. V následujících tabulkách jsou uvedeny přesné nastavení jednotlivých evolučních algoritmů. Mutace jedince je pravděpodobnost, že bude jedinec vybrán pro mutování. Mutace bitu je pak pravděpodobnost mutace jednotlivých parametrů v případě, že byl jedinec vybrán pro mutování.

V tabulce 3.1 jsou uvedeny nastavení evolučních algoritmů s ohodnocovací funkcí pevný protihráč, velikost populace je 40, počet generací je 300, pravděpodobnost křížení je 0.9, počet her pro ohodnocení jednoho jedince je 400 a v každé hře hrají čtyři hráči.

| název | mutace jedince | mutace bitu | křížení |
|--------------|-----------------------|--------------------|----------------|
| Basic 1 | 0.08 | 0.12 | jednobodové |
| Basic 2 | 0.12 | 0.18 | jednobodové |
| Basic 3 | 0.16 | 0.24 | jednobodové |
| Basic 4 | 0.08 | 0.12 | dvoubodové |
| Basic 5 | 0.12 | 0.18 | dvoubodové |
| Basic 6 | 0.16 | 0.24 | dvoubodové |
| Basic 7 | 0.08 | 0.12 | uniformní |
| Basic 8 | 0.12 | 0.18 | uniformní |
| Basic 9 | 0.16 | 0.24 | uniformní |

Tabulka 3.1: pevný protihráč - nastavení evolučních algoritmů

V tabulce 3.2 jsou pak uvedeny nastavení evolučních algoritmů s ohodnocovací funkcí všichni proti všem. Velikost populace je 15, počet generací je 300, pravděpodobnost křížení je 0.9 a v každé hře opět hrají čtyři hráči.

| název | mutace jedince | mutace bitu | křížení |
|--------------|-----------------------|--------------------|----------------|
| EbdFourPl 1 | 0.08 | 0.12 | jednobodové |
| EbdFourPl 2 | 0.12 | 0.18 | jednobodové |
| EbdFourPl 3 | 0.16 | 0.24 | jednobodové |
| EbdFourPl 4 | 0.08 | 0.12 | dvoubodové |
| EbdFourPl 5 | 0.12 | 0.18 | dvoubodové |
| EbdFourPl 6 | 0.16 | 0.24 | dvoubodové |
| EbdFourPl 7 | 0.08 | 0.12 | uniformní |
| EbdFourPl 8 | 0.12 | 0.18 | uniformní |
| EbdFourPl 9 | 0.16 | 0.24 | uniformní |

Tabulka 3.2: všichni proti všem - nastavení evolučních algoritmů

A konečně v tabulce 3.3 jsou uvedeny nastavení evolučních algoritmů s ohodnocovací funkcí učící se pevný protihráč, velikost populace je 40, počet generací je 300, pravděpodobnost křížení je 0.9, počet her pro ohodnocení jednoho jedince je 400 a v každé hře hrají čtyři hráči.

Nejdříve jsme spustili evoluční algoritmy s ohodnocovací funkcí učící se pevný protihráč a všichni proti všem. Nejlepší strategie vyšlechtěné těmito evolučními algoritmy sehrály proti sobě turnaj a nejlepší tři jedinci byli pevní protihráči

| název | mutace jedince | mutace bitu | křížení |
|-------------------|----------------|-------------|-------------|
| Changing Rivals 1 | 0.08 | 0.12 | jednobodové |
| Changing Rivals 2 | 0.12 | 0.18 | jednobodové |
| Changing Rivals 3 | 0.16 | 0.24 | jednobodové |
| Changing Rivals 4 | 0.08 | 0.12 | dvoubodové |
| Changing Rivals 5 | 0.12 | 0.18 | dvoubodové |
| Changing Rivals 6 | 0.16 | 0.24 | dvoubodové |
| Changing Rivals 7 | 0.08 | 0.12 | uniformní |
| Changing Rivals 8 | 0.12 | 0.18 | uniformní |
| Changing Rivals 9 | 0.16 | 0.24 | uniformní |

Tabulka 3.3: učící se pevný protihráč - nastavení evolučních algoritmů

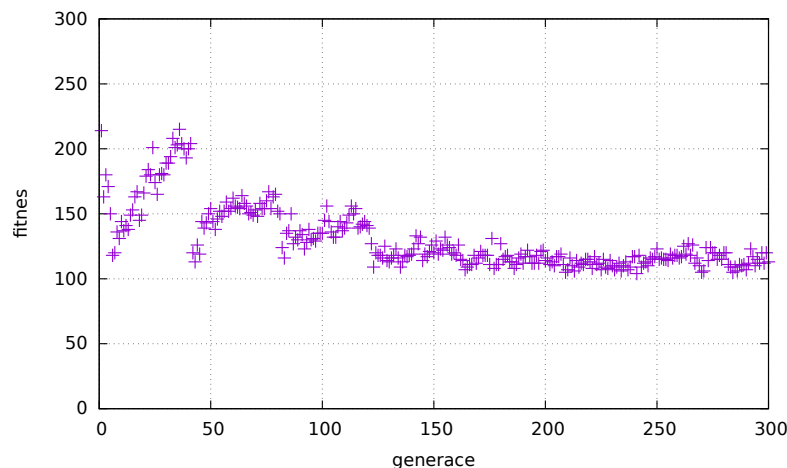
u evolučních algoritmů s ohodnocovací funkcí pevný protihráč. Po spuštění všech těchto uvedených evolučních algoritmů jsme vzali nejlepší jedince z poslední generace a sehráli jsme mezi nimi turnaj. Každá dvojice jedinců proti sobě hrála 2000 her. Jeden a ten samý jedinec proti sobě nehrál. Přičemž 1000 her byl první na řadě jeden jedinec a dalších 1000 her byl první na řadě druhý jedinec. Výsledné skóre jednotlivých jedinců je uvedeno v tabulce 3.4.

3.3 Interpretace výsledků

Grafické znázornění výsledků je uvedeno v příloze A na grafech 7.1 až 7.27. Každý graf na ose x znázorňuje, kolikátá generace probíhá, a na ose y znázorňuje fitness nejlepšího jedince z této generace. Fitness je počítáno jako celkový počet výher při ohodnocování. Tedy pro každý typ ohodnocovací funkce má fitness jiný význam. Při použití ohodnocovací funkce pevný protihráč (na grafech 7.1 až 7.9) je fitness přímo ukazatel kvality jedince. Jde o počet výher při sehrání 400 her proti pevným soupeřům. Tito soupeři jsou po celou evoluci stejní, a tedy je zde možné mezigenerační srovnání. Při použití ohodnocovací funkce učící se pevný protihráč (na grafech 7.19 až 7.27) je toto mezigenerační srovnání možné vždy jen po určitý počet generací (v tomto případě 40), protože potom se soupeři změni. Tento jev je dobře vidět například na obrázku 3.2. Zde je vidět, že v generaci jedna až čtyřicet se jedinci výrazně zlepšují. To je dáno tím, že na začátku jsou pevní soupeři velmi hloupí. V generaci 41 se soupeři vymění za dosud nejlepší vyšlechtěné jedince a je zde tedy vidět propad fitness jedinců. To je dáno tím, že se soupeři, proti kterým se fitness testuje, zlepšili. Tuto situaci pak můžeme sledovat každých čtyřicet generací, ovšem v menším a menším měřítku. A konečně při použití ohodnocovací funkce všichni proti všem (na grafech 7.10 až 7.18) je fitness jedince v rozmezí nula až 4000. Je to dáno tím, že každý hráč hraje celkem $\frac{15 \cdot 14 \cdot 13 \cdot 12}{4} = 8190$ her. Mezigenerační srovnání zde ale neurčuje míru zlepšení jedinců, protože se v každé generaci testuje proti jiným jedincům.

3.3.1 Vliv mutace a křížení

Celkový výsledek evoluce závisí na mnoha faktorech, které neovlivníme. Velký vliv má například první generace, která je generovaná náhodně. V této sekci se ale pokusíme zhodnotit jednotlivé nastavení evoluce na základě grafů z výsledků.



Obrázek 3.2: Changing rivals 2

Například lze na grafech 7.3 a 7.6 pozorovat, že mutace jedince 0.16 a mutace bitu 0.24 je již příliš velká. U vývoje fitness lze pozorovat několik poklesů, které mohou být způsobené právě velkou mutací. Naproti tomu na grafu 7.1 je vidět stabilní (i když ne moc velký) progres po celou dobu evoluce. Dále srovnáme vývoj fitness u evoluce s rozdílným typem křížení, ale stejnou pravděpodobností mutace. Na grafech 7.4 a 7.7 je výrazný progres na začátku evoluce, ale pak vývoj jedinců víceméně stagnuje. Naproti tomu na grafu 7.1 takto strmý progres není patrný a i přes to mají v poslední generaci jedinci srovnatelné fitness ohodnocení. Z toho lze usuzovat, že jednobodové křížení způsobuje pravidelné drobné zlepšení jedinců, zatímco dvoubodové a uniformní křížení způsobuje na začátku výrazné zlepšení jedince, které pak již není jak zlepšovat (našli jsme lokální maximum) a tedy zbytek evoluce jedinci stagnují.

3.3.2 Vyhodnocení závěrečného turnaje

Jednotlivé přístupy uvedené v tabulce 3.4 nejdříve srovnáme podle použité ohodnocovací funkce. Na první pohled je vidět, že největší úspěchy má ohodnocovací funkce pevný protihráč a učící se pevný protihráč. Průměrné skóre jedince vyšlechtěného s ohodnocovací funkcí pevný protihráč je $\frac{243847}{9} \doteq 27094$, s ohodnocovací funkcí učící se pevný protihráč to je $\frac{238524}{9} \doteq 26503$. A konečně s ohodnocovací funkcí všichni proti všem je průměrné skóre jedince $\frac{219629}{9} = 24403$. Ohodnocovací funkce pevný protihráč a učící se pevný protihráč jsou podobné. Vždy se testuje proti pevné sadě soupeřů, přičemž v jednom případě se po nějaké době tito soupeři mění. Není tedy překvapující, že průměrně dosáhli jedinci vyšlechtění těmito způsoby podobné skóre. Je ovšem vidět výrazně nižší průměrné skóre u ohodnocovací funkce všichni proti všem. To může být způsobeno malým počtem jedinců v generaci. Při evoluci s tímto typem ohodnocovací funkce jsme nastavili velikost populace v generaci na 15. Větší populaci v generaci nebylo možné nastavit, protože by evoluční algoritmus běžel příliš dlouho.

Nyní zhodnotíme nastavení evolučních algoritmů prvních šesti nejlepších jedinců. V této šestici je čtyřikrát nastavena největší mutace jak bitu, tak jedince a dvakrát nastavena nejmenší mutace jak bitu tak jedince. Dále je v první šestici třikrát zastoupena ohodnocovací funkce pevný protihráč, dvakrát učící se

pevný protihráč a jednou ohodnocovací funkce všichni proti všem. Co se týče typu křížení, tak je v první šestici jednou zastoupeno jednobodové křížení, dvakrát zastoupeno dvoubodové křížení a třikrát zastoupeno uniformní křížení. Je zde tedy vidět zastoupení širokého spektra možných nastavení evoluce. Z toho lze usuzovat, že evoluce, kromě jejího počátečního nastavení, závisí také na náhodě. Náhoda se uplatňuje především při generování první generace a při křížení a mutaci. K získání relevantnějších výsledků by bylo třeba každý evoluční algoritmus spustit například desetkrát a započítat průměrné skóre nejlepších jedinců.

| Název | počet výher |
|-------------------|-------------|
| Basic 4 | 31161 |
| Basic 9 | 31022 |
| Changing Rivals 6 | 29996 |
| Changing Rivals 9 | 29985 |
| Basic 1 | 28858 |
| EbdFourPl 9 | 27230 |
| EbdFourPl 1 | 27159 |
| Basic 5 | 27150 |
| Basic 7 | 27088 |
| Changing Rivals 8 | 26918 |
| EbdFourPl 5 | 26918 |
| EbdFourPl 3 | 26495 |
| Basic 3 | 26394 |
| Changing Rivals 4 | 26279 |
| Changing Rivals 7 | 26145 |
| Basic 2 | 26175 |
| Changing Rivals 1 | 26019 |
| Changing Rivals 5 | 25428 |
| EbdFourPl 8 | 25002 |
| EbdFourPl 2 | 24549 |
| Changing Rivals 2 | 24476 |
| EbdFourPl 4 | 24438 |
| Basic 6 | 23312 |
| Changing Rivals 3 | 23278 |
| Basic 8 | 22687 |
| EbdFourPl 6 | 19368 |
| EbdFourPl 7 | 18470 |

Tabulka 3.4: Výsledky šlechtění

3.3.3 Styl hry strategie

V této sekci zhodnotíme, jakými pravidly se strategie řídí, na základě konkrétních vah jedince. Tedy se pokusíme zjistit, jak by měl člověk hrát, aby hrál podobně jako vyšlechtěné strategie. Srovnáme prvních šest jedinců, kteří vyhráli turnaj. Konkrétné váhy těchto jedinců lze najít v příloženém DVD. Všech šest

jedinců klade při prvních dvou tazích hry největší důraz na získání kamene. Dále čtyři z nich považují za druhou nejdůležitější surovinu obilí. Z toho lze usuzovat, že se tyto strategie zaměřují na stavbu měst.

Dále zhodnotíme, zda strategie spíše postaví cestu, nebo koupí akční kartu (v případě, že má na obě akce suroviny). Koupí akční karty určuje jeden parametr z jedince a postavení cesty součet čtyř parametrů. U prvních dvou jedinců má parametr na koupí akční karty větší hodnotu, než součet čtyř parametrů na postavení cesty. Tedy první dva jedinci preferují koupí akční karty před postavením cesty. Oproti tomu další čtyři jedinci mají větší součet parametrů na postavení cesty, než parametr na koupí akční karty. Tedy tyto čtyři jedinci preferují postavení cesty před koupí akční karty.

Pokud bychom chtěli zhodnotit, zda strategie spíše postaví vesnici než cestu, tak už to nebude takto jednoduché, protože na postavení vesnice používá strategie složitější vzorec. Například ale u prvního, druhého, čtvrtého a šestého jedince vždy strategie preferuje postavení vesnice před postavením cesty. Je to dáno tím, že už hodnota parametru *villageGen* je větší jak součet čtyř parametrů pro postavení cesty. Třetí strategie v pořadí oproti tomu preferuje postavení vesnice před postavením cesty jenom v případě, pokud vesnice sousedí s výhodným přístavem. A konečně čtvrtá strategie preferuje postavení vesnice jenom pokud bude vesnice sousedit s třemi výhodnými stěnami, nebo bude postavena u výhodného přístavu.

Takovýchto pravidel lze z konkrétních vah jedinců vypočítat více. Například za jakých podmínek strategie spíše postaví město, než vesnici. Zde však již neexistuje takto jednoduchá odpověď, protože nelze obecně říci, co strategie preferuje. V tomto případě záleží na konkrétním umístění vesnice a města.

4. Implementace hry

4.1 Technologie

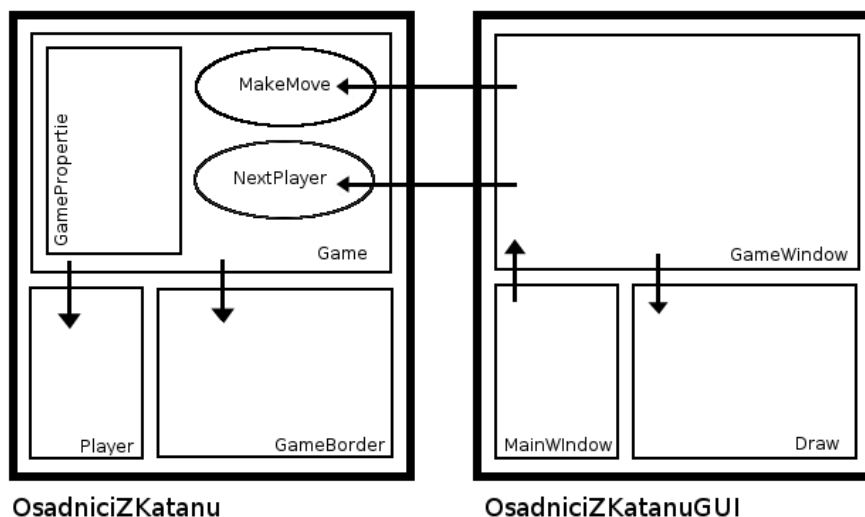
K implementaci hry Osadníci z katanu byl použit jazyk c# a jako cílovou platformu jsme zvolili systém Windows. Pro vývoj aplikace bylo použito vývojové prostředí Visual Studio 2012 a pro tvorbu grafické části aplikace byly zvoleny technologie WPF (Windows Presentation Foundation). Při programování byla použita dokumentace jazyka c# MSDN [3].

4.2 Návrh

Celá aplikace hry Osadníci z katanu je implementovaná ve dvou projektech. První má název `osadniciZKatanuGUI`. V tomto projektu je implementovaná grafická stránka hry a obsluha událostí vytvořených hráčem. Druhý projekt má název `osadniciZKatanu` a je koncipovaný jako knihovna. Tuto knihovnu pak využívá projekt `osadniciZKatanuGUI`. Tento koncept jsme zvolili, aby bylo možné sehrávat hry mezi strategiemi bez nutnosti kompilace projektu `osadniciZKatanuGUI`.

V projektu `osadniciZKatanu` je implementováno jádro hry. Hlavní třída tohoto projektu je třída `Game`. Tato třída obsahuje seznam hráčů ve hře a popis hracího plánu. Dále pak obsahuje metody na ovládání hry. Hráč je reprezentován třídou `Player`. Popis hracího plánu je uložen v třídě `GameBorder`. Projekt `osadniciZKatanuGUI` pak obsahuje instanci třídy `Game`. Na této třídě volá metodu `MakeMove` v případě, že hráč vytvořil takové události, které určují tah.

Návrh celé aplikace je zobrazen na obrázku 4.1, kde jsou znázorněné oba dva projekty. Obdélníky představují třídy, elipsy znázorňují metody a šipky reprezentují volání metod.



Obrázek 4.1: diagram návrhu aplikace

4.3 Jádro hry

Jádro hry je implementované v knihovně `osadniciZKatanu`. Hlavní komponenty knihovny jsou tyto:

- třída `Game` - zajišťuje logiku hry.
- třída `GameProperties` - obsahuje informace o hře.
- třída `Player` - zajišťuje logiku hráče a uchovává informace o stavu hráče.
- třída `PlayerProperties` - obsahuje informace o hráči.
- třída `GameBorder` - obsahuje popis hrací plochy.
- třída `Move` - třída, od které dědí třídy popisující konkrétní tah - pro každý druh tahu je třeba uchovat rozdílný typ informací.

4.3.1 Třída `Game`

Třída `Game` reprezentuje konkrétní instanci hry. Třída `GameWindow` v projektu `osadniciZKatanuGUI` obsahuje instanci třídy `Game`, na které volá metody ovládající hru a dle informací z této třídy zobrazuje jednotlivé komponenty grafického okna hry. Hlavní metody na ovládání hry jsou následující:

- metoda `MakeMove` - jako parametr požaduje popis tahu. Tah je reprezentovaný třídou `Move`. Třída rozpozná, o jaký typ tahu se jedná, a pokud je tah korektní, tak ho provede. Na konci zkontroluje, jestli počet bodů hrajícího hráče nestačí na vítězství. Pokud ano, nastaví stav hry na konec hry.
- metoda `NextState` - hra se může nacházet celkem ve čtyřech stavech. Jsou to začátek, první dva tahy hry, hra a konec hry. Tato metoda nastaví stav hry na následující v pořadí.
- metoda `NextPlayer` - nastaví aktuálního hráče na následujícího v pořadí.
- metoda `GetMaterials` - má dvě implementace. První z nich nemá žádný parametr. Tato implementace provede produkci suroviny každé stěny na hrací desce. Druhá implementace má jeden parametr, celé číslo z intervalu dva až dvanáct, a provede produkci surovin stěn s tímto pravděpodobnostním číslem.
- metoda `RollTheDice` - simuluje hod dvěma hracími kostkami.

Třída `Game` obsahuje seznam hráčů ve hře a instanci třídy `GameProperties`. Každý hráč je reprezentován instancí třídy `Player`. Dále pak třída definuje stav hry. Stav hry představuje enum, který má hodnoty `start`, `firstPhaseOfGame`, `game` a `endGame`

4.3.2 Třída `GameProperties`

Třída `GameProperties` reprezentuje stav hry a výchozí nastavení hry. Tedy informace o tom, kolik surovin produkuje vesnice, kolik město, kolik a jakých karet je v balíčku akčních karet, nebo jaký je minimální počet bodů na vítězství. Dále třída obsahuje informace o hracím plánu. Tyto informace jsou uloženy v instanci třídy `GameBorder`. Jde o údaje na jaké stěně je jaká surovina, kde je přístav, nebo kde je výchozí pozice zloděje. Třída `GameProperties` tyto informace načte z xml souborů. Tento systém zajišťuje pohodlné změny v nastavení hry, tedy například pokud bychom se rozhodli změnit minimální počet bodů pro vítězství nebo počet akčních karet.

4.3.3 Třída `Player`

Třída `Player` reprezentuje hráče. Nezáleží, jestli jde o hráče fyzického nebo o hráče řízeného strategií. Třída `Game` obsahuje instance této třídy reprezentující jednotlivé hráče ve hře. Na těchto hráčích pak volá metody zajišťující logiku hráče. Mezi hlavní metody této třídy patří:

- metoda `AddPort` - třída uchovává informace o tom, jaké má hráč k dispozici přístavy. Pokud hráč postaví vesnici na přístavu, tak třída `Game` zavolá tuto metodu.
- metoda `AddRoad` - zajišťuje přidání cesty do seznamu cest hráče. Toto přidání obsahuje testování, zda postavením této cesty nevznikla nová nejdelší cesta.
- metoda `AddVillage` - přidá novou vesnici do seznamu vesnic hráče.
- metoda `AddTown` - přidá nové město do seznamu měst hráče.

Tato třída také obsahuje instanci třídy `PlayerProperties`, která obsahuje základní informace o hráči.

4.3.4 Třída `PlayerProperties`

Třída `PlayerProperties` reprezentuje stav hráče. Ve třídě jsou uloženy informace, kolik má daný hráč bodů, kolik má rytířů, jestli má největší armádu, nebo jestli má právě nejdelší cestu ve hře. Dále obsahuje barvu hráče, seznam vesnic a měst, seznam cest, seznam karet se surovinami a seznam akčních karet daného hráče.

4.3.5 Třída `Move`

Třída `Move` popisuje tah. Ve hře je několik typů tahů a každý z nich potřebuje pro popis jiný typ informací. Proto od této třídy dědí další třídy popisující konkrétní tah hráče. Tah reprezentujeme jako dvojici. První část je výměna surovin a druhá je nějaký konkrétní tah. Část na výměnu surovin je pro každý tah stejná a tedy je tato část implementovaná přímo ve třídě `Move`. Tah výměny surovin je reprezentovaný jako seznam dvojic surovin. První surovina ve dvojici určuje, co chceme měnit, a druhá surovina ve dvojici, za co chceme surovinu vyměnit. Kurz

výměny je daný jednoznačně podle přístavů hráče, takže ho není třeba uvádět v popisu tahu. Ve hře jsou implementované tyto typy tahů. Každá z uvedených tříd dědí od třídy `Move`.

- třída `BuildRoadMove` - obsahuje informace o tom, kde se má cesta postavit. Tato informace je uložena v instanci třídy `Edge`.
- třídy `BuildVillageMove` a `BuildTownMove` - určují, na jakém vrcholu se má postavit vesnice nebo město. Vrchol je určen třídou `Vertex`.
- třídy `BuyActionCardMove` a `CouponMove` - Nepotřebují žádné další informace o tahu, koupě akční karty je jednoznačná. Použití akční karty jeden bod je také jednoznačné a nepotřebuje žádné další informace o tahu.
- třída `FirstPhaseGameMove` - určuje první dva tahy hry. Jde o postavení vesnice a cesty. Tento tah je speciální a použije se jenom na začátku hry. V rámci konzistence jsem se rozhodl tento tah reprezentovat jako jeden, i když jde v podstatě o tahy dva.
- třídy `KnightMove` a `ThiefMove` - jsou si podobné, protože obě definují přemístění zloděje. V případě třídy `KnightMove` se ještě navíc přičte jeden rytíř hrajícímu hráči. Stěna, kam se má rytíř přemístit, je reprezentovaná třídou `Face`. Dále je hráč, který má být okraden, reprezentovaný barvou hráče.
- třída `MaterialsFromPlayersMove` - obsahuje jeden typ suroviny, který chce hrající hráč získat od soupeřů.
- třída `TwoMaterialsMove` - obsahuje dva typy surovin, které chce hrající hráč získat.
- třída `TwoRoadMove` - definuje dvě cesty, které chce hráč postavit. Cesty jsou opět určeny třídou `Edge`.
- třída `NothingToDoMove` - Tento tah značí, že hrající hráč již nemůže udělat žádný tah.

4.3.6 Třída `GameBorder`

Třída `GameBorder` obsahuje informace o hrací desce. Třída využívá tyto datové struktury:

- třída `Edge` - reprezentuje hranu hracího plánu. Obsahuje informace o souřadnicích konců této hrany na hrací desce, informaci, zda zde leží cesta a jakého hráče a ID hrany. Dále třída obsahuje reference na sousední hrany a sousední vrcholy.
- třída `Vertex` - definuje vrchol hracího plánu. Obsahuje informace o souřadnicích vrcholu, zda na vrcholu leží nějaká stavba, popřípadě jaká a kterého hráče a zda zde leží přístav. Mimo to třída obsahuje reference na sousední vrcholy, hrany a stěny.

- třída **Face** - představuje stěnu hracího plánu. Obsahuje informace o souřadnicích stěny, surovině, která na stěně leží, pravděpodobnostní číslo na stěně a zda zde leží zloděj. Opět třída obsahuje reference na sousední vrcholy.
- třída **Coord** - reprezentuje souřadnice x a y na hracím plánu.

Třída **GameBorder** pak obsahuje seznam vrcholů, hran a stěn hracího plánu. Dále tato třída obsahuje metody, které slouží k nalezení vrcholu, hrany nebo stěny, podle zadané souřadnice. Ty využívá grafická verze hry, když hráč chce postavit budovu nebo cestu na nějaké souřadnici.

4.4 Grafické rozhraní

Grafické rozhraní hry je implementováno v projektu **osadniciZKatanuGUI**. Hlavní třídy tohoto projektu jsou následující:

- třída **MainWindow** - představuje okno s nastavením nové hry. Toto okno se spustí při spuštění programu **osadniciZKatanuGUI.exe**.
- třída **GameWindow** - reprezentuje okno hry.
- třída **Draw** - se stará o vykreslení konkrétního stavu hry. Tedy například které prvky okna mají být aktivní, a které ne.

4.4.1 Třída MainWindow

Při spuštění aplikace se spustí třída **MainWindow**. Ta implementuje okno s nastavením hry. Okno je implementováno pomocí technologie WPF, tedy grafická stránka hry je popsána v souboru **MainWindow.xaml** a obsluha událostí je implementovaná v souboru **MainWindow.xaml.cs**. Po stisknutí tlačítka Nová hra třída vytvoří instanci třídy **GameWindow**, které předá v parametrech nastavení hry. Jde o počet hráčů, jazyk hry (implementovaná je angličtina a čeština), zda se má použít výchozí hrací deska, nebo vygenerovat náhodnou, a za které hráče bude hrát strategie a za které fyzický hráč.

4.4.2 Třída GameWindow

Třída **GameWindow** implementuje hlavní okno hry. Okno je implementováno pomocí technologie WPF, a tedy je grafická stránka hry implementována v jiném souboru, než metody na obsluhu událostí okna. V konstruktoru této třídy se vytvoří instance třídy **Game**. Dále se vytvoří instance třídy **MyGameLogic**, pokud hra obsahuje hráče, za které hraje strategie. Třída **MyGameLogic** implementuje interface **IGameLogic**. Dále se vytvoří instance třídy **Draw**.

Třída **GameWindow** definuje enum, který reprezentuje stav hry. Tento stav určuje, které komponenty okna mají být aktivní/neaktivní a co znamená klik na hrací plochu. Ve hře jsou následující stavy hry:

- stav **firstPhaseMove** znamená, že probíhají první dva tahy hry.

- na přemístění zloděje máme dva stavy hry. První je `moveThief`, který znamená, že hráč má vybrat stěnu, kam umístí zloděje. Druhý má název `moveThiefPartTwo` a znamená, že hrající hráč má vybrat hráče, kterého okrade. Pokud je pouze jeden možný hráč, tak se tento stav přeskočí. Stav na použití akční karty rytíř jsou podobné a mají název `knightMove` a `knightMovePartTwo`. Mají podobný význam jako stavy na přemístění zloděje.
- na použití akční karty „dvě cesty“ máme stavy `twoFreeRoadsFirstRoad` a `twoFreeRoadsSecondRoad`. První znamená, že hráč má postavit první cestu a druhý značí postavení druhé cesty.
- stavy na použití akční karty „dvě suroviny zadarmo“ jsou `twoMaterials` a `twoMaterialsPartTwo`. První značí vybrání první suroviny a druhý vybrání druhé suroviny zadarmo.
- událost použití akční karty „surovina od hráčů“ má jeden stav s názvem `materialsFromPlayers`, který znamená, že hráč má vybrat surovinu, kterou chce získat.
- na výměnu surovin jsou implementovány dva stavy hry. První má název `changingMaterials` a druhý `changingMaterialsPartTwo`. První znamená, že hráč má vybrat, kterou surovinu chce vyměnit, a druhý kterou surovinu chce získat.

Tah hráče probíhá následovně. V případě, že je na tahu fyzický hráč, tak třída `GameWindow` zachytává akce hráče (kliknutí na hrací plochu, stisknutí tlačítka) a v případě, že vykonané akce tvoří kompletní tah (některý tah se skládá z více událostí), tak třída zavolá metodu `MakeMove` třídy `Game`. Pokud je tento tah korektní, tak instance třídy `Game` tah provede. Pokud hráč klikne na tlačítko další hráč, tak se zavolá metoda `NextPlayer` třídy `Game`. V případě, že je na tahu strategie, tak se zavolá metoda `GenerateMove`, kterou definuje interface `IGameLogic`. Tato metoda vrátí popis tahu. Dále se zavolá metoda `MakeMove`, v jehož parametru bude tento popis tahu.

Při každé události na hrací ploše, která změní stav hry, se zavolá metoda `DrawStatue` třídy `Draw`.

4.4.3 Třída `Draw`

Projekt `osadniciZKatanuGUI` také implementuje třídu s názvem `Draw`, která se stará o vykreslení stavu hry. Hlavní metoda této třídy je `DrawStatue`, která jako parametr dostane referenci na instanci třídy `Game` a aktuální stav hry. Podle toho, v jakém stavu je hra, vykreslí příslušné komponenty (aktivuje a deaktivuje jednotlivé komponenty okna podle toho, jestli je nebo není možné je v tomto stavu hry použít). Dále třída obsahuje funkce na vykreslení města, vesnice, cesty nebo zloděje na hrací desku. Třída se také stará o prvotní vykreslení hrací plochy.

5. Implementace AI

Umělá inteligence v řešení se skládá ze tří částí. První část je evoluce, která se stará přímo o šlechtění jedinců, a druhá je simulátor, který má na starosti sehrání zápasů mezi jedinci. Evoluce pak využívá simulátor k ohodnocení jedinců. Třetí část se pak stará přímo o ohodnocení jednotlivých tahů a vygenerování nejlepšího tahu.

5.1 Simulátor

Simulátor hry osadníci z katanu je implementovaný v projektu `simulator`. Jeho grafická podoba je pak implementovaná v projektu `simulatorGUI`. Tento simulátor slouží k sehrání her, kde za každého hráče hraje strategie. Simulátor používáme k sehrání více her mezi několika strategiemi za účelem jejich porovnání. Porovnávání strategií pak využíváme při ohodnocení jednotlivých jedinců při evoluci.

5.1.1 Implementace simulátoru

Jádro simulátoru je implementováno ve třídě `Simulator`, která zajišťuje sehrání jedné hry. Při sehrávání her se pak vždy volá metoda na této třídě s názvem `Run`. Tato metoda vrací status hry po jejím sehrání. Tedy počty bodů jednotlivých hráčů, počet kol, kdo postavil jaké vesnice atd. Údaje z tohoto statusu se uloží do celkové statistiky a spustí se další hra. Po sehrání všech her se tato celková statistika vytiskne na konzoli. Třídou `Simulator` pak využívá i grafická verze simulátoru. Implementace metody `Run` obsahuje cyklus, který se provádí do té doby, než nastane konec hry. V jednom cyklu se provede jeden tah jednoho hráče. Jeden takový cyklus je znázorněn v následující ukázce:

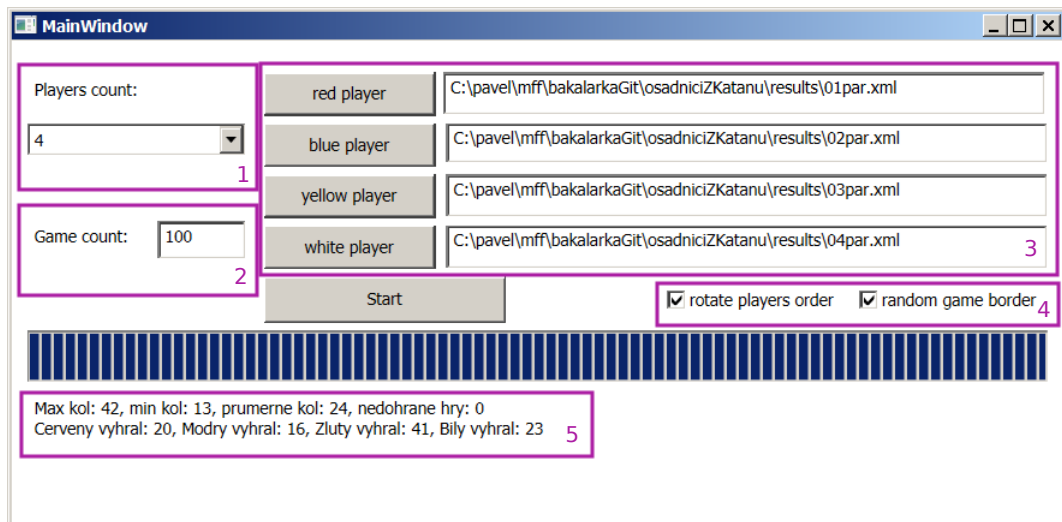
```
//na začátku hodíme kostkami
gm.RollTheDice();
//pokud padla sedmička, tak je třeba přemístit zloděje
//jinak stěny produkují suroviny
if (gm.GmProp.FallenNum != 7) { gm.GetMaterials(gm.GmProp.FallenNum); }
else { gm.GmProp.NeedToMoveThief = true; }

int counter = 0;
do
{
    //pojistka proti zacyklení
    if (counter > MAX_MOVES) { throw new TooManyMovesException(); }
    //strategie actAI generuje tah
    mvDes = actAI.GenerateMove(gm.GmProp, gm.ActualPlayer.PlProp);
    //tento tah se také provede
    moveToStr = gm.MakeMove(mvDes);
    counter++;
} while (!(mvDes is NothingToDoMove));
gm.NextPlayer();
```

Pro sehrání prvních dvou tahů hry je cyklus mírně upraven, nehází se kostkou a stěny neprodukují suroviny.

5.1.2 Ovládání

Ovládání grafické verze simulátoru V této sekci popíšeme ovládání grafické verze simulátoru. Okno simulátoru je zobrazeno na obrázku 5.1. Počet hráčů v jedné hře lze zvolit v boxu s číslem jedna, v boxu s číslem dva navolíme počet her a v boxu s číslem tři pak vybereme jedince, kteří proti sobě mají sehrát hry. Implicitní pořadí hráčů ve hře je první červený, druhý modrý, třetí žlutý a čtvrtý je modrý hráč. V boxu s číslem čtyři pak lze zvolit, že se bude pořadí hráčů spravedlivě střídát. Dále lze v tomto boxu navolit, zda chceme hrát na předpřipraveném hracím poli (které je stejné jako předpřipravené hrací pole v projektu `osadniciZKatanu`), a nebo chceme každou hru generovat náhodný hrací plán. Po sehrání her se v boxu číslo pět zobrazí statistika her. Jde konkrétně o maximální počet kol v jedné hře, minimální počet kol v jedné hře, průměrný počet kol v jedné hře a počet nedohraných her. Ten je v typickém případě nula. Jde o pojistku proti zacyklení. V případě, že hra přesáhne určitý počet kol, řádově stovky, tak se hra ukončí a její výsledky nebudou započítány ve statistice. Dále pak statistika ukazuje počet výher jednotlivých hráčů.



Obrázek 5.1: okno simulátoru

Ovládání konzolové verze simulátoru Program `simulator.exe` je konzolová aplikace, která na vstupu požaduje dva parametry. První parametr je xml soubor se seznamem jedinců, kteří mají proti sobě sehrát turnaj. Druhý parametr je číslo, kolik mají proti sobě dva jedinci sehrát her. Turnaj pak probíhá stylem každý s každým. Tedy každá dvojice jedinců spolu sehraje několik her. jedinec sám se sebou turnaj nehraje. Na výstupu je pak po řadě uveden počet výher jednotlivých jedinců. Pod tímto seznamem je uvedena tabulka, kde je znázorněno, v jaké hře získal jaký jedinec kolik bodů. Na řádce k a sloupci l je uvedeno, kolikrát vyhrál k -tý jedinec s l -tým jedincem při hře, kde začínal jedinec k . Počet výher jedince l je pak doplněk do celkového počtu her. Tuto aplikaci jsme používali pro ohodnocení jednotlivých jedinců vyšlechtěných různými typy evolucí v kapitole 3.

5.2 Evoluce

Evoluce je implementovaná v projektu s názvem `evolution`. Jde o konzolovou aplikaci, která na vstupu vyžaduje xml soubor, který popisuje nastavení evoluce.

5.2.1 Implementace evoluce

Evoluce probíhá v cyklu přes generace. V každém cyklu se nejdříve ohodnotí všichni jedinci v populaci a potom se jedinci zkříží a provede se mutace. Hlavní komponenty projektu `evolution` jsou:

- třída `Individual` reprezentuje jednoho jedince. Tato třída obsahuje pole celých čísel.
- třída `Population` reprezentuje populaci, tedy skupinu jedinců. Třída obsahuje seznam jedinců a implementuje metodu na vytvoření nové náhodné populace.
- interface `IOperator`, který definuje rozhraní operátoru. Operátorem nazýváme jednak mutaci a jednak křížící funkci. Tento interface má definovanou jednu metodu, a to sice `Operate`, která v parametru dostane populaci a vrátí novou populaci, kde se na všech jedincích provedla požadovaná operace.
- interface `ISelector`, který definuje rozhraní selektoru. Selektor má metodu `Select`, která na vstupu požaduje v parametru populaci a vrací populaci s přeuspořádanými jedinci. Vždy dva jedinci vedle sebe byli vybráni pro zkřížení. V projektu je implementovaná pouze ruletová selekce, ale je zde volná cesta pro další možný vývoj.
- interface `IFitnessEvaluator` definuje rozhraní pro implementaci ohodnocovací funkce. Interface definuje metodu `Evaluate`, která v parametru požaduje populaci. Každého jedince z populace pak ohodnocovací funkce ohodnotí a nastaví mu jeho novou fitness.
- třída `EvolutionAlgorithm` má na starosti samotnou evoluci. Implementuje metodu `Evolve`, která na vstupu dostane v parametru populaci. Metoda nejdříve provede selekci a potom na každého jedince použije všechny operátory, které jsou v evolučním algoritmu nastavené.
- třída `EvolutionAlgorithmProperties` obsahuje nastavení evoluce. Implementuje metodu `LoadFromXml`, která v parametru dostane název xml souboru s nastavením populace. Metoda poté z tohoto xml souboru načte nastavení evoluce.

5.2.2 Nastavení a spuštění evoluce

Program `evolution.exe` je konzolová aplikace, která na vstupu požaduje jeden parametr. Tento parametr je název xml souboru s nastavením evoluce. Parametry evoluce jsou popsány v sekci 2.4. U každé ze tří ohodnocovacích funkcí nastavujeme jiné parametry, a tedy používáme tři typy xml souborů k popsání evoluce. Tři konkrétní typy xml souborů s nastavením evoluce jsou uvedeny v příloze C.

5.3 Osadníci z katanu AI

Umělá inteligence pro Osadníky z Katanu je implementována v projektu s názvem `osadniciZKatanuAI`. V projektu je implementován interface `IGameLogic`, který definuje metodu `GenerateMove`, která vrací popis tahu. Tato metoda požaduje dva parametry. První parametr je reference na třídu `GameProperties`, která popisuje stav hry a druhý parametr je reference na třídu `PlayerProperties`, která popisuje stav hrajícího hráče. Na základě těchto dvou tříd implementace interfacu vygeneruje tah, který hrající hráč zahraje. V projektu je definovaná jedna implementace tohoto interfacu s názvem `MyGameLogic`. Tato implementace při zavolání metody `GenerateMove` nejdříve vygeneruje všechny možné tahy hrajícího hráče, které ohodnotí. Potom vrátí tah s největším ohodnocením. Vzorce na ohodnocení jednotlivých tahů jsou popsány v sekci 1.4.

6. Uživatelská dokumentace

6.1 Nastavení nové hry

Při spuštění hry se zobrazí dialog s nastavením hry jako na obrázku 6.1. V tomto dialogu lze nastavit jazyk hry - je na výběr z českého a anglického jazyka, hrací plán - lze zvolit zda chceme generovat náhodný hrací plán nebo využít předpřipravený, a je možné zvolit počet hráčů. V aplikaci je možné zvolit dva až čtyři hráče. Dále je u každého hráče možné zvolit, zda za něj bude hrát fyzický hráč nebo strategie. U strategie lze zvolit, jaký jedinec se má použít. Při výběru strategie zvolíme xml soubor s jedincem. Tento jedinec je stejný pro všechny hráče, za které hraje strategie. Pokud není vybrán žádný explicitní jedinec, tak je použit nejlepší vyšlechtěný jedinec z kapitoly 3. Dále je možné zobrazit ID vrcholů, stěn a hran. Toto ID se využívá při popisu tahů strategie. Když je strategie na tahu, vypíše, jaký tah provedla. K popisu stěn, hran a vrcholů používá právě tato ID označení. Dále je možné zvolit zobrazení všech možných tahů, které může strategie udělat včetně jejich ohodnocení.

The screenshot shows a window titled 'MainWindow' with a light blue background. It contains several sections of controls:

- Jazyk:** A dropdown menu currently set to 'čeština'.
- Hrací plán:** A dropdown menu currently set to 'výchozí'.
- Počet hráčů:** Radio buttons for 'dva', 'tři', and 'čtyři', with 'čtyři' selected.
- Player Color/Strategy Selection:**
 - červený (red):** Radio buttons for 'hráč' (selected) and 'počítač'.
 - modrý (blue):** Radio buttons for 'hráč' (selected) and 'počítač'.
 - žlutý (yellow):** Radio buttons for 'hráč' (selected) and 'počítač'.
 - bílý (white):** Radio buttons for 'hráč' (selected) and 'počítač'.
- Buttons:** 'Nová hra' (New game) and 'Vybrat AI' (Select AI).
- Checkboxes:** 'u AI ukázat možné tahy' and 'u AI ukázat popisy tahů', both currently unchecked.
- Input Field:** A text box next to the 'Vybrat AI' button, currently empty.

Obrázek 6.1: formulář pro nastavení nové hry

6.2 Herní okno

Hrací plán na obrázku 6.2 je rozdělen do několika boxů. Číslo v boxu jedna udává kolikáté kolo hry se odehrává. V boxu číslo dva je počet akčních karet v balíčku a v boxu číslo tři je počet cest, vesnic a měst, které může ještě hrající hráč postavit. Boxy čtyři až osm popisují hrací plán. Příklad přístavu je uvedený v boxu číslo čtyři. Jedná se konkrétně o přístav s kurzem dvě cihly za jednu libovolnou surovinu. Oproti tomu v boxu číslo osm je uvedený přístav s kurzem tři libovolné suroviny za jednu libovolnou surovinu. Příklad stěny je v boxu číslo šest. Tato stěna produkuje obilí v případě, že na hrací kostce padne číslo tři. Speciální stěna je v boxu číslo pět. Na této stěně je poušť a tedy tato stěna

neprodukuje žádnou surovinu. Navíc je na této stěně výchozí pozice zloděje. Dále je v boxu číslo sedm uvedený příklad vrcholu a hrany. Na vrcholu má modrý hráč postavenou vesnici a na hraně cestu.

V boxu s číslem jedenáct jsou informace o hrajícím hráči. Tedy jeho barva a počet bodů a rytířů. V boxu deset je seznam koupených a zatím nepoužitých akčních karet hrajícího hráče a v boxu s číslem devět jeho aktuální zásoba surovin. V boxu s číslem dvanáct jsou tlačítka na ovládání hry. Vždy jsou aktivní pouze ty, které může hrající hráč použít. Tedy tlačítko koupit akční kartu je aktivní pouze tehdy, když má hrající hráč potřebné suroviny na koupi akční karty a zároveň ještě nepostavil žádnou stavbu. A nakonec v boxu třináct je seznam potřebných surovin na jednotlivé akce hráče.



Obrázek 6.2: okno hry

6.3 Ovládání hry

Ve hře se hráči střídají na tahu, kde hrající hráč může postavit jednu stavbu. Stavbou rozumíme postavení cesty, vesnice, města nebo koupě akční karty. Dále může hrající hráč použít na tahu jednu akční kartu. Hrající hráč může měnit suroviny v kterékoli fázi tahu. Poslední možná akce hry je přemístění zloděje.

6.3.1 První dva tahy hry

Na začátku hry každý hráč postaví jednu vesnici a jednu cestu, přičemž cesta musí sousedit s vesnicí. Poslední hráč postaví dvě vesnice a dvě cesty, a zase další hráči v opačném pořadí než na začátku staví po jedné cestě a vesnici. Tedy na konci první fáze hry má každý hráč postavené dvě vesnice a dvě cesty. Vesnici hráč postaví kliknutím levým tlačítkem myši na vrchol hrací plochy a cestu postaví kliknutím na hranu hrací plochy. Příklad vrcholu a hrany s vesnicí a cestou je označen na obrázku 6.2 v boxu číslo sedm.

Rytíř Tato akční karta vyžaduje přemístění zloděje a vybrání hráče, kterého zloděj okrade (pokud není tento hráč určen jednoznačně). Přemístění zloděje a vybrání hráče se ovládá stejně jako je popsáno v sekci 6.3.3.

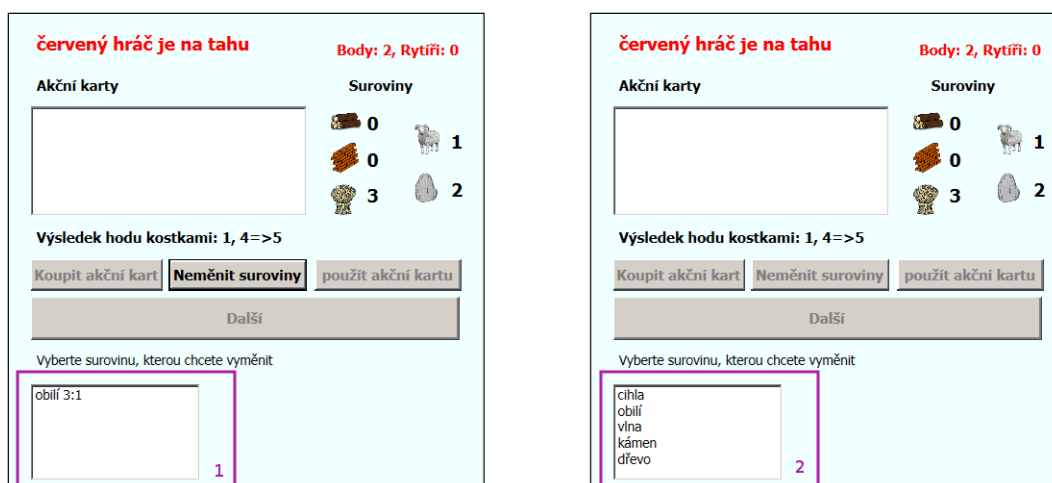
Dva materiály zadarmo Při použití této akční karty se hráči zobrazí stejná nabídka, jako je zobrazená na obrázku 6.4 vpravo v boxu dva. Dvojklikem si hráč vybere první a druhou surovinu.

Dvě cesty zadarmo Hráč postaví dvě cesty zadarmo (bez placení surovinami) kliknutím na dvě hrany na hracím plánu, stejně jako při postavení cesty, za kterou zaplatí.

Surovina od hráčů Hráči se po použití této akční karty zobrazí nabídka surovin, jako na obrázku 6.4 vpravo v boxu dva. Dvojklikem hráč vybere surovinu, kterou chce získat od soupeřů. V surovinách se mu pak přičte tolik kusů této suroviny, kolik jí měli všichni soupeři dohromady. Soupeřům se naopak tato surovina odečte.

6.3.5 Výměna surovin

Vyměnit suroviny může hráč v jakékoli fázi tahu. Kliknutím na tlačítko vyměnit suroviny se zobrazí nabídka zobrazená na obrázku 6.4 vlevo v boxu s číslem jedna. Jde o seznam surovin, které může hrající hráč vyměnit s kurzem, který je u nich uvedený. Na uvedeném příkladě například červený hráč může vyměnit tři obilí za jednu libovolnou surovinu. Při dvojkliku levým tlačítkem myši na zvolenou surovinu se zobrazí druhá nabídka, která je zobrazena na stejném obrázku vpravo v boxu s číslem dva. Jde o seznam všech surovin, ze kterých si hrající hráč vybere tu, kterou potřebuje. Opět surovinu vybere dvojklikem.



Obrázek 6.4: okno hry - výměna surovin

6.3.6 Konec hry

Hra končí v okamžiku, kdy nějaký hráč dosáhl alespoň deseti bodů. Potom se zobrazí dialog znázorňující pořadí hráčů. Poté je možné navolit další nastavení pro novou hru.

7. Závěr

7.1 Zhodnocení výsledků

Cíl bakalářské práce byl naprogramovat implementaci hry Osadníci z Katanu a vytvořit umělou inteligenci do této hry za pomoci evolučních algoritmů. V projektu je naimplementovaná třída strategií, která je parametrizovaná vektorem čísel, které určují váhu jednotlivých tahů ve hře. Spustili jsme několik různých evolučních algoritmů a sehráli více než sto miliónů her. Mezi nejlepšími vyšlechtěnými jedinci jsme pak sehráli turnaj a nejlepšího jedince jsme nastavili jako defaultního protihráče v implementované hře osadníci z katanu.

Podle konkrétních vah nejlepších vyšlechtěným jedinců jsme se pokusili zhodnotit styl hry vyšlechtěné strategie. Zjistili jsme, že strategie při evoluci u většiny jedinců poznala, že postavení vesnice je důležitější než postavení cesty a postavení cesty je většinou důležitější jak koupě akční karty. Dále při prvních dvou tazích hry strategie preferovala umístění u kamene a obilí. Tedy surovin potřebných pro postavení města. Je to logické, protože pokud hráč postaví město, tak získá dvojnásobek možných surovin a může přebytečné suroviny měnit za ostatní.

Všechny evoluční algoritmy probíhaly se standardním nastavením hry. Toto nastavení hry je uloženo v xml souboru. Zde jsou uvedeny údaje o tom, kolik je jakých akčních karet ve hře, počet budov a cest na začátku hry každého hráče, minimální počet bodů na vítězství, kolik produkuje suroviny vesnice a další. Toto nastavení lze snadno změnit a po kompilaci celého projektu se změny provedou.

7.2 Další vývoj

Aplikace byla psána tak, aby byl její další vývoj co nejméně náročný. Zejména lze snadno rozšiřovat třídu strategií, uživatelské rozhraní a pravidla hry.

Vývoj umělé inteligence Třídou strategií definují pravidla pro ohodnocení tahů popsaná v sekci 1.4. Tyto pravidla lze velmi snadno prakticky bez omezení vyvíjet. Například u prvních dvou tahů hry ohodnocujeme jednotlivé vrcholy pouze na základě stěn, se kterými sousedí. Zde by bylo vhodné započítat do výsledného fitness i širší okolí vrcholu. Strategie by měla zjistit, zda je možnost další expanze do okolí, nebo zda jsou okolní vrcholy již obsazené vesnicemi soupeřů. Tyto skutečnosti by se mohly projevit ve výsledném ohodnocení tahu. Dále při postavení nové cesty pouze zjišťujeme, zda se vytvořila nová nejdelší cesta hráče, zda se vytvořila nová nejdelší cesta vůbec a zda se vytvořilo nové místo pro postavení vesnice. Jako možné rozšíření lze do vzorce na výpočet fitness tohoto tahu zahrnout, zda jsme postavili cestu, která vede ke stěně se surovinou, kterou ještě nemáme. Při změně těchto pravidel by bylo nutné znovu spustit evoluční algoritmy a vyšlechtit nové jedince.

Vývoj hry Hra jde dále vyvíjet a upravovat. Lze například velmi snadno změnit hrací plán za jiný. Stačí vytvořit obrázek se základním tvarem plánu a umístit ho do okna hry. Dále je třeba popsat tento hrací plán v xml souboru. V tomto

popisu je uvedeno kde leží vrchol, kde hrana, kde stěna, na jaké stěně je jaká surovina a kde je jaký přístav. Dále lze implementovat různá rozšíření hry. Jako například přístavní mistr, což je karta, kterou získá hráč, který má alespoň tři vesnice, nebo jednu vesnici a jedno město na přístavu.

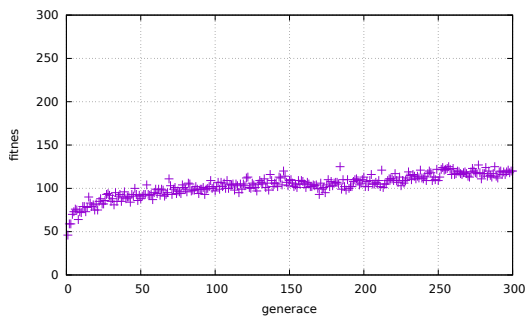
Vývoj uživatelského rozhraní Uživatelské rozhraní lze také velmi snadno vyvíjet. Ve hře je oddělena grafická podoba hry a logika hry. Některé implementované metody nejsou úplně intuitivní, jako například způsob vybrání surovin při jejich výměně, a bylo by možné jejich výběr zlepšit.

Seznam použité literatury

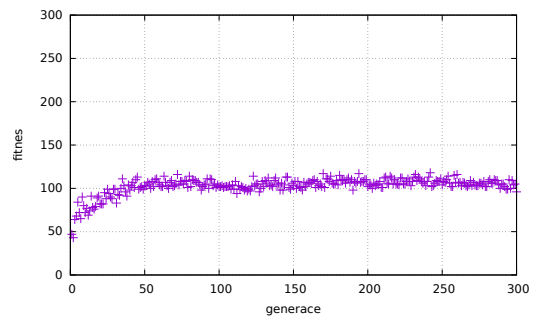
- [1] CatanGmbH. Settlers of Catan oficial website. [online], 5. 4. 2016.
<http://www.catan.com/>.
- [2] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.
- [3] Microsoft. MSDN dokumentace jazyka c#. [online], 5. 4. 2016.
<https://msdn.microsoft.com/en-us/library/618ayhy6.aspx>.

Přílohy

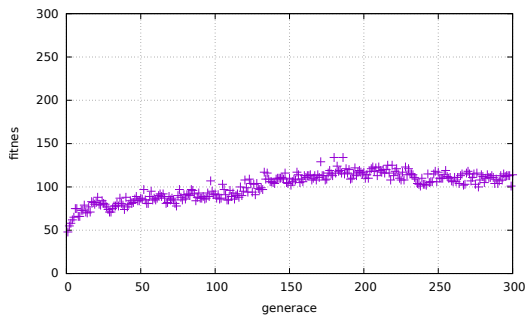
Příloha A - výsledky evoluce



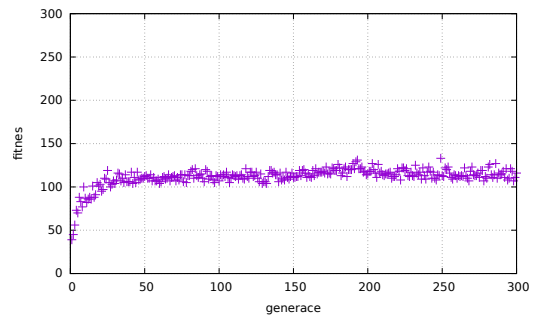
Obrázek 7.1: Basic 1



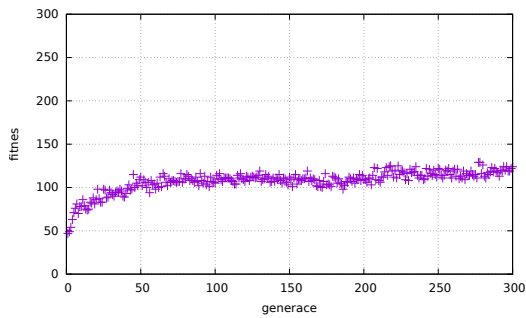
Obrázek 7.2: Basic 2



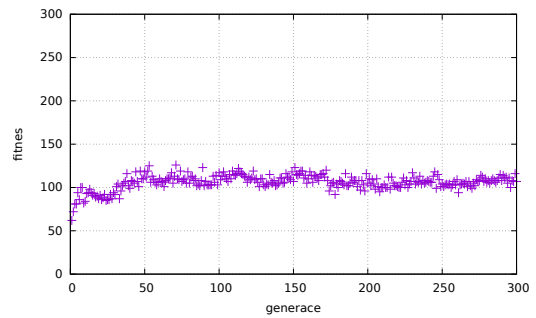
Obrázek 7.3: Basic 3



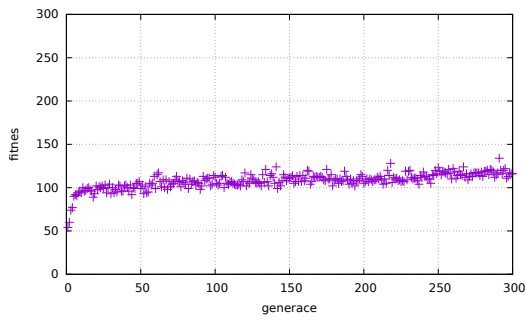
Obrázek 7.4: Basic 4



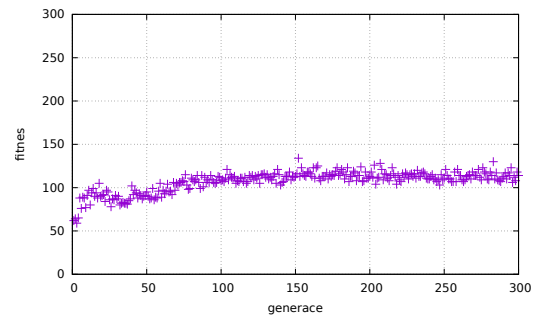
Obrázek 7.5: Basic 5



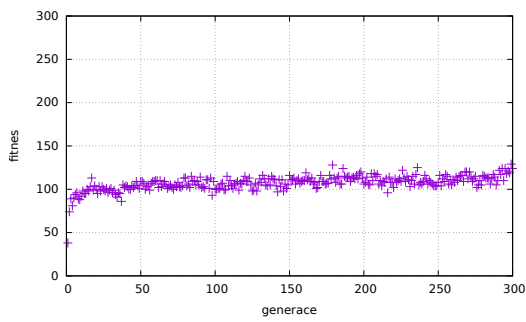
Obrázek 7.6: Basic 6



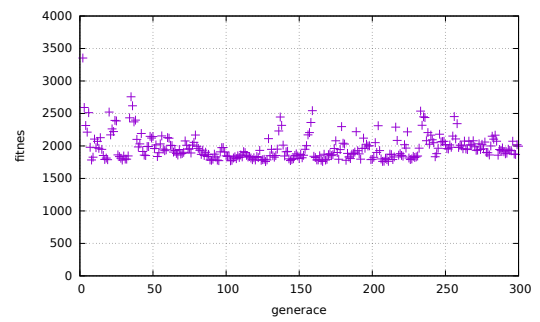
Obrázek 7.7: Basic 7



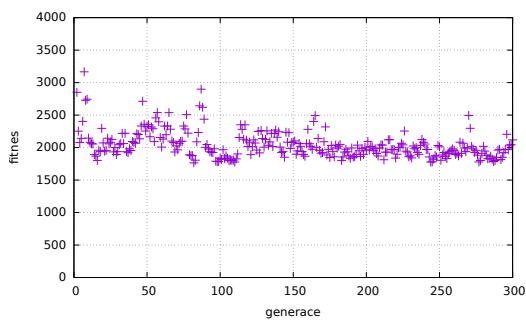
Obrázek 7.8: Basic 8



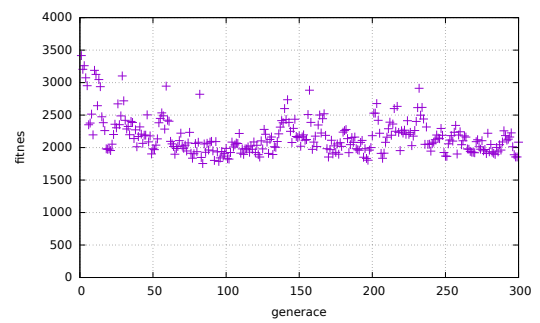
Obrázek 7.9: Basic 9



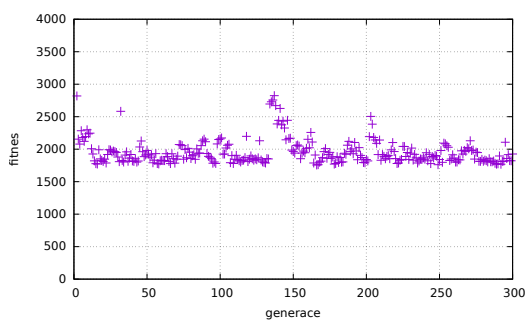
Obrázek 7.10: EbdFourPl 1



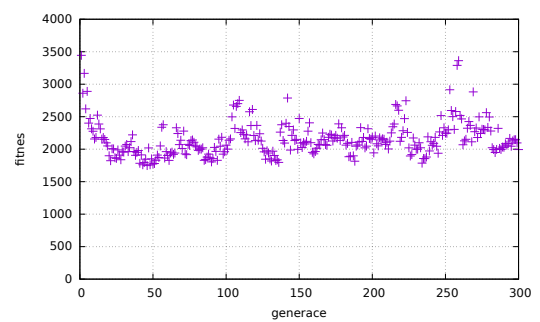
Obrázek 7.11: EbdFourPl 2



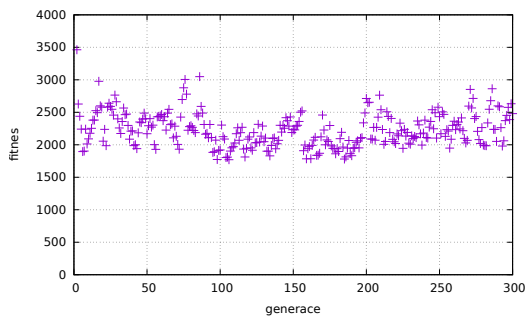
Obrázek 7.12: EbdFourPl 3



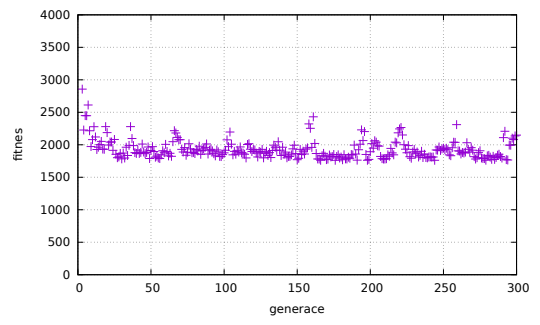
Obrázek 7.13: EbdFourPl 4



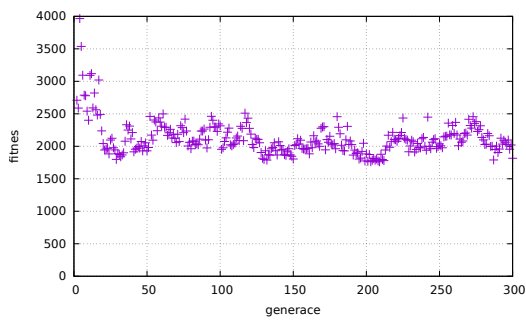
Obrázek 7.14: EbdFourPl 5



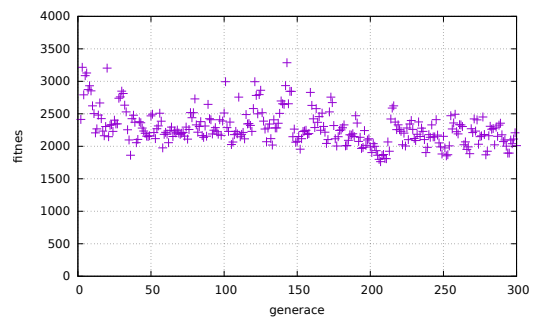
Obrázek 7.15: EbdFourPl 6



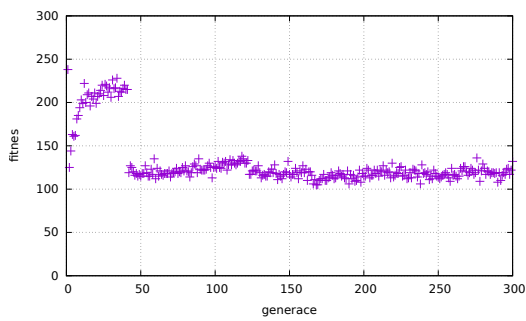
Obrázek 7.16: EbdFourPl 7



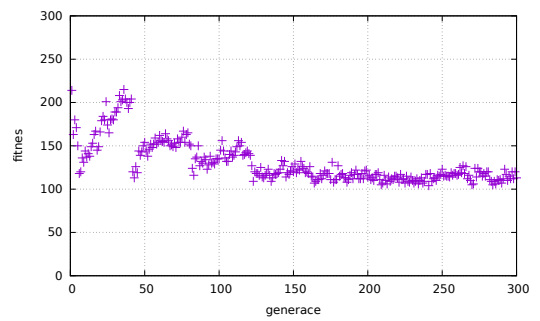
Obrázek 7.17: EbdFourPl 8



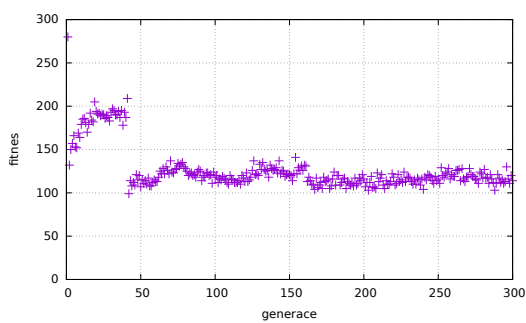
Obrázek 7.18: EbdFourPl 9



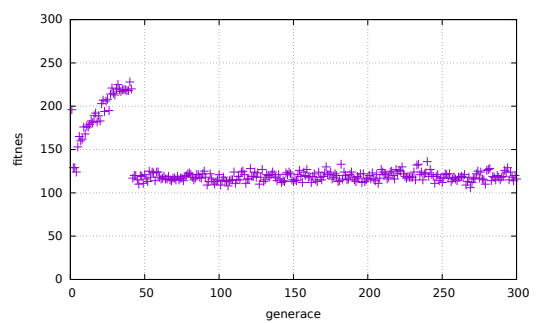
Obrázek 7.19: Changing rivals 1



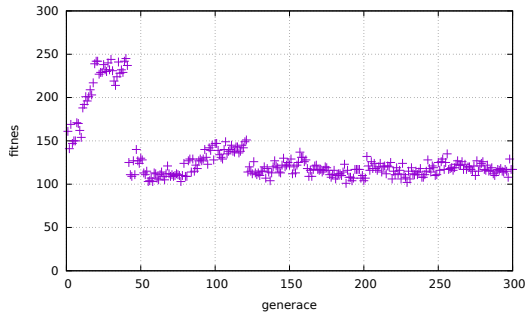
Obrázek 7.20: Changing rivals 2



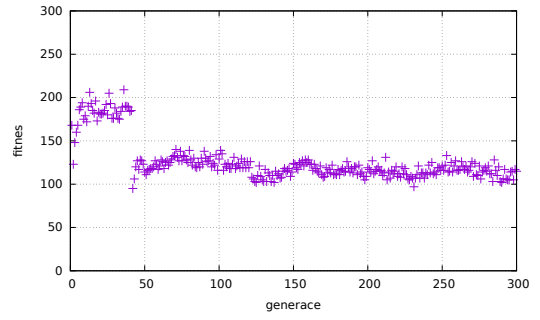
Obrázek 7.21: Changing rivals 3



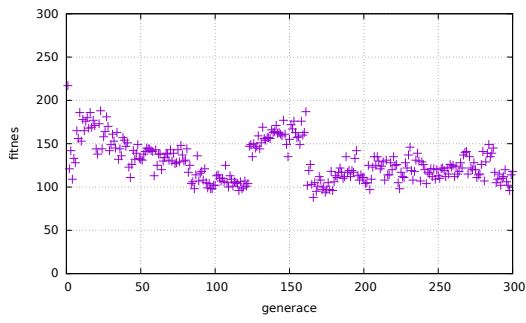
Obrázek 7.22: Changing rivals 4



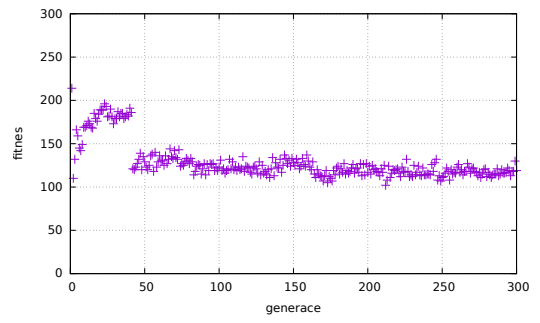
Obrázek 7.23: Changing rivals 5



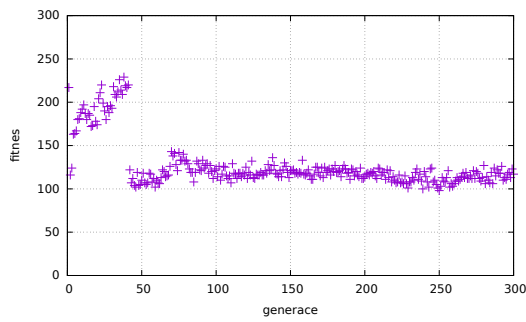
Obrázek 7.24: Changing rivals 6



Obrázek 7.25: Changing rivals 7



Obrázek 7.26: Changing rivals 8



Obrázek 7.27: Changing rivals 9

| Název | počet výher |
|----------------------------------|--------------------|
| Changing Rivals Small Matting 9 | 22617 |
| Changing Rivals Small Matting 8 | 24162 |
| Changing Rivals Small Matting 7 | 24611 |
| Changing Rivals Small Matting 6 | 27587 |
| Changing Rivals Small Matting 5 | 25240 |
| Changing Rivals Small Matting 4 | 25178 |
| Changing Rivals Small Matting 3 | 28953 |
| Changing Rivals Small Matting 2 | 24384 |
| Changing Rivals Small Matting 1 | 30616 |
| Changing Rivals Medium Matting 9 | 21759 |
| Changing Rivals Medium Matting 8 | 25646 |
| Changing Rivals Medium Matting 7 | 26751 |
| Changing Rivals Medium Matting 6 | 27423 |
| Changing Rivals Medium Matting 5 | 20362 |
| Changing Rivals Medium Matting 4 | 26039 |
| Changing Rivals Medium Matting 3 | 25916 |
| Changing Rivals Medium Matting 2 | 25959 |
| Changing Rivals Medium Matting 1 | 27901 |
| Changing Rivals 9 | 30158 |
| Changing Rivals 8 | 27173 |
| Changing Rivals 7 | 26273 |
| Changing Rivals 6 | 30005 |
| Changing Rivals 5 | 25555 |
| Changing Rivals 4 | 26827 |
| Changing Rivals 3 | 23479 |
| Changing Rivals 2 | 24902 |
| Changing Rivals 1 | 26524 |

Tabulka 7.1: Výsledky šlechtění - různá pravděpodobnost křížení

Příloha B - parametry jedince

| Název parametru | Zkratka |
|-------------------------------|-------------------------|
| weightGoodNumbers | <i>goodNum</i> |
| weightPortThreeOne | <i>ThreePort</i> |
| weightPortTwoOne | <i>TwoPort</i> |
| weightMissingMaterial | <i>missMat</i> |
| weightEdgeGeneral | <i>edgeGen</i> |
| weightWood | <i>wood</i> |
| weightBrick | <i>brick</i> |
| weightGrain | <i>grain</i> |
| weightSheep | <i>sheep</i> |
| weightStone | <i>stone</i> |
| weightRoadGeneral | <i>roadGen</i> |
| weightLongestRoad | <i>longRoad</i> |
| weightExtensionRoad | <i>extRoad</i> |
| weightSpaceForVillage | <i>villageSpace</i> |
| weightThiefMoveBase | <i>thiefMove</i> |
| weightThiefMoveOtherBuilding | <i>thiefOthBd</i> |
| weightThiefMoveMyBuilding | <i>thiefMyBd</i> |
| weightThiefMoveProb | <i>thiefProb</i> |
| weightVillageGeneral | <i>villageGen</i> |
| weightVillageGoodNumbers | <i>villageGoodNum</i> |
| weightVillagePortThreeOne | <i>villagePortThree</i> |
| weightVillagePortTwoOne | <i>villagePortTwo</i> |
| weightTownGeneral | <i>townGen</i> |
| weightTownGoodNumbers | <i>townGoodNum</i> |
| weightBuyActionCardGeneral | <i>buyActCard</i> |
| weightUseActionCardGeneral | <i>useActCard</i> |
| weightUseMatFromPIGeneral | <i>actMatFromPl</i> |
| weightUseTwoMatRoadGeneral | <i>actTwoMatRoad</i> |
| weightUseTwoMatVillageGeneral | <i>actTwoMatVillage</i> |
| weightUseTwoMatTownGeneral | <i>actTwoMatTown</i> |
| weightUseTwoMatActGeneral | <i>actTwoMatAct</i> |
| weightUseTwoRoadGeneral | <i>actTwoRoad</i> |

Tabulka 7.2: Seznam parametrů jedince

Příloha C - xml soubory

Příklad jedince s konkrétními váhami

```
<bestParameter weightGoodNumbers="27874" weightPortThreeOne="19582"
weightPortTwoOne="80960" weightMissingMaterial="95751"
weightEdgeGeneral="97430" weightWood="40598" weightBrick="41559"
weightGrain="51904" weightSheep="37971" weightStone="58879"
weightRoadGeneral="24532" weightLongestRoad="26949"
weightExtensionRoad="3273" weightSpaceForVillage="16262"
weightThiefMoveBase="39535" weightThiefMoveOtherBuilding="49818"
weightThiefMoveMyBuilding="79247" weightThiefMoveProb="13457"
weightVillageGeneral="45371" weightVillageGoodNumbers="61612"
weightVillagePortThreeOne="83486" weightVillagePortTwoOne="19280"
weightTownGeneral="97735" weightTownGoodNumbers="42754"
weightBuyActionCardGeneral="50459" weightUseActionCardGeneral="24078"
weightUseMatFromPlGeneral="19762" weightUseTwoMatRoadGeneral="52566"
weightUseTwoMatVillageGeneral="82423" weightUseTwoMatTownGeneral="60341"
weightUseTwoMatActGeneral="59119" weightUseTwoRoadGeneral="21599"/>
```

Nastavení evoluce s ohodnocovací funkcí pevný protihráč

```
<?xml version="1.0" encoding="UTF-8"?>
<eva popSize="40" generationCount="300" upperBoundaryEachIndividual="100000">
  < matingManner type="OnePtXOver" matingProb="0.9"/>
  < mutation mutationProb="0.08" mutationChangeBitProb="0.12"/>
  < evaluatorManner type="basic" playersCountInGame="4">
    < firstRival>01par.xml</firstRival>
    < secondRival>02par.xml</secondRival>
    < thirdRival>03par.xml</thirdRival>
    < gameCount>400</gameCount>
  </evaluatorManner>
</eva>
```

Nastavení evoluce s ohodnocovací funkcí učící se pevný protihráč

```
<?xml version="1.0" encoding="UTF-8"?>
<eva popSize="40" generationCount="300" upperBoundaryEachIndividual="100000">
  < matingManner type="OnePtXOver" matingProb="0.9"/>
  < mutation mutationProb="0.08" mutationChangeBitProb="0.12"/>
  < evaluatorManner type="changeRivals" playersCountInGame="4">
    < gameCount>400</gameCount>
    < changingTime>40</changingTime>
  </evaluatorManner>
</eva>
```

Nastavení evoluce s ohodnocovací funkcí všichni proti všem

```
<?xml version="1.0" encoding="UTF-8"?>
<eva popSize="15" generationCount="300" upperBoundaryEachIndividual="100000">
  < matingManner type="OnePtXOver" matingProb="0.9"/>
  < mutation mutationProb="0.08" mutationChangeBitProb="0.12"/>
  < evaluatorManner type="ebdWithEbd" playersCountInGame="4">
  </evaluatorManner>
</eva>
```

Příloha D - struktura naměřených dat

Naměřená data jsou uložena ve složce `results`. V této složce jsou výsledky spuštěných evolučních algoritmů a výsledky turnajů mezi jedinci.

Výsledky evoluce Výsledky evoluce jsou uloženy ve složce `evolution`. Tato složka obsahuje následující typy souborů:

- `[nazevEvolucnihoAlgoritmu].xml` - nastavení konkrétního evolučního algoritmu.
- `[nazevEvolucnihoAlgoritmu].pdf` - graf s vývojem fitness konkrétního evolučního algoritmu.

Dále jsou pak konkrétní výsledky evolučního algoritmu uloženy ve složce s názvem `[nazevEvolucnihoAlgoritmu]Results`. Tato složka obsahuje následující dva typy souborů:

- `[cisloGenerace]-[hodnotaFitness]-best.xml` - soubory tohoto typu obsahují nejlepšího jedince v dané generaci.
- `[cisloGenerace]-[prumerneFitness]-allPopulation.xml` - tyto soubory obsahují všechny jedince v dané generaci.

Výsledky turnajů Složka `tournaments` obsahuje výsledky turnaje nejlepších vyšlechtěných jedinců mezi sebou. Nastavení turnaje je vždy uloženo v souboru `[nazevTurnaje].xml` a výsledek turnaje je v souboru `[nazevTurnaje].txt`. Jsou zde tyto turnaje:

- `firstTournament` - jde o turnaj jedinců vyšlechtěných pomocí ohodnocovací funkce učící se pevný protihráč a všichni proti všem. První tři jedinci z tohoto turnaje jsou pevní protihráči při evoluci s ohodnocovací funkcí pevný protihráč. V souboru `firstTournament.xml` jsou jedinci uloženi v pořadí `ChangingRivals9 – ChangingRivals1` a `EbdFourP19 – EbdFourP11`. Ve stejném pořadí jedinců jsou uvedeny i výsledky turnaje.
- `bestMatingTournament` - toto je turnaj jedinců s ohodnocovací funkcí učící se pevný protihráč. Tento turnaj byl sehrán, abychom zjistili nejlepší hodnotu pravděpodobnosti křížení. V nastavení turnaje jsou jedinci uloženi v pořadí `ChangingRivalsSmallMating9 – ChangingRivalsSmallMating1`, `ChangingRivalsMediumMating9 – ChangingRivalsMediumMating1` a nakonec `ChangingRivals9 – ChangingRivals1`. Opět ve stejném pořadí jedinců jsou uvedeny i výsledky turnaje
- `finalTournament` - je turnaj 27 nejlepších jedinců vyšlechtěných evolučními algoritmy. Vítěz z tohoto turnaje je nastaven jako defaultní protihráč v implementované hře `Osadníci z Katanu`. Jde o jedince vyšlechtěné evolučními algoritmy s názvem po řadě `ChangingRivals9 – ChangingRivals1`, `EbdFourP19 – EbdFourP11` a `Basic9 – Basic1`.

Příloha E - obsah přiloženého DVD

- `OsadniciPrace.pdf` - tato práce v elektronické podobě.
- složka `osadniciZKatanu` obsahuje celý projekt, včetně zkompileovaných binárních souborů. Soubory jsou zkompilevané na operačním systému Windows 7. Je možné otevřít soubor `osadniciZKatanu.sln` v programu Visual Studio a provést kompilaci.
- složka `results` obsahuje výsledky evoluce.

Následuje seznam binární souborů jednotlivých aplikací v projektu:

- `osadniciZKatanuGUI/bin/Release/osadniciZKatanuGUI.exe` - samotná hra osadníci z katanu.
- `simulator/bin/Release/simulator.exe` - konzolová verze simulátoru.
- `simulatorGUI/bin/Release/simulatorGUI.exe` - grafická verze simulátoru.
- `evolution/bin/Release/evolution.exe` - konzolová verze evoluce.
- `evolutionGUI/bin/Release/evolutionGUI.exe` - grafická verze evoluce.