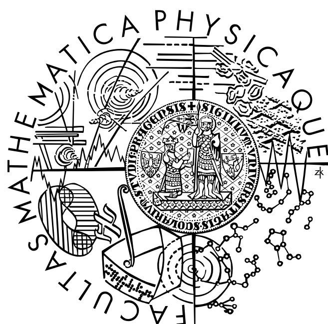
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Radek Fürbach

Implementace umělé inteligence v simulátoru strategické hry

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Jiří Vyskočil Ph.D.

Studijní program: Informatika programování

2008

Poděkování

Děkuji vedoucímu práce RNDr. Jiřímu Vyskočilovi za hodnotné rady.

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 12.12.2008 …………………

Název práce: Implementace umělé inteligence v simulátoru strategické hry
Autor: Radek Fürbach
Katedra: Katedra teoretické informatiky a matematické logiky
Vedoucí bakalářské práce: RNDr. Jiří Vyskočil Ph.D.
e-mail vedoucího: Jiri.Vyskocil@mff.cuni.cz

Abstrakt:

Cílem této práce je porovnání několika vybraných metod umělé inteligence ve specifikované strategické hře. Práce obsahuje tři části.

První část specifikuje model strategické hry, na kterém jsou simulovány experimenty. Definuje objekty vyskytující se ve hře, vztahy mezi nimi a použité algoritmy.

Druhá část specifikuje umělou inteligenci použitou ve strategické hře. Vysvětluje genetické algoritmy a uvádí několik metod selekcí, křížení a mutací. Popisuje některé základní umělé neuronové sítě a jejich architektury.

Poslední část popisuje několik různých algoritmů umělé inteligence vycházejících z teorie v 2. části. Dále porovnává jejich výkonnost na simulovaných experimentech.

Klíčová slova: umělé neuronové sítě, genetické algoritmy, diskrétní simulace;

Title: The Implementation of an Artificial Intelligence in a Strategy Game Simulator
Author: Radek Fürbach
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: RNDr. Jiří Vyskočil Ph.D.
Superviser's e-mail address: Jiri.Vyskocil@mff.cuni.cz

Abstract:

The aim of this thesis is a comparison of a few selected methods of an artificial intelligence in a specified strategy game. The thesis contains three parts.

The first part specifies a model of the strategy game, whereat are simulated some experiments. It defines objects that occur in the game, relation among them, and used algorithms.

The second part specifies of the artificial intelligence that is used in the strategy game. It explains the genetic algorithm and shows a few methods of so called selection, crossing, and mutation. It describes some basic artificial neural networks and their architectures.

The last part describes several algorithms of the artificial intelligence using theory from the second part. It compares their efficiency on the simulated experiments.

Keywords: artificial neural network, genetic algorithms, discrete simulation;

Obsah

1 Úvod 1

2 Specifikace strategické hry 3
2.1 Specifikace 3
2.2 Hráč 3
2.3 Události 4
2.4 Prostředí 7
2.5 Jednotky 7

3 Umělá inteligence 19
3.1 Popis úlohy 19
3.2 Řešení úlohy 20
3.3 Požadavky pro řešení podúloh 21
3.4 Těžba surovin 21
3.5 Výroba jednotek 23
3.6 Útok jednotek 26
3.6 Popis neuronové sítě použité v rozhodování o výrobě jednotek 29
3.7 Genetické algoritmy 36
3.8 Použití genetických algoritmů 39

4 Experimenty 41
4.1 Popis experimentů 41
4.2 Implementační parametry 43
4.3 Parametry strategické hry 44
4.4 Provedené experimenty: 44
4.5 Porovnání fitness funkcí 44
4.6 Porovnání přenosových funkcí 48
4.7 Porovnání architektur neuronových sítí 50

5 Závěr 52

Literatura 53

A Uživatelské rozhraní 54

B Stručný uživatelský pohled 55

C Obsah přiloženého CD 56

D Instalace 57

Kapitola 1

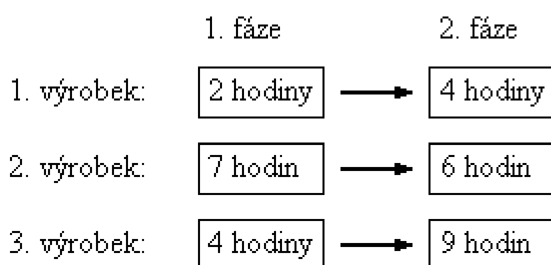
Úvod

V této práci jsem se rozhodl zabývat metodami umělé inteligence. Vybral jsem si umělou inteligenci, protože mi připadá zajímavým nástrojem na řešení mnoha složitých problémů. Jako cíl jsem si kladl specifikaci a implementaci umělé inteligence řešící určitý problém z oboru plánování a rozvrhování.

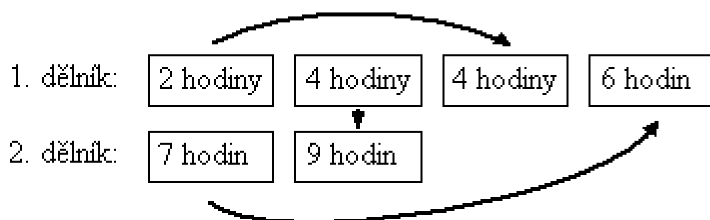
Plánování je činnost vytvoření plánu akcí tak, aby se z počátečního stavu postupným aplikováním akcí z plánu dosáhlo koncového stavu. Problematika rozvrhování se zabývá, jak optimálně rozvrhnout akce z plánu mezi dostupné zdroje. Více o problematice plánování a rozvrhovaní na [1].

Příklad plánování a rozvrhování:

Dva dělníci vyrábějí 3 různé výrobky. Výroba výrobků je rozdělena do 2 fází trvající určitou dobu. Cílem úlohy je rozvrhnout fáze jednotlivých výrobků mezi dělníky tak, aby výrobky byly co nejdříve hotové.



Řešení:



Šipky na obrázku znázorňují závislost výroby jednotlivých fází. Celková doba je rovna 16 hodin.

Z oboru plánování a rozvrhování jsem si vybral úlohu plánování a rozvrhování akcí jednotek ve strategických hrách. Tato úloha se od klasické problematiky plánování a rozvrhování liší tím, že nelze jednoznačně popsat optimum. Dalším rozdílem je, že se tato úloha musí řešit v omezeném čase. Během výpočtu se nesmí strategická hra zastavit na dlouhou dobu, aby se zachovala její hratelnost.

V této práci specifikuji a implementuji strategickou hru. Na specifikované strategické hře definuji úlohu (pro umělou inteligenci). Hlavním cílem této práce je navrhnout a implementovat umělou inteligenci řešící definovanou úlohu.

Kapitola 2 specifikuje strategickou hru. Definuje objekty vyskytující se ve hře a vztahy mezi nimi. Ve 3. kapitole se definuje úloha nad strategickou hrou, navrhuje se řešení a popisují se použité algoritmy z oboru umělé inteligence. Kapitola 4 popisuje několik metod umělé inteligence a porovnává výkonnost na simulovaných experimentech.

Kapitola 2

Specifikace strategické hry

Popisovaná hra je typu strategické hry pro 2 hráče. Cílem hry je zničit všechny soupeřovy jednotky. Za tímto účelem může každý hráč těžit suroviny, vystavět budovy a vybudovat v nich armádu potřebnou ke zničení soupeře. Hra se odehrává v armádním prostředí a probíhá v reálném čase. Každý hráč má pod správou několik jednotek a v průběhu hry jim může zadávat příkazy. Je to jediný způsob, kterým může zasahovat do průběhu hry. Hráč může zadávat příkazy typu: přemístění pohyblivé jednotky na určitou pozici, vyrobit jednotku, těžit suroviny atd. Jednotky plní zadané příkazy samostatně a nezávisle na ostatních jednotkách. Jednotky si nijak nepomáhají a nepředávají si jakékoliv informace.

2.1 Specifikace

Strategická hra je reprezentovaná diskrétní simulací. Modelem simulace je prostředí reprezentované 2 rozměrnou sítí indexů určující typ povrchu. Simulace probíhá deterministicky až na zásahy hráče. Během simulace se postupně provádějí události. Události jsou uloženy ve frontě. V jednom kroku simulace se odebere událost z vrcholu fronty, zpracuje se a zařadí se zpět na konec fronty. Událostí můžeme myslet vykreslení stavu simulace do okna, provedení příkazu uživatele, detekce, zda neskončila hra, provedení akce jednotky atd.

2.2 Hráč

Hráč je prvek schopný vlastního uvažování, který má pod správou několik **jednotek** (viz kapitola 2.5) a může jim zadávat příkazy a tím zasahovat do průběhu simulace. *Hráč* se rozděluje na typy:

- uživatel

  Reprezentuje živého člověka.

- umělá inteligence.

  Reprezentuje strojově vykonávaný algoritmus.

Každý *hráč* ve hře je reprezentován objektem, který obsahuje informace:

- identifikátor hráče (celé číslo)

  Identifikátor jednoznačně určuje *hráče*.

- počet natěžených surovin (celé kladné číslo)

  Reprezentuje bohatství *hráče*.

- maximální počet jednotek (celé kladné číslo)

  Udává maximální počet *jednotek*, které mohou být momentálně pod správou daného *hráče*. Tento limit lze zvýšit výstavbou speciálních budov k tomu určených. Slouží především pro statistické výpisy.

- aktuální počet jednotek (celé číslo v rozsahu 0 - maximální počet jednotek)

  Udává počet *jednotek*, které jsou momentálně pod správou daného *hráče*. Slouží především pro statistické výpisy.

- vylepšené technologie (seznam boolových hodnot)

  Určuje aktuálně dosažené technologie .

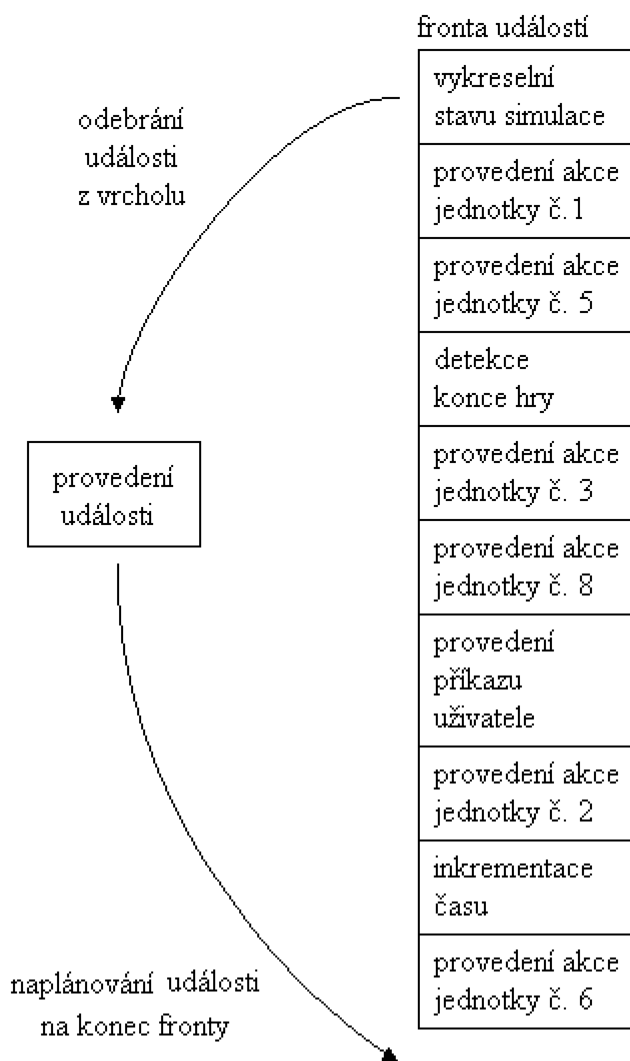
- postavené typy budov (seznam boolových hodnot)

  Určuje typy budov, které *hráč* momentálně vlastní.

2.3 Události

Událost je činnost prováděná v určitém okamžiku simulace. V simulaci jsou prováděny tyto typy událostí:

- provedení akce jednotky
- zadání příkazu uživatelem
- zadání příkazů umělou inteligencí
- vykreslení stavu simulace
- detekce konce hry
- inkrementace času



Na obrázku je znázorněný příklad simulace v určitém stavu. Momentálně se provádí událost *vykreslení stavu simulace,* ve které se vykreslí aktuální stav simulace na obrazovku. Po vykreslení se tato událost naplánuje na konec fronty. Potom se budou provádět dvě události *provedení akce jednotky,* ve kterých se provede jeden krok akce *jednotky* číslo 1 a *jednotky* číslo 5. Po provedení akcí jednotek se v události *detekce konce hry* zjišťuje, zda nenastaly podmínky ukončující simulace. Pokud nenastanou podmínky k ukončení simulace, pak se událost *detekce konce hry* naplánuje na konec fronty. Dále se začnou provádět další události ve frontě dokud se neukončí simulace. Simulace se může ukončit při vyhodnocování události *detekce konce hry*, když nastanou podmínky potřebné k ukončení hry. Podmínka k ukončení simulace nastává v době, když jeden z hráčů nevlastní už žádné *jednotky*.

2.3.1 Provedení akce jednotky

Během této události *jednotka* provádí určitou akci. Typ akce je závislý na zadaném příkazu. *Jednotka* se může přesunout, zaútočit, vyrobit jinou *jednotku* atd. Pokud je daná *jednotka* zničena nebo pokud dokončí svou akci, pak se tato událost dál neprovádí. Jinak se automaticky naplánuje na konec fronty. První naplánovaní události tohoto typu je v době zadání příkazu.

2.3.2 Zadání příkazu uživatelem

Tato událost zadaná příkaz *jednotce*. Při zadání příkazu se automaticky naplánuje událost *provedení akce jednotky*. Událost je potřebná k tomu, aby uživatel mohl zasahovat do chodu simulace. Při kliknutí myši nebo zmáčknutí klávesy se tato událost naplánuje na konec fronty.

2.3.3 Zadání příkazů umělou inteligencí

Tato událost vygeneruje a zadá příkazy *jednotkám*, které jsou pod správou *umělé inteligence*. Při zadání příkazu se automaticky naplánuje událost *provedení akce jednotky*. Událost je potřebná k tomu, aby *umělá inteligence* mohla zasahovat do chodu simulace. Po provedení události se tato událost automaticky naplánuje na konec fronty.

2.3.4 Vykreslení stavu simulace

Aby se uživatel mohl správně rozhodnout, jaký příkaz zadá, musí mít přehled o průběhu simulace. Z tohoto důvodu je potřebná událost, která vykreslí aktuální stav simulace do okna. Po vykreslení je tato událost automaticky znovu naplánována na konec fronty.

2.3.5 Detekce konce hry

Tato událost zjišťuje, zda nenastala podmínka k ukončení simulace. Pokud nastane podmínka k ukončení simulace, pak se simulace ukončí. Jinak se událost znovu naplánuje na konec fronty. Konec simulace nastane, pokud jeden z *hráčů* splní cíl strategické hry. Cílem strategické hry pro všechny *hráče* je zničit všechny *nepřátelské jednotky* (*jednotky* vlastněné protihráčem).

2.3.6 Inkrementace času

Událost *inkrementace času* zajišťuje "realtimeovost" simulace a přehled o čase. Tato událost uspí aplikaci po dobu několika milisekund a inkrementuje čítač času (celé číslo) o 1. Uspáním aplikace zpomaluje průběh celé simulace. Pokud by byl průběh simulace příliš rychlý, bylo by pro uživatele velmi obtížné včas reagovat na některé situace. Čas v celé simulaci je reprezentován čítačem času. Po provedení *inkrementace času* se událost znovu naplánuje na konec fronty.

2.4 Prostředí

Všechny *jednotky* vyskytující se v simulaci jsou umístěny v určitém okolí zvaném *prostředí*. *Prostředí* je reprezentované 2 rozměrným polem indexů určující typ povrchu (konečný výčet). Dále obsahuje seznam *jednotek* vyskytující se v simulaci. Typická *jednotka* pokrývá více elementárních polí. Rozměry prostředí se pohybují v řádech tisíců. *Prostředí* si lze představit jako krajinu s jednotkami, které se v ní vyskytují.

2.5 Jednotky

Jednotka je objekt v simulaci, který je schopen vykonávat akce. Všechny *jednotky* mají umístění v *prostředí* a rozdělují se do několika typů. *Jednotka* je například vozidlo, budova, střela, strom, kouř atd.

2.5.1 Typy jednotek

V simulaci se vyskytují *jednotky* několika typů. Nejobecnějším typem *jednotek* je *základní jednotka*. *Základní jednotky* se dělí na *hmotné jednotky* a *nehmotné jednotky*. *Hmotné jednotky* se dále dělí na *živé jednotky* a *neživé jednotky*. *Živé jednotky* se dělí na *pohyblivé jednotky* a *nepohyblivé jednotky*. *Pohyblivé jednotky* se dále rozdělují na *útočné jednotky* a *neútočné jednotky*.

Formální specifikace pomocí gramatiky jak popisuje Michal Chytil [2]:

základní jednotka	→	nehmotná jednotka
	|	hmotná jednotka
hmotná jednotka	→	neživá jednotka
	|	živá jednotka

živá jednotka	→	nepohyblivá jednotka
	|	pohyblivá jednotka
pohyblivá jednotka	→	neútočná jednotka
	|	útočná jednotka

2.5.1.1 Základní jednotka

Tento typ *jednotky* je nejobecnějším typem ve hře. Všechny *jednotky* jsou potomci tohoto typu. *Základní jednotka* má nadefinováno pouze:

- identifikátor (celé kladné 4-bytové číslo)

  Identifikátor jednoznačně určuje jednotku.

- dvourozměrné souřadnice (odpovídající dvojici celých čísel)

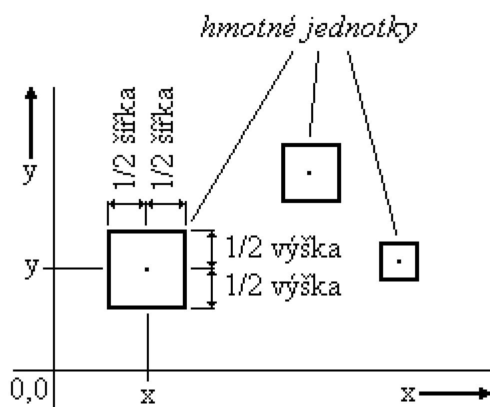
  Souřadnice jsou v jednotkách o velikosti jednoho elementárního pole z *prostředí*.

2.5.1.2 Nehmotné jednotky

Nehmotná jednotka je potomkem *základní jednotky*. Nemůže být vlastněna žádným *hráčem*, nelze na ni zaútočit a nemůže se nikam přesunout. *Nehmotná jednotka* reprezentuje nekonečně tenký bod, který nevyplňuje žádné místo v *prostřední*. Slouží především pro pomocné účely a grafické doplňky. Pomocí ní se vytvářejí drobné animace jako například kouř od raket.

2.5.1.3 Hmotná jednotka

Hmotná jednotka je rozšíření *základní jednotky*. Vlastní navíc informace o své velikosti (výška, šířka (celá kladná čísla)). Výška i šířka je ve stejných jednotkách jako souřadnice. Souřadnice a velikost *jednotky* tvoří obdélníkový výřez, který určuje místo v *prostředí*, které daná *jednotka* pokrývá. Různé *hmotné jednotky* nemůžou v jednom okamžiku pokrývat stejné místo. *Hmotná jednotka* je jakákoliv *jednotka*, která pokrývá v *prostředí* určité místo. Například vozidlo, budovy, stromy atd.



Na obrázku je znázorněný umístění *hmotných jednotek* v prostředí.

2.5.1.4 Neživá jednotka

Tento typ jednotky je zkonkrétnění *hmotné jednotky*. *Neživá jednotka* reprezentuje hmotný nezničitelný objekt. Například surovinový důl, strom či kus skály. *Jednotky* tohoto typu nemůžou být vlastněny žádným *hráčem*, nelze je zničit a nemůžou se přesunout na jinou pozici.

2.5.1.5 Živá jednotka

Živá jednotka je rozšíření typu *hmotné jednotky*. Na tyto *jednotky* je možné zaútočit. Všechny *jednotky*, které můžou být vlastněny *hráčem* jsou typu *živé jednotky*. *Živá jednotka* je navíc rozšířená o atributy:

- odolnost (celé kladné číslo)

  Určuje kolik *jednotka* vydrží

- aktuální poškození (celé číslo v rozsahu 0 - odolnost)

  Určuje jak hodně je *jednotka* momentálně poškozena. Pokud je aktuální poškození větší než odolnost, pak je jednotka zničena.

- obranu (typ (index) a mohutnost (reálné kladné číslo))

  Určuje jak hodně se zvětší aktuální poškození při zásahu *jednotky* střelou. Přesný výpočet poškození popsán v akci **útočení** (viz kapitola 2.5.3.3).

- vlastnící hráč (index)

  Určuje *hráče*, který vlastní danou *jednotku*.

- informace o prostředí (ukazatel na *prostředí*)

2.5.1.6 Nepohyblivá jednotka

Tento typ jednotky je zkonkrétnění *živé jednotky*. *Nepohyblivé jednotky* nejsou schopny se přemístit. Příkladem *nepohyblivé jednotky* jsou budovy.

2.5.1.7 Pohyblivá jednotka

Pohyblivá jednotka je rozšíření *živé jednotky*. Navíc obsahuje informace o typu podvozku (typ (index) a rychlost (reálné kladné číslo)) a je schopen pomocí vlastního algoritmu se přesunout z jednoho místa do druhého místa. Všechny letadla, vozidla a střely jsou jednotky tohoto typu..

2.5.1.8 Neútočné jednotky

Tato *jednotka* reprezentuje *pohyblivou jednotku*, která nemůže zaútočit na jakoukoliv jinou *jednotku*. Příkladem *neútočné jednotky* je stavěč budov.

2.5.1.9 Útočné jednotky

Útočná jednotka je rozšíření *pohyblivé jednotky*. Tento typ *jednotky* může útočit na ostatní *živé jednotky*. Reprezentuje například vozidla a letadla. *Útočná jednotka* má navíc nadefinováno:

- dostřel (celé kladné číslo)

  Určuje maximální vzdálenost *nepřátelské jednotky*, na kterou ještě *útočná jednotka* může zaútočit.

- útok (typ (index) a síla (celé reálné číslo))

  Určuje jak hodně se zvětší aktuální poškození při zásahu *jednotky* střelou. Přesný výpočet poškození popsán v akci *útočení* (viz kapitola 2.5.3.3).

  interval střelby (celé číslo)

  Interval střelby určuje po jaké době od minulého vystřelení je *jednotka* schopna znovu vystřelit.

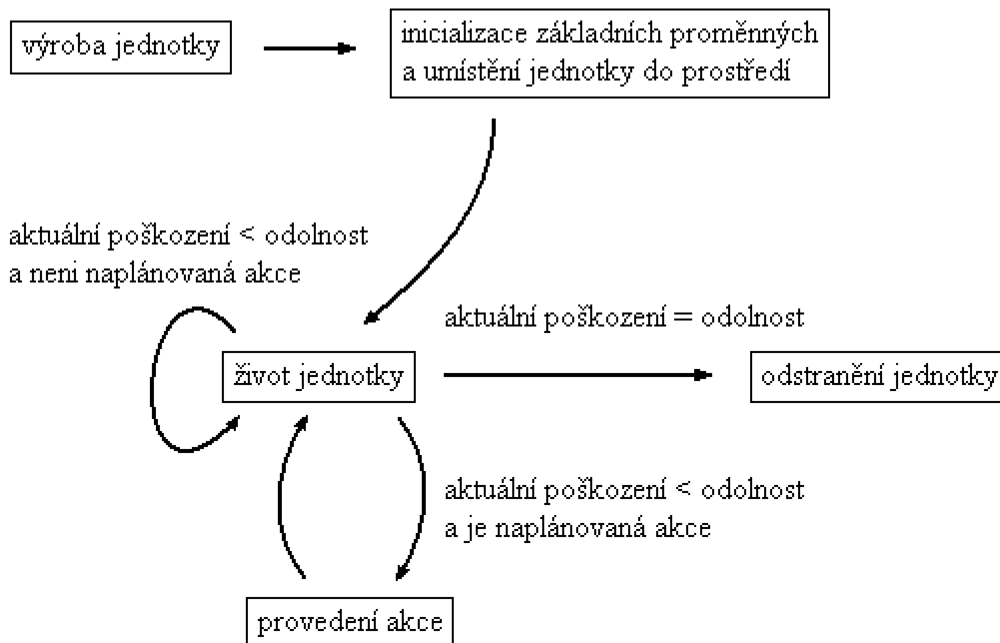
Nepřátelská jednotka

Nepřátelská jednotka je relativní pojem vůči *hráči*. Všechny *jednotky* vlastněné jiným *hráčem* se označují jako *nepřátelské jednotky*.

2.5.2 Životní cyklus jednotky

Životní cyklus *živé jednotky* začíná v době, kdy je vyrobena. Po vyrobení proběhne inicializace základních proměnných (souřadnice, vlastnící hráč atd.) a umístění *jednotky* do *prostředí*. Během svého života *jednotka* může provádět své akce. Při napadení *jednotky* se zvětšuje její aktuální poškození. Životní cyklus *živé jednotky* končí v době, kdy aktuální poškození dosáhne celkové odolnosti. Pak je daná *jednotka* odstraněna z *prostředí*.

Formální znázornění pomocí konečného automatu:



2.5.3 Akce jednotek

Akce je činnost, kterou *jednotka* může provést, když nastane událost *provedení akce jednotky*.

Jednotka ve hře má nadefinováno několik akcí, které může během simulace provést. Akce *jednotek* jsou:

- výroba jednotky
- přemístění
- útočení
- těžba surovin

2.5.3.1 Výroba jednotky

Akce *vyrobení jednotky* vytvoří novou *živou jednotku* určitého typu. Je to jediný způsob, kterým lze vytvořit novou *jednotku*. Tuto akci můžou provést pouze některé *živé jednotky* (některé budovy a stavěč) a v případě, že vlastnící *hráč* má dostatečný počet surovin k vyrobení dané *jednotky*. Akce v*ýroba jednotek* je rozdělena na 3 fáze:

- započetí výroby
- průběh výroby
- konec výroby

Započetí výroby

Započetí výroby je 1. fáze akce *výroby jednotek*. Při zadání příkazu k vyrobení *jednotky* se naplánuje událost *provedení akce jednotky,* ve které se bude provádět akce *výroba jednotek*. Dále se zapamatuje typ vyráběné *jednotky* a nastaví se čítač doby stavby (celé kladné číslo) na hodnotu určující dobu výroby.

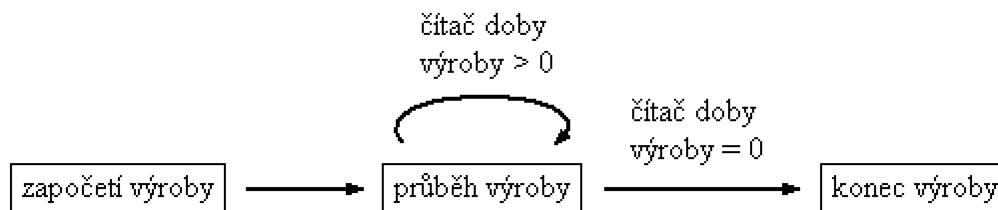
Průběh výroby

V této fázi se při každém provedení akce *výroba jednotek* pouze sníží čítač doby stavby o 1 dokud není roven 0. Po snížení čítače doby stavby se událost *provedení akce jednotky* znovu naplánuje na konec fronty. Při průběhu výroby nemůže *jednotka* provádět jakoukoliv jinou akci (pohyb, výroba jiné jednotky, útočení, těžení)

Konec výroby

Konec výroby nastane v okamžiku, kdy je čítač doby stavby roven 0. V tomto okamžiku se vytvoří nová *jednotka* zadaného typu a inicializují se její proměnné. Vlastnící *hráč* se nastaví na stejného *hráče* jako u vyrábějící *jednotky*. Souřadnice vyrobené *jednotky* se nastaví na poblíž souřadnic vyrábějící *jednotky*, tak aby vyrobená *jednotka* stála vedle vyrábějící *jednoty*. Událost *provedení akce jednotky* už se znovu nenaplánuje.

Formální znázornění pomocí konečného automatu:



2.5.3.2 Přemístění

Akce *přemístění* slouží k tomu, aby *jednotky* během simulace mohli změnit svoji polohu. Tuto akci dokážou provést všechny *pohyblivé jednotky* (vozidla, letadla a střely). Všechny *jednotky* se o směru pohybu rozhodují reaktivně. *Jednotky* si neplánují trasu. Při zadání příkazu přemístit *jednotku* se naplánuje událost *provedení akce jednotky* a zapamatují se souřadnice cíle. V každé fázi akce *přemístění* se vypočítává směr pohybu a *jednotka* se posouvá tímto směrem o určitý dílek vzdálenosti. Po každém posunutí *jednotky* se událost *provedení akce jednotky* znovu naplánuje dokud se *jednotka* nepřemístí k cíli. *Jednotka* se k cíli pohybuje přímým směrem dokud nenarazí na *překážku*. Při naražení na *překážku* ji jednotka objede a potom dále pokračuje v přímém směru k cíli.

Překážka

Každý typ podvozku pohyblivé jednotky má nadefinováno po jakých druzích povrchů se může v *prostředí* pohybovat. *Překážkou* se rozumí místo v *prostředí* s typem povrchu, po kterém se daná jednotka nemůže pohybovat. Další možnou *překážkou* může být místo, kde se nachází jiná *hmotná jednotka*. Pak je poloha *překážky* určena obdélníkovým výřezem tvořeným souřadnicemi a velikostí této *hmotné jednotky*.

Volné místo - místo, které není *překážka*

Objíždění překážky

Každá *pohyblivá jednotka* má okolo sebe umístěno několik pomyslných kontrolních bodů rozmístěných do kruhu o průměru šířky jednotky. V těchto bodech jednotka zjišťuje, zda na jejich pozicích je *volné místo* či *překážka*. Na objíždění *překážky* se používají 2 algoritmy. První algoritmus vyhodnotí, pod jakým úhlem by se musel vydat, aby *překážku* objel zleva a zprava a rozhodne se pro stranu s menší odchylkou úhlu. Úhel objetí *překážky* zjistí

pomocí kontrolních bodů. Dotáže se v každém bodě, zda je v něm volné místo nebo *překážka* a úhel objetí určí pomocí polohy sebe, cíle a nějakého kontrolního bodu, ve kterém je volné místo. Ze všech možných směrů si vybírá ten, který má nejmenší odchylku od směru k cíli. Tento algoritmus si nic nepamatuje a proto se může stát, že při objíždění *překážky* složitějšího tvaru se v půlce objíždění rozhone *překážku* objet druhým směrem a následně se tak zacyklit. K tomu slouží druhý algoritmus, který si pamatuje stranu objíždění a snaží se objet *překážku* stále stejným směrem, přestože by se mohlo zdát objetí z druhé strany ideálnější. Používá stejné kontrolní body jako v 1. algoritmu a výsledný směr určí nejlevějším či nejpravějším volným bodem od směru *překážky*. Pro výběr správného algoritmu se jednotka rozhoduje podle posledního směru pohybu. Pokud se směr posledního pohybu výrazně neliší od směru cíle, pak se přepne do prvního algoritmu. Pokud se směr posledního pohybu výrazně liší od směru cíle, pak se přepne do druhého algoritmu. Jinak algoritmus nemění.

Rychlost

Rychlost je maximální změna souřadnic *pohyblivé jednotky* během zpracování jedné události *provedení akce jednotky*. Pro každý typ podvozku a typ terénu je definován koeficient, který určuje s jakou výhodou či penalizací se může jednotka s tímto typem podvozku po tomto typu terénu pohybovat. *Rychlost* jednotky po určitém povrchu je určena hodnotou rychlosti jednotky vynásobenou koeficientem definovaným typem svého podvozku a povrchu, na kterém daná jednotka stojí.

2.5.3.3 Útočení

Útočení je jediný způsob, jakým lze zničit *nepřátelskou jednotku*. Všechny *útočné jednotky* (vozidla a letadla) jsou schopny akce *útočení*. Tato akce se provádí automaticky bez zadání příkazu *hráčem*. Po provedení *útočení* se tato akce znovu naplánuje v události *provedení akce jednotky* dokud existuje *útočná jednotka*. Akce *útočení* je rozděleno do 2 fází:

- čekání na výstřel
- výstřel

Čekání na výstřel

Během této fáze se v každém provedení akce *útočení* zmenší čítač doby výstřelu (celé kladné číslo) o 1 dokud není roven 0. Pokud je čítač doby výstřelu roven 0, pak se zjišťuje zda se nevyskytuje *nepřátelská jednotka* ve vzdálenosti dostřelu *útočné jednotky*. Pokud se vyskytuje nepřátelská jednotka v dostřelu *jednotky* a čítač doby výstřelu je roven 0, pak nastává fáze *výstřel*.

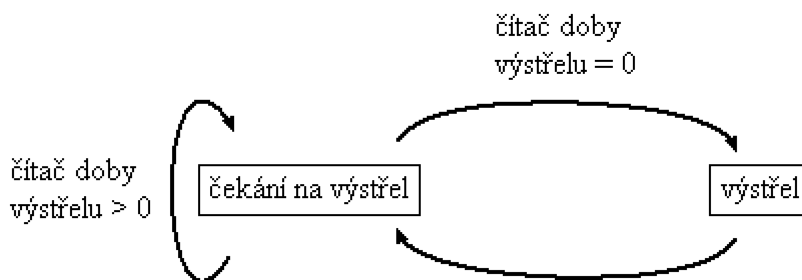
Výstřel

Při vykonávání fáze *výstřel* se nastaví čítač doby výstřelu na interval střelby *útočné jednotky*. Dále se vytvoří *pohyblivá jednotka* reprezentující střelu a zadá se jí příkaz *přemístit* na cíl (*jednotka* na kterou se útočí). Střela se přemísťuje k cíli standardním způsobem a po dosažení cíle se vypočítá *výsledné poškození*, to se přičte k *aktuálnímu poškození* cíle a pak střela zanikne. Pokud aktuální poškození dosáhne maximálního stupně poškození, pak daná *jednotka* zaniká. Po fázi *výstřel* okamžitě opět nastává fáze *čekání na výstřel*.

Výpočet výsledného poškození

Pro každý typ obrany *živé jednotky* a typ útoku *útočné jednotky* je definovaný koeficient, který určuje, jak je daný typ útoku výhodný proti dané obraně. Výsledné poškození je rovno rozdílu mohutnosti obrany a síly útoku vynásobené tímto koeficientem.

Formální znázornění pomocí konečného automatu:



2.5.3.4 Těžba surovin

Těžba surovin je způsob, jak *hráč* může získat suroviny potřebné k vyrábění *jednotek*. Tuto akci můžou provádět jen *těžící jednotky*. *Těžící jednotky* můžou natěžit suroviny jen ze *surovinového dolu*. Po natěžení surovin se suroviny odnáší do určité budovy (*základní budova*). Při zadání příkazu k těžbě surovin se naplánuje událost *provedení akce jednotky,* ve které se bude provádět akce *těžba surovin*. Po každém provedení *těžby surovin* se tato akce znovu naplánuje do události *provedení akce jednotky* dokud se jednotce nezadá jiný příkaz.

Těžící jednotka

Těžící jednotka je *pohyblivá jednotka*, která je schopna provádět akci *těžba surovin*. Je rozšířena o atribut náklad (celé kladné číslo) udávající počet surovin, které aktuálně nese.

Surovinový důl

Surovinový důl je *neživá jednotka*, v kterém může *těžící jednotka* natěžit suroviny.

Základní budova

Základní budova je *nepohyblivá jednotka* (budova), do které se donášejí suroviny.

Akce *těžba surovin* je rozdělena do 4 fází:

- přemístění k surovinovému dolu
- těžba v surovinovém dole
- přemístění k základní budově
- vyložení surovin

Přemístění k surovinovému dolu

Během této fáze se *jednotka* přemisťuje standardním způsobem k *surovinovému dolu*. Až se *jednotka* přemístí k *surovinovému dolu*, pak se nastaví čítač doby těžby (celé kladné číslo) na hodnotu určující dobu těžení v *surovinovém dole* a nastane fáze *těžba v surovinovém dole*.

Těžba v surovinovém dole

V každém provedení fáze *těžby v surovinovém dole* se zmenšuje čítač doby těžby o 1 dokud není roven 0. Pokud je čítač doby těžby roven 0, pak se *těžící jednotce* nastaví atribut náklad na počet natěžených surovin. Po nastavení nákladu nastává fáze *přemístění k základní budově*.

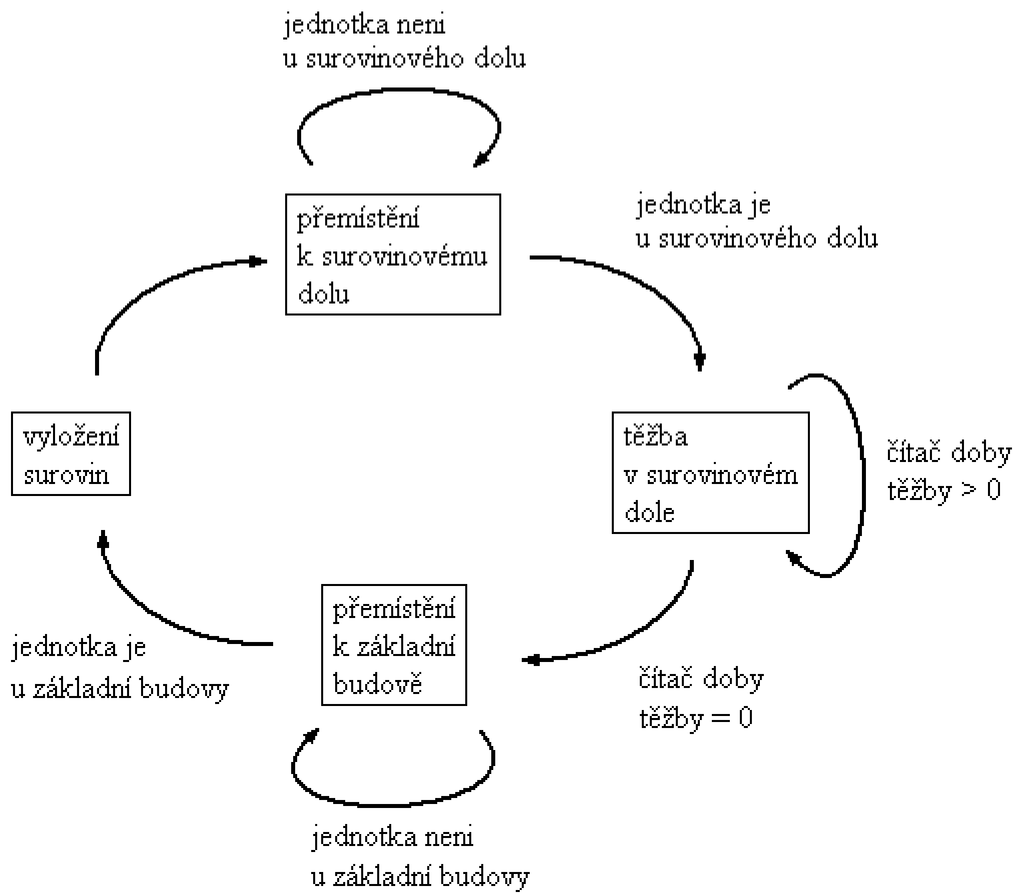
Přemístění k základní budově

Během této fáze se *jednotka* přemisťuje standardním způsobem k *základní budově*. Až se *jednotka* přemístí k *základní budově*, pak nastane fáze *vyložení surovin*.

Vyložení surovin

Při vykládání surovin se u vlastnícího *hráče* zvětší počet natěžených surovin o náklad a proměnná náklad se vynuluje. Po provedení této akce nastane opět fáze *přemístění k surovinovému dolu*.

Formální znázornění pomocí konečného automatu:



Kapitola 3

Umělá inteligence

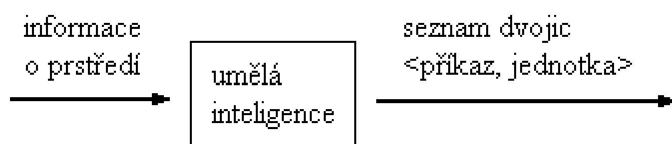
Průběh simulace je ovlivňován příkazy *hráčů*. Úkolem každého *hráče* je zadávat příkazy *jednotkám* tak, aby daný *hráč* zničil všechny *nepřátelské jednotky*. V této kapitole se specifikuje umělá inteligence řešící problém, jak dosáhnout splnění úkolu *hráče*. *Hráč* může zadávat příkazy:

- přemístění pohyblivé jednotky na určitou pozici
- vyrobit jednotku
- těžit suroviny

Útoky *jednotek* nejsou zadané *hráčem*. *Jednotky* útočí automaticky, pokud se v blízkosti nachází *nepřátelská jednotka*.

3.1 Popis úlohy

Cílem úlohy je v každém časovém okamžiku se rozhodnout, jaké příkazy zadat *jednotkám*, aby *umělá inteligence* dosáhla zničení všech *nepřátelských jednotek*. Úloha spočívá v tom, jak z informací o *prostředí* v každé situaci vygenerovat seznam příkazů *jednotkám*. Položky výstupního seznamu jsou reprezentovány dvojicí <příkaz, *jednotka*>. Po vygenerování výstupního seznamu se každý příkaz v tomto seznamu zadá *jednotce* z příslušné dvojice. Celá úloha se řeší během události *provedení příkazů umělé inteligence*. Výpočet rozhodování umělé inteligence je prováděn v rámci provádění příslušné události v simulaci.



3.2 Řešení úlohy

V důsledku zjednodušení chování *umělé inteligence* jsem se rozhodl rozdělit úlohu na 3 nezávislé podúlohy:

- těžba surovin
- výroba jednotek
- útok jednotek

Každá podúloha je řešena samostatně. Vstupy každé podúlohy jsou informace z *prostředí*. Výsledky všech 3 podúloh jsou seznamy dvojic <příkaz,*jednotka*>. Výsledný seznam vznikne sjednocením všech tří podseznamů.

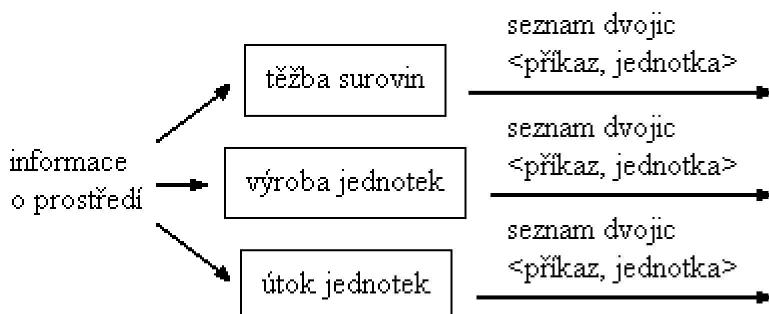
výhody:

- Řešit nezávislé podúlohy zvlášť je mnohem jednodušší než řešit celou úlohu najednou.

nevýhody:

- Správná řešení všech tří podúloh nemusí vést k optimálnímu řešení globální úlohy.

Grafické znázornění rozdělení úlohy:



3.3 Požadavky pro řešení podúloh

Řešení každé podúlohy by mělo splňovat:

- rychlé

  Během výpočtu rozhodování *umělé inteligence* se nesmí simulace zastavit na dlouhou dobu, aby se zachovala hratelnost hry.

- reaktivní

  V situaci, kdy se útočí nebo vyrábí mnoho *jednotek*, se počty *jednotek* často mění a je potřeba na tyto změny včas zareagovat..

- snadno reprezentovatelné (reprezentace *umělé inteligence* by měla být přehledná, bez zbytečně složitých struktur a se snadno dohledatelnými informacemi)

  Chtěl bych porovnat vliv několika modifikací na umělé inteligenci a proto potřebuji reprezentaci, která bude pro tyto účely snadno editovatelá.

V rámci zjednodušení jsem se rozhodl řešit dvě ze tří podúloh ad-hoc heuristikou.

3.4 Těžba surovin

K výrobě *jednotek* musí mít hráč dostatečný počet surovin. *Hráč* může získat suroviny pouze těžbou ze *surovinového dolu*. Pokud *hráč* těží suroviny rychleji než jeho protivník, pak má oproti němu velkou výhodu ve výrobě a počtu *jednotek*.

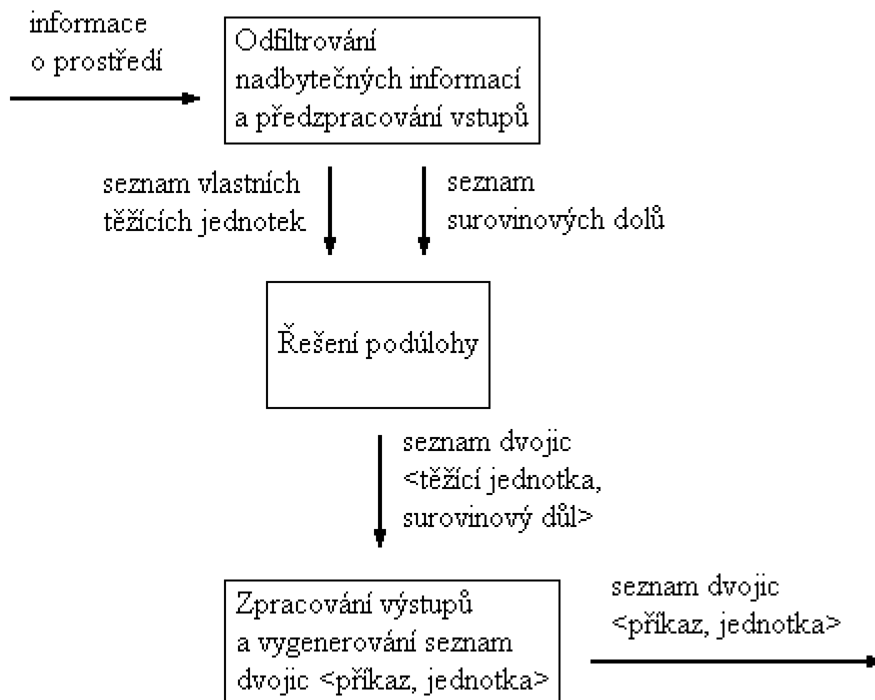
3.4.1 Popis podúlohy

Cílem podúlohy je v každém okamžiku simulace pro každou svoji *těžící jednotku* určit, zda bude těžit a v jakém *surovinovém dole*. Tato podúloha spočívá v tom, jak z informací o *prostředí* vygenerovat seznam dvojic <*těžící jednotka*, *surovinový důl*>. Vstupem je seznam svých *těžících jednotek* a seznam všech *surovinových dolů*. *Surovinové doly* z výstupního seznamu určují doly, v kterých začnou těžit *jednotky* z příslušných dvojic.

Podúloha je rozdělena do 3 fází:

- Odfiltrování nadbytečných informací a předzpracování vstupů
- Řešení podúlohy
- Zpracování výstupů a vygenerování seznamu dvojic <příkaz, jednotka>

Grafické znázornění rozdělení podúlohy:



3.4.2 Odfiltrování nadbytečných informací a předzpracování vstupů

Z informací o *prostředí* se vybere pouze seznam vlastních *těžících jednotek* a seznam *surovinových dolů*. Ke správnému rozhodnutí těžby surovin musí mít *umělá inteligence* informace, kde se nachází *surovinové doly* a jaké má prostředky k těžbě surovin.

3.4.3 Řešení podúlohy

Rozhodl jsem se řešit podúlohu *těžby surovin* prostřednictvím ad-hoc heuristiky. Ad-hoc heuristika mi připadá plně dostačující pro tuto podúlohu. Podle mého názoru by použití metod umělé inteligence nepřineslo znatelně lepší řešení. Navíc implementace metod umělé inteligence by byla značně časově náročnější a komplikovanější. Podúloha je řešena algoritmem, který

vytvoří výstupní seznam ze všech *jednotek*, které jsou schopny těžit a momentálně nevykonávají jakoukoliv akci. *Surovinový důl* pro každou dvojici určí nejbližším *surovinovým dolem* od *jednotky* z příslušné dvojice.

3.4.4 Zpracování výstupů a vygenerování seznamu dvojic <příkaz, jednotka>

Pro každou dvojici <*těžící jednotka*, *surovinový důl*> z výstupního seznamu se vygeneruje dvojice <příkaz, *jednotka*>. Příkaz je roven těžbě surovin ze *surovinového dolu* a *jednotka* je rovna *těžící jednotce* z příslušné dvojice.

3.5 Výroba jednotek

Jednotky vyskytující se v simulaci se výrazně liší tím, jaké akce jsou schopny vykonat. *Jednotky* můžou útočit, stavět, vyrábět atd. Aby byl *hráč* schopný vykonávat tyto akce, potřebuje k tomu různé typy *jednotek*. V každé situaci simulace musí být schopný určit, jaké typy *jednotek* bude vyrábět. Pokud bude vyrábět převážně *jednotky* jednoho typu, pak bude nad protihráčem zaostávat v ostatních oblastech.

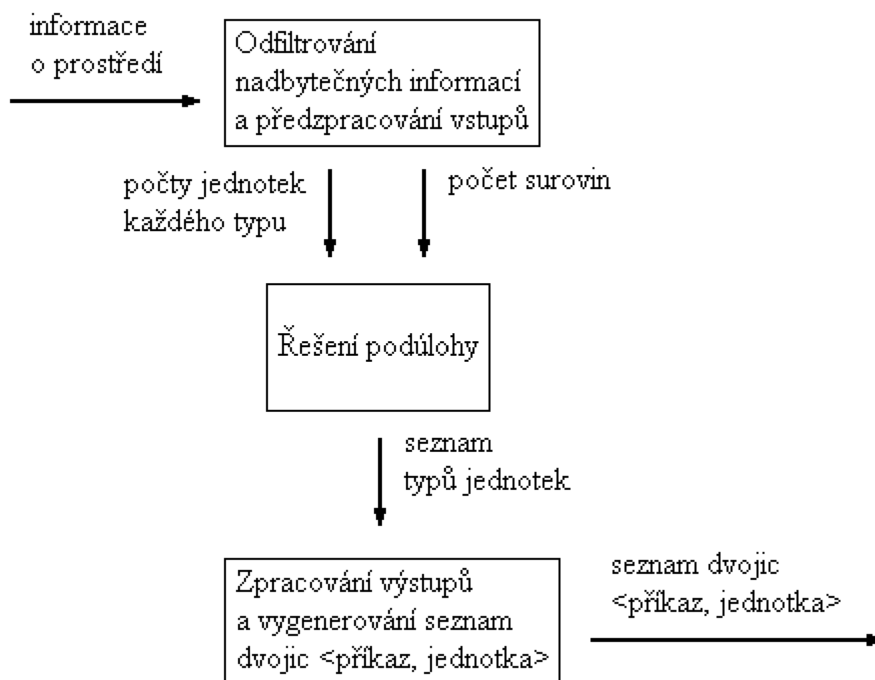
3.5.1 Popis podúlohy

Cílem podúlohy je v každém časovém okamžiku se rozhodnout, jaké *jednotky* vyrobit, aby *umělá inteligence* byla dostatečně bojeschopná a technologicky nezaostávala nad protihráčem. V této podúloze se řeší jak z informací o *prostředí* vytvořit seznam typů *jednotek*. Výstupní seznam určuje, jaké typy *jednotek* se začnou vyrábět během události *provedení příkazů umělé inteligence*.

Podúloha je rozdělena do 3 fází:

- Odfiltrování nadbytečných informací a předzpracování vstupů
- Řešení podúlohy
- Zpracování výstupů a vygenerování seznamu dvojic <příkaz, jednotka>

Grafické znázornění rozdělení podúlohy:



3.5.2 Odfiltrování nadbytečných informací a předzpracování vstupů

Ze všech informací o *prostředí* se pro tuto podúlohu vyberou pouze počty *jednotek* od každého typu a počet natěžených surovin. *Umělá inteligence* musí mít informace o počtu svých jednotek, aby byla schopna správně určit, jaké jednotky je potřeba v dané situaci vyrobit. Počet surovin je důležitý, aby *umělá inteligence* mohla rozhodnout, kolik *jednotek* je schopna vyrobit.

3.5.3 Řešení podúlohy

Podúloha *výroby jednotek* mi připadá nejdůležitější ze všech tří podúloh. Myslím si, že vymýšlet dostatečně dobrou ad-hoc heuristiku pro tuto podúlohu by bylo příliš složité. Z tohoto důvodu jsem se rozhodl podúlohu *výroby jednotek* řešit prostřednictvím metod umělé inteligence. Navíc se domnívám, že prostřednictvím metod umělé inteligence dosáhnu výrazně lepšího řešení než ad-hoc heuristikou.

Z metod umělé inteligence se nabízejí možnosti:

- prohledávání stavového prostoru
- pravidlové expertní systémy
- Bayesovské sítě
- neuronové sítě

Myslím si, že vyzkoušení všech metod by zabralo příliš času a přesáhl bych rámec bakalářské práce. Na základě tohoto názoru jsem se rozhodl vybrat pouze jednu možnost. Z těchto možností jsem si vybral **neuronové sítě** (viz kapitola 3.6). Připadají mi pro danou podúlohu nejvhodnější. *Neuronové sítě* jsou rychlé, reaktivní a snadno reprezentovatelné. Doba výpočtu *neuronových sítí* lze snadno odhadnout z počtu *neuronů*. *Neuronová síť* má stejnou odezvu na všechna vstupní data (i v nejhorším případě zaručuje odhadnutelně dlouhou odezvu). Z tohoto důvodu *neuronové sítě* nezpůsobí neplynulost hry.

Nevýhody ostatních možností:

- prohledávání stavového prostoru

  U velkých stavových prostorů je prohledávání příliš dlouhé. Existují sice optimalizované algoritmy jako např. A* (více v [9]), ale pro efektivní běh potřebují dobré heuristiky. Tyto heuristiky by bylo obtížné vytvořit a navíc nezaručují konstantní odezvu.

- pravidlové expertní systémy

  Nejsou k dispozici odvozovací pravidla a jejich vymýšlení by bylo příliš komplikované.

- Bayesovské sítě

  K učení Bayesovských sítí je potřeba mít vzorová data. Abych měl vzorová data, musel bych znát pro několik vybraných vzorových situací optimální nebo alespoň suboptimální rozhodnutí. Protože neznám ani suboptimální rozhodnutí, nejsem schopen Bayesovskou sít zaučit. Navíc *neuronové sítě* v určitých případech mohou dosáhnout rychlejšího ohodnocení vzoru jak je popsáno v článku [8].

3.5.4 Zpracování výstupů a vygenerování seznamu dvojic <příkaz, jednotka>

Výsledkem podúlohy je seznam typů *jednotek*. Pro každý typ *jednotky* "J" z výsledného seznamu se zjistí, jaký typ *jednotky* "T" může vyrobit danou *jednotku* typu "J". Potom se náhodně vybere *jednotka* "t" typu "T", která momentálně nevykonává žádnou akci. Pokud taková jednotka "t" existuje, pak se vygeneruje dvojice <příkaz, *jednotka*>. Příkaz je roven vyrobit *jednotku* typu "J". *Jednotka,* které bude příkaz zadán je *jednotka* "t".

Pokud seznam typů *jednotek* obsahuje n stejných typů "J", pak se může vygeneruje až n dvojic <příkaz, *jednotka*> (pokud je dostatek *jednotek* typu "T", které momentálně nevykonávají žádnou akci). Příkazy jsou pro všechny dvojice totožné. *Jednotky,* kterým budou příkazy zadány jsou pokaždé jiné *jednotky* typu "T".

Pseudo-algoritmus:

S - Seznam typů jednotek
V - Seznam dvojic <příkaz, *jednotka*>

```
while seznam S není prázdný do
begin
        J  := odeber první položku ze seznamu S
        T := zjisti jaký typ jednotky může vyrobit jednotku typu J
        t  := najdi jednotku typu T která momentálně nevykonává žádnou akci
        if Existuje t then
                vlož do seznamu V dvojici (vyrábět jednotky typu J; t)
end
```

3.6 Útok jednotek

Aby *hráč* zničil všechny *nepřátelské jednotky* a zároveň zabránil zničení všech svých *jednotek* musí se rozhodnout kolika *jednotkami* se bude bránit a kolika *jednotkami* bude útočit. Dále se může rozhodnout střežit některé strategické pozice jako mosty.

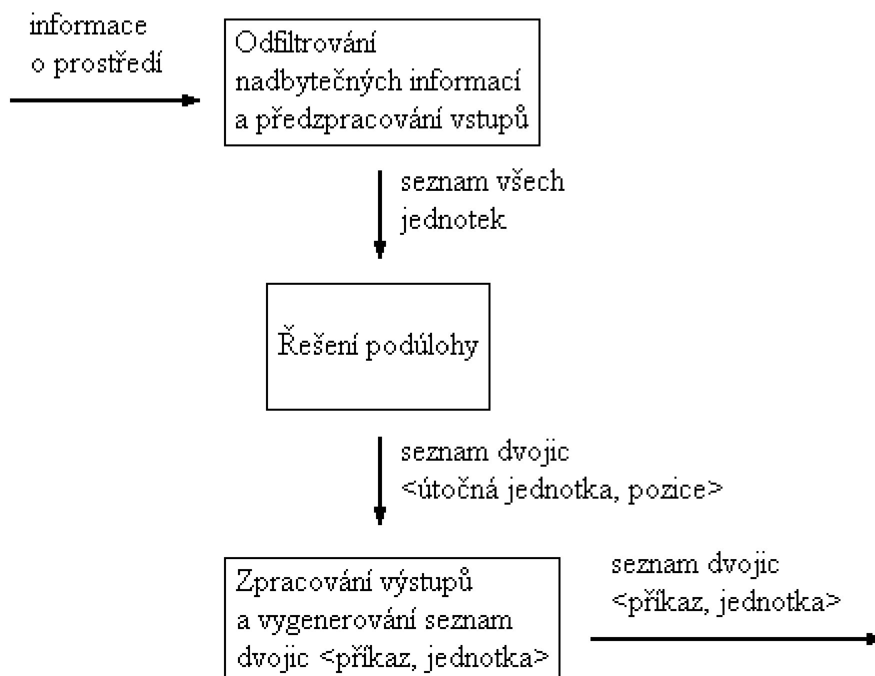
3.6.1 Popis podúlohy

Cílem podúlohy je v každém časovém okamžiku se rozhodnout, kolik a na jakou pozici poslat *útočné jednotky*, aby *umělá inteligence* dosáhla zničení všech *nepřátelských jednotek* a zabránila zničení vlastních *jednotek*. Úloha spočívá v tom, jak z informací o pozicích a typech vlastních

i nepřátelských *jednotek* vygenerovat seznam dvojic <*útočná jednotka*, pozice>. Položka z výstupního seznamu reprezentuje příkaz přemístění *jednotky* na pozici z příslušné dvojice. Podúloha je rozdělena do 3 fází:

- Odfiltrování nadbytečných informací a předzpracování vstupů
- Řešení podúlohy
- Zpracování výstupů a vygenerování seznamu dvojic <příkaz, jednotka>

Grafické znázornění rozdělení podúlohy:



3.6.2 Odfiltrování nadbytečných informací a předzpracování vstupů

Vstupní informace pro tuto podúlohu jsou základní informace (pozice, typ, vlastnící hráč) o všech *jednotkách* vyskytující se v *prostředí*. Všechny ostatní informace z *prostředí* jsou odfiltrovány. *Umělá inteligence* potřebuje informace o typech a pozicích svých a nepřátelských *jednotek*, aby byla schopna rozhodnout, jaké *jednotky* se mají přesunout na určitou pozici.

3.6.3 Řešení podúlohy

V rámci zjednodušení jsem se rozhodl řešit tuto podúlohu ad-hoc heuristikou. Podle mého názoru je ad-hoc heuristika dostatečná pro řešení podúlohy *útoku jednotek*. Domnívám se, že implementace řešení této podúlohy pomocí metod umělé inteligence by zabrala příliš času a přesáhla by rámec bakalářské práce. Navíc si myslím, že tato podúloha není tak důležitá jako podúloha *výroby jednotek*. Podúloha *útoku jednotek* se řeší jednoduchým algoritmem. Všechny *útočné jednotky*, které jsou pod správou *umělé inteligence*, jsou označeny buď jako defenzivní nebo jako ofenzivní. Po vyrobení *útočné jednotky* se *jednotka* automaticky označí jako defenzivní. Během jedné události *provedení příkazů umělé inteligence* algoritmus vykoná postupně tyto kroky:

- převedení několika ofenzivních jednotek na defenzivní
- převedení několika defenzivních jednotek na ofenzivní
- bránění defenzivními jednotkami
- útok ofenzivními jednotkami

ofenzivní jednotka - *útočná jednotka*, která je označena jako ofenzivní

defenzivní jednotka - *útočná jednotka*, která je označena jako defenzivní

ofenzivní i *defenzivní jednotka* je relativní pojem vzhledem k *umělé inteligenci*, která má tyto jednotky pod svoji správou.

3.6.3.1 Převedení několika ofenzivních jednotek na defenzivní

Počet *defenzivních jednotek* ku počtu *útočných jednotek* by měl být vyšší než pevně zvolené N z <0,1>. Pokud tato podmínka neplatí pak se přeznačují náhodně zvolené *ofenzivní jednotky* na *defenzivní jednotky* dokud nezačne podmínka platit. Tento krok je důležitý proto, aby *umělá inteligence* měla pod správou dostatek *defenzivních jednotek* potřebných k bránění své základny.

3.6.3.2 Převedení několika defenzivních jednotek na ofenzivní

Během tohoto kroku se snižuje čítač intervalu útoku (celé číslo) dokud není roven 0. Pokud je čítač intervalu útoku roven 0, pak se nastaví čítač na hodnotu intervalu útoku a přeznačují se náhodně zvolené *defenzivní jednotky* na *ofenzivní jednotky* dokud ještě platí podmínka o poměru *defenzivních* a *útočných jednotek*. Tento krok slouží k tomu, aby *umělá inteligence* měla v době útoku pod správou dostatek *ofenzivních jednotek*. Přeznačování na *ofenzivní jednotky* je prováděno jednou za určitou dobu.

3.6.3.3 Bránění defenzivními jednotkami

V tomto kroku probíhá bránění základny. Během bránění se zjišťuje, zda se v určité vzdálenosti od základny nevyskytuje *nepřátelská jednotka*.Pokud ano, pak se vygeneruje seznam se všemi *defenzivními jednotkami* a pozicí určenou souřadnicemi této *nepřátelské jednotky*.

3.6.3.4 Útok ofenzivními jednotkami

Pokud *umělá inteligence* vlastní některé *ofenzivní jednotky*, pak s nimi útočí na náhodně zvolenou nepřátelskou budovu. Během tohoto kroku vygeneruje seznam všech *ofenzivních jednotek* a pozicí určenou souřadnicemi náhodně zvolené nepřátelské budovy.

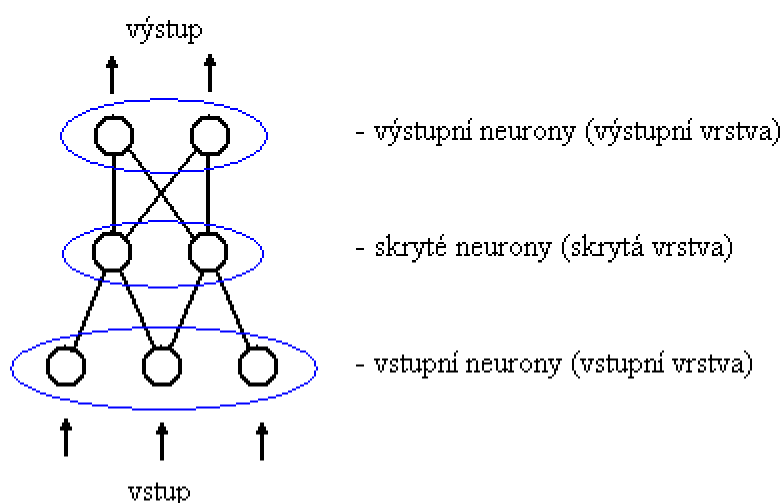
3.6.4 Zpracování výstupů a vygenerování seznamu dvojic <příkaz, jednotka>

Pro každou dvojici <*útočná jednotka*, pozice> z výstupního seznamu se vygeneruje dvojice <příkaz, *jednotka*>. Příkaz je roven přemístění na danou pozici a *jednotka* je rovna *útočné jednotce*.

3.6 Popis neuronové sítě použité v rozhodování o výrobě jednotek

Následující kapitola bude popisovat základní principy *neuronových sítí* (dále NS) jak je uvedeno v [3]. NS je struktura několika vzájemně propojených **neuronů** (viz kapitola 3.6.1). Každý spoj má nadefinovanou svoji synaptickou váhu, která určuje "sílu" spoje. *Neurony* jsou v síti uspořádány do několika vrstev. Vrstvy NS se rozdělují na vstupní, skryté a výstupní. Podle toho, v jaké vrstvě se vyskytuje *neuron*, ho nazýváme vstupním, skrytým či výstupním *neuronem*.

Příklad *neuronové sítě*:



3.6.1 Neuron

Neuron je základní element NS. Každý *neuron* má několik vstupů a jen jeden výstup. *Neurony* jsou mezi sebou propojeny tak, že vstupy skrytých a výstupních *neuronů* je napojeny na výstupy některých vstupních a skrytých *neuronů*. Výstupní hodnota *neuronu* je určena součtem součinů hodnot spojených *neuronů* a synaptických vah spojů, k tomuto výsledku se přičte práh *neuronu* a dosadí se do přenosové funkce. Formální vzoreček:

$$Y = S(\sum_{i=1}^{N}(w_i x_i) + \Theta)$$

x_i jsou vstupy *neuronu*

w_i jsou synaptické váhy

Θ je práh

$S(x)$ je přenosová funkce *neuronu* (někdy aktivační funkce)

Y je výstupní hodnota *neuronu*

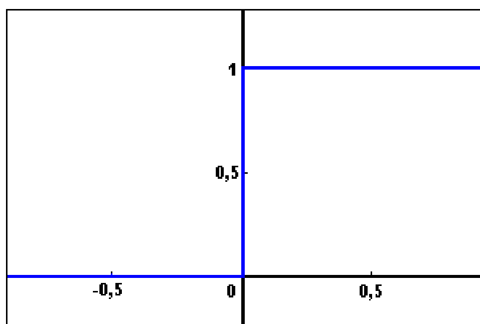
3.6.2 Učení neuronových sítí

Učení NS je proces nastavení synaptických vah a prahů *neuronů* tak, aby NS vracela správný výsledek pro odpovídající vstup. Učení NS se rozděluje na učení s učitelem a učení bez učitele. U učení NS s učitelem se porovnávají výstup NS s požadovaným výstupem (výstup učitele). Parametry NS se nastavují tak, aby tento rozdíl byl co nejmenší. U učení bez učitele NS dostane vstupní vzory s určitými vlastnostmi. Parametry NS se nastaví tak, aby rozdíl výstupů vzorů s podobnými vlastnostmi byl co nejmenší.

3.6.3 Přenosové funkce:

Často používané přenosové funkce jsou skoková funkce, funkce radiální báze, sigmoidální funkce a funkce hyperbolické tangenty.

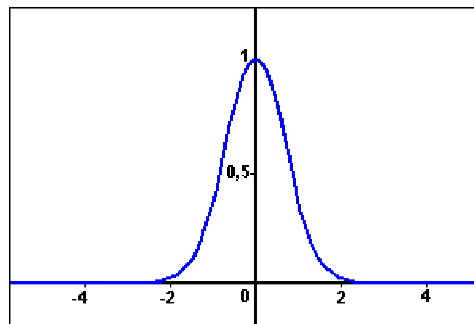
Skoková přenosová funkce



$$f(x) = 0 \quad \text{pro} \quad x < 0$$

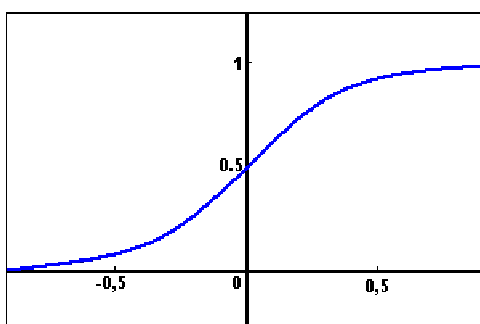
$$f(x) = 1 \quad \text{pro} \quad x \geq 0$$

Přenosová funkce radiální báze



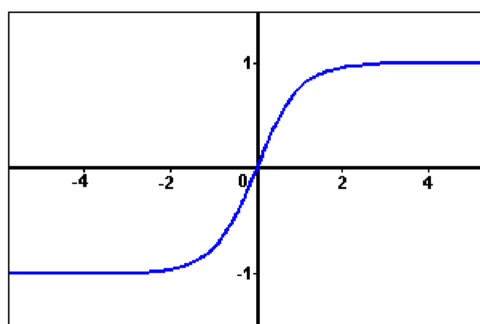
$$f(x) = e^{-kx^2}$$

Sigmoidální přenosová funkce



$$f(x) = \frac{1}{1 + e^{-kx}}$$

Přenosová funkce hyperbolické tangenty



$$f(x) = \frac{2}{1 + e^{-kx}} - 1$$

3.6.4 Architektura neuronových sítí

Architektura NS popisuje, jak jsou *neurony* uspořádány, jak jsou mezi sebou propojeny a kolik obsahují vrstev. NS je n-vrstvá, jestliže počet skrytých a výstupních vrstev je rovný n. Síť je dále charakterizovaná počty *neuronů* v jednotlivých vrstvách a přenosovými funkcemi.

3.6.5 Použití neuronové sítě

NS je použita pro řešení podúlohy *výroby jednotek*. Vstupní informace podúlohy jsou počty *jednotek* od každého typu a počet natěžených surovin. Výstupem podúlohy je seznam typů *jednotek* určující, jaké *jednotky* se budou v daném okamžiku vyrábět. Základním rysem NS je, že počty vstupních i výstupních *neuronů* musejí být pevné délky. Počet typů *jednotek* vyskytující se v simulaci je konečné číslo, tudíž není potřeba nijak předzpracovávat vstupní informace pro NS. Výstupní seznam typů *jednotek* může být libovolně dlouhý. Výstupem NS jsou priority stavby *jednotky* od každého typu a počet *jednotek*, který se začne během toho kroku vyrábět. Výstup NS je potřeba převést do formátu výstupu podúlohy.

3.6.6 Použitá architektura

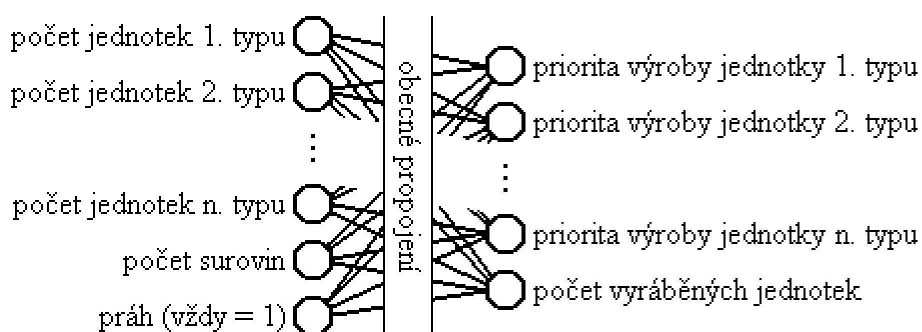
Pro podúlohu *výroby jednotek* jsem použil **dopřednou neuronovou síť**. *Dopředná neuronová síť* je jedna z nejjednodušších architekturou NS a často používaná pro jednoduché problémy. V *dopředné neuronové síti* jsou *neurony* propojeny tak, že výstup *neuronu* je poskytován pouze *neuronům* na vyšších vrstvách. Velkou výhodou *dopředných sítí* je, že se snadno a dobře zaučují. Všechny hodnoty *neuronů* a synaptických vah jsou reálná čísla z rozsahu <-1;1>. Přenosovou funkci jsem zvolil funkci radiání báze. Na základě mých experimentů (viz kapitola 4.5) ji považuji za nejvhodnější přenosovou funkci.

3.6.7 Popis vstupních a výstupních neuronů

Neuronová síť obsahuje *neurony* typu:

- neuron počtu jednotek
- neuron počtu surovin
- neuron priority jednotky
- neuron počtu vyráběných jednotek
- neuron prahu

Grafické znázornění zapojení *neuronové sítě*:



3.6.7.1 Neurony počtů jednotek

Pro každý typ *jednotky* NS obsahuje jeden vstupní *neuron* tohoto typu. Tyto *neurony* určují počet vlastněných *jednotek* daného typu. NS tak získává informace o aktuálním počtu *jednotek* každého typu. Abych stlačil počet jednotek do rozsahu <-1;1>, pak hodnotu *neuronu* nastavuji na počtu jednotek daného typu /10. Pokud je počet *jednotek* daného typu větší než 10, pak je hodnota *neuronu* rovná 1.

3.6.7.2 Neuron počtu surovin

NS obsahuje jeden vstupní *neuron počtu surovin*, který určuje počet surovin natěžených *umělou inteligencí*. Počet surovin se pohybuje v řádech stovek až tisíců. Aby vstupní hodnota nepřesahovala rozsah <-1;1>, tak se hodnota *neuronu* rovná počtu surovin /1000. Pokud je počet natěžených surovin větší než 1000, pak je hodnota *neuronu* rovna 1.

3.6.7.3 Neurony priority výroby jednotky

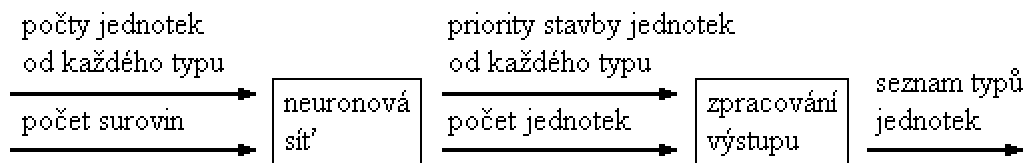
Pro každý typ *jednotky* NS vlastní jeden výstupní *neuron* tohoto typu. Tyto *neurony* určují prioritu výroby *jednotky* daného typu. Priority určují jaké typy *jednotek* se budou během tohoto kroku vyrábět. Pokud je priorita záporná, pak se *jednotka* daného typu nevyrábí. Přesnější popis určení výroby typů *jednotek* podle priorit je popsán v kapitole 3.6.8.

3.6.7.4 Neuron počtu vyráběných jednotek

NS obsahuje jeden výstupní *neuron* tohoto typu. Jeho hodnota určuje kolik *jednotek* se během daného kroku začne vyrábět. Protože hodnota neuronu je z rozsahu <-1;1> a počet vyráběných *jednotek* musí být celé kladné číslo, pak se výsledná hodnota *neuronu* vynásobí deseti a zaokrouhlí se na celé číslo. Pokud je hodnota *neuronu* záporná, pak je výsledná hodnota rovna 0.

3.6.7.5 Neuron prahu

Prahy všech *neuronů* jsou reprezentované vstupním *neuronem prahu*. *Neuron prahu* je propojený se všemi ostatními *neurony* a jeho hodnota je vždy rovna 1. Hodnota prahu každého *neuronu* je určena synaptickou vahou spoje s tímto *neuronem*.



3.6.8 Zpracování výstupu

Výsledný seznam typů *jednotek* vznikne tak, že se do něj postupně přidávají typy *jednotek* s nejvyšší prioritou. Po každém přidání se priorita daného typu zmenší na polovinu. Do seznamu se přidávají typy *jednotek* s nejvyšší prioritou dokud velikost seznamu není rovna výslednému počtu *jednotek*, který se má začít během tohoto kroku vyrábět.)

Pseudo-algoritmus:

```
P[J] - priorita výroby jednotky typu J
Vpj - výsledný počet jednotek
S    - seznam typů jednotek

i := 0
while i < Vpj do
begin
        J  := najdi typ jednotky s nejvyšší prioritou P
        vlož do seznamu S typ jednotky J
        P[J] := P[J] / 2
        i := i +1
end
```

3.6.9 Nastavení parametrů neuronové sítě

Chování NS je určeno hodnotami synaptických vah. Potřebuji nastavit synaptické váhy tak, aby pro každou situaci NS vracela výstup vedoucí ke správnému řešení podúlohy *výroby jednotek*. Nabízejí se možnosti:

- pevné nastavení
- učení bez učitele
- učení s učitelem

Pevné nastavení parametrů je u většiny NS velmi obtížné a složité. Při učení bez učitele NS rozdělí prostor vstupních dat do několika shluků. Protože pro moji úlohu by bylo obtížné vymyslet vztah shluků a řízení, rozhodl jsem se vyloučit tuto možnost. Každou NS jsem schopný ohodnotit (viz kapitola 3.6.10), proto jsem se rozhodl využít tuto zpětnou vazbu a vybral jsem si učení s učitelem.

3.6.10 Proces učení

K učení NS nemám k dispozici vzorová data s požadovaným výstupem. Abych měl vzorová data s požadovaným výstupem, musel bych znát pro několik vybraných vzorových situací optimální nebo alespoň suboptimální rozhodnutí. I se suboptimálním rozhodnutí by byla NS schopna se něco naučit. Protože neznám ani suboptimální rozhodnutí, nemůžu měřit kvalitu NS tak, že na vzorových datech porovnám výstup NS s požadovaným výstupem. Z tohoto důvodu nemohu použít k učení NS algoritmy jako je například backpropagace, která učí NS tím, že se snaží minimalizovat tento rozdíl. V mém případě jsem schopen zjistit kvalitu NS ohodnocením výsledného stavu strategické hry po nasimulování několika rozhodnutí dané NS. Všechny NS ohodnocuji stejným globálním kritériem, abych bych schopný NS srovnávat mezi sebou. Na základě této zpětné vazby jsem schopen učit NS například pomocí metody simulovaného žíhání (více v [10]). V případě použití metody simulovaného žíhání by se stavový prostor parametrů NS prohledával tak, že by se v každém kroku náhodně pozměnily parametry NS a pokud by toto pozměnění vedlo ke zlepšení kvality, pak by se toto pozměnění přijalo. Pokud pozměnění nevede ke zlepšení kvality, pak toto pozměnění může být přesto přijato s určitou pravděpodobností. Tento algoritmus pracuje pouze s jednou NS, kterou postupně pozměňuje. Myslím si, že je lepší tento postup rozšířit tak, aby pracoval se skupinou NS. Potom nové NS může generovat křížením několika nejúspěšnějších NS ze skupiny. Na tomto principu fungují **genetické algoritmy** (viz kapitola 3.7), které používám k nastavení parametrů NS.

3.7 Genetické algoritmy

Tato kapitola popisuje základní princip *genetických algoritmů*. Více na [4,5,6,7]. *Genetický algoritmus* (dále GA) je nedeterministický postup hledání řešení. Princip GA využívá poznatky z genetické evoluce v přírodě. GA se používají na řešení složitějších problémů, které nejsme schopni řešit pomocí běžných metod.

3.7.1 Podmínky aplikace

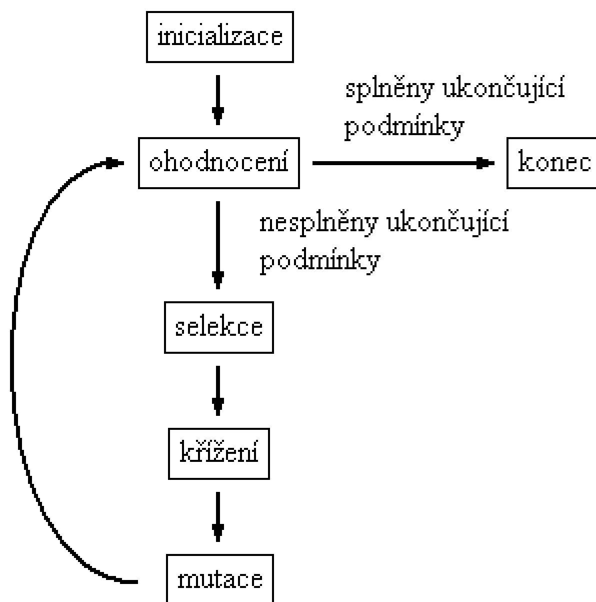
Abychom mohli problém řešit pomocí GA, pak musí splňovat několik podmínek:

- každé řešení musí jít převést do jednotné reprezentace (binárního kódu), ten nazýváme **genotyp**. Řešení reprezentované tímto *genotypem* nazýváme **fenotyp**.
- každé řešení musí jít ohodnotit alespoň dvoustavovou hodnotou. Čím více stavovou hodnotou dokážu ohodnotit řešení, tím plynuleji bude genetický vývoj konvergovat ke správnému řešení. Této ohodnocující hodnotě říkáme **fitness**. Ohodnocující funkci nazýváme **fitness fukcí**.

3.7.2 Princip genetických algoritmů

Princip GA spočívá v náhodném vygenerování několika *genotypů*, tuto skupinu *genotypů* nazýváme **populace**. Všechny řešení reprezentované těmito *genotypy* v *populaci* ohodnotíme *fitness funkcí* a vybereme několik *genotypů* s vysokou hodnotou *fitness*. Tomuto výběru říkáme **selekce**. Z vybraných *genotypů* pak vytvoříme novou *populaci* pomocí náhodného **křížení** a **mutace**. Celý postup několikrát opakujeme. Každou iteraci tohoto cyklu se nazýváme **generací**. Algoritmus většinou končí po dosažení dostatečného řešení nebo po předem zvolené době. Úspěšnost postupu závisí na konkrétním zvolení typu *selekce*, *křížení*, *mutace* a *fitness* funkce.

Grafické znázornění běhu algoritmu:



3.7.2.1 Ohodnocení

Ohodnocení je proces, který z *genotypu* vrací *fitness* hodnotu určující úspěšnost řešení reprezentované tímto *genotypem*. Tato hodnota je velmi důležitá, aby algoritmus *selekce* mohl správně určit jaká řešení vybere.

3.7.2.2 Selekce

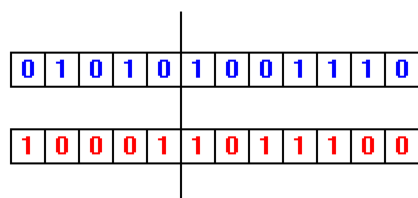
Algoritmus *selekce* určuje, jakým způsobem bude probíhat výběr *genotypů* do nové populace. Je důležitá proto, aby v každé další *generaci populace* obsahovala lepší *genotypy*.

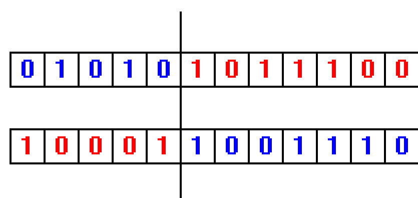
3.7.2.3 Křížení

Křížení je proces v kterém se zaměňují části *genotypů* mezi jedinci. *Křížení* je důležité proto, aby vznikly nové *genotypy*. Z hlediska, že *selekce* vybere *genotypy* s vysokou *fitness*, je velká pravděpodobnost, že kombinací těchto *genotypů* vznikne lepší *genotyp* (*genotyp* s vyšší *fitness*). U *genotypů* reprezentovaných binárním kódem rozdělujeme k*řížení* na *jednobodové křížení* a *vícebodové křížení*.

Grafické znázornění jednobodového křížení:

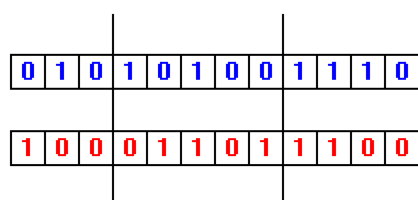
původní genotypy   výsledné genotypy

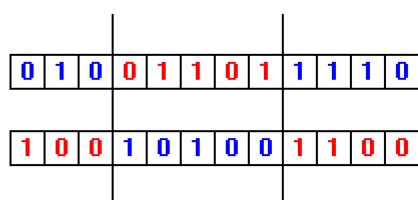




Grafické znázornění vícebodového křížení (na obrázku je znázorněné 2 bodové křížení):

původní genotypy   výsledné genotypy



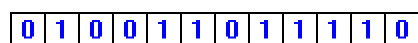


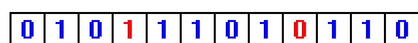
3.7.2.4 Mutace

Může nastat situace, kdy všechny *genotypy* v populaci mají na jednom konkrétním místě stejnou hodnotu. Pak po jakémkoliv *křížení* hodnota *genotypu* na daném místě zůstane vždy stejná. Pomocí *křížení* nejsme schopni vytvořit *genotyp*, který bude mít tuto hodnotu jinou než ostatní *genotypy*. Proto je nutné, aby po *křížení* nastala *mutace*. Při *mutaci* se náhodně změní náhodná část *genotypu*. *Mutace* jsou zpravidla prováděny s nízkou pravděpodobností, aby se *genotypy* výrazně nelišily od *genotypů* z předchozí *generace*.

Grafické znázornění mutace genotypu reprezentovaným binárním kódem:

původní genotyp   výsledný genotyp





3.7.3 Fitness krajina

Fitness krajina je n-rozměrná nadrovina udávající hodnotu *fitness* v závislosti na *genotypu*. Tvar *fitness krajiny* může výrazně ovlivnit průběh genetického vývoje. Volba *fitness funkce* určuje zpětnou vazbu genetického vývoje. Pokud je zpětná vazba příliš velká, pak je obtížné určit směr genetického vývoje

a nastává náhodné prohledávání stavového prostoru. Dále by *fitness krajina* měla obsahovat co nejméně lokálních maxim. Při velkém počtu lokálních maxim je pro genetický vývoj obtížné najít globální maximum. Může se stát, že genetický vývoj uvázne v některém lokálním maximu. Proti uváznutí v lokálním maximu pomáhá fáze mutace.

3.8 Použití genetických algoritmů

Pomocí GA nastavuji synaptické vády *neuronové sítě* řešící podúlohu *výroby jednotek*. V této kapitole popsuji mojí implementaci metod *ohodnocení*, *selekce*, *křížení* a *mutace*.

3.8.1 Genotyp neuronové sítě

Genotyp je tvořen řetězcem hodnot synaptických vah *neuronové sítě*. Všechny hodnoty *genotypu* jsou reálná čísla. Každá synaptická váha má v řetězci definovanou určitou pozici, aby se dalo zjistit jaké číslo odpovídá jaké synaptické váze. Pro stejnou architekturu *neuronových sítí* jsou všechny *genotypy* stejně dlouhé.

3.8.2 Implementace ohodnocení

Ohodnocení je implementováno funkcí, která vrací reálné číslo udávající dosažené bohatství *hráče* po odehrání několika kroků hry. *Ohodnocení genotypu* je rozděleno do několika fází:

- vygenerování počátečního stavu
- simulace strategické hry
- ohodnocení výsledného stavu

3.8.2.1 Vygenerování počátečního stavu

Během této fáze se vytvoří *prostředí* a několik základních *jednotek* vlastněné *umělou inteligencí*. Dále se z *genotypu* nastaví synaptické váhy *neuronové sítě*.

3.8.2.2 Simulace strategické hry

V této fázi probíhá simulace strategické hry řízené jednou *umělou inteligencí*. Během simulace *umělá inteligence* zadává příkazy *jednotkám*, které má pod správou. Rozhodování o zadávání příkazů je závislé na vstupním *genotypu*. Po simulaci několika kroků strategické hry nastává fáze *ohodnocení výsledného stavu*.

3.8.2.3 Ohodnocení výsledného stavu

Během *ohodnocení výsledného stavu* se zjišťuje úspěšnost *umělé inteligence* a vypočítává se *fitness* hodnota (viz kapitola 4.4). *Fitness* hodnota závisí na velikosti bohatství (počty jednotek, surovin atd.) *umělé inteligence* dosaženém ve výsledném stavu.

3.8.3 Implementace selekce

Algoritmus *selekce* použitý v mé implementaci GA vybere do nové *populace* n *genotypů* s nejvyšší *fitness* hodnotou. Každý vybraný *genotyp* je m-krát zkopírován tak, aby velikost nové *populace* byla stejná jako velikost původní *populace*.

3.8.4 Implementace křížení

Během fáze *křížení* se nejdříve náhodně spárují *genotypy*. Po spárování *genotypů* se každý pár s určitou pravděpodobností zkříží. Pravděpodobnost zkřížení páru je rovna 75%. Křížení *genotypů* v páru se provádí *jednobodovým křížením* (vygeneruje se náhodné číslo z rozsahu <0, délka *genotypu*> a zamění se navzájem všechny hodnoty *genotypů* jejichž pozice jsou menší nebo rovny vygenerovanému číslu) .

3.8.5 Implementace mutace

Ve fázi *mutace* se vezme každý *genotyp* a s pravděpodobností 50% se zmutuje. Zmutování *genotypu* se provádí tak, že se každá hodnota *genotypu* s pravděpodobností 10% nahradí náhodným číslem z rozsahu <-1,1>.

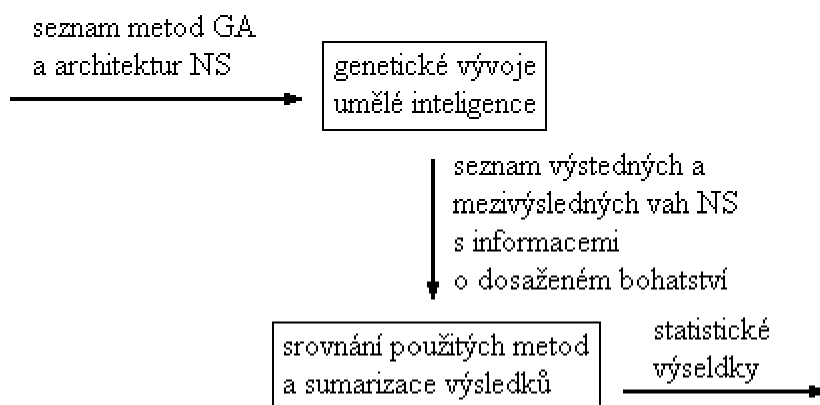
Kapitola 4

Experimenty

V této kapitole popisuji provedené experimenty na umělé inteligenci použité `vpodúloze` *výroby jednotek*. Cílem experimentů je porovnání různých architektur *neuronových sítí* (NS) s učením vah pomocí různých metod *genetických algoritmů* (GA). Všechny experimenty byly provedeny na prázdné mapě (umělá inteligence neměla protivníka). Domnívám se, že počet hráčů nemá vliv na srovnání kvalit různých metod. Myslím si, že řešení je natolik robustní, že při učení NS na mapě s protivníkem se NS dokáže adaptovat.

4.1 Popis experimentů

V rámci jednoho experimentu se porovnává vliv různých metod GA a různých architektur NS na genetický vývoj *umělé inteligence*. Vstupem experimentu je seznam metod GA a architektur NS. Pro každou položku vstupního seznamu se provede genetický vývoj *umělé inteligence* používající příslušné metody a architektury. Genetické vývoje jsou na sobě nezávislé. Po všech genetických vývojích nastane srovnání výsledných *umělých inteligencí* a sumarizace výsledků.



4.1.1 Genetický vývoj umělé inteligence

Vstupní parametry genetického vývoje jsou metody GA a architektura NS. Genetický vývoj vrací výsledné a mezivýsledné ohodnocení vah NS spolu s informacemi o dosaženém bohatství. Cílem vývoje je najít ohodnocení vah NS tak, aby co nejlépe řešila zadanou podúlohu (naučit *umělou inteligenci* používat všechny typy *útočných jednotek* ve hře).

4.1.2 Srovnání použitých metod a sumarizace výsledků

V této fázi se z výsledných a mezivýsledných informací o dosaženém bohatství vytvoří statistické informace. Je porovnáván jak průběh genetického vývoje tak cílový stav vývoje (nejlepší *umělá inteligence*).

4.1.2.1 Průběh vývoje

V průběhu vývoje je hodnoceno:

- 1. výskyt umělé inteligence používající 1 typ útočných jednotek
- 1. výskyt umělé inteligence používající 2 typy útočných jednotek
- 1. výskyt umělé inteligence používající 3 typy útočných jednotek
- 1. výskyt umělé inteligence používající 4 typy útočných jednotek
- absolutní kolísavost

  Průměrný rozdíl největších *fitness* hodnot v *populaci* mezi dvěmi po sobě jdoucími *generacem*. Absolutní kolísavost by měla být co nejmenší. Pokud genetický vývoj příliš kolísá, pak je malá pravděpodobnost, že *populace* bude v další *generaci* lepší než v předchozí *generaci*.

  $$Y = \frac{1}{n} * \sum_{i=0}^{n-1} \left| X_{i+1} - X_i \right|$$

- relativní kolísavost

  Absolutní kolísavost vztažená vůči maximální hodnotě *fitness*.

  $$Y = \frac{1}{n} * \sum_{i=0}^{n-1} \left| X_{i+1} - X_i \right| * \frac{1}{Max(\{X_i \mid i \in \{0,\ldots,n\}\})}$$

- absolutní střední odchylka růstu

  Střední odchylka nejlepších *fitness* hodnot v *populaci* mezi dvěmi po sobě jdoucími *generacemi*. Absolutní střední odchylka růstu by měla být co nejmenší. *Fitness* hodnot v *populaci* by měla růst plynule a ne skokově (*fitness* hodnota několik *generací* stagnuje a pak najednou výrazně vzroste). Pokud *fitness* hodnota několik generací stagnuje, pak se z genetického vývoje stává náhodné prohledávání stavového prostoru.

  $$Y = \frac{1}{n} * \sum_{i=0}^{n-1} \left| \frac{X_n - X_0}{n} - \left(X_{i+1} - X_i \right) \right|$$

- relativní střední odchylka růstu

  Absolutní střední odchylka růstu vztažená vůči celkovému růstu.

$$Y = \frac{1}{n} * \sum_{i=0}^{n-1} \left| \frac{X_n - X_0}{n-1} - (X_{i+1} - X_i) \right| * \frac{1}{(X_n - X_0)}$$

n - počet *generací* genetického vývoje

X_i - Největší *fitness* hodnota *genotypu* v i-té *generaci*

$X_0 = 0$

4.1.2.2 Cílový stav

V cílovém stavu vývoje je hodnocena nejlepší umělá inteligence z celého vývoje. Je posuzováno:

- dosažený počet útočných jednotek
- počet používaných typů útočných jednotek
- střední odchylka používaných útočných jednotek

$$Y = \frac{1}{m} * \sum_{i=1}^{m} \left| \left(\sum_{j=1}^{m} X_j / m \right) - X_i \right|$$

m - počet typů *útočných jednotek*

X_i - počet *útočných jednotek* i-tého typu

4.2 Implementační parametry

Všechny experimenty byly provedeny na pracovní stanici Pentium 4 CPU 2.66GHz, 1,00 GB RAM s operačním systémem Windows XP Media Center Edition SP2 a vývojovém prostředí Delphi 7. V *genetických algoritmech* byl použit standardní generátor náhodných čísel vývojového prostředí Delphi 7 (vestavěná funkce Random). Všechny experimenty byly provedeny se zafixovaným náhodným semínkem, aby se daly opětovně ověřit výsledky experimentů. Semínko náhody bylo nastaveno na 48889. Provedení experimentů je nezávislé na implementačních parametrech kromě generátoru náhody.

4.3 Parametry strategické hry

Parametry specifikované strategické hry jsou upraveny pro genetický vývoj. Úprava je provedená tak, aby simulace strategické hry byla co nejrychlejší. Aby se pomocí GA nalezlo alespoň "trochu" dobré řešení, musí GA ohodnotit kolem 1000 až 10000 řešení (jak se ukázalo v kapitole 4.6.1). Oproti komerčně známým strategickým hrám jako je například Command and Conquer nebo Dune 2 je průběh specifikované strategické hry výrazně rychlejší. *Jednotky* jsou schopny během jednoho kroku se posunout o větší vzdálenost a pootočit se o větší úhel. Výroby všech *jednotek* trvají kolem dvou až pěti kroků akce *výroby jednotek*. *Těžícím jednotkám* byla výrazně navýšena nosnost surovin. V události *inkrementace času* byla minimalizována doba uspání aplikace.

4.4 Provedené experimenty:

- porovnání fitness funkcí
- porovnání přenosových funkcí
- porovnání architektur neuronových sítí

4.5 Porovnání fitness funkcí

V tomto experimentu se porovnávají čtyři typy *fitness funkce*. Konkrétní implementace *Fitness funkce* velmi ovlivňuje průběh genetického vývoje. Porovnání *fitness funkcí* bylo realizováno na jednovrstvé *neuronové síti*. Pro výpočet hodnoty neuronů nebyla použita přenosová funkce radiální báze. Genetický vývoj probíhal 100 *generací* na *populaci* o 50 *genotypů*. Doba simulace strategické hry byla 100 cyklů

4.5.1 Porovnávané fitness funkce

Porovnávané fitness funkce:

- hrubá fitness funkce
- jemná fitness funkce
- fitness funkce s limitem
- fitness funkce s penalizací

4.5.1.1 Hrubá fitness funkce

Hrubá fitness funkce ohodnocuje *umělou inteligenci* jen za počet vyrobených *útočných jednotek.*

4.5.1.2 Jemná fitness funkce

Tato *fitness funkce* navíc od *hrubé fitness funkce* zohledňuje počet typů postavených budov (*nepohyblivá jednotka*). Výroba *jednotek* je složitý úkol a proto je lepší nejdříve naučit *umělo inteligenci* stavět různé typy budov a potom v nich vyrábět *jednotky*.

4.5.1.3 Fitness funkce s limitem

Fitness funkce s limitem se od *jemná fitness funkce* liší tím, že obsahuje limit pro počet *útočných jednotek* stejného typu. Pokud *umělá inteligence* vyrobí více *útočných jednotek* stejného typu než je daný limit, pak bude ohodnocena stejně jako by jich vyrobila přesně do velikosti limitu. Tento limit nazývám **limitem ohodnocení**. Limit nutí *umělou inteligenci* používat *jednotky* různých typů. Navíc snižuje počet lokálních maxim ve *fitness krajině*. Například pro *umělou inteligenci*, která ze všech natěžených surovin vyrobí *jednotky* stejného typu, bude těžké se naučit vyrábět víc typů *jednotek*, protože by se nejprve musela naučit našetřit suroviny, čímž by dočasně zmenšila svoji *fitness*. V tomto experimentu jsem zvolil limit rovný 10.

4.5.1.4 Fitness funkce s penalizací

Tato *fitness funkce* je ještě přísnější než *fitness funkce s limitem* a dokonce penalizuje *umělou inteligenci* za překročení limitu. Velikost limitu jsem nechal stejnou jako u *fitness funkce s limitem.*

$$Hrubá\ fitness = \prod_{i=1}^{n} (X_i + 1)$$

$$Jemná\ fitness = \prod_{i=1}^{n} (X_i + 1) * (B + 1)$$

$$Fitness\ s\ limitem = \prod_{i=1}^{n} (Min(\{X_i,\ Lim\}) + 1) * (B + 1)$$

$$Fitness\ s\ penalizací = \prod_{i=1}^{n} (Max(\{0,\ Lim - |\ X_i - Lim\ |\}) + 1) * (B + 1)$$

X_i - počet *útočných jednotek* i-tého typu

n - počet typů *útočných jednotek*

B - počet typů postavených budov

Lim - limit *útočných jednotek*

Min - funkce vracející minimum z množiny

Max - funkce vracející maximum z množiny

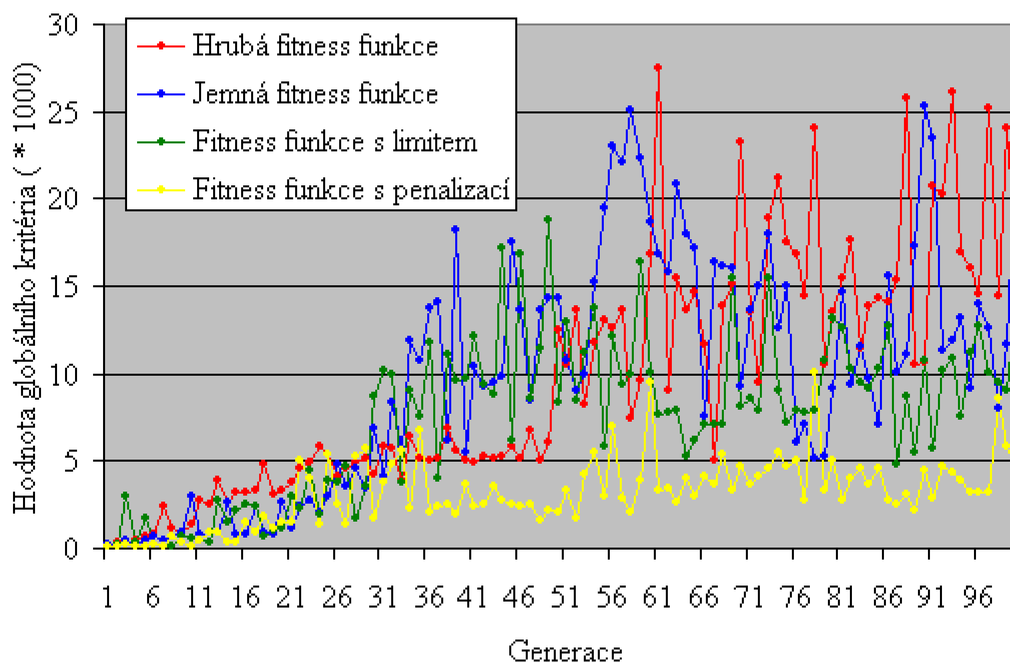
4.5.2 Výsledek experimentu

Protože v každém genetickém vývoji byla použita jiná funkce pro výpočet *fitness* hodnoty, nemůžu genetické vývoje mezi sebou srovnávat prostřednictvím *fitness* hodnoty. Z tohoto důvodu jsem se rozhodl srovnat genetické vývoje podle globálního kritéria. Genetické vývoje jsou porovnávány podle součinu počtu *útočných jednotek* každého typu.

$$Hodnota\ globálního\ kritéria = \prod_{i=1}^{n}(X_i + 1)$$

X_i - počet *útočných jednotek* i-tého typu

n - počet typů *útočných jednotek*



Průběh vývoje	Hrubá fitness	Jemná fitness	Fitness s limitem	Fitness s penalizací
1. výskyt 1 typu útoč. jednotek	1. generace	1. generace	1. generace	1. generace
1. výskyt 2 typů útoč. jednotek	1. generace	1. generace	1. generace	1. generace
1. výskyt 3 typů útoč. jednotek	3. generace	10. generace	1. generace	1. generace
1. výskyt 4 typů útoč. jednotek	18. generace	10. generace	22. generace	11. generace
Absolutní kolísavost	3064,06	3025,54	2820,4	1450,79
Relativní kolísavost	11,16%	11,94%	14,99%	14,39%
Absolutní střední odchylka růstu	3011,99	2351,45	2243,33	1179,53
Relativní střední odchylka růstu	18,59%	9,79%	15,58%	26,48%

Cílový stav	Hrubá fitness	Jemná fitness	Fitness s limitem	Fitness s penalizací
Počet útoč. jednotek	55	56	42	31
Počet typů útoč. jednotek	4	4	4	4
Střední odchylka útoč. jednotek	6,25	8,5	2,25	0,88

4.5.3 Zhodnocení výsledků

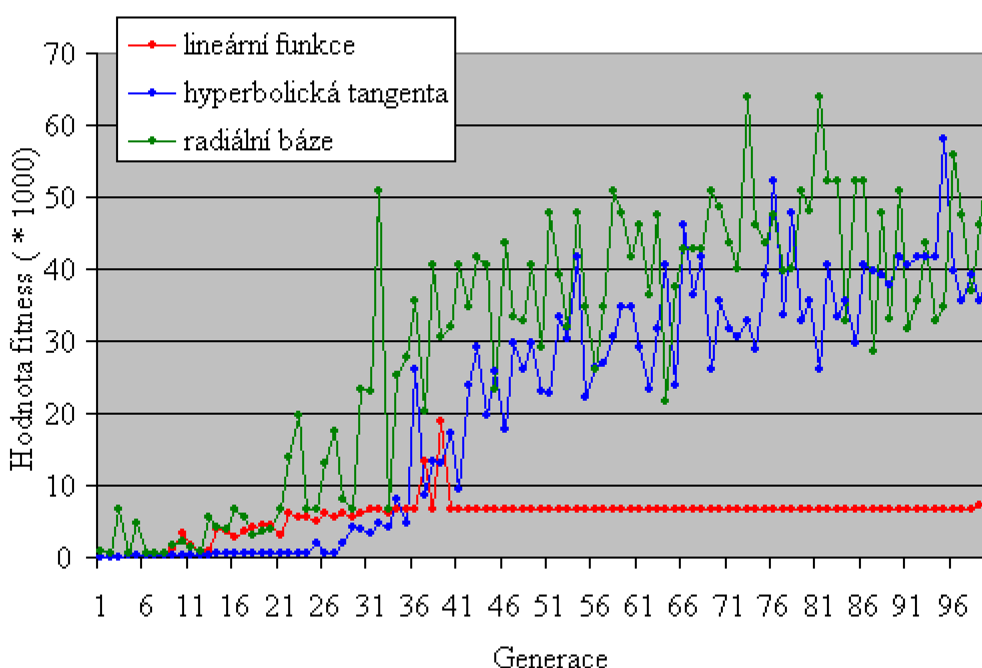
Genetické vývoje používající *hrubou* a *jemnou fitness funkci* dosáhly největšího počtu *jednotek*. Mezi průběhem těmito dvěma vývoji nebyl výrazný rozdíl. Pouze mezi 30. a 50. *generací* se *jemná fitness funkce* odchýlila od *hrubé fitness funkce*. Genetické vývoje s *fitness funkcí s penalizací* a s *fitness funkcí s limitem* rychleji naučily *umělé inteligence* používat více typů *útočných jednotek*. Navíc *umělé inteligence* vyvinuté oběma vývoji dosahovali vyváženějšího počtu *útočných jednotek* (od každého typu vlastnili přibližně stejný počet *útočných jednotek*). Protože genetický vývoj používající *fitness funkcí s penalizací* dosáhl výrazně menšího počtu *jednotek* než u všech ostatních vývojů, hodnotím *fitness funkci s penalizací* jako nejhorší z porovnávaných funkcí. Vývoj používající *fitness funkci s limitem* byl značně lepší než vývoj *fitness funkce s penalizací* a výrazně se nelišil od vývoje používající *jemnou* a *hrubou fitness funkce*. Navíc průběh vývoje s *fitness funkcí s limitem* měl menší rozkmit hodnot globálního kritéria než vývoje s *hrubou* a *jemnou fitness funkcí*.

4.6 Porovnání přenosových funkcí

Tento experiment srovnává vliv *přenosových funkcí* na průběh genetického vývoje. Nejčastější přenosové funkce jsou skoková funkce, funkce radiální báze, sigmoidální funkce a funkce hyperbolické tangenty. Funkce hyperbolické tangenty = 2 * sigmoidální funkci -1. Na základě této vlastnosti se domnívám, že genetický vývoj obou těchto funkcí by byl téměř totožný. Proto jsem se rozhodl tyto funkce mezi sebou nesrovnávat. Skoková funkce nabývá pouze hodnot 0 nebo 1. Kdyby všechny výstupní parametry byly pouze dvoustavové, potom by *umělá inteligence* nebylo schopna přesně uspořádat typy *jednotek* podle priority výroby. Byla by pouze schopna rozdělit typy *jednotek* do dvou skupin. Přestože by rozdělení do dvou skupin stačilo k rozhodování o výrobě *jednotek*, domnívám se, že by rozhodování *umělé inteligence* bylo velmi omezené. Na základě tohoto názoru jsem se rozhodl skokovou přenosovou funkce zamítnout. V tomto experimentu srovnávám lineární funkci, funkci hyperbolické tangenty a funkci radiální báze. Parametr u funkce radiální báze a hyperbolické tangenty byl rovný 1. Porovnání *přenosových funkcí* bylo realizováno na jednovrstvé neuronové síti. Genetický vývoj probíhal 100 *generací* na *populaci* o 50 *genotypů*. Za ohodnocující funkci jsem zvolil *fitness funkci s limitem*. Na základě předchozího experimentu mi připadá nejvhodnější. *Ohodnocující limit* byl rovný 10. Doba simulace jedné strategické hry byla 100 cyklů.

4.6.1 Výsledek experimentu

V grafu jsou znázorněny *fitness* hodnoty nejlepších *umělých inteligencí* v *populaci* během každé *generace*.



Průběh vývoje	lineární funkce	hyperbolická tangenta	radiální báze
1. výskyt 1 typu útoč. jednotek	1. generace	1. generace	1. generace
1. výskyt 2 typů útoč. jednotek	3. generace	2. generace	1. generace
1. výskyt 3 typů útoč. jednotek	9. generace	25. generace	1. generace
1. výskyt 4 typů útoč. jednotek	37. generace	36. generace	22. generace
Absolutní kolísavost	618,45	5036,12	8592,84
Relativní kolísavost	3,25%	8,67%	13,45%
Absolutní střední odchylka růstu	899,92	4752,96	6262,73
Relativní střední odchylka růstu	13,52%	12%	10,78%

Cílový stav	lineární funkce	hyperbolická tangenta	radiální báze
Počet útoč. jednotek	28	37	42
Počet typů útoč. jednotek	4	4	4
Střední odchylka útoč. jednotek	2	1,25	2,25

4.6.2 Zhodnocení výsledků

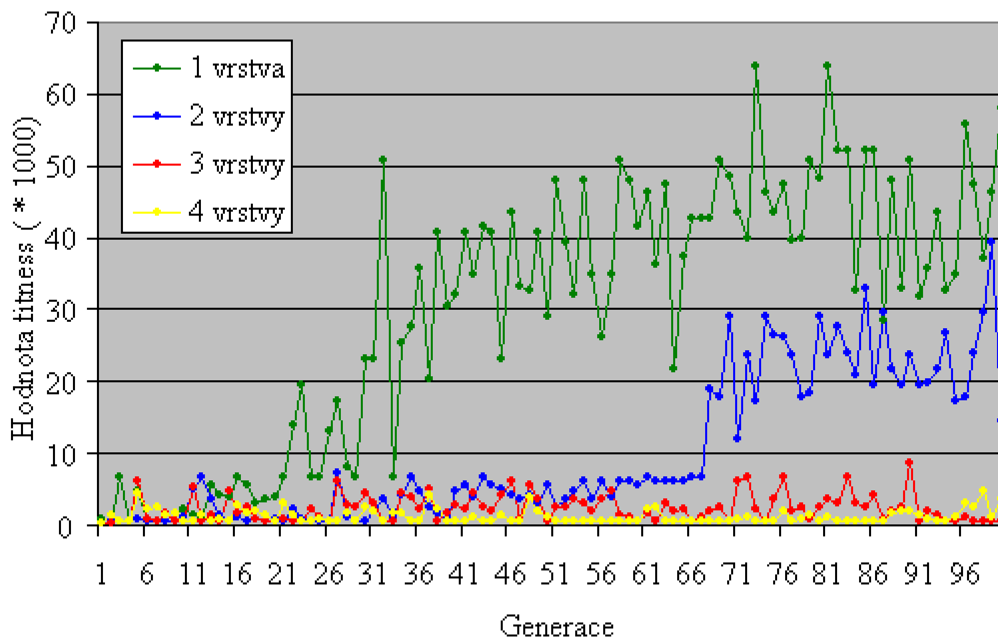
Genetický vývoj používající lineární funkci nalezl v 37. a 39. *generaci umělou inteligenci* používající 4 typu *útočných jednotek*. Kromě těchto dvou *generací* už nenalez žádnou jinou *umělou inteligenci* používající 4 typy *útočných jednotek*. Proto lineární funkci hodnotím jako nejhorší funkci z porovnávaných *přenosových funkcí*. Vývoj s přenosovou funkcí radiální báze naučil umělou inteligenci používat více typů jednotek rychleji než vývoje používající ostatní přenosové funkce. Navíc výsledná *umělá inteligence* dosáhla většího počtu *útočných jednotek* než všechny ostatní *umělé inteligence*. Přestože genetický vývoj používající *přenosovou funkci* radiální báze nejvíce kolísal, hodnotím funkci radiální báze za nejvhodnější *přenosovou funkcí* pro moji úlohu.

4.7 Porovnání architektur neuronových sítí

Třetí experiment zkoumá vývoj *umělé inteligence* na různých typech architektur NS. Zkoumané architektury NS jsou 1, 2, 3 a 4 vrstvé *neuronové sítě*. Skryté vrstvy obsahují 10 *neuronů*. Jako ohodnocující funkci jsem si vybral *fitness funkci s limitem*. Ohodnocující limit byl rovný 10. Na základě výsledku z 2. experimentu jsem se rozhodl za přenosovou funkci zvolit funkci radiální báze. Parametr funkce radiální báze byl rovný jedné Genetický vývoj probíhal 100 *generací* na *populaci* o 50 *genotypů*. Doba simulace strategické hry byla 100 cyklů.

4.7.1 Výsledek experimentu

V grafu jsou znázorněny *fitness* hodnoty nejlepších *umělých inteligencí* v *populaci* během každé *generace*.



Průběh vývoje	1 vrstva	2 vrstvy	3 vrstvy	4 vrstvy
1. výskyt 1 typu útoč. jednotek	1. generace	1. generace	1. generace	1. generace
1. výskyt 2 typů útoč. jednotek	1. generace	1. generace	1. generace	1. generace
1. výskyt 3 typů útoč. jednotek	1. generace	5. generace	5. generace	2. generace
1. výskyt 4 typů útoč. jednotek	22. generace	68. generace	11. generace	nikdy
Absolutní kolísavost	8592,84	3110,16	1788,59	757,22
Relativní kolísavost	13,45%	7,93%	20,53%	15,65%
Absolutní střední odchylka růstu	6262,73	2977,73	1349,6	675,5
Relativní střední odchylka růstu	10,78%	20,51%	272,65%	18,61%

Cílový stav	1 vrstva	2 vrstvy	3 vrstvy	4 vrstvy
Počet útoč. jednotek	42	34	27	41
Počet typů útoč. jednotek	4	4	4	3
Střední odchylka útoč. jednotek	2,25	2,5	4,25	5,56

4.7.2 Zhodnocení výsledků

U NS s vyšším počtem vrstev se výrazně zhoršil průběh genetického vývoje. Naučení *umělé inteligence* používat více typů *útočných jednotek* trvalo mnohem déle než u jednovrstvé NS. Se zvětšením počtu vrstev NS se zvětšuje i velikost *genotypu,* která exponenciálně ovlivňuje velikost prohledávaného prostoru. Myslím si, že toto zvětšení prohledávaného prostoru u vícevrstvých NS je hlavní příčinou toho, že GA nedokázaly najít tak schopné *umělé inteligenc*e jako u jednovrstvé NS. Dále se domnívám, že u velkých struktur je nižší pravděpodobnost, že jejich zkřížením dosáhneme lepšího jedince (*genotyp* s vyšší hodnotou *fitness*). Myslím si, že toto je další příčina zhoršení průběhu genetického vývoje u NS s vyšším počtem vrstev.

Kapitola 5

Závěr

Cílem této práce bylo specifikovat strategickou hru, popsat úlohu a navrhnout umělou inteligenci řešící danou úlohu na specifikované strategické hře.

Ve specifikaci jsem popsal základní princip mé strategické hry, nadefinoval jsem prostředí, objekty a vztahy mezi nimi. Specifikaci jsem psal tak, aby byla snadno srozumitelná a aby čtenář pochopil základní princip systému nad kterým definuji a řeším úlohu.

Na zmíněné strategické hře jsem nadefinoval úlohu a navrhl její řešení. Při návrhu řešení jsem uplatnil poznatky z oboru umělé inteligence. Všechny použité metody umělé inteligence jsem popsal tak, aby měl čtenář dostatečný přehled nad postupem řešení dané úlohy.

Specifikovanou strategickou hru jsem implementoval včetně několika metod umělé inteligence řešící danou úlohu. Na implementované strategické hře jsem provedl několik experimentů srovnávající výkonnost implementovaných metod umělé inteligence. Na závěr jsem zhodnotil výsledky experimentů a popsal výhody a nevýhody porovnávaných metod.

Do budoucna bych chtěl provést srovnání prezentovaných metod i s dalšími metodami jako jsou různé druhy prohledávání stavového prostoru. Dále bych chtěl provést experiment srovnávající výkonnost navržené umělé inteligence s lidským hráčem (případně více hráči). Jako další rozšíření bych viděl v implementaci metod umělé inteligence i do podúlohy útoku jednotek. Umělá inteligence by pak rozhodovala i o rozestavení jednotek na mapě.

Literatura

[1] Ghallab M, Nau D.S., Traverso P. *Automated Planning: Theory and Practice*. Elsevier, 2004, ISBN 1-558-60856-7

[2] Chytil M. *Automaty a gramatiky*. SNTL, 1984,

[3] Rojas R. *Neural Networks: A Systematic Introduction*. Springer, 1996, ISBN 3-540-60505-3

[4] Koza J.R. *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*. Stanford University Computer Science Department technical report, 1990,

[5] Koza J.R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992, ISBN 0-262-11170-5

[6] Koza J.R. *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, 1994, ISBN 0-262-11189-6

[7] Koza J.R., Bennett, F.H., Andre D., Keane M.A. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann, 1999, ISBN 1-55860-543-6

[8] Zhang R., Bivens A. J. *Comparing the use of bayesian networks and neural networks in response time modeling for service-oriented systems*. Association for Computing Machinery, 2007, ISBN 978-1-59593-717-9, High Performance Distributed Computing, Strany: 67 - 74

[9] Chen Z. *Computational Intelligence for Decision Support*. CRC Press, 2000, ISBN 0-849-31799-1

[10] Laarhoven P. J. M., Aarts E. H. L. *Simulated Annealing: Theory and Applications*. Springer, 1987, ISBN 90-277-2513-6,

A Uživatelské rozhraní

Uživatelské prostředí obsahuje okno, na kterém se vykresluje aktuální stav strategické hry a panel pomocí něhož se zadávají příkazy *jednotkám*. Ovládání *jednotek* je vykonáváno pomocí myši. Při zmáčknutí levého tlačítka myši a následném tažením po okně je uživatel schopen označit *jednotky* vyskytující se v obdélníku určeném body aktuální polohy myši a polohy myši při zmáčknutí levého tlačítka. Označené jednotky a jejich akce se vykreslí na ovládací panel. Při zadání příkazu se příkaz aplikuje pouze na označené jednotky. Při kliknutí pravého tlačítka myši do okna modelu je dán příkaz přesunout jednotky na pozici, kam bylo kliknuto. Pokud jednotka má určitou speciální akci, pak se tato akce při označení jednotky vykreslí na ovládací panel. Po kliknutí na akci na ovládacím panelu se zadá tato akce všem označeným jednotkám, které jsou schopny provedení této akce. Prostřednictvím šipek na klávesnici lze posouvat s obrazem strategické hry.

B Stručný uživatelský pohled

Hra obsahuje 5 typů jednotek: jednotku na těžení a stavbu budov, tři typy vozidel a jedno letadlo. Stavěč slouží k těžení surovin a ke stavbě budov. Těží se pouze jeden druh suroviny a stavěč ho může donést jen do základní budovy. Jeden typ vozidla je průzkumník. Pohybuje se obzvlášť rychle, ale jeho útočná síla je nízká. Další typ vozidla je protiletecké vozidlo s velkým dostřelem a malou odolností. Může střílet i na pozemní jednotky. Poslední typ vozidla je mohutný tank, pomalý ale silný a hodně odolný. Počet jednotek vlastněných hráčem je omezený kapacitou dep. Každé depo může uskladnit určitý počet vozidel. Jedná se jen o limit, hráč nemusí nijak do dep zajíždět. Další budovy jsou základní budova, výrobna vozidel, výrobna letadla a budova výzkumu.

C Obsah přiloženého CD

K bakalářské práci je přiložené CD, které obsahuje implementovanou strategickou hru, zdrojové kódy strategické hry a samotný text této práce.

Přiložené CD obsahuje tři adresáře:

- Bakalářská práce

  Adresář obsahuje text bakalářské práce ve formátu pdf.

- Projekt

  V tomto adresáři je uložený spustitelný program s mapou a ukázkovými neuronovými sítěmi.

- Zdroj

  Adresář obsahuje zdrojové kódy ke strategické hře. Zdrojové kódy jsou psané ve vývojovém prostředí Deplhi 7.

D Instalace

Program běží na operačním systému Windows XP. Program není potřeba instalovat, stačí pouze zkopírovat celou složku na disk. Po spuštění programu se automaticky spustí strategická hra a zobrazí se ovládací panel a okno zobrazující stav hry. Hra se ovládá prostřednictvím myši a ovládacího panelu jak je popsáno v příloze A. Při zmáčknutí klávesy "x" se zobrazí pomocný panel, který slouží k nastavení a spuštění genetického vývoje umělé inteligence. Prostřednictvím tlačítka "NS" se otevře okno na nastavení neuronové sítě, která je použita v rozhodování umělé inteligence. Neuronová síť lze nahrát zmáčknutím tlačítka "Nahrát". Tlačítko "Simulation" otevře okno pro nastavení parametrů genetického vývoje. Po potvrzení parametrů se automaticky spustí genetický vývoj umělé inteligence. Pro spuštění simulace musí být nejprve nastavena neuronová sít. Po dokončení každé generace evolučního vývoje program uloží dvě nejlepší neuronové sítě v generaci na disk. Dále uloží doposud nejlepší dosaženou neuronovou sít z celého vývoje. Také připíše záznam do souboru "Gensfile.txt" obsahující fitness hodnotu nejlepšího jedince a počty útočných jednotek od každého typu. Pro zobrazení chování vyvinuté umělé inteligence se nejprve musí nahrát vyvinutá neuronová síť (pomocí tlačítka NS), zaškrtnout volba "vykreslovat simulaci" a spustit simulace. Při nastavení parametrů genetického vývoje nastavte počet jedinců na 1 jedinec, počet generací na 1 a vyberte stejnou přenosovou funkci, jaká byla použita ve vývoji.