Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Simulační jádro pro rozvoj evolučních algoritmů

Bc. Jindřich Mikule

Diplomová práce

2021

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jindřich Mikule**
Osobní číslo:	**I17214**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Simulační jádro pro rozvoj evolučních algoritmů**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

V teoretické části práce se student zaměří na problematiku genetických a evolučních algoritmů, představí základní kategorizaci a využití těchto algoritmů. Následně se bude student věnovat popisu autonomní evoluce algoritmu v řízeném prostředí. Součástí práce bude také disputace o kauzualitě v evolučním programování.
V pratkické části student navrhne a implementuje simulační jádro, které umožní běh evolučních algoritmů v řízeném prostředí. Vstupními proměnnými budou kritéria na úrovni fitness funkce a startovní podmínky a limitní omezení. Jádro umožní sledování rozvoje kolonie algoritmů (tj. sledování stavového prostoru na úrovni agregovaných zastoupení jednotlivých jedinců).

Rozsah pracovní zprávy:	**60**
Rozsah grafických prací:	**10**
Forma zpracování diplomové práce:	**tištěná**

Seznam doporučené literatury:

*HYNEK, Josef. Genetické algoritmy a genetické programování. Praha: Grada, 2008. ISBN 8024726955.
*MITCHEL, Melanie. An Introduction to Genetic Algorithms. Cambridge: MIT Press, 2002. ISBN 0262631857.
*JACOBSON, Lee a Burak KANBER. Genetic algorithms in Java basics. New York: Apress, 2015. Expert's voice in Java. ISBN 1484203291.

Vedoucí diplomové práce:	**doc. Ing. Tomáš Brandejský, Dr.** Katedra softwarových technologií
Datum zadání diplomové práce:	**6. listopadu 2020**
Termín odevzdání diplomové práce:	**15. května 2021**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 30. listopadu 2020

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47 b zákona č.111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 21. 5. 2021

Bc. Jindřich Mikule

PODĚKOVÁNÍ

Tímto chci poděkovat vedoucímu práce doc. Dr. Ing. Tomáši Brandejskému a Ing. Josefu Brožkovi, Ph.D. za jejich odborné vedení, cenné rady, připomínky, podněty a veškerou poskytnutou pomoc.

THANKS

I'd like to thank the mentor doc. Dr. Ing. Tomáš Brandejský and Ing. Josef Brožek, Ph.D. for their professional guidance, valuable advice, comments, suggestions and all the help provided.

ANOTACE

Diplomová práce se zaměřuje na vývoj aplikace, která je schopná simulovat vybrané aspekty sociologického vývoje společnosti. Součástí aplikace je sledování stavového prostoru na úrovni agregovaných zastoupení hodnot atributů jednotlivých entit. V úvodní části je provedena rešerše evolučních algoritmů a simulátorů. Následně se práce zaobírá disputací, jak propojit diskrétní simulátor a evoluční algoritmy. Konec práce je věnován implementaci a ukázce vyvinuté aplikace.

KLÍČOVÁ SLOVA

Simulace, simulační jádro, evoluční programování, sociologická simulace, agent, diskrétní simulace.

TITLE

Simulation engine for the development of evolutionary algorithms.

ANNOTATION

The diploma thesis focuses on the development of an application that is able to simulate selected aspects of the sociological development of society. The application includes the monitoring of the state space representation at the level of aggregate values of attributes of individual entities. The introductory part summarizes the evolutionary algorithms and the simulators. Subsequently, the work deals with the disputation of merging discrete simulator and evolution algorithms. The end is devoted to the implementation and demonstration of developed application.

KEYWORDS

Simulation, simulation core, evolution programing, sociology simulation, agent, discrete simulation

OBSAH

Seznam obrázků, tabulek **11**

Seznam zkratek a pojmů **13**

Úvod **15**

1 Evoluční algoritmy **16**

1.1 Motivace k zavedení evolučních algoritmů 16

1.2 Historie Evolučních algoritmů 17

1.3 Pojmy 17

1.3.1 Chromozom 17

1.3.2 Gen 18

1.3.3 Alela 18

1.3.4 Genotyp 19

1.3.5 Fenotyp 19

1.3.6 Jedinec 19

1.3.7 Populace 19

1.3.8 Generace 19

1.4 Genetické algoritmy 20

1.5 Operátory genetického algoritmu 20

1.5.1 Selekce 21

1.5.2 Křížení 22

1.5.3 Mutace 22

2 Simulace **24**

2.1 Simulace 24

2.1.1 Systém, abstrakce 25

2.1.2 Model systému, událost 26

2.1.3 Proměnné 27

2.1.4 Entity systému 27

2.1.5 Atributy prvků 28

2.1.6 Proces 28

2.1.7 Aktivity 28

2.1.8 Scéna a scénář 30

2.1.9 Parametrizace simulačního modelu 30
2.2 Základní rozdělení 31
2.2.1 Spojitá simulace 32
2.2.2 Diskrétní simulace 32
2.2.3 Stochastická simulace 33
2.2.4 Deterministická simulace 33
2.2.5 On-line simulace 33
2.2.6 Off-line simulace 34
2.3 Metody synchronizace simulačního výpočtu 35
2.3.1 Metoda snímání aktivit 35
2.3.2 Metoda plánování událostí 35
2.3.3 Metoda interakce procesů 36
2.3.4 Třífázová metoda 36
3 Autonomní vývoj jedince v simulátoru 38
3.1 Řízené prostředí simulace 38
3.2 Evoluční algoritmus v řízeném prostředí 39
3.3 Fitness funkce 40
3.4 Kauzalita GA 41
4 Analýza řešeného problému 42
4.1 Analýza požadavků 42
4.2 Případy užití 44
4.2.1 Teoretický popis 45
4.2.2 Diagram užití aplikace 46
4.2.3 Specifikace případů užití 48
4.3 Diagram tříd/ UML 49
5 Vlastní řešení, implementace 51
5.1 Řešení 51
5.2 Implementace 52
5.2.1 Simulační jádro 52
5.2.2 Entita 53
5.2.3 PostOffice 57

5.2.4 Configuration 58
5.3 Parametrizace – nastavení aplikace simulátoru 58
5.3.1 Startovní podmínky 59
5.3.2 Ukončovací podmínka 59
5.3.3 Vstupní kritéria 59
5.3.4 Limitní omezeni 61
6 Ladění, simulační experimenty 63
6.1 Validace a ladění 63
6.1.1 Situace A, kdy je v populaci nadbytek produkce 63
6.1.2 Situace B, kdy se populace drží na úrovni přežití. 65
6.1.3 Situace C, kdy populace nakonec vyhladoví 67
6.2 Nastavení vstupních parametrů simulace 69
6.3 Experiment 1 70
6.4 Experiment 2 - krádeže 71
6.5 Experiment 3 – krádeže s potrestáním 72
6.6 Ostatní experimenty 72
Závěr 74
Použitá literatura 75
Přílohy 79

SEZNAM OBRÁZKŮ, TABULEK

Obrázek 1 Příklad mutace – přepnutí bitu....23
Obrázek 2 Ukázka události....27
Obrázek 3 Metoda interakce procesů....36
Obrázek 4 Značení pro aktéra....45
Obrázek 5 Značení pro případ užití....46
Obrázek 6 Značení pro relace....46
Obrázek 7 Diagram užití simulátoru pro vývoj evolučních algoritmů....48
Obrázek 8 Simulátor – implementace....53
Obrázek 9 Entita – ukončení životního cyklu....55
Obrázek 10 Entita – implementace upkeep funkcí....55
Obrázek 11 Entita – zpracování záměru....56
Obrázek 12 Entita – vyhodnocení záměrů....56
Obrázek 13 Entita – čtení zprávy....56
Obrázek 14 Entita – zpracování zprávy....57
Obrázek 15 PostOffice – komunikace....58
Obrázek 16 PostOffice – de/alokace mailboxů....58
Obrázek 17 Conf - singleton.....58
Obrázek 18 Parametrizace validace 1....64
Obrázek 19 Výsledek validace A – velikost populace....64
Obrázek 20 Výsledek validace A – produktivita....65
Obrázek 21 Parametrizace validace B....66
Obrázek 22 Výsledek validace B – velikost populace....66
Obrázek 23 Výsledek validace B – produktivita....67
Obrázek 24 Parametrizace validace C....68
Obrázek 25 Výsledek validace C – velikost populace....68
Obrázek 26 Výsledek validace C - produktivita....69
Obrázek 27 Výsledky simulace....70
Obrázek 28 Výsledky simulace....71
Obrázek 29 Výsledky simulace....72
Obrázek 30 PyCharm....83
Obrázek 31 Nastavení entity....84
Obrázek 32 Nastavení záměrů....85

Obrázek 33 Nastavení simulace 85
Obrázek 34 Ovládací panel 86
Obrázek 35 Grafický výstup 86
Obrázek 36 UML 91

Tabulka 1 Příklad pořadové selekce 22
Tabulka 2 Úvodní specifikace 43
Tabulka 3 Požadavková specifikace na funkce 43
Tabulka 4 Požadavková specifikace nefunkcionální 44
Tabulka 5 Nastavení parametrů pro simulační experimenty 70
Tabulka 6 Formát zpráv 81
Tabulka 7 Parametry simulátoru 82
Tabulka 8 Algoritmus snímání aktivit 92
Tabulka 9 Algoritmus plánování aktivit 93
Tabulka 10 Algoritmus interakce procesů 94
Tabulka 11 Algoritmus třífázové metody 95

SEZNAM ZKRATEK A POJMŮ

Termín	Význam
ADT	Abstraktní datový typ, matematická struktura skládající se z jedné nebo více domén, tj. tříd matematických objektů + jedné nebo více matematických operací nad těmito prvky.
AI	Artificial intelligence je zkratka pro umělou inteligenci (UI).
Booleovská hodnota	Boolean je logický datový typ reprezentovaný jednou ze dvou hodnot true (pravda, 1) nebo false (nepravda, 0).
CPU	Centrální procesorová jednotka (zkratka CPU, anglicky central processing unit) je v informatice označení základní elektronické součásti v počítači, která umí vykonávat strojové instrukce, ze kterých je tvořen počítačový program a obsluhovat jeho vstupy a výstupy.
Determinismus	Vlastnost procesu, jehož každý stav je určen předcházejícím. Tzn., že je opakovatelný a závisí pouze na vnějších vstupech
DNA	Deoxyribonukleová kyselina, látka, která se vyskytuje ve všech buněčných organismech. Nese genetickou informaci, v níž je předurčeno, jak má organismus vypadat a jak budou probíhat veškeré životní procesy daného organismu.
Drag&drop	Drag and drop (táhni a pusť) je v informatice operace používaná v grafickém uživatelském rozhraní, kdy uživatel v počítači „uchopí“ pomocí ukazovacího zařízení (např. myši) virtuální objekt (např. soubor nebo ikonu) a přesune ho „přetažením“ na jiné místo (nebo na jiný objekt).
GA	Genetic algorithm,genetický algoritmus je metaheuristika inspirovaná procesem přirozeného výběru, který patří do třídy EA.
GUI	Grafické uživatelské rozhraní (anglicky Graphic User Interface, známe pod zkratkou GUI) je uživatelské rozhraní, které umožňuje ovládat počítač pomocí interaktivních grafických ovládacích prvků.
GP	Genetic programming, je technika evolučních programů, která je vhodná pro konkrétní úkol tím, že na populaci programů aplikuje operace podobné přírodním genetickým procesům.
EA	Evolution algorithm, je podskupina evolučního výpočtu, obecný populačně založený optimalizační algoritmus.
Epigenetika	Epigenetika (=mimo genetiku) je vědní podobor genetiky, jenž studuje změny v genové expresi (a tedy obvykle i ve fenotypu), které nejsou způsobeny změnou nukleotidové sekvence DNA.
ES	Evolution Strategy, je technika optimalizace založená na myšlenkách evoluce. Patří do obecné třídy metodik evolučního výpočtu.
FXML	Značkovací jazyk pro vytváření GUI pro JavaFX aplikace. Jazyk je založen na bázi XML, poskytující strukturu pro vytváření uživatelského rozhraní odděleně od aplikační logiky.
IDE	Integrated Development Environment, je software usnadňující práci programátorů, většinou zaměřené na jeden konkrétní programovací jazyk. Obsahuje editor zdrojového kódu, kompilátor, případně interpret a většinou také debugger.
JVM	Java Virtual Machine je sada počítačových programů a datových struktur, která využívá modul virtuálního stroje ke spuštění dalších počítačových programů a skriptů vytvořených v jazyce Java.
Metaheuristika	Metaheuristika je metodologie vyšší obecnější úrovně, která může být použita jako řídící strategie pro návrh základní heuristiky pro řešení specifických optimalizačních problémů; př. rozvrhování předmětů: simulované žíhání a výměna dvou předmětů.
Monte Carlo	Třída algoritmů pro simulaci systému. Základním konceptem je použití náhodnosti k řešení problémů, které by mohly být v zásadě deterministické.
Multithreading	V počítačové architektuře je multithreading schopnost centrální procesorové jednotky (CPU) nebo jednoho jádra ve vícejádrovém procesoru vykonat více procesů, vhodně podporovaných operačním systémem.

NTP	Zkratka z anglického Network Time Protocol - časový protokol sítě.
OS	Operační systém (OS) je sada programů umožňující co nejefektivnější využití hardwaru počítače a hlavním úkolem tohoto systému je zabezpečit běh a programovou podporu aplikačních programů.
Stochastismus	Vlastnost procesu, jehož každý stav je určen náhodou.
RAM	RAM je zkratkou pro anglické označení random access memory, pro kterou se vžil český překlad operační paměť. Jde o velmi rychlou paměť, která uchovává právě zpracovávaná data.
Replikace	Opakování děje simulace, kde se stejná výpočetní úloha prování n-krát.
UML	Unified Modeling Language. Ve volném překladu z anglického jazyka zkratka znamená unifikovaný modelovací jazyk.
Variable	Proměnná, symbolický název spojený s hodnotou a jehož přidružená hodnota může být změněna.
Virtualizace	je označení postupů, technik a prostředků, které umožňují v počítači přistupovat k dostupným zdrojům jiným způsobem, než jakým fyzicky existují.

ÚVOD

Odvětvím přitahujícím velkou pozornost široké i odborné veřejnosti v oblasti informačních technologií jsou v současné době umělé inteligence. Laickou pozornost aktivovala komerční produkce vědecko-fantastické literatury. Reálná aplikace umělé inteligence se stále rozšiřuje a zaměřuje se většinou na snahu o maximálně autonomní systémy, které budou schopné opakovat řešení podobných tříd problémů, ve kterých však existuje určitý stupeň volnosti některých vstupních proměnných, čímž tyto problémy nejde (nebo je velmi obtížné) řešit tradičními programátorskými, nebo matematickými postupy.

Téma mojí diplomové práce se diametrálně odlišuje od obvyklých problémů, které umělé inteligence řeší, a proto jsem nemohl odolat vývoji simulačního jádra pro rozvoj evolučních algoritmů. V práci se nejprve zaměřuji na teoretické uvedení do obecné problematiky vybrané části umělé inteligence. Následně provádím teoretickou rozpravu nad simulátory. Výsledkem mého bádání (a předmětem další práce) je stanovení obecných principů, jak propojit diskrétní simulátor a evoluční algoritmy tak, aby bylo možné simulovat vybrané aspekty sociologického vývoje společnosti.

Engine aplikace je navržen s důrazem na mé vlastní zkušenosti s programováním, proto jsem zvolil programovací jazyk Python, který pro řešení daného typu problému není obecně vhodný. Aplikace umožňuje sledování stavového prostoru na úrovni agregovaných zastoupení hodnot atributů jednotlivých entit. Program simulačního jádra využívá několik nástrojů a modulů programovacího jazyka Python.

Simulátor najde uplatnění primárně na akademické půdě. Koncepce simulátoru umožňuje provádět experimenty uplatnitelné v oborech sociologie, ekonomie, historie a do jisté míry i v některém z odvětví antropologie, etnografie a dalších.

.

1 EVOLUČNÍ ALGORITMY

Evoluční algoritmy jsou algoritmy umělé inteligence, které se inspirovaly v biologické evoluci. Snaží najít řešení vytvořením generické populace a využití jejich operací. Mezi základní mechanismy, které jsou mimo jiné odvozeny od biologické evoluce, patří: reprodukce, mutace, rekombinace a selekce. Pojem evoluční algoritmy vznikl na základě podmětu sjednocení a metod, které se principem evoluce zaobírají. Do této kategorie se řadí:

a) Genetické algoritmy
b) Genetické programování
c) Evoluční programování
d) Evoluční strategie

Oproti jiným algoritmům se snaží dosáhnout výsledků využitím modelů evolučních procesů, aby tak nalezly řešení náročných a rozsáhlých úloh. Populace se časem přiblíží k řešení.

1.1 Motivace k zavedení evolučních algoritmů

Zástupci jakéhokoliv živočišného druhu jedné populace mezi sebou bojují o možnost se rozmnožovat a o svůj holý život. Přežijí pouze jen ti nejsilnější zástupci svého druhu a své předpoklady pro přežití vtisknou do svých potomků. Tito jedinci jsou opět vystaveni zkouškám přežití a znovu mají možnost usilovat o reprodukci v rámci své populace. O tomto jevu se píše v Darwinově teorii. Darwinova evoluční teorie se zakládá na tezi přirozeného výběru, podle které přežívají jen nejlépe přizpůsobiví jedinci populace. [4][5] [6]

Tato myšlenka už existovala před Darwinem, například už Darwinův dědeček Erasma Darwin se tímto problémem zabýval. Ačkoliv Charles Darwin a jeho následníci jí rozvinuli do podoby umožňující vysvětlit různorodost forem života, přizpůsobování živých organismů proměnlivému prostředí. Koncepce evoluce spočívá v kombinaci následujících pěti principů:[5]

1. Živé organizmy se v průběhu času vyvíjejí.
2. Evoluční změny mají charakter větvení.
3. Nové živočišné druhy vznikaly rozdělením na izolované populace a podpopulace.
4. Evoluční změny se nedějí najednou, ale postupně.
5. Mechanizmus adaptivní změny je přirozený výběr.

Právě tato odborná literatura byla inspirací pro akademiky minulého století, kteří bádali nad aplikováním teorie do výpočetní techniky. Tím historicky přispěli k vytvoření nových algoritmů umělých inteligencí.

1.2 Historie Evolučních algoritmů

O evoluční programování roste zájem až na přelomu století, přibližně posledních třicet let, zaznamenaly úžasný nárůst zájmu veřejnosti. Historicky vznikl koncept původně ze čtyř větví. První z nich se nazývá evoluční programování (EP) a důkladně jej popsal Lawrence J. Fogel v roce 1960 ve svých publikacích. Dalším z nich jsou genetické algoritmy (GA) a za jejich vysvětlením stojí John H. Holland ve svých publikacích z roku 1962 nebo 1975. Třetí přístup – genetické programování (GP) je dílem Johna Kozy a jeho kolegů ze Stanfordské univerzity z roku 1988. Posledním z nich jsou evoluční strategie a pod jejich vznik se podepsali studenti z Berlínské univerzity Rechenberg, Schwefel, Bienert publikacemi z roku 1965.[1][2][6] [33]

Právě J. H. Holland dal genetickému algoritmu jeho podobu. Fungování vychází z idejí schémat, ze kterých se staví větší celky, až se postaví výsledné řešení. Původní myšlenka se zakládala na vylepšování jedinců v populaci reprezentovaných pomocí řetězce 0 a 1. Ti pomocí přirozeného výběru a operátorů mutace, křížení a inverze vytvořili novou generaci jedinců. Rechenberg používal pouze dva jedince v populaci s operací mutace. Podobně L. J. Fogel používal pouze mutaci. Cílem Hollandova snažení bylo naučit počítače adaptovat se na okolní prostředí. [1][2][6]

V současné době prožívají GA velký nástup a s tím i různorodost. Algoritmy se používají například v neuronových sítích nebo pro tvorbu dalších programů (tzv.genetické programování). V roce 1992 John Koza ze Stanfordské univerzity vydal zajímavou publikaci, která popisovala originální modifikaci genetického algoritmu, kterou nazval genetické programování. Při tomto přístupu je původní reprezentace chromozómů znakovými řetězci nahrazena složitějšími funkcemi. V nejjednodušší verzi genetického programování se funkce rovnají výrazům obsahujícím proměnné, konstanty, základní aritmetické operace a elementární funkce.[33]

1.3 Pojmy

K podrobnějšímu popisu a vysvětlení algoritmu je dobré si říci několik biologických pojmů, které budou v jistém přeneseném slova smyslu figurovat v této diplomové práci.

1.3.1 Chromozom

Chromozom je soubor parametrů, které definují navrhované řešení problému za pomoci evolučního algoritmu. Zjednodušeně řečeno chromozom popisuje vlastnosti jedince.

V přírodě je chromozom tvořen dvoušroubovicí kyseliny deoxyribonukleové (DNA). DNA je stočena a vytváří tím prostorový útvar známý jako chromozom. Zde je chromozom souborem genů jedince. Je reprezentován numerickým nebo textovým řetězcem. Takový řetězec je složen z hodnot jednotlivých genů seřazených do posloupnosti. Chromozom nebo řetězec, který chromozom reprezentuje, může být vyjádřen například pomocí binární sekvence znaků: 1101011101.

Pro přehlednější vyjádření genetické informace lze využít jiného typu reprezentace. Geny, ze kterých je chromozom složen, mohou být zapsány ve dvojici s pořadím genu v chromozomu. První prvek udává pozici v chromozomu a druhý vlastní hodnotu genu. Řetězec pak bude vypadat například takto: [8]

INTEGER{(0; 9)(1; 6)(2; 5)(3; 7)(4; 3)(5; 9)(6; 4)}

FLOAT{(0; 0,03)(1; 1,342)(2; 6,2)(3; 7,034)(4; -3,45)(5; 9,4837)(6; 4)}

1.3.2 Gen

V souvislosti s evolučním algoritmem gen reprezentuje vlastnosti jedince. Na rozdíl od přírody je v tomto kontextu význam pojmu gen značně zjednodušen. Je reprezentován znakem, který popisuje vlastnost genu, např. genetický algoritmus nejčastěji používá pro reprezentaci genu binární hodnotu, tedy 0 nebo 1. Ale obecně může být pro zápis vlastnosti genu využit jiný zápis, např. dekadický. Lze se tedy odkázat na předchozí kapitolu, ve které byl na příkladu vysvětlován pojem Chromozom. Gen je tedy reprezentován jedním prvkem chromozomu z jeho znakové sekvence na jeho určité pozici. Na dekadickém zápise chromozomu lze vidět tučně zvýrazněný příklad genu:[8]

INTEGER{(0; 9)(1; 6)**(2; 5)**(3; 7)(4; 3)(5; 9)(6; 4)}

FLOAT{(0; 0,03)(1; 1,342)(2; 6,2)(3; 7,034)**(4; -3,45)**(5; 9,4837)(6; 4)}

Jako znázornění můžeme uvést i problém obchodního cestujícího. Obchodní cestující má za úkol navštívit předem daný počet měst a trasa mezi nimi má být co možná nejkratší nebo co možná nejrychlejší. Gen bude v tomto případě popisovat délku jednoho spojení mezi dvěma městy. Pro tento problém bude zřejmě výhodnější využít dekadického zápisu genomu.

1.3.3 Alela

Alela je konkrétní symbol nebo hodnota, která na určité pozici v chromozomu genu nabývá. Na příkladu dekadického zápisu chromozomu lze vidět gen, ve kterém je tučně zvýrazněna alela:[8]

INTEGER{(0; 9)(1; 6)(2; **5**)(3; 7)(4; 3)(5; 9)(6; 4)}

FLOAT{(0; 0,03)(1; 1,342)(2; 6,2)(3; 7,034)(4; -**3,45**)(5; 9,4837)(6; 4)}

1.3.4 Genotyp

Genotyp je souhrn všech genů jedince. Představuje tedy celou jeho genetickou výbavu. Pro evoluční algoritmus platí, že v chromozomu je obsažena tatáž informace jako v genotypu. V přírodě může genotyp obecně obsahovat větší množství informací, než které jsou obsaženy v jednom chromozomu. Například u člověka je genotyp složen ze 46 chromozomů a každý z těchto chromozomů obsahuje množství genů reprezentovaných v DNA.[8]

1.3.5 Fenotyp

Fenotyp je soubor všech pozorovatelných vlastností a znaků živého organismu. Představuje výsledek spolupůsobení genotypu, epigenetiky a prostředí čili to, jak organismus v daném znaku (znacích) nakonec skutečně vypadá. Přeneseně řečeno fenotyp je zobrazení genetické informace do konkrétního stavu na konkrétních pozicích řetězce.

Příkladem fenotypu může být grafické znázornění jedince z populace řešení „Problému obchodního cestujícího“.[7][8]

1.3.6 Jedinec

Jedinec je nositelem genetické informace a je základním prvkem populace. Je definován určitým genotypem. Každý jedinec má unikátní genetickou výbavu a v rámci GA je jedinec definován jedním chromozomem o pevné délce.[7][8]

1.3.7 Populace

Populace je množinou všech jedinců. Přeneseno na GA je populace partikulárním řešením zadaného problému.[7][8]

1.3.8 Generace

Generace je skupina jedinců, na kterou je aplikována teorie o vývoji druhů. Generace je v podstatě reprezentována jako jedna iterace *t* reprodukce jedinců populace *P(t)*.[7]

1.4 Genetické algoritmy

Genetické algoritmy jsou nejpoužívanější kategorií evolučních algoritmů. Tuto převahu je možné vysvětlit tím, s jakou jednoduchostí lze řešit jeden anebo skupinu složitých problémů. Jednotlivé populace časem jednoduše zaproximují k řešení.

Genetické algoritmy jsou založeny na myšlence darwinovského principu evoluce. Hledání optimálního řešení probíhá formou soutěže v rámci populace postupně se vyvíjejících entit. K tomu aby bylo možné posoudit, kteří členové populace mají větší šanci podílet se na dalším vývoji hledaného řešení, musí být tato schopnost individuí kvantifikovatelná. V této souvislosti se mluví o ohodnocení, míře kvality, vhodnosti či míře kvality, vhodnosti, síle či reprodukční schopnosti jednotlivce (v zahraniční odborné literatuře se označuje jako *fitness*). Jedinci s lepším ohodnocením (s vyšší hodnotou zdatnosti neboli *fitness*) mají přirozeně větší šanci přežít déle a podílet se na vytváření následující generace. Použitím rozmanitých technik křížení a reprodukce potom vznikne nová generace jednotlivců, ve které jsou vlastnosti předchůdců částečně zděděny po rodičích a částečně ovlivněny náhodnými mutacemi v procesu reprodukce. Opakuje-li se tento evoluční cyklus mnohokrát, obvykle po desítkách až stovkách opakování vznikne populace s jedinci, kteří mají vysoké ohodnocení a mohou představovat dostatečné či dokonce optimální řešení daného problému. Protože tento evoluční proces v sobě zahrnuje značný díl náhodnosti, je zřejmé, že každý běh příslušného algoritmu se bude odvíjet odlišným způsobem. Z téhož důvodu se v některých případech poměrně snadno může stát i to, že celá populace v procesu vývoje zdegeneruje a nejlepší jedinec bude reprezentovat pouze lokální optimum, které může být velmi odlišné od optima globálního. Proto se chování genetických algoritmů na konkrétní úloze obvykle popisuje prostřednictvím různých statistik, které shrnují nejlepší a nejhorší průměrné hodnoty sledovaných ukazatelů.[3]

1.5 Operátory genetického algoritmu

Operátory v oblasti genetických algoritmů představují sadu nástrojů, pomocí nichž lze z rodičovské generace získat další generaci – potomky. Pro vznik nové generace, musí nastat řada událostí. První věc, kterou jedinec před reprodukcí udělá – najde si protějšek, se kterým by jeho generace pokračovala, a jeho geny by byly zachovány. Druhá a třetí událost nastává po reprodukci - jedinci si vymění genetické informace mezi sebou a předají je svému potomkovi. Všechny tyto události mají velké množství implementací. Pro účely řešení úloh za pomoci GA

je pokaždé potřeba vybrat operátor, který je nejvhodnější v závislosti na požadovaném výsledku. [8]

1.5.1 Selekce

V biologii by se dal tento postup pospat jako rituál namlouvání. Jedinci si vybírají, jaký partner je pro ně nejvhodnější pro stvoření svých potomků. U savců většinou samec ukazuje svou dominanci nad svými vrstevníky a snaží se samici dokázat, že on je nejvhodnějším kandidátem pro reprodukci. Někteří teritoriální jedinci si své samice chrání a v případě vniknutí na jejich území a namlouvání samic z jeho harému bere jako útok na jejich právo rozmnožování. Jak bylo zmíněno už v této kapitole, vyhrají pouze ti nejsilnější. Těm slabším bude upřeno privilegium na rozmnožování a jejich genom nadobro vyhyne. Z toho vyplývá, že evoluce zajistila velice rozmanitou množinu možností výběru nejzdatnějších z dané populace. Pokud máme principy evoluce imitovat, stojíme před úkolem, jak najít k této množině alternativu, která umožní co nejlépe definovat, kteří jedinci určí správný směr evoluce.[3][8]

Operátory selekce se mimo jiné ještě dělí na:

(1) *Proporcionální metody* – platí, že pravděpodobnost výběru jedince je přímo úměrná velikosti jeho ohodnocení. Jedinec s větším ohodnocením tedy bude vybrán s větší pravděpodobností, něž jiný jedinec hůře hodnocený. Selekční metody z této skupiny pracují přímo s ohodnocením jedinců, proto nevyžaduji jejich seřazení podle velikosti ohodnocení. U proporcionálních metod se můžeme setkat s pojmem: „problém měřítka“ (scalling problem). Jedná se o stagnaci při příliš nízké hodnotě selektivního tlaku, nebo předčasnou konvergenci při příliš rychle zúženém výběru.[9]

(2) *Pořadová selekce* – od proporcionálních metod se liší v kritériu, podle něhož jsou vybíráni rodičovští jedinci. Nejprve se seřadí populace podle kvality ohodnocením jejích jedinců, těm je přiděleno číslo určující pořadí v rámci populace. Díky tomu, že následný výběr neprobíhá podle ohodnocení, ale pořadí, odpadá u této metody možnost výrazně nevyvážené pravděpodobnosti selekce jedince, kdy jeden či několik kvalitních jedinců ovládne výběr na úkor méně kvalitních individuí. Metoda může být tak jako zástupci proporcionální selekce implementována v podobě upraveného ruletového kola, či výběru na (jednotkové) úsečce. Pořadová selekce umožňuje jednoduchou kontrolu selekčního tlaku a zároveň řeší problém měřítka (scaling problem).[9]

C	*FR*	*R*	*Šance výběru*

A	5	5	33,33 %
B	2	4	26,67 %
C	0,5	1	6,67 %
D	1,5	3	20 %
E	1	2	13,33 %

Tabulka 1 Příklad pořadové selekce

Kde *C* je chromozom nesoucí vlastnosti jedince; *FR* je Fitness Rank; *R* je Rank neboli ohodnocení pořadovou selekcí (tedy od jedné až do *N*): 1 je nejhorší a *N* je nejlepší. V tabulce je názorně demonstrován příklad pořadové selekce. Systém ohodnotí jedince fitness rankem podle kvality chromozomu, který je ovšem v tabulce označován pouze jako zástupná reprezentace pro účely zjednodušení příkladu. Na základě fitness ranku jim přiřadí pořadí (takzvaný rank) a následně dopočítá procentuální šanci jejich výběru.

1.5.2 Křížení

Křížení je jeden z elementárních operátorů GA, který se dále dělí dle principu výměny genetické informace mezi chromozomy patřící rodičovským jedincům. Před použitím operátoru křížení je nezbytné použít některou z metod selekce a tím vybrat z populace několik jedinců. Vybraní jedinci se vzájemně spárují a nadále si vymění část svých genů a vytvoří nového potomka. V této práci je uveden v drtivé většině příklad, ve kterém se množí dva jedinci, nicméně je možné rozmnožovat se i mezi více jedinci.

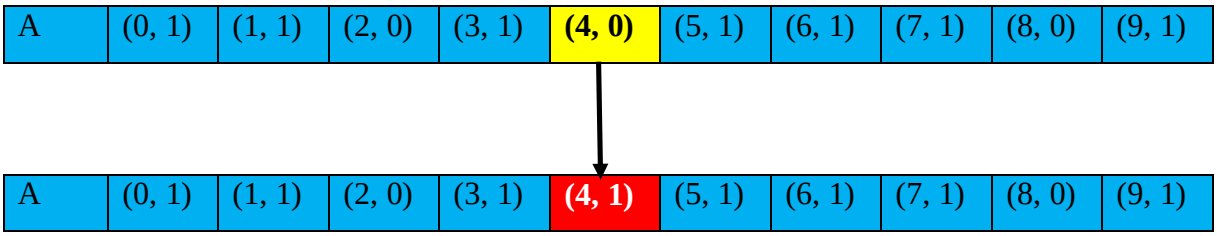
Křížení se dá vyjádřit jako procedura, v rámci které dochází k vzájemné výměně informace mezi dvěma rodičovskými chromozomy a k následnému vytvoření dvou potomků. Existuje více možností, jak vybrat jednu či více částí rodičovského chromozomu. Výměna genetické informace s partnerským rodičovským chromozomem se může také odlišovat. [7]

1.5.3 Mutace

Mutace je genetický operátor, který mění jeden nebo více hodnot genů v chromozomu jedince. Výsledkem mutace může být zanesení zcela nové genetické informace do genomu jedince. Díky mutacím lze u genetického algoritmu zabránit uvíznutí v lokálním optimu a diverzifikovat tak populaci. Jedná se o velmi primitivní operaci: změnu jednoho, či více náhodně vybraných genů v chromozomu. [7]

Výskyt mutace udává pravděpodobnost, kterou si uživatel nastaví jako argument při spouštění programu, avšak to závisí na konkrétní implementaci aplikace GA. Pravděpodobnost výskytu by měla být nastavena s velkou opatrností, příliš velká hodnota bude mít za následek degradaci GA na náhodný vyhledávací algoritmus a neúměrné snížení efektivity algoritmu. Rozumná míra pravděpodobnosti bývá od 0,001 % do 0,05 %. [7][9]

Na následujícím obrázku je znázorněn příklad mutace přepnutím bitu. Hodnota první alely v genu reprezentuje pozici neboli index genu v chromozomu. Hodnota druhé alely v chromozomu představuje hodnotu genetické informace. V tomto případě se jedná o nulu a jedničku, kde jednička reprezentuje kvalitu jedince a nula jeho nekvalitu. Nejprve se vygeneruje náhodné číslo od 0 do 9. Tyto hodnoty se získají z první alely v prvním genu a první alely v posledním genu. Vygenerované číslo je 4. Tím byla zvolena pozice, jejíž genetická informace bude logickým operátorem negace přepnuta z hodnoty 0 na 1. Výsledkem bude zanesení zcela nové genetické informace do genomu jedince.



Obrázek 1 Příklad mutace – přepnutí bitu

2 SIMULACE

Význam tohoto slova lze definovat tím, že se vysvětlí základní termíny model, simulace, systém ve vztahu k tomuto odvětví. Termíny vycházejí většinou z anglických technických slov a v češtině mohou mít i jiný význam. Je dobré držet se anglických termínů, protože většina publikací a knih se jich striktně drží.

Ve dvacátém prvním století je běžné upřednostňovat počítačovou simulaci, před rekonstrukcí reálně vypadající situace. Důvody jsou elementárně jednoduché: úspora času, šetření nákladů a nulová rizika zranění. Jako příklad lze uvést vyšetřování letecké katastrofy. Rekonstrukce takové situace by byla nesmyslná a životy ohrožující, právě v tomto případě lze použít simulátory. Ty naplníme všemi dostupnými informacemi z černých skříněk, trosek letadel, výpovědí očitých svědků a nastavíme je jako vstupní podmínky simulátoru. Po spuštění simulátoru lze sledovat, zda se letadlo chová stejně a na základě toho posoudit, zda byla příčina katastrofy odhalena a případně upravovat parametry, dokud nezískají vyšetřovatelé cenné informace. Vyšetřování katastrof je jeden z méně obvyklých způsobů, jak lze simulátory použít. Běžně se používají při výuce pilotů, řidičů, marketingové analýze a existuje dalších mnoho způsobů, jak lze simulátory uplatnit. [13]

Na základě získaných poznatků ze simulace se může člověk rozhodovat, zjišťovat, jak daný systém funguje, provádět optimalizace simulovaných procesů.

2.1 Simulace

Počítačová simulace je pokus o simulaci reálného sytému za pomoci výpočetní techniky. V obecné mluvě značí simulace předstírání nemoci, bezvědomí, duševní poruchy, bolesti apod. Profesionálně vzato, tento význam slova simulace by měl patřit někam do odvětví sociální psychologie. Pojem simulace, jak ho chápe aplikovaná informatika a jak ho chápou i ostatní obory, když aplikují výpočetní techniku, má však zcela jiný obsah. Stručně řečeno, v této oblasti je simulace chápána jako výzkumná technika, jejíž podstatou je náhrada zkoumaného dynamického systému jeho simulátorem s tím, že se simulátorem experimentuje s cílem získat informace o původním zkoumaném dynamickém systému.

Aby šlo o simulaci, musí být cílem experimentů se simulátorem snaha dozvědět se něco o simulovaném systému. Když je simulátor realizován jako výpočet na binárním počítači, může se složkou simulace stát i experimentování se simulátorem, jehož cílem je získat informaci o něm samotném, a ne o simulovaném systému: nastane to tehdy, když např. zjišťujeme, zda v příslušném programu není programátorská chyba nebo zda v něm není použita nevhodná

numerická metoda; tento proces se nazývá ověření správnosti modelu neboli verifikace modelu. [11]

Existuje několik pojmů náležící simulaci. Patří mezi ně systém, model, proměnné stavu systému (variables), entity a atributy entit, list processing, aktivity a zpoždění.

2.1.1 Systém, abstrakce

Pod pojmem **abstrakce** je ukryt převod určitého reálného objektu O_r na objekt abstraktní O_a, který obsahuje zjednodušenou verzi objektu reálného O_r. Tento nově vzniklý objekt O_a se v modelování a simulaci nazývá systém. [14]

Systém musí obsahovat zvládnutelné aspekty objektu reálného a nepotřebné aspekty jsou zanedbány. Tento problém lze také vyjádřit tak, že nad zkoumaným objektem jsou vymezovány systémy. Jako příklad systému může být supermarket s x kasami a y prodavači, kteří mají z nakupujících zákazníků.

Systém lze rozdělit podle toho, zda uvažuje existenci času:

- Statický – je systém, jenž nepřihlíží k významu času a to znamená, že pojem času zanedbává a neváže se k němu. Všechny jeho prvky mají spojité chování.
- Dynamický – je systém, jehož čas se nezanedbává a zároveň je chápán ve smyslu „Newtonovské“ fyziky. Simulace se jinými, než dynamickými systémy nezabývají, a proto se pojem statická nebo dynamická simulace téměř nepoužívají.

Podle predikovatelnosti:

- Stochastické – používají generátor pseudonáhodných čísel, nebo pravděpodobnost (např. Metoda Monte Carlo.)
- Deterministické – výsledek je opakovatelný a závisí pouze na vstupních datech a možné interakci s vnějším světem.

Okolí zkoumaného objektu obsahuje objekty, které není třeba zkoumat, ale přesto je třeba jejich existenci brát v úvahu, protože nějakým způsobem ovlivňují zkoumaný objekt. Takovéto okolí nazýváme **okolí systému**. Pod tímto okolím si lze představit například davy zákazníků vstupující do obchodního domu. [14]

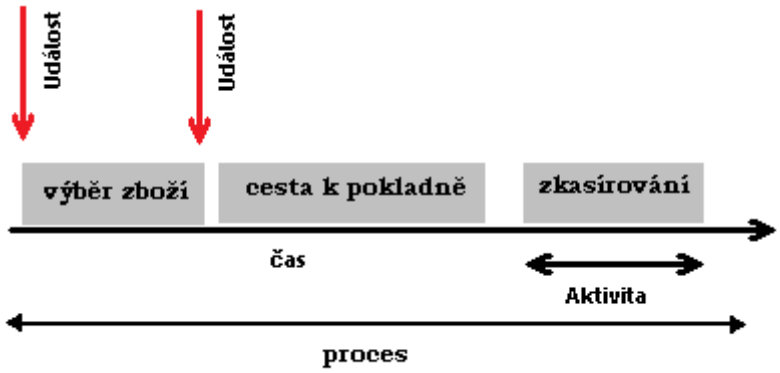
2.1.2 Model systému, událost

Model je reprezentací aktuálního systému. Termín model lze použít pro analogii mezi dvěma systémy, a to modelovaným (originálem) a modelujícím. Vztah mezi těmito modely je dán následovně: každému prvku modelovaného systému je přiřazen prvek modelujícího systému, každému atributu prvku je přiřazen atribut prvku a pro hodnoty atributů a je dána nějaká relace. V simulaci se však uplatní jen „simulační model“, jedná se o modely, které splňují následující požadavky[13][11]:

1. Jejich modelující i modelované systémy jsou dynamické systémy.
2. Existuje zobrazení existence modelovaného systému do existence modelujícího systému. Je-li t_1 časový okamžik, v němž existuje modelovaný systém M_1, je mu přiřazen okamžik $\tau(t_1)=t_2$, v němž existuje modelovaný systém M_2, a tak je zobrazením τ přiřazen i stavu $S(t_1)=\sigma_1$ systému M_1 stav $S(t_2)=\sigma_2$ systému.
3. Mezi stavy σ_1 a σ_2 jsou splněny požadavky na vztahy mezi prvky a jejich atributy, jak již bylo definováno pro modely obecně.
4. Zobrazení τ je neklesající, to znamená, že pokud v originálním systému nastane stav S_1 před stavem S_2, pak v modelujícím systému musí též nastat stav S_1 před stavem S_2 nebo alespoň ve stejnou dobu, pokud se nejedná o kvalitní systém, nikdy však nesmí stav S_2 nastat před stavem S_1.

Událost (event) se dá vysvětlit jako případ, který po výskytu změní stav systému. Existují dva druhy událostí vnější a vnitřní událost v odborné literatuře nazývané jako endogenní a exogenní událost.[12][15]

Na následujícím obrázku je popsán příklad nákupu v supermarketu. Příklad názorně ukazuje na reálné situaci, co představují jaké pojmy v systému. Zákazník přijde do obchodu a vybere si zboží, které si chce nakoupit. Zahájení výběru zboží je událost, výběr jednoho druhu zboží je událost, ukončení výběru zboží je událost. Na obrázku je událost znázorněna popiskem s nápisem event (přeloženo z anglického slova událost). Každá událost zahajuje aktivitu, která je na obrázku znázorněna popiskem aktivity (přeloženo z angličtiny jako činnost). Výběr zboží je aktivita, cesta k pokladně je aktivita, zkasírování je aktivita. Každá aktivita trvá určitou dobu. Tu vyměřují události v systému například událost zahájení výběru zboží a ukončení výběru zboží trvá určitý čas, po který tato aktivita běží. Jedna anebo několik vzájemně souvislých aktivit dává dohromady proces. Na obrázku je popsán popiskem proces. A to celé je simulační systém.



Obrázek 2 Ukázka události

2.1.3 Proměnné

Proměnné stavu systému ukládají všechny informace potřebné k definování toho, co se děje v systému, na adekvátní úrovni (tj. pro dosažení požadovaného výstupu) v daném časovém bodě. To co mohou být proměnné stavu systému v jednom případě, nemusí být stejné v jiném případě, i když fyzický systém může být stejný. Během procesu modelování se však jakékoliv opomenutí snadno projeví, ale na druhé straně mohou být zbytečně stavové proměnné eliminovány. Po určení proměnných při modelování systému, lze na jejich základě založit sledování stavů v systému spolu s atributy entit. Ty zůstávají v modelu diskrétních událostí konstantní v časových intervalech a mění hodnotu pouze v určitých předem definovaných bodech zvaných *čas události.* [12]

2.1.4 Entity systému

Systém se také skládá z entit (prvků). Entity reprezentují jak fyzické, tak i logické složky zkoumaného objektu. Entity lze dělit na:

- **Permanentní** entity – zůstávají v systému během celé doby jeho existence.
- **Temporální** entity – zůstávají v systému jenom určitou dobu

  - mohou zaniknout dvěma způsoby: 1) přímo v systému; anebo 2) přesunutím se do okolí systému (tím pro systém přestanou existovat).

  i. *Exogenní* entity – vznikají vně systému, tzn. v jeho okolí
  ii. *Endogenní* entity – vytváří se uvnitř samotného systému

Prvky (entity) rozlišujeme také na *mobilní* a *stabilní*. Stabilní entity představují většinou infrastrukturu. Mobilní entity mohou měnit polohu v průběhu simulace, buď to samy anebo být přemístěné s jinou entitou. [14][15]

2.1.5 Atributy prvků

Prvky (entity) systému mají svoje vlastnosti tzv. **atributy**, které nabývají v čase různých hodnot v případě dynamického systému. Atributy lze dále klasifikovat na standardní a referenční. Pod standardním atributem si lze představit například reálné číslo, text či booleovskou hodnotu, zkrátka to, co se v programování nazývá jednoduchý datový typ. Referenční atributy přiřazují prvku ukazatel na jiný prvek, tedy mezi prvky definují vazby (vztahy, relace).

Nakonec je ještě třeba si definovat stav dynamického systému v čase *t*, zkráceně **stav systému**. Stav systému je dán prvky, které existují v systému v čase *t* a hodnotami jejich atributů.[14]

2.1.6 Proces

Proces je posloupnost přirozeně na sebe navazujících aktivit, které spolu tvoří jistý logický celek. Pro názornost lze proces znázornit graficky (viz obrázek v kapitole 2.1.2). V grafu je vidět, že uvedený proces se skládá z tří aktivit (a_1, a_2, a_3). Po startu procesu *P* se nejprve vykoná aktivita a_1, po její ukončení aktivita a_2 a na konec aktivita a_3, jejíž ukončení se chápe zároveň jako konec procesu *P*. [15]

2.1.7 Aktivity

Aktivita je základní akční jednotkou simulace, která je obrazem jedné činnosti v simulovaném systému. Přičemž pro ni platí následující:

- má určité časové trvání,
- (potencionálně) mění stav systému.

Běh simulačního systému je představovaný vykonáváním jednotlivých aktivit, a to ve stejném pořadí, ve kterém se vykonávají jim odpovídající činnosti v simulovaném systému.

Pro implementaci každé individuální aktivity a v rámci simulačního programu je principiálně možné využít různé techniky, z kterých uveďme aspoň dvě základní: (i) technika ***interogativního plánování*** uplatňuje od okamžiku zahájení aktivity *a* periodické testování splnění dané ukončovací podmínky této aktivity, (ii) technika ***imperativního plánování*** je založená na realizaci „čekání“ od časového okamžiku *t* zahájení aktivity a až do daného časového okamžiku $t+\Delta t$.[15]

Tak jako můžeme mluvit o časové existenci systému, můžeme mluvit i o časové existenci aktivity jako důležitého prvku simulovaného systému. Časová existence aktivity je charakterizovaná množinou reálných čísel (časových okamžiků), ve kterých aktivita existuje, čímž může měnit stav systému. Z tohoto hlediska rozeznáváme dva druhy aktivit (i) ***spojitá*** (ii) ***nespojitá***.

Spojitá aktivita může měnit stav systému po celou dobu jejího trvání. V takovém případě je časová existence aktivity charakterizovaná intervalem reálných čísel $<t_1,t_2>$. Spojité aktivity jsou typicky realizované pomocí interogativního plánování.

Diskrétní aktivita je charakterizována tím, že může změnit stav systému jen v okamžiku ukončení aktivity; před tímto okamžikem diskrétní aktivita stav systému změnit nemůže. Tuto situaci můžeme vidět i tak, že aktivita existuje jen v okamžiku jejího ukončení a tedy její existence je charakterizována jednoprvkovou množinou reálných čísel $\{t_2\}$. Ukončení diskrétní aktivity a tím vyvolávanou změnu stavu systému nazýváme **událost** (*U*).

Výskyt diskrétních (resp. spojitých) aktivit v simulovaném systému výrazně ovlivňuje charakter simulačního modelu a vede k rozeznávání některých typů simulace[15]:

- V případě že simulovaný systém obsahuje jenom spojité aktivity, hovoříme o spojité simulaci.
- Pokud simulovaný systém obsahuje jenom diskrétní aktivity, mluvíme o diskrétní simulaci.
- V případě že simulovaný systém obsahuje spojité i diskrétní aktivity, tak příslušnou simulaci označujeme jako kombinovanou (diskrétně spojitou) simulaci.

Příklad aktivity

Pohyb dopravního prostředku po základním volném úseku dopravní infrastruktury, obslužná činnost klienta při obsluze klienta v bance, zpracování výrobku na jednom pracovišti výrobního podniku apod.

Spojitá aktivita:

Přesun vlaku z pozice A do pozice B v případě, že je respektována dynamika pohybu v závislosti na vlastnostech vlaku a venkovních podmínek.[15]

Diskrétní aktivita:

Přesun vlaku z pozice A do pozice B v případě, že nemáme důvod podrobně sledovat dynamiku pohybu vlaku. Za předpokladu, že dopředu známe pevnou dobu přesunu (to jest charakteristickým znakem diskrétní aktivity) a zajímavá je pouze událost ukončení přesunu (pro tento případ je tedy možno aplikovat techniku imperativního plánování).[15]

2.1.8 Scéna a scénář

Před spuštěním simulačního programu musíme nejprve konfigurovat **scénu**, na které se bude odehrávat simulační děj. Pod scénou rozumíme množinu všech permanentních prvků (entit) systému s hodnotami jejich atributů. Poté musíme zadat pravidla generování a zániku temporálních prvků (jsou to zpravidla zákazníci vyžadující obsluhu). Nakonec musíme vybrat a zadat algoritmy obsluhy zákazníků. Tím se pro jedno spuštění simulačního programu specifikuje dynamické chování systému. [15]

Scénář je popis dynamického chování systému pro jedno spuštění simulačního programu a je tedy určený[15]:

- scénou se svými vlastnostmi,
- pravidly vstupu, výstupu, generování a zániku temporálních prvků,
- rozhodovacími a řadicími algoritmy popisujícími procesy obsluhy.

2.1.9 Parametrizace simulačního modelu

Běh simulačního programu podle jediného definovaného scénáře je nazýván ***simulační pokus*** (experiment) se simulačním modelem. Ze simulačního pokusu se dozvíme, jak by se choval systém při použitém scénáři (Co se stane když…?). Před spuštěním simulačního experimentu musí experimentátor ještě nastavit parametry samotného simulačního modelu platné pro toto spuštění (tj. **parametrizace** simulačního modelu). Parametrizuje se zde například doba trvání simulačního experimentu, režim interakce s uživatelem, výběr sledovaných výstupních statistik a mnoho dalších parametrů. Jak je z předchozího patrné,

scénář definuje množinu hodnot **vstupních parametrů** a výsledkem experimentu je množina **výstupních hodnot** charakterizujících chování systému. Vstupními parametry mohou být hodnoty proměnných, datové soubory ale i deklarativně (například grafickou formou) či procedurálně vyjádřené algoritmy. Datové vstupní parametry nemusí být deterministické, naopak v typických případech je mnoho vstupních parametrů představováno náhodnými proměnnými. V takovém případě jsou samozřejmě příslušné výstupní hodnoty realizacemi náhodných proměnných (výstupních náhodných proměnných). Proto je nutné realizovat celou sérii pokusů (tzv. replikace pokusu) se stejným scénářem ale s různými instancemi hodnot vstupních náhodných proměnných. Přirozeně na výstupu také dostaneme pro každou replikaci množinu výstupních hodnot instancí – výstupních náhodných proměnných. Tyto hodnoty se potom statisticky zpracují například výpočtem aritmetického průměru odpovídajících hodnot a ze všech replikací se odhadne střední hodnota výstupní náhodné proměnné. Simulační pokus potom tedy nedává informaci o např. délce fronty, ale o střední hodnotě délky fronty.[15]

2.2 Základní rozdělení

Přesnější vyjádření charakteru simulovaného systému lze dělit na několik skupin:

1. Spojitá simulace
2. Diskrétní simulace
3. Diskrétně-spojitá simulace (kombinovaná simulace)

Jestliže simulovaný systém je spojitý, jestliže se hodnoty jeho atributů mění v čase jen spojitě, mluví se o **spojité simulaci** (continuous simulation). Jestliže je simulovaný systém diskrétní, tj. nastávají v něm nespojité změny v čase, mluví se o **diskrétní simulaci** (discrete event simulation). Je-li simulovaný sytém tak říkajíc kombinovaný, to jest má-li jak vlastnosti typické pro spojité systémy, tak vlastnosti typické pro diskrétní systémy, mluví se o kombinované **diskrétně-spojité simulaci** nebo častěji o kombinované simulaci.[11]

Laicky se dá říci, že nejzásadnější rozdíl mezi spojitou a diskrétní simulací je jejich spojitost (resp. diskrétnost) aktivit. Aktivita je základní stavební jednotkou simulace, představuje činnost, kterou entity v simulátoru vykonávají, každá aktivita trvá svou danou dobu simulačního času a během změny simulačního času se mění stav simulace. U spojitých aktivit může nastat změna stavu kdykoliv při vykonávání aktivity, u diskrétní aktivity je změna stavu až při skončení činnosti, kde je vyvolaná změna stavu systému, která se nazývá událostí. Vykonáváním jednotlivých aktivit v určitém pořadí vytvářejí průběh simulačního programu. [13]

2.2.1 Spojitá simulace

Systém můžeme nazvat systémem se spojitým chováním v případě, že všechny jeho prvky se chovají spojitě. Systém je spojitý, pokud se hodnoty jeho atributů mění v čase pouze spojitě. Původní stav modelu se mění v nový vždy spojitě bez ohledu na to, čím byla změna vyvolána. [23]

U těchto simulací se přestáváme zajímat o jednotlivé události, ale snažíme se modelovat systém jako celek. Informace o něm ukládáme do stavových proměnných, které průběžně měníme. Jejich změnu určujeme řešením soustav časových diferenciálních rovnic. V drtivé většině případů se rovnice nedají vyřešit analyticky, tudíž se řeší numericky. [24]

Tento typ modelování se nejvíce využívá ve fyzice, kde s jeho pomocí můžeme ze zadaného počátečního stavu odhadovat jeho vývoj v budoucnu, jako například předpovědi v meteorologii, geologii nebo astronomii.[24]

2.2.2 Diskrétní simulace

V některých případech nás však zajímají pouze změny stavu systému, ke kterým dochází při výskytu určitých událostí, a změny, ke kterým dochází v reálném systému mezi těmito událostmi, jsou z hlediska účelu zkoumání daného systému nezajímavé. V takovém případě tedy pomocí modelu zaznamenáváme pouze tyto relevantní události a časové okamžiky, ve kterých nastávají. Potom hovoříme o diskrétní simulaci. Tento modelový přístup často nachází uplatnění při zkoumání různých ekonomických jevů, podnikových procesů a činnosti systémů hromadné obsluhy.[19]

V literatuře zabývající se diskrétní simulací se můžeme setkat například s pojmy procesně orientovaný přístup a událostně orientovaný přístup. Oba tyto přístupy vycházejí ze stejné myšlenky, kterou je časový krok proměnlivé délky. Smyslem časového kroku proměnlivé délky je přeskočení časového úseku, během kterého v modelu nenastala žádná událost, tedy vybrání minima ze všech v danou chvíli známých časů výskytu událostí, které mají nastat a posun do tohoto času. V tomto čase je potom třeba zpracovat všechny události, které se mají v daném okamžiku odehrát, pokud je to možné, vygenerovat nové časy výskytu těchto událostí a vrátit se ke kroku, ve kterém vybíráme minimální čas.[19]

2.2.3 Stochastická simulace

Stochastická simulace je simulace systému, který v sobě má proměnné, které se mohou měnit stochasticky (náhodně) s individuálními pravděpodobnostmi. Realizace těchto náhodných proměnných se generují a vkládají do modelu systému. Výstupy modelu se zaznamenávají a pak se proces opakuje s novou sadou náhodných hodnot. Tyto kroky se opakují, dokud není shromážděno dostatečné množství dat. Nakonec distribuce výstupů ukazuje nejpravděpodobnější odhady a rámec očekávání ohledně toho, v jakém rozsahu hodnot se proměnné budou více či méně pravděpodobně snižovat. Nejběžnějším příkladem implementace této skupiny simulátorů je metoda Monte Carlo.[21]

2.2.4 Deterministická simulace

V matematickém modelování deterministické simulace neobsahují žádné náhodné proměnné ani žádný stupeň náhodnosti a skládají se většinou z rovnic, například diferenciálních rovnic. Tyto simulace mají dopředu známé vstupy a vedou k jedinečné sadě výstupů v kontrastu se stochastickou simulací, která zahrnuje pouze náhodné proměnné. Deterministická simulace je obvykle navržena tak, aby zachytila základní mechanismus nebo přirozený proces. Liší se od statistických modelů (například lineární regrese), jejichž cílem je empiricky odhadnout vztahy mezi proměnnými. Deterministický model je považován za určitou aproximaci reality, kterou je jednodušší vytvořit a interpretovat než stochastický model. Takové modely však mohou být velmi komplikované s velkým počtem vstupů a výstupů. Jediná fixovaná sada výstupů může být generována mnoho sadami vstupů. Z tohoto důvodu je dobré zohlednit nepřesnosti parametrů a modelů rozhodující o průběhu simulace možná ještě více než u standardních statistických modelů.[21]

2.2.5 On-line simulace

Tento typ je používán pro simulace s krátkodobým plánováním a předpokládá, že je nutné v reálném čase aplikovat vhodnou variantu plánu, která zabezpečí optimální provoz příslušného systému. Pokud je potřeba operativně řídit nebo plánovat, lze využívat online simulace, která se od klasické off-line simulace liší tím, že:[22]

- Vybudovaný simulační model je přímo napojený na reálný systém. To znamená, že hodnoty jeho vybraných stavových proměnných jsou aktualizované v reálném čase. To může být například informační nebo dispoziční systém.

- Časové trvání simulačních pokusů, které odrážejí různé varianty evoluce procesů reálného systému, musí být podstatně rychlejší nežli evoluce odpovídajících procesů v realitě, aby doporučení vyplývající z jejich realizací byly ještě aplikovatelné na aktuální situaci v reálném systému.

Všechny výpočty se musejí stihnout za velmi krátký časový úsek, a proto vyvstává mnoho problémů, které se musí řešit při návrhu systému. V systému nelze realizovat velký počet nebo prověřovat velký počet provozních variant a z nedostatku času také nelze realizovat dostatečný počet replikací apod. [22]

Online simulaci lze aplikovat jako přirozenou podporu pro operativní plánování nebo řízení. Obecně lze konstatovat, že platný provozní plán spojený s příslušným reálným systémem je nutné modifikovat jednou z následujících metodik: [22]

- Reaktivní plánování – plánování založené na realizacích simulovaných provozních výhledů. Je prováděno v případě, že platný provozní plán se natolik rozchází s realitou, že již není použitelný.
- Iniciativní plánování – kontinuální provádění úprav platného plánu v případě, kdy se v blízké budoucnosti předvídá problém.

2.2.6 Off-line simulace

Pro potřeby strategického a taktického plánování je tradičně využívána off-line simulace. Ta je charakterizována realizacemi umělých vstupů do simulátoru. Tento typ simulace není napojen na reálný systém, ale většinou na historická data nebo na generátor pseudonáhodných čísel. Hlavním cílem je navrhnout optimální nebo alespoň racionální částečně optimální řešení pro zkoumaný systém z dlouhodobého hlediska. Důležitým faktorem úspěchu nalezení optimálního řešení je sběr a analýza vstupních dat nad historickými nebo prognostickými soubory dat. Nelze přehlédnout ani analýzu a sběr výstupních dat, které jsou realizovány archivací výstupních dat ze simulačních experimentů a jejich následné statistické zpracování.

Využitím off-line simulace umožňuje v rámci promyšlené série simulačních experimentů pohodlně prověřovat chování zkoumaného systému bez nutnosti se rozhodovat o praktické realizaci některé z prověřovaných variant zkoumaného systému v reálném čase. [22]

2.3 Metody synchronizace simulačního výpočtu

Během simulace se neustále mění simulační čas, který je odrazem času v reálném světě. Pro aktivity běžící v simulátoru je potřeba zajistit, aby závisely na simulačním čase. Simulátor se musí držet časové kontinuity a vždy musí plynout kupředu a nikdy zpět.

2.3.1 Metoda snímání aktivit

Metoda snímání aktivit je metodou univerzální, která je použitelná pro realizaci jak spojité, tak i diskrétní simulace. Tato metoda se též někdy označuje za dvojfázovou metodu. Přístup metody snímání aktivit se označuje jako synchronní. Její princip spočívá ve snímání všech právě běžících (registrovaných) aktivit v simulátoru. Snímání probíhá v obvykle pevně daných přírůstcích simulačního času. Časové přírůstky jsou označovány jako snímací perioda. Během snímání každé aktivity dochází k jejímu vyhodnocování, tj. testuje se, zda v simulačním čase $t_s = n \times \tau$ (kde n vyjadřuje počet snímacích period od počátku simulace) došlo:[14]

- v případě diskrétní simulace ke splnění výskytu její koncové události U
- v případě spojité simulace ke změně hodnot příslušných dynamických atributů a případně ke splnění aktivační podmínky (například k dosažení dané prahové hodnoty atributy nebo výskytu interakce mezi aktivitami) pro vykonání specifické operace.

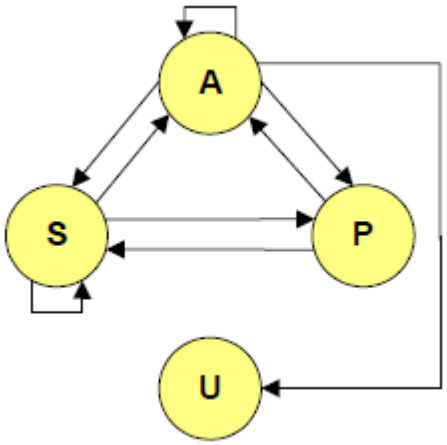
Rozepsaný algoritmus metody snímání aktivit lze najít v příloze G.

2.3.2 Metoda plánování událostí

Metoda plánování událostí je jednou z nejrozšířenějších metod pro realizaci diskrétní simulace. Přístup metody plánování událostí se označuje jako asynchronní. Jak již bylo dříve definováno, ukončení diskrétní aktivity se nazývá událost. Princip této metody spočívá v plánování událostí do budoucnosti. Jelikož u diskrétní simulace je charakteristické, že doba trvání každé aktivity je předem známa, určí se pro každou aktivitu událost, která bude představovat její konec. Tato událost může vytvořit či odstartovat jinou aktivitu. Informace o příslušných událostech je nutné uchovávat v kalendáři událostí nebo časové ose, a to od počátku vytvoření až do jejího výskytu. Kalendář událostí je typicky realizován ADT prioritní frontou. Rozepsaný algoritmus metody plánování události lze najít v příloze G. [14]

2.3.3 Metoda interakce procesů

Metoda interakce procesů je metoda pro realizaci diskrétní simulace. Převážně vychází z předchozí metody plánování událostí. Každý proces je zde rozdělen na jednotlivé aktivity s ním spojené. Pokud je proces aktivován, provede se pouze aktuální aktivita procesu spojená se změnou stavu systému.



Obrázek 3 Metoda interakce procesů

Na výše uvedeném obrázku je zachycena metoda interakce procesů respektive stavový přechodový diagram. Proces se může nacházet v jednom ze čtyřech stavů. Stav je na obrázku ilustrovaný žlutým kolečkem s počátečním písmenem názvu stavu. Šipky symbolizují směr přechodu stavu, jakým může přejít z jednoho stavu procesu do druhého a dokonce se i ze sebe samého. Nelze přejít opačným směrem, než ukazuje šipka. Procesy se tedy mohou nabývat následujících stavů:[20]

1. aktivní – proces je právě obsluhován; je prováděn výpočet
2. ukončený – proces je ukončený; již nebude obsluhován
3. suspendovaný – vykonávání procesu je naplánováno; proces čeká
4. pasivní – proces čeká; jeho vykonávání není naplánováno.

Rozepsaný algoritmus metody interakce procesů lze najít v příloze G.

2.3.4 Třífázová metoda

Třífázová metoda je metoda pro realizaci diskrétní simulace. Na rozdíl od ostatních synchronizačních metod, metoda třífázová rozlišuje dva druhy aktivit: ***plánované***, které se plánují předem a čas jejich vykonávání je předem znám a aktivity ***podmínkové***, jejichž výskyt

je dán určitou aktivační podmínkou. V každém simulačním kroku se nejprve obslouží aktuální plánovaná událost, tedy událost s nejnižším časem výskytu t_u. Později je testováno, zda nebyla splněna podmínka pro některou z čekajících podmínkových aktivit. Pokud ano, tak se podmínková aktivita vykoná. Nejčastějším případem je, že podmínkové aktivity čekají na uvolnění zdrojů aktivitami plánovanými, tím se lze vyhnout soupeření dvou aktivit o stejný druh zdroje. Rozepsaný algoritmus metody interakce procesů lze najít v příloze G.[20]

3 AUTONOMNÍ VÝVOJ JEDINCE V SIMULÁTORU

Kapitola se zabývá myšlenkou samovolného vývoje entit v řízeném prostředí a vysvětlením klíčových prvků této problematiky. Konkrétně jsou nejprve objasněny pojmy a význam řízeného prostředí, které je řídícím nástrojem celé simulace. Navazuje popis evolučního algoritmu v řízeném prostředí. Oddíl končí disputací o kauzalitě evolučního algoritmu.

3.1 Řízené prostředí simulace

K tomu aby naplňoval systém svoje cíle, používá řízené prostředí simulace a svoje prostředky. Simulace se používá v mnoha kontextech, jako technologie pro ladění nebo optimalizaci výkonu, bezpečnostní inženýrství, testování, školení, vzdělávání a videohry. Počítačové experimenty se často používají ke studiu simulačních modelů. Simulace je také používána s vědeckým modelováním přírodních systémů nebo lidských systémů k získání vhledu do jejich fungování jako v ekonomii. Simulaci lze použít k ukázání případných skutečných účinků alternativních podmínek a postupů. Simulace se také používá, když skutečný systém nemůže být zapojen, protože nemusí být přístupný, nebo může být nebezpečné nebo nepřijatelné zapojit ho nebo je navržen, ale ještě není postaven anebo prostě nemůže existovat. Mezi klíčové problémy v simulaci patří získávání platných zdrojů informací o příslušném výběru klíčových charakteristik a chování, použití zjednodušujících aproximací a předpokladů v rámci simulace, přesnost a validita výsledků simulace. V tomto ohledu řízené prostředí ovlivňuje dění a vývoj evolučního algoritmu v simulaci. Simulace je velmi dobrý nástroj, velice univerzální, avšak v mnoha ohledech nesouvisí s evolučními algoritmy.

Případným spojením simulace a evolučních algoritmů lze uvažovat o hned několika vyvstalých problémech, které tímto spojením vznikají. Entity v evolučních algoritmech zanikají vždy po vzniku nové generace. Evoluční algoritmus provede selekci dvojic za pomoci fitness funkce a následně z těchto jedinců a jejich potomků vytvoří novou generaci jedinců a tento proces se neustále dokola opakuje. Simulační entity jsou v tomto ohledu velmi odlišné. Entity v systému reprezentují fyzické nebo logické složky zkoumaného objektu a dají se rozlišovat na entity permanentní (=zůstávají v systému během celé doby jeho existence) a temporální (=vyskytují se v systému jenom určitou dobu). Z toho lze usuzovat, že vznik nebo zánik entit v simulačních systémech je závislý na čase. Entity vznikají v daný simulační čas a také i v daný simulační čas zanikají. Na druhou stranu entity v evolučním algoritmu vznikají po každé nové generaci a umírají spolu s ní a čas této události není znám a čas ani není pro vznik (ev. zánik) důležitý. Tím tedy nastává problém, že evoluční algoritmy jsou do jisté míry závislé na svých

nových generacích a simulace jsou závislé na svém simulačním čase. Bez úprav, tak jak jsou entity navrženy nyní, jsou tyto dvě technologie nekompatibilní. K tomu aby byli propojitelné je potřeba naplnit několik požadavků:

1. Entita má autonomní chování nezávislé na generacích.
2. Alespoň jeden z atributů entity musí být závislý na čase.
3. Řízené prostředí simulace poskytne entitě nástroje k tomu, aby mohla jednat a případně ovlivňovat okolní entity. Nástroji se rozumí minimálně ohodnocení jedinců, výběr partnera do páru, křížení a rozmnožování.
4. Entity mohou existovat pouze ve vyměřenou dobu (tj. entity jsou temporální). Řízené prostředí zajistí jejich alokaci a dealokaci ve správný čas.
5. Časová kontinuita simulace musí být dodržena.

Řízené prostředí tedy ovlivňuje veškeré dění a vývoj evolučního algoritmu. Jeho účelem v simulátoru je sledovat veškeré stavy atributů, poskytovat entitám potřebné prostředky a nástroje (například fitness funkce, selekce, křížení a podobně), postarat se o včasnou alokaci a dealokaci entit. V praxi to znamená, že simulátor poskytne časovou značku kdykoliv jí entity potřebují, srovnává jejich atributy závislé na čase, poskytne jim nástroje a nechá entity rozhodovat o jejich načasování. V poslední řade zaznamená potřebné statistické údaje, aby se mohli vyhodnotit výstupy průběhu simulace.

3.2 Evoluční algoritmus v řízeném prostředí

Z popisu řízeného prostředí je patrné, že v simulátoru je evoluční algoritmus odlišný od toho klasického z první kapitoly. V řízeném prostředí simulátoru je téměř nezbytné, aby se každá entita vyvíjela paralelně nebo alespoň pseudoparalelně k ostatním entitám v systému. Vysvětlení je prosté, entita se pak bude chovat v systému realističtěji a bude autonomní. Dále je nutné, aby entita měla atributy závislé na čase. Jako příklad „časových“ atributů lze uvést zdroje jedince, které si vytváří podle toho, kolik simulačního času uběhlo. Pokud bude entita v systému nějakým způsobem jednat, je nutné, aby to vykonávala za pomoci svých nástrojů. Nástroje mohou nabývat do různých podob, ale nejčastěji se vyskytují formou záměrů. Ty jsou vlastně podmíněné proměnné – buď jedinec má záměr tuto činnost vykonávat, nebo nikoliv. Základními dvěma záměry musí být vždy: (i) výběr nejvhodnějšího jedince (ii) rozmnožování se. Jestliže se pár entit rozmnoží, přidá do systému novou samostatnou entitu. K tomu aby si mezi sebou entity mohly vyměňovat informace o svém stavu nebo požadavku, musí si o tom

posílat zprávy. Jakmile obdrží entity zprávu, vyhodnocují ji a případně na ní reagují posláním další zprávy. Populace se rozšiřuje částečně v závislosti na schopnosti interakce s ostatními entitami.

Postup evoluční algoritmu v řízeném prostředí simulátoru bude vypadat následovně:

1. Nastav hodnotu simulačního času $t=0$ a hodnotu konečného simulačního času t_k.
2. Náhodně vygeneruj počáteční populaci *P(0)*; vykonej kroky číslo 3 a 4 pro všechny jedince v populaci *P(0)*; pokračuj krokem číslo 6.
3. Přidej do systému jedince a uveď jej do chodu.
4. Inicializuj počáteční hodnoty jedince; nastav hodnotu záměrů *I(e)* na *false*; nastav hodnotu existenčního času entity te_k.
5. Nastav hodnotu aktuálního času entity t_e ; pokud je $t_e > te_k$ odstraň jedince ze systému.
6. Přijmi zprávy od ostatních jedinců *M*; nastav hodnotu časových atributů.
7. Zpracuj a vyhodnoť záměry entit *I(e)*; pokud vznikne nový jedinec *J*, pokračuje krokem číslo 3.
8. Nech rozhodnout se jedince o jeho záměrech a nastav hodnotu *I(e)* podle jejich rozhodnutí.
9. Vyhodnoť všechny příchozí zprávy *M*.
10. Nastav hodnotu simulačního času *t*; pokud je $t < t_k$, ukonči simulaci a vrať výsledky simulace; jinak pokračuj krokem číslo 5.

3.3 Fitness funkce

Fitness funkce je zvláštní typ objektivní funkce, která se používá jako souhrnné vyjádření zásluhy o tom, jak blízko je dané konstrukční řešení k dosažení stanovených cílů. Fitness funkce se používají v genetickém programování a evolučních algoritmech pro vedení simulací směrem k optimálním návrhovým řešením. Zejména v oblasti genetického programování a evolučních algoritmů je v této kapitole představováno jako konstrukční řešení z pole různých prostředků. Původní myšlenkou evolučního algoritmu je, že po každém kroku průběhu simulace se odstraní nejhorší entity z celé populace. V řízeném prostředí simulace je to velice podobné, ale s tím rozdílem, že generace nejsou krokově závislé, avšak na místo toho využívají nástroje prostředí, aby byli jedinci co nejúspěšnější. Fitness funkce v řízeném prostředí ovlivňuje vývoj populace tedy jiným způsobem nežli v klasickém evolučním algoritmu.

Každá entita má své prostředky, nástroje, aby ovlivnila své chování a tak zapříčinila vývoj populace. Za vhodného jedince je považován ten, který má nejvíce kvalitních prostředků. Například v této práci je nejvhodnější jedinec v populaci ten, který umí vygenerovat nejvíce peněžních zdrojů a tím pádem i nejvíce svých potomků. Zároveň platí, že každý jedinec má své vlastní prostředky, kterými ovlivňuje vývoj populace. Například prostředky pro odstranění konkurenčního jedince.

3.4 Kauzalita GA

Obyčejně v evolučních algoritmech čas nehraje žádnou roli a nemá pro ně žádný význam a je spíše přenesen do podoby jednotlivých generací. Nicméně pokud uvažujeme o vývoji evolučního algoritmu v řízeném prostředí, tak se z času náhle stává jeho základní jednotka. Entity jsou v takovémto systému pouze dočasné, respektive existují pouze po určitou dobu. Pro entitu to znamená, že její čas pro reprodukci je limitován a pokud se nestihne rozmnožit za tuto dobu, její geny zanikají. Na druhou stranu se může stát, že se rozmnoží i s desítkami různých entit najednou. Páry rodičů jsou vybírány sekvenčně, tj. vyberou se postupně všechny páry a pak je stvořena nová generace. Zatímco u řízených EA je vybrán rodičovský pár a vzápětí je stvořen nový jedinec a s ním i nová generace. Entity ve svém vymezeném čase jednají za pomoci nástrojů, které jim poskytlo prostředí a snaží se tak ovlivnit dění v systému. Příkladem takového nástroje je operace výběru partnera a zplození nového jedince.

Zatímco v klasickém evolučním algoritmu se pracuje s populacemi shodných jedinců, tak v řízeném prostředí simulace, se operuje s populací unikátních jedinců. Každé individuum má jedinečné parametry, z čehož vyplývá, že populace skládající se s takových jedinců se drží blíže k přírodním zákonitostem a přirozenému výběru nežli pravidlům klasického evolučního algoritmu. Evoluce je pak dána přenosem svých parametrů na potomky. Genetické operátory křížení, mutace a selekce jsou pak nástroji v rukou jedince pro vznik nové generace. Simulace takového vývoje je pak chápána spíše jako evoluce agentů v simulačním prostředí, kteří mohou být reprezentováni mimo jiné i pomocí konečných automatů s tím rozdílem, že nevyvíjím konečný automat, ale jeho parametry. [32]

4 ANALÝZA ŘEŠENÉHO PROBLÉMU

Analýza problému v softwarovém inženýrství je jedna z prvních a zásadních kroků při vývoji nového produktu. Je nutná při rozhodování o potřebách a podmínkách, které jsou kladeny na nový produkt. Lze si jen těžko představit dobře vedený projekt bez řádné systémové analýzy. V rámci této kapitoly bude právě taková analýza provedena a její postup bude podrobně popsán.

4.1 Analýza požadavků

Analýza požadavků se obyčejně provádí v následujícím sledu aktivit: [26][27]

1. Sběr požadavků: komunikace se zákazníky a uživateli za účelem získání jejich požadavků na systém.
2. Analýza požadavků: identifikování nejasných, nekompletních, nebo protichůdných požadavků a následné řešení těchto nesrovnalostí.
3. Zaznamenání požadavků: dokumentování požadavků v různých formách, jako běžný textový dokument, případy užití (use case), nebo specifikace procesů.

Vzor zápisu požadavků je vypracován v následující tabulce. Praktickou ukázku zaznamenávání požadavků si lze názorně demonstrovat na případu této diplomové práce. Obyčejně se jednotlivé požadavky zanesou přehledně do tabulky, ve které lze v případě potřeby snáze požadavek dohledat. Sloupec „Označení“ je jednoznačný identifikátor požadavku, se kterým je možné na požadavek zkráceně odkázat a také umožňuje rychlejší orientaci. Sloupec „Požadavek“ je celá formulace požadavku vyčerpávajícím způsobem.

Označení	Požadavek
INTRO 1	Účelem této podkapitoly je stanovit požadavky na simulační systém pro rozvoj evolučních algoritmů, zejména vytvořit simulační jádro používané pro analýzu vývoje populace a experimentování s populací.
INTRO 2	Předmětem dodávky je poskytování všech příslušných služeb týkajících se simulačního systému pro rozvoj evolučních algoritmů, včetně dodávky, otestování systému a školení za podmínek stanovených v této technické specifikaci.
INTRO 3	Následující standardní terminologie se používá k označení stavu požadavků v této požadavkové specifikaci: **Musí** – označuje požadavek, který je povinný **Měl by** – označuje požadavek, který je žádoucí **Může** – označuje požadavek, který je volitelný, nepovinný

Tabulka 2 Úvodní specifikace

Funkční požadavky

Požadavky lze obvykle dělit do dvou prioritních skupin: ty co vyjadřují nějakou funkci a ty co nikoliv. Funkční požadavky objasňují, co se musí udělat a identifikuje nutné operace, aktivity a akce, které musí být vykonány. Jakmile analýza funkčních požadavků skončí, bude použita jako základ pro další funkční analýzu. V tabulce jsou vypsány všechny funkční požadavky, které byli kolektovány rozhovorem od zákazníka.

Označení	Požadavek
SYSTEM 1	Systém **musí** simulovat vývoj populace v interním simulačním čase
SYSTEM 2	Systém **musí** zobrazovat výsledky průběhu simulace na grafický nebo tabulkový výstup
SYSTEM 3	Systém **musí** splňovat základní principy vytváření nových generací podle teorie o evolučních algoritmech
SYSTEM 4	Systém **musí** splňovat základní principy diskrétní simulace
SYSTEM 5	Systém **musí** používat statistické a numerické matematické metody
SYSTEM 6	Systém **musí** umožňovat rozmnožování jedinců mezi dvojicemi entit
SYSTEM 7	Páry entit **mohou** být stejné po celou dobu jejich života
SYSTEM 8	Systém **musí** umožňovat nastavení velikosti vstupní množiny
SYSTEM 9	Systém **musí** být schopen ovlivnit délku života každé entity na začátku simulace ve vstupní množině dat
SYSTEM 10	Systém **musí** být schopen zpracovávat alespoň 500 entit v jednu časovou jednotku simulace
SYSTEM 11	Systém **musí** umožnovat entitám běžet paralelně k ostatním entitám a mít vlastní prostředky, které budou odděleně od zbytku entit
SYSTEM 12	Systém **musí** umožňovat entitě asynchronně komunikovat s ostatními entitami – přijímat a odesílat požadavky
SYSTEM 13	Systém **musí** umožňovat entitě zpracovávat, vyřizovat a odpovídat na požadavky ostatních entit
SYSTEM 14	Systém **musí** umět poskytovat čas, s vysokou přesností v řádu nanosekund
SYSTEM 15	Entita **musí** umět pracovat se základní simulační časovou jednotkou
SYSTEM 16	Systém **musí** umožňovat entitě příjem a výdej zdrojů alespoň za jednu simulační časovou jednotku
SYSTEM 17	Příjem zdrojů **musí** být ovlivněný talentem entity
SYSTEM 18	Entita **musí** být schopna rozhodovat se o svých záměrech, které bude v daném časovém úseku vykonávat
SYSTEM 20	Entita sama o sobě **musí** být schopna rozhodnout, kdy se má ukončit svůj životní cyklus, následně potom **musí** dealokovat prostředky

Tabulka 3 Požadavková specifikace na funkce

Nefunkční požadavky

Požadavky, které nedefinují služby a operace jsou takzvaně nefunkční. Ty stanovují omezení, jež jsou kladené na proces vývoje či na systém samotný. Tabulka ukazuje, jaké

nefunkční požadavky bude systém mít. Některé z nich jsou navrženy člověkem, který si říká software architekt. Ten je odvodí z funkčních požadavků. Software architekt je zkušený programátor, který zná technologie a ví, jaká mají omezení a k čemu se hodí.

Označení	Požadavek
NP 1	Systém **musí** být implementován programovacím jazykem Python
NP 2	Systém **by měl** být multiplatformní a podporovat operační systémy Windows, Linux, macOS
NP 3	Systém **musí** být udržitelný minimálně 2 roky
NP 3	Systém a subsystémy **musí** být navrženy pro nepřetržitý provoz s minimem prostojů.
NP 4	Systém **musí** dodržovat všechny nařízení, vyhlášky a zákony v zemi odběratele
NP 5	Systém **může** poskytovat vyšší úroveň zabezpečení
NP 6	Odběratel **musí** mít dostatečnou kapacitu výpočetního výkonu vzhledem k provozu a funkční požadavky
NP 7	Všechny softwarové moduly a sub aplikace **musí** být řádně navrženy a dodávány.

Tabulka 4 Požadavková specifikace nefunkcionální

4.2 Případy užití

V softwarovém a systémovém inženýrství je případ užití („Use-case“ z anglosaské fráze) seznam kroků, který obvykle definuje interakci mezi tzv. rolí a systémem. Za rolí může být člověk (uživatel) nebo externí systém. Za systémem je téměř vždy aplikace nebo nějaký modul aplikace.

Pro názornou vizualizaci jednotlivých případů se používají diagramy užití („Use Case diagram“). Ty zachycují zjednodušený popis chování aktéra při práci se systémem a také jaké funkce na systému může vykonávat. Z toho jednoznačně vyplývá, že Diagram užití znázorňuje pouze funkční požadavky systému a tím pádem zobrazuje interakci mezi aktérem a systémem.

Kreslení a návrh diagramu lze sestavit do jednotlivých posloupností aktivit:[26][28]

1. Vyhledávání hranic systému.
2. Nacházení aktérů, jež interagují se systémem.
3. Nalezení případů užití.
4. Upřesnění případů užití.

5. Definování alternativní scénářů.
6. Opětovné provádění těchto kroků, dokud se neustálí případy užití, hranice systému a typy aktérů.

4.2.1 Teoretický popis

V rámci diagramu užití se používají následující prvky:[28][29]

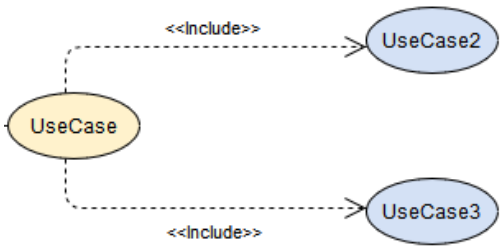
- *Aktér* – je role, která komunikuje s jednotlivými případy užití. V této roli může být obsazen uživatel nebo externí systém. Aktérem tedy může být např. uživatel, administrátor, SMS server nebo dokonce čas. Aktér inicializuje nějaký případ užití (např. uživatel vloží příspěvek do fóra). Zde bychom hovořili o tom, že je aktér aktivní. Aktér sám však může být případem užití iniciován (např. externí SMS server je iniciován případem užití Poslat SMS). V tomto případě hovoříme o pasivním aktérovi a zakreslujeme ho v diagramu napravo. Z toho plyne, že aktér nemusí být vždy fyzická osoba, ale může se jednat o hardwarové zařízení či další systém. Aktéry lze dále rozlišit na primární a pomocné. V jazyce UML jsou aktéři označováni pomocí panáčka s titulkem pod ním tímto způsobem:

Obrázek 4 Značení pro aktéra

- *Případ užití* –„specifikuje část funkcionality systému, kterou využívá aktér a která plní určitý cíl.“. Případ užití je popsán pomocí množiny scénářů, které jsou prováděny se stejným záměrem. V jazyce UML jsou případy užití označovány pomocí oválu, v jehož středu je popisek případu. Značení může být provedeno tímto stylem:

Obrázek 5 Značení pro případ užití

- *Komunikační asociace (relace)* – označuje se pomocí čáry, jež je zakončena šipkou a slouží k zobrazení toku informací mezi ostatními prvky. Komunikační asociace s popisem významu asociace se může značit následujícím stylem:



Obrázek 6 Značení pro relace

- *Hranice systému* – je znázorněna rámečkem okolo případů užití a je označena názvem systému. V jazyce UML jsou pro hranice systém vyhrazeny obdélníky.

4.2.2 Diagram užití aplikace

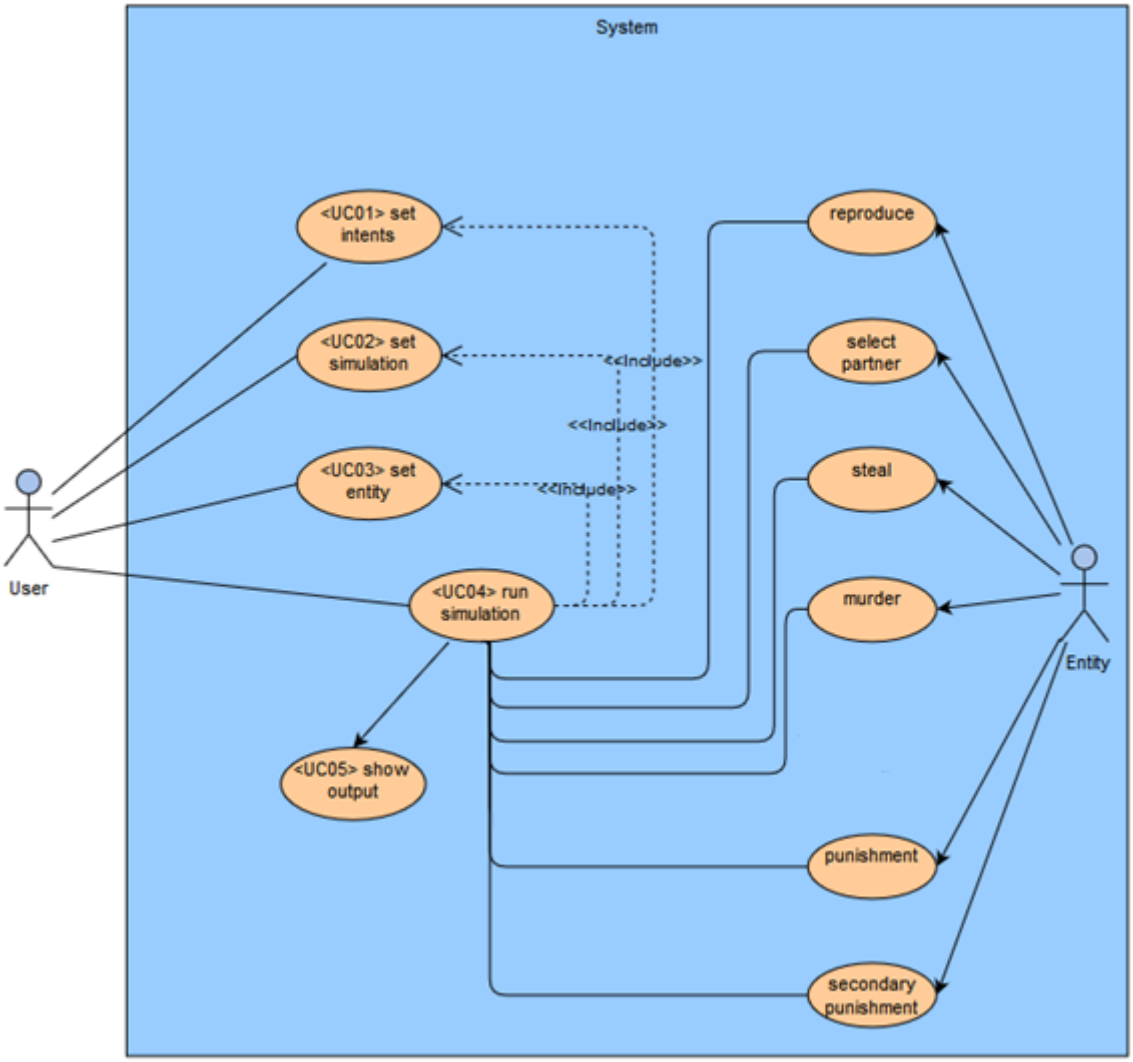
Diagram užití je jeden z nástrojů, který pomáhá analytikům navrhnout funkční systém a zároveň zadavateli demonstrovat, zdali je systém navržený správně. Právě z toho důvodu je nutný diagram pro simulátor pro rozvoj evolučních algoritmů.

Na obrázku, který je následuje, jsou ukázány případy užití, které by v aplikaci simulátoru neměly chybět. Na levé straně je aktér, v tomto případě uživatel, který bude systém aplikace obsluhovat. Aktér nejprve inicializuje případ užití a tím se stane aktivním. Případ užití se následně spustí a provede potřebné kroky k jeho realizaci. Uprostřed je systém, v jehož středu figurují případy užití a komunikační asociace. Pro lepší představu je dobré objasnit si smysl několika vybraných případů užití:

- UC01 set intents – tento use case umožňuje zvolit si nastavení jednotlivých záměrů, které bude entita při spuštění simulace vykonávat. Systém vygeneruje formulář, ve kterém je seznam všech záměrů. Záměr reprodukce a selekce je povinný a musí být

vybrán vždy. Uživatel si může vybrat jeden či více záměrů z tohoto seznamu a následně systém uloží toto nastavení.

- UC02 set simulation - use case umožňuje zvolit si nastavení počátečních podmínek simulace. Systém nejprve vygeneruje formulář, ve kterém se nachází všechna textová pole s nastavení simulace. Každé pole je popsáno a jednoznačně identifikováno. Uživatel vyplní pole a systém pak nastavení uloží.
- UC03 set entity - umožňuje nastavit všechny parametry, které se týkají entity. Systém nejprve vygeneruje formulář, ve kterém se nachází všechna textová pole s nastavením parametrů entity. Každé pole je popsáno a jednoznačně identifikováno. Uživatel vyplní pole a systém pak nastavení uloží.
- UC04 run simulation - umožňuje systému spustit simulaci. Systém vygeneruje tlačítko pro spuštění simulace. Uživatel poklepe na tlačítko. Systém vygeneruje vstupní množinu entit simulace. Systém spustí souběh entit. Aktér uživatel tímto aktivizuje pasivního aktéra entitu. Entita interaguje s ostatními jedinci a používá k tomu záměrů, které systém uložil při UC01. Záměry, které entita může použít, jsou: reproduce – požadavek na rozmnožení s vybraným partnerem; select partner – požadavek na výběr partnera; steal – požadavek krádeže zdrojů od jiné entity; murder – požadavek eliminace jiné entity; punishment – požadavek na potrestání zločince; secondary punishment – požadavek na potrestání již trestaného zločince.
- UC05 show output – umožňuje systému zobrazit výstup ze simulace.



Obrázek 7 Diagram užití simulátoru pro vývoj evolučních algoritmů

4.2.3 Specifikace případů užití

V minulé kapitole je graficky znázorněno, co bude systém vykonávat v reakci na aktéry. Nicméně kromě názvů jednotlivých případů užití o nich není známo vůbec nic. Z toho důvodu existuje jiný nástroj a to „specifikace užití“ (anglicky „Use case specification“). Ten definuje konkrétní funkcionality, co systém obsahuje a jak (popřípadě kým) je spuštěn. Jedná se o extra dokument, který se většinou k diagramu užití přikládá v případě potřeby většího detailu. Nemá na pevno danou šablonu, může nabývat podoby tabulky, textu, strukturovaného seznamu anebo eventuálně kombinace všech předchozích. I když nemá předem danou strukturu, je nutné vyzdvihnout jednotlivé části specifikace, které jsou potřebné k její definici. Konkrétní specifikace případů užití pro aplikaci simulátoru lze nalézt v příloze E. [29]

Pro jeden případ užití jsou následující jednotlivé části specifikace:[30]

1. **Krátký popis** – V první části se stručně popíše případ užití, stačí jedna až dvě věty. Měl by vysvětlovat, jakou má funkčnost, proč ji uživatel spouští anebo jakou má pro uživatele přidanou hodnotu.
2. **Aktéři** – Další část jmenuje aktéry, kteří se případu užití účastní.
3. **Podmínky pro spuštění** – Každý případ užití může mít definované určité podmínky, které musí být před jeho spuštěním splněny. Pokud jakékoliv podmínky před spuštěním existují, zde je nezbytné je uvést.
4. **Základní tok** – Jedná se o strukturovaný seznam kroků, kde je popsána interakce mezi aktéry a jednotlivými případy užití. Body jsou zapisovány jako scénář, ve kterém se střídají vždy aktér a systém. Vše je popsáno opět z pohledu uživatele a toho, jaký má na něj případ užití vliv. Častou chybou je popisovat, co systém zobrazí, co uživatel zapíše do formuláře a podobně. Popis by měl však být úplně odstíněn od toho, jak systém vypadá, měl by se zaměřit na to, jak funguje. Základní tok neřeší možné chyby a předpokládá bezproblémový průběh, kde v posledním kroku dojde ke splnění cíle případu užití.
5. **Alternativní tok** Specifikace může obsahovat několik alternativních toků (scénářů), které umožňují reagovat na odchylky od scénáře základního. Jedná se o poruchy nebo chyby, ať již ze strany uživatele (špatně zadal heslo) nebo systému (nepodařilo se vytisknout dokument). Alternativní tok se vždy vztahuje k určitému bodu hlavního toku a řeší jeho nestandardní verzi. Většinou je na konci odkázán opět na nějaký bod hlavního toku, kde zas hlavní tok pokračuje dále.
6. **Podmínky pro dokončení** – Podobně, jako může mít případ užití podmínky před spuštěním, může mít i podmínky pro dokončení.

4.3 Diagram tříd/ UML

Zkratka UML za sebou skrývá iniciály prvních písmen ve slovním spojení Unified Modeling Language. Ve volném překladu z anglického jazyka zkratka znamená unifikovaný modelovací jazyk. Na rozdíl od psaných programovacích jazyků zahrnuje grafickou syntaxi. To znamená, že má pravidla pro sestavování jednotlivých prvků jazyka do větších objektů. Vedle syntaxe UML obsahuje i grafickou sémantiku. To jsou jednoznačná pravidla říkající jednotlivým syntaktickým výrazům jejich význam.[31]

V současné době má jazyk UML největší význam při návrhu softwarových systémů, protože objektově orientovaný návrh každé složitější aplikace je nezbytným předpokladem pro její úspěšnou a rychlou implementaci. Pro objektově orientovaný návrh je samozřejmě možné použít různé podpůrné prostředky, zejména další odlišné typy diagramů, UML je však významný také v tom ohledu, že přesně specifikuje, co má daný diagram obsahovat, což je velmi důležité zejména při sdílení informací mezi jednotlivými analytiky a vývojáři. Dále je již z principu UML nutné, aby vytvářené grafy měly vnitřní konzistenci a přesně danou sémantiku, což u jiných typů grafů nemusí být obecně zaručeno. UML diagramů existuje několik typů lišících podle toho, jaké se pomocí nich plánují či zpracovávají úlohy. Tyto diagramy se od sebe odlišují především repertoárem použitých značek, způsobem jejich vzájemného propojení a s nimi související sémantikou. Výhodou UML diagramů je existence otevřeného a rozšiřitelného standardu a univerzálnost použití na jakýkoliv objektově orientovaný jazyk. Pro taky mluví skutečnost, že je podporován celou řadou vývojových nástrojů určených specificky jen pro návrh UML a nebo integrovaným prostředím, které dokáže z kódu programovacího jazyka navrhnout diagram a i obráceně. Konkrétní specifikaci UML diagramu pro aplikaci simulátoru lze nalézt v příloze F. [31]

5 VLASTNÍ ŘEŠENÍ, IMPLEMENTACE

V této kapitole jsou popsány nejdůležitější části aplikace. První část se zaměřuje na obecný výklad použitých principů a programovacích technik, zatímco druhá část se specializuje na tvorbu desktopové aplikace a klíčové části kódu a pokračuje řešením vlastní problematiky aplikace.

5.1 Řešení

Aplikace je psaná v dobře známém programovacím jazyce Python. Je to rozšířený jazyk, který v sobě obsahuje všechny potřebné nástroje k uvedení aplikace do chodu. Obsahuje knihovny, které umí pracovat s GUI, s procesy, s kolekcemi a zejména s diskrétní simulací. Python je dobře přizpůsoben pro implementaci vícevláknových nebo víceprocesových aplikací. Multiprocessing se dobře uplatní v poměrně výpočetně náročné simulaci, tím se totiž značně zkrátí doba, po kterou se simulace vyhodnocuje.

Simulátor si naplánuje a spustí simulované události nebo procesy v zadaný simulační čas, přesně tak jak je typické pro diskrétní simulátory. Každému jedinci, který je iniciovaný, je přiřazena poštovní schránka a jedna instance simulátoru. Následně je jedinec zpracován jako proces. Jedinec vykoná svůj životní cyklus a pak vyčká po jednu simulační časovou jednotku. Tento postup se opakuje do té doby, než nastane událost, která ukončí proces jedince, nebo popřípadě nastane podmínka, která ukončí běh celé simulace.

Životní cyklus zpracování simulačního procesu jedince je řešen v celkem šesti fázích:

a) Narození jedince – inicializace atributů a proměnných jedince, přiřazení a spuštění simulačního procesu.
b) Upkeep – odečtení životních nákladů, získání zdrojů, ověření ukončovací podmínky jedince, ukončení životního cyklu.
c) Zpracování intentů – vykonání kódu, který zpracovává procesy záměrů.
d) Rozhodování o intentech – vyhodnocení podmínek záměrů.
e) Zpracování zpráv – přijmutí zpráv z „poštovní schránky“ entity, čtení zpráv, vykonání kódu zpracovávajícího jednotlivé zprávy.

V diplomové práci je pro komunikaci mezi entitami použito řešení sdílení zpráv, kde komunikace mezi jedinci/vlákny probíhá za pomoci centrální pošty – dictionary, v němž klíčem je unikátní označení entity a daty seznam zpráv – poštovní schránka. Obě tyto kolekce musí být ve sdílené a synchronizované paměti v případě využití více procesů. Pro zdůvodnění potřeby

sdílené paměti je nutné nahlédnout do teorie operačních systémů. Při vytváření nového procesu se program rozdělí (např. pomocí os.fork) do n procesů. Jak je všeobecně známo, každý proces má svůj vymezený díl operační paměti, který je oddělený od ostatních procesů tj. procesy spolu nesdílí paměť. Jejich data jsou pouze zkopírována metodou „copy-on-write“ a aktuální stavy všech proměnných v nových procesech jsou v takovém stavu, ve kterém byly zkopírováné z původního procesu. Každá poštovní schránka přijímá krátké zprávy od všech jedinců, kteří chtějí komunikovat s jiným jedincem. Zpráva se musí skládat minimálně z identifikátoru zprávy. Obvykle s sebou nese i informace o odesílateli, aby adresát mohl například posoudit, jestli má o takového partnera zájem nebo komu má nazpátek odpovědět. Informace o formátech a typech zpráv lze najít v příloze A. Verze všech potřebných knihoven v příloze H.

5.2 Implementace

Součástí praktické části bylo implementovat simulační jádro pro rozvoj evolučních algoritmů. Tato kapitola popisuje klíčové části programu a vysvětluje funkčnost jednotlivých funkcí. Přehled použitých nástrojů je k nalezení v příloze C.

5.2.1 Simulační jádro

Jádro simulátoru je založeno na open-source simulátoru Simulus. Ten implementuje procesně orientovaný přístup simulace s několika pokročilými funkcemi, které usnadňují modelování a vykonávání simulačních úloh – událostí nebo procesů. Jeho hlavní předností oproti jiným simulátorům je podpora paralelní a distribuované simulace.

Simulus může být využíván dvěma způsoby, přičemž oba přístupy se dají vzájemně kombinovat. Jedním ze způsobů je plánování událostí. Uživatel může naplánovat události. Simulus posléze zajišťuje, že jsou všechny události seřazeny podle časových značek. Když dojde k události, simulus posune čas simulace na událost a následně se zavolá event handler události, což je vlastně uživatelsky definovaná funkce. Při zpracování události může uživatel naplánovat nové události do simulované budoucnosti.

Druhým způsobem jsou procesy. Uživatel může vytvářet procesy a spouštět je. Každý proces je samostatným vláknem. Během jeho zpracovávání může být proces pozastaven do stavu uspaný nebo zablokovaný. Proces obnoví svoji činnost buďto po uplynutí zadaného času nebo po zrušení podmínek pro blokování zdrojů (resource). V této práci je používán přístup ryze procesový.

Simulus má základní objekt simulace, „simulator“, který se při vytváření musí pojmenovat unikátním identifikátorem. Simulátorů je možné vytvořit tolik, kolik je potřeba. Každý simulátor si vytvoří svůj vlastní fyzický proces. Maximálně se však vytvoří n procesů, kde n je počet procesorů resp. u jednoprocesorových počítačů počet jader procesoru. Simulátory je možné synchronizovat do grup, čímž se umožňuje spustit simulaci paralelně.

Simulátor si udržuje svůj vlastní seznam událostí a hlídá si čas simulace. Je možné naplánovat mu události a procesy pomocí funkcí sched() a process(), jejichž jediným povinným parametrem je funkce. Každý jedinec je procesem, přičemž je naplánovaný a spuštěný každou simulační časovou jednotku, dokud se pro něj nesplní ukončovací podmínky.

Následuje ukázka konkrétní implementace funkcí simulátoru. Funkce initialize() vytvoří potřebný počet paralelně běžících simulátorů. init_set() naplánuje běh procesů jednotlivých entit. Funkce run() nejprve nastaví synchronizaci simulátorů a následně spustí simulaci.

```
    def run(self):
        self.g = simulus.sync(self.sims), lookahead=1, enable_smp=True)
        self.g.run(self.until, show_runtime_report=True)

    def initializate(self):
        sims = []
        for i in range(NUM_PROCESSES):
            sim = simulus.simulator('sim%d' % i)
            sims.append(sim)

        return sims

    def init_set(self):
        for i in tqdm(range(NUM_ENTITIES)):
            # find simulator for the new entity
            sim = self.sims[self.sim_insert_index]
            # we create unique name for entity mailbox
            entity_name = 'Entity%d' % counter_entities.value
            mbox = self.post_office.create_mailbox(entity_name)

            ent = Entity(self, sim, mbox, entity_name, val_talent())

            # prepare entity life simulation
            p = sim.process(ent.live)
            ...
```

Obrázek 8 Simulátor – implementace

5.2.2 Entita

Klíčový objekt celé aplikace. Představuje jedince, který se snaží být v rámci simulace co nejúspěšnější. Na začátku svého životního cyklu je inicializován a posléze spuštěn jako proces

vybrané instance simulátoru. Dále pokračuje ve vykonávání svého životní cyklu, jak bylo popsáno v kapitole 5.1 řešení.

Atributy třídy

V této třídě jsou obsaženy tyto popisné atributy:

- *name* – unikátní identifikátor jedince, podle kterého jej ostatní poznají a podle kterého je založený mailbox na zprávy.
- *money* – představují zdroje pro jedince, které jsou důležitou komoditu. Ze zdroje je entita živá tzn., má životní náklady a zároveň si za každý časový interval vydělá určitý počet zdrojů závislých na jejím talentu. Pokud má dostatek zdrojů, může uvažovat o dítěti.
- *talent* – je talent vydělávat zdroje. Talentovanější jedinec si může pořídit více dětí a tím být ve společnosti úspěšnější.
- *productivity* – je statistická proměnná, která určuje míru zapojení a využití zdrojů ve vztahu k výsledkům ekonomické činnosti. Každý výdaj a příjem *money* je započítán. Produktivita je považována za míru úspěšnosti v evoluci jedince a z toho vyplívá, že jí můžeme nazývat fitness funkcí.

Proměnné záměrů:

- *isLookingForPartner* – proměnná, která určuje, zda jedinec hledá svůj protějšek nebo jestli již rozeslal žádosti ostatním entitám.
- *intentLookForYourNewPartner* – ukazuje záměr, jestli jedinec právě teď hledá svého partnera do páru. To je dáno tím, zdali má dostatek zdrojů, aby mohl mít potomka.
- *intentSteal* – ukazuje záměr, jestli je entita rozhodnuta vykonat nelegální činnost a okrást jinou entitu za účelem zisku zdrojů.
- *intentMurder* – ukazuje záměr, jestli je entita rozhodnuta vykonat nelegální činnost a odstranit jinou entitu za účelem likvidace konkurence.
- *intentPunishment* – ukazuje záměr, jestli je entita rozhodnuta potrestat jinou entitu, která byla přistižena při nelegální činnosti. Trestem můžou být tři možnosti: zabavení všech zdrojů, vyhoštění jedince, vyhoštění entity a jeho potomků.

- *intentSecondaryPunishment* – ukazuje záměr, jestli entita, která byla přistižena při nelegální činnosti, bude potrestána zvláště výjimečným trestem (vyhoštění entity a jejich potomků).

Komunikační proměnné:

- *post_office* – objekt, přes který všechny entity mezi sebou komunikují. Každá entita má vlastní poštovní schránku, do které dostává od ostatních entit zprávy.
- *messages* – fronta, do které se nahrávají nově příchozí zprávy od ostatních jedinců.

Metody a funkce

Metoda live() je strukturovaná do několika fází upkeep, zpracování intentů, rozhodování o intentech, zpracování zpráv. Celá metoda je „obalená“ do nekonečného cyklu, který je ukončen ve chvíli, kdy se simulační čas posune na čas úmrtí jedince a booleovská proměnná *dead* se nastaví na *true*.

```
    self.dead = self.dead or
                self.death_simulation_timestamp <= self.sim.now
    if self.dead:
            self.env.__remove_entity(self.name)
            self.recorder.save_record(['pop', self.sim.now, self.name, -1])

            break
```

Obrázek 9 Entita – ukončení životního cyklu

V průběhu každé první fáze je třeba vykonat income jedince. Což v praxi znamená, že si musí vydělat zdroje, odečíst životní náklady. V každé z těchto funkcí se vypočítává i produktivita entity. Jak lze vidět na ukázce, probíhá v ní výpočet mzdy a životních nákladů.

```
    def get_salary(self):
        self.money += val_salary(self)
        self.productivity += val_salary(self)

    def pay_live_costs(self):
        self.money -= VAL_LIVE_EXPENSE
        self.productivity -= VAL_LIVE_EXPENSE
```

Obrázek 10 Entita – implementace upkeep funkcí

Druhou fází probíhá zpracování intentů. Jakmile je entita rozhodnuta, tak se v následujícím čase vykonají záměry, které jsou vyhodnocovány ve třetí fázi. Zpracování záměrů obvykle vyústí v interakci s nějakou jinou entitou vložením zprávy do její poštovní schránky. Příklad zpracování záměru – hledaní partnera:

```
    if self.intentLookForYourNewPartner:
       message_object = ["LOOKING_FOR_PARTNER", self.name, self.talent,
```

```
                        self.money]
        self.post_office.notify_to('Seznamka1', message_object)
        self.isLookingForPartner = False
```

Obrázek 11 Entita – zpracování záměru

Oproti jiným zprávám, které se obvykle doručují jiným entitám, se zpracování záměru hledání partnera zasílá společnému objektu Seznamce. Ta má svojí oddělenou schránku, ve které shromažďuje všechny žádosti entit a na konci nejmenší jedné časové simulační jednotky odpoví všem entitám, kterého partnera jim vybrala.

Ve třetí fázi se vyhodnocují podmínky rozhodnutí o úmyslu jedince. Každý záměr je poskládaný z jedné nebo více podmínek, které se budou muset splnit, aby se entita rozhodla pro danou činnost v nadcházejícím časovém úseku. Některé z hodnot, které jsou v ukázce uváděny, jsou nastavitelné hodnoty za pomoci konfigurační části GUI nebo souboru conf.py.

```
self.intentLookForYourNewPartner = self.money > VAL_CHILD_EXPENSE and
                                   self.isLookingForPartner
self.intentSteal = self.time_i_havent_a_child > self.live_length *
                   VAL_INTENT_STEAL and is_on_intent_want_to_steal
self.intentMurder = (self.time_i_havent_a_child > self.live_length *
                    VAL_INTENT_MURDER) and is_on_intent_want_to_murder
self.intentPunishment = is_on_intent_execute_punishment
```

Obrázek 12 Entita – vyhodnocení záměrů

Ve čtvrté fázi dochází ke čtení krátkých zpráv z fronty zpráv. Ta je při každém incomu vyzvednuta ze schránky pošty. Zpracování zpráv probíhá přečtením, vyhodnocením a následně smazáním zprávy. Tento proces se opakuje, dokud není fronta zpráv úplně prázdná. Formát se liší s každým typem zprávy (datovým typem a počtem položek) a proto je nutné počínat si opatrně při jejich čtení. Jednotlivé typy a formáty zpráv lze najít v příloze A.

```
while self.mbox:
    message = self.mbox.pop()
    ...
```

Obrázek 13 Entita – čtení zprávy

Ukázkou pro zpracování zprávy je například vyhodnocení typu zprávy „máš dítě“. Pokaždé se porovná, jestli odpovídá typ zprávy a následuje vyhodnocování. Pokud entita je prvním rodičem, stane se v této roli „matkou“ a přivede svého potomka do systému a zařadí ho do plánovače procesů simulátoru, aby se spustil jeho životní cyklus. Ať už je v pomyslné roli otce či matky, musí zaplatit výdaje spojené s početím následovníka, který ponese část jejich genu.

```
if message[0] == "YOU_HAVE_A_CHILD":
    if message[3]:
        mbox = self.post_office.create_mailbox(message[2])
        ent = Entity(self.env, self.sim, mbox, message[2], message[4])
        # give a child birth to life
        p = self.sim.process(ent.live)
        ent.p = p
        # we watch historical count of entities
        counter_alive_entities.increment()

    self.money -= VAL_CHILD_EXPENSE
    self.productivity -= VAL_CHILD_EXPENSE
    self.children.append(message[2])
    ...
```

Obrázek 14 Entita – zpracování zprávy

5.2.3 PostOffice

PostOffice je hlavním komunikačním uzlem aplikace. Má na starosti komunikaci mezi entitami a ostatními objekty simulace. Jeho jedinou proměnou je *mailboxes*, která je instance dictionary o jejíž sdílení mezi procesy se stará multiprocessing manager. Kolekce se chová stejně jako standartní dictionary pythonu, s tím rozdílem, že používá u svých operací mutex. Každá poštovní schránka, kterou obsahuje tato kolekce, je reprezentována sdíleným listem, který je uzamykatelný mutexem. Každá entita si mezi sebou předává jednu a tu samou instanci třídy post_office a díky tomu mají jedinci možnost posílat zprávu komukoliv v populaci. K této komunikaci jsou využívány metody:

- notifyTo – zpráva se pošle konkrétnímu jedinci
- notifyRandom – zpráva se pošle náhodnému jedinci

Následuje ukázka konkrétní implementace komunikačních funkcí.

```
    def notify_to(self, owner, message_object):
        mailbox = self._mailboxes.get(owner)
        if mailbox is not None:
            mailbox.append(message_object)

    def notify_random(self, message_object):
        choice = random.randint(0, counter_alive_entities.value - 1)
        for index, key in enumerate(self._mailboxes.keys()):
            if index == choice:
                mailbox = self._mailboxes.get(key)
                if mailbox is not None:
                    mailbox.append(message_object)
                break
```

Obrázek 15 PostOffice – komunikace

Kromě funkcí pro komunikaci s jednotlivými entitami existují ještě dvě funkce, které spravují mailboxy. Jedna vytvoří novou poštovní schránku, vloží jí do dictionary *mailboxes* a vrátí na ní referenci. Druhá funkce maže (nepoužívané) schránky. Funkce alokují a dealokují mailboxy jedinců, vždy při jejich narození nebo úmrtí.

```
    def create_mailbox(self, owner):
        self._mailboxes[owner] = self.manager.list()
        return self._mailboxes[owner]

    def remove_mailbox(self, owner):
        self._mailboxes.pop(owner)
```

Obrázek 16 PostOffice – de/alokace mailboxů

5.2.4 Configuration

Tato třída je singleton, což znamená, že může mít pouze jednu instanci třídy. Obsahuje informace o vstupním nastavení simulátoru a entity. Používá se v aplikaci simulátoru k ukládání a čtení konfigurace. Kompletní přehled, co který atribut znamená, lze najít v příloze B.

```
class Singleton(type):
    _instances = {}

    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            cls._instances[cls] = super(Singleton, cls)
                                        .__call__(*args, **kwargs)
        return cls._instances[cls]


class Conf(metaclass=Singleton):
    ...
```

Obrázek 17 Conf - singleton.

5.3 Parametrizace – nastavení aplikace simulátoru

Pomocí aplikace simulátoru můžeme nastavovat vstupní kritéria a tím změnit vývoj jedinců v simulaci. Kritéria jsou nastavována pomocí proměnných systému a na základě nich mohou ovlivňovat produktivitu a tím pádem i fitness funkci evolučního algoritmu. Startovní podmínky simulace a entit mají za úkol nastavovat všechno ohledně vygenerování počáteční množiny entit a její mohutnosti, ukončovací podmínku simulace a počet paralelně běžících procesů. Vstupní kritéria pomáhají docílit simulovaných scénářů a tím dosáhnout požadovaných předpokladů experimentu.

5.3.1 Startovní podmínky

Před zahájením simulace má uživatel možnost nastavit si počáteční podmínky a tím parametrizovat generování vstupní množiny entit. Součástí startovních podmínek je mohutnost vygenerované množiny jedinců. Kvalita jedinců je uživatelsky nastavitelná, ale toto téma bude podrobněji rozebráno později. Nastavením větší množiny lze dosáhnout větší diverzifikace vygenerovaných atributů. Výhodou menší množiny je lepší pozorovatelnost jednotlivých aspektů průběhu simulace. K vyprodukování náhodných čísel je použit generátor pseudonáhodných čísel s využitím zvoleného seedu (lze také nastavovat). Tím se vytvoří série náhodných čísel, která bude i při příštím generování opakovatelná a tím pádem může být i věrohodně zopakován experiment. Velikost vygenerované množiny je defaultně nastavena na 1000 jedinců, ale konečné rozhodnutí zůstává pouze na uživateli, který si může nastavit libovolnou hodnotu.

5.3.2 Ukončovací podmínka

Po parametrizaci startovních podmínek se musí definovat ukončovací podmínka, aby se mohla tato v průběhu simulace vyhodnotit a její běh mohl skončit. V evolučním algoritmu se ukončovací podmínka nastavuje jako cílový počet generací. V simulacích je to simulační čas – interval, po který simulace může běžet. V programu, který je součástí této diplomové práce je tomu stejně jako u simulací. Uživatel si může nastavit hodnotu „Simulation time“, která nastaví interval od nuly, až do *n*. Po uplynutí této doby simulace končí. Délka simulace je defaultně nastavena na 500 simulačních časových jednotek, ale uživatel si ji může nastavit do libovolné výše, tedy dokud nenarazí na limitní omezení.

5.3.3 Vstupní kritéria

Simulací samotnou se prokazují hypotézy a jejich funkčnost navrhovanými scénáři. Ty se s defaultním nastavením hodnot simulace sice neprokáží, ale mohou posloužit jako počáteční seznámení se s aplikací. Jakmile se uživatel obeznámí s tím, kde se spustí běh simulace, může se pustit do parametrizace vstupních kritérií. Pokud jsou nastavené správně, například podle doporučení, která budou následovat, potom se může postupným experimentováním a laděním dojít k požadovanému výsledku. Nastavením atributů entity ovlivníme vygenerování hodnot, jak ve vstupní množině, tak i při narození nového jedince. Řeč je tu o nastavování popisných, „životní“ atributů a udělování povolení záměrů entit. Popisné atributy charakterizují jedince,

popisují čím je jedinečný, pomáhají při interakci s ostatními jedinci. Vztah mezi těmito atributy bude popsán na několika příkladech. Vysvětlení používaných výrazů v kapitole 5.2 a v příloze B.

Pro řízení a nastavování scénářů je důležité si ukázat následující:

1. Jak ovlivnit produktivitu a příjmy.
    - Nastavením hodnoty Talent na vyšší hodnoty se výrazně zvyšuje příjem zdrojů.
    - Snížení hodnoty Live expense na minimum zvyšuje zůstatek zdrojů entity a tím i její produktivitu.
    - Zvýšení jedné nebo obou hodnot Live length dojde k prodloužení délky života a tím přijde více příležitostí k získání zdrojů. Dojde k vytváření větších úspor, ze kterých si mohou entity pořídit více potomků.
2. Jak ovlivnit míru zločinnosti.
    - Úplným řešením odstraněním zločinnosti je zakázat záměr Murder, Steal a tím entitám znepřístupnit tyto nástroje.
    - Čím větší náklady pro život, tím méně mají jedinci zdrojů a jejich míra frustrace je větší a tím se zvětšuje šance, že jedinci spáchají zločin. Zvýšením konstanty Live expense nad číslo, které je větší než hodnota horní hranice intervalu talentu, se zajistí pokles výdělku jedince a tím se zvýší míra frustrace jedince.
    - Výše násobku Intent murder/steal přímo ovlivňuje zločinnost. Čím menší je hodnota intentu, tím větší sklony k zločinu budou jedinci mít.
    - Jedinec, který má příliš krátký život na to, aby za něj stihl normálními prostředky vydělat na dítě, je frustrovaný a má sklony k zločinu. Výrazným zmenšením mezních hodnot intervalu Life lenght se způsobí, že má větší tendenci ke zločinu.
    - Čím dražší Child expense tím trvá entitě déle, než získá dostatečný počet zdrojů na výdaje za dítě. Pokud bude hodnota příliš vysoká, entity začnou mít sklony ke zločinu.
3. Jak ovlivnit úmrtnost jedinců.
    - Interval hodnoty Life length se nastaví na menší hodnotu. Tím se zkrátí doba, po kterou jsou entity v systému a tím umírají častěji.

- Nastavením velké hodnoty na Child expense se entitám sníží dostupnost potomstva a tím pádem jich budou mít méně a vývoj populace může začít klesat.
- Povolením záměrů execute punishment, secondary punishment, murder může v neprosperující společnosti zajistit několik potrestaných zlodějů i s jejich potomky, to logicky může způsobit pokles populace.

4. Jak ovlivnit porodnost.
    - Lepší podmínky na náklady Child expense za dítě pro rodiče způsobí větší porodnost. Čím menší hodnota se nastaví, tím je více příležitostí pro entity mít potomka.
    - Zvýšení jedné nebo obou hodnot Live length způsobí prodloužení délky života a tím může dojít k více příležitostem početí dítěte. Zatímco tyto entity hledají partnery, jejich potomci nelení a také se párují. Délka života jedinců může tedy ovlivnit do jisté míry porodnost.
    - Snížení hodnoty Live expense na minimum zvyšuje zůstatek zdrojů entity. Pokud se bude tato hranice držet pod spodní hranicí intervalu talentu, budou mít entity vždy dostatek zdrojů na zaplacení nákladů za dítě.

5.3.4 Limitní omezeni

Limitní omezení nastavují jednoznačné stropní hranice simulátoru. Počet entit, který se může v jeden okamžik nacházet v systému, závisí na mnoha aspektech. Aspekty, které je uživatel schopen ovlivnit, by se dali generalizovat do dvou kategorií: (i) fyzické hranice procesoru, (ii) matematické omezení. CPU má vliv na počet paralelně běžících procesů a počet spočítaných instrukcí za jednotku času (rychlost výpočtů). V rámci praktické práce byl simulátor testován na procesoru Intel Core i5-5200U (3MB Cache, 2.20 GHz); Počet jader procesoru: 4; Počet souběžných procesů: 4; Během testování bylo zjištěno, že v extrémně klidovém režimu (tj. při ukončení veškerých nedůležitých procesů) může být spuštěných v jeden moment přibližně 200 000 jedinců. Matematické omezení, které simulace evoluce jedinců má, může mít příliš exponenciální průběh. Pokud si uživatel nastaví přehnaně ideální podmínky pro častý vznik nových generací, simulace bude rychle aproximovat k exponenciálnímu průběhu.

Dalším omezením, které aplikace simulátoru má, je interval hodnot, do kterého se musí všechny parametry simulace vejít. V rámci aplikace diplomové práce se parametry na všech

polích nemohou nastavovat na hodnotu nižší než 0 a zároveň se nemohou nastavovat na hodnotu vyšší než sys.maxint.

6 LADĚNÍ, SIMULAČNÍ EXPERIMENTY

Kapitola nejdříve popíše průběh simulování vytvořeného modelu a provede srovnání hodnot, které simulaci poskytne. Na základě simulačních experimentů se pokusem zjistí chování entit v populaci. Další experimenty budou prováděny postupným povolováním (resp. zakazováním) vybraných záměrů. Kompletní příručka ke grafickému rozhraní je k nalezení v příloze D.

6.1 Validace a ladění

Pro účely odzkoušení simulátoru je nutné ověřit si funkčnost simulátoru za pomoci jeho parametrizace a několika vybraných scénářů. V tomto případě je validace systému provedena pomocí pozorování systému, který se nastaví na tři různé scénáře. Ověřuje se chování, že validovaný systém udává stejné výsledky, jako by udával původní reálný systém. Reálným systémem se v tomto případě uvažuje o reálné populaci jedinců v uzavřeném prostředí.

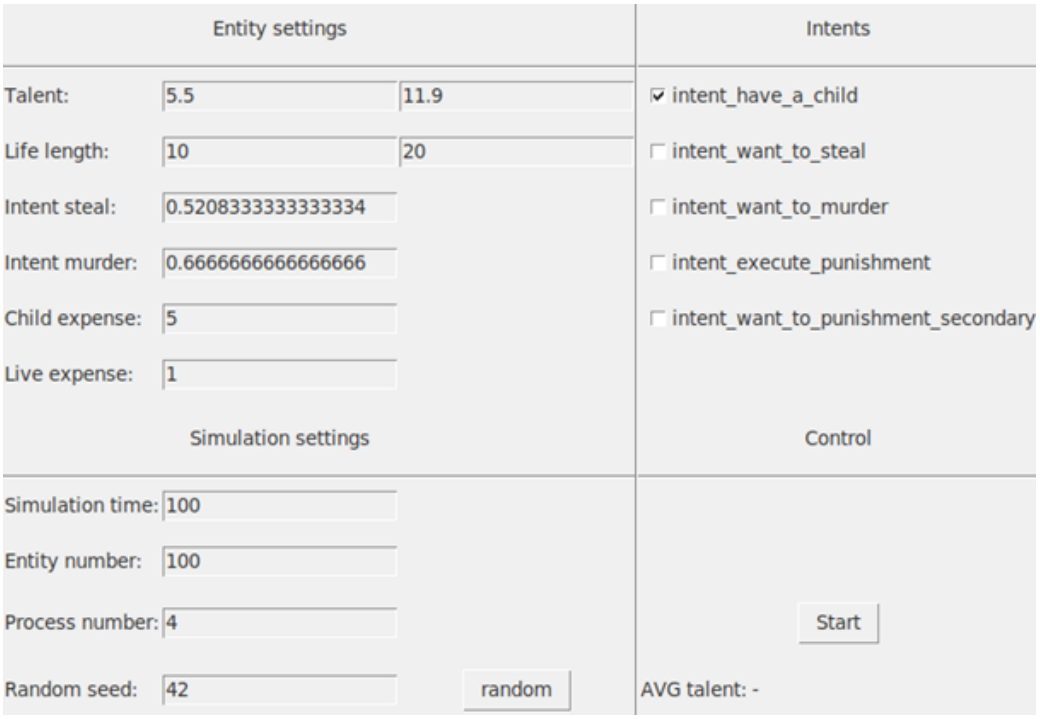
První situace si bere za cíl nastavit podmínky, které by umožňovali jedincům vytvářet nadbytek produkce. Logicky lze usoudit, že pokud každý jedinec nebo alespoň většina jedinců v systému bude mít dostatek zdrojů, bude mít více než dva potomky a tudíž bude taková populace růst exponenciálním způsobem, i jejich produktivita bude mít obdobný průběh.

Druhá situace se pokusí nastavit parametry simulace a jedinců tak, aby produktivita jedinců stagnovala nebo oscilovala kolem jedné hodnoty. V chaotickém systému, jakým tento systém bezesporu je, lze pouze stěží nasimulovat takovou situaci. Dříve či později se vždy přikloní k rostoucímu nebo klesajícímu trendu. Stagnace je z tohoto pohledu pouze dočasným stavem. Z toho důvodu se bude simulovat situace pouze na vybraném časovém úseku.

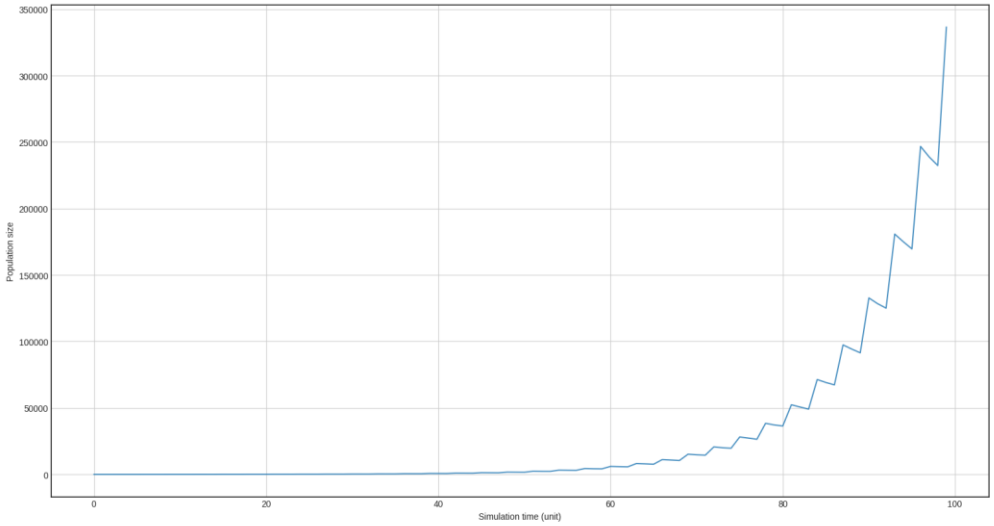
Třetí validační scénář nastíní situaci, kdy jedinci nebudou mít téměř žádnou produktivitu. Všichni jedinci postupně vyhladoví a tím pádem nezaloží další generaci potomků. Populace vyhyne a produktivita bude nulová.

6.1.1 Situace A, kdy je v populaci nadbytek produkce

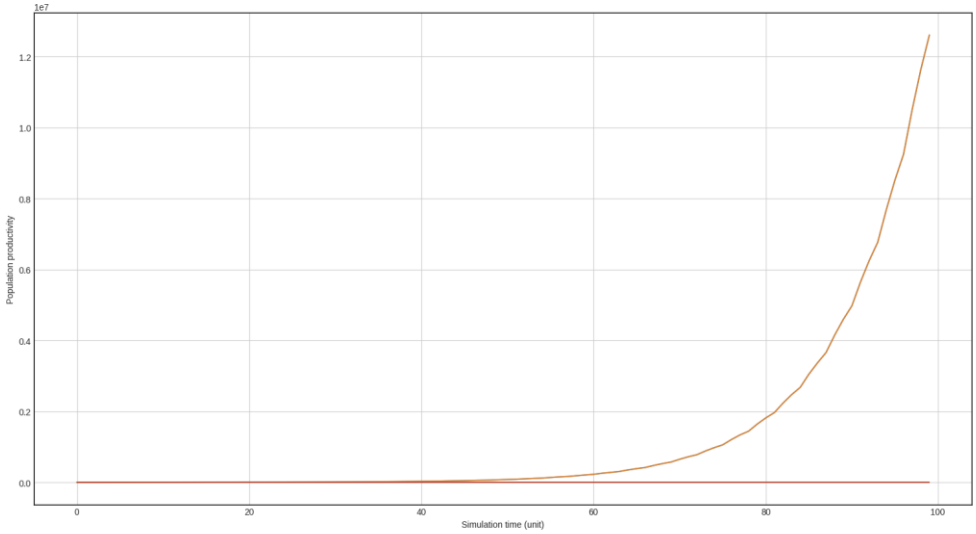
Na obrázku je vidět grafické rozhraní parametrizace vstupních kritérií a startovních podmínek. V tomto případě mají všechny entity maximálně prosperovat. Parametrizování simulačních nastavení pro připravovaný scénář je jednoduché. Stačí nastavit Live expense a Child expense na nízké hodnoty a nakonec pro rychlejší zisk zdrojů i spodní hranici talentu, aby entity byly co nejproduktivnější (avšak s rozvahou, aby se nenarazilo na limitní omezení).



Obrázek 18 Parametrizace validace 1



Obrázek 19 Výsledek validace A – velikost populace

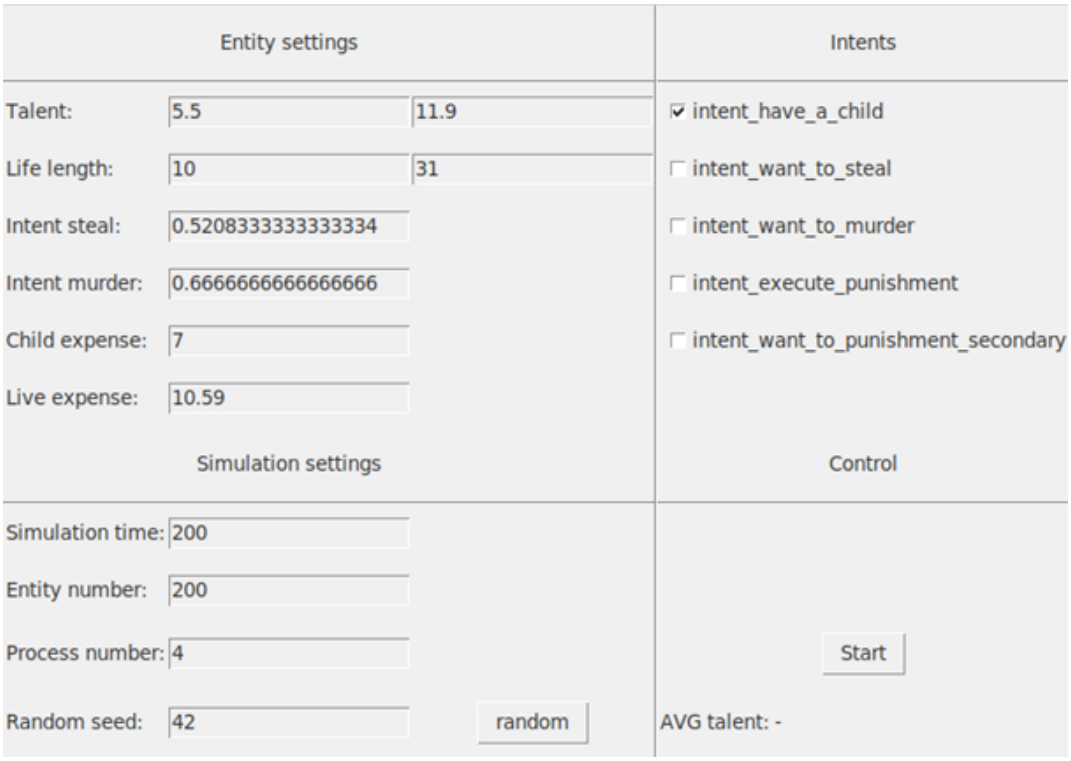


Obrázek 20 Výsledek validace A – produktivita

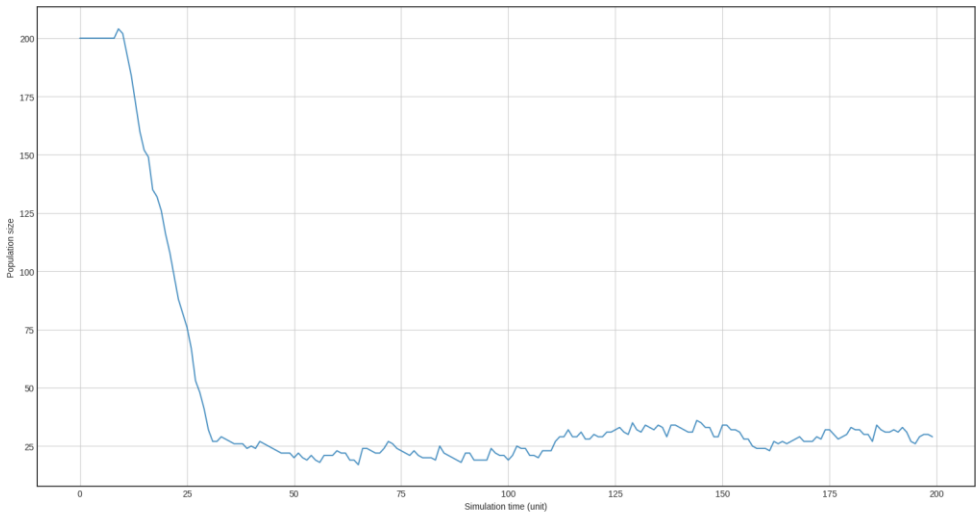
Předpokládaný výsledek byl naplněn. Podle obrázků lze jasně zjistit, že průběhy obou výstupů simulace téměř odpovídají exponenciálnímu průběhu.

6.1.2 Situace B, kdy se populace drží na úrovni přežití.

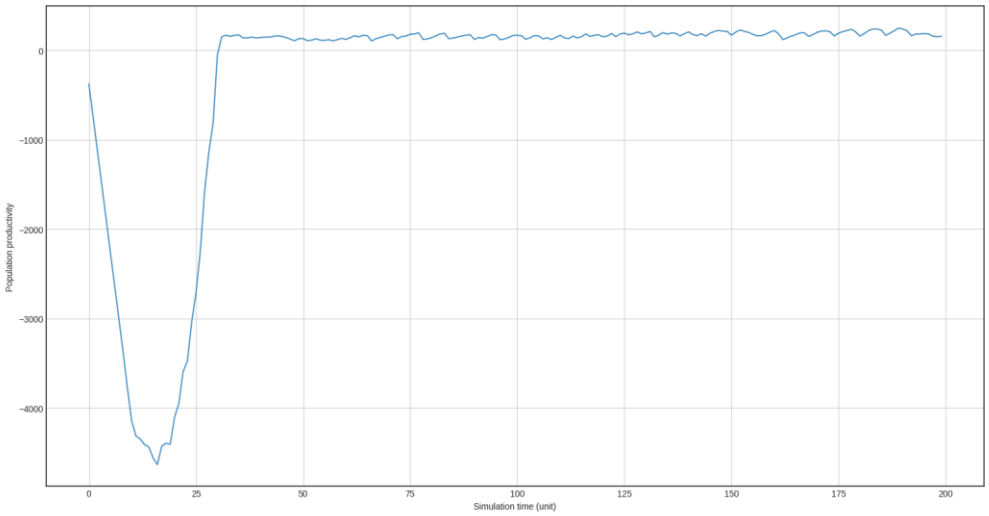
Jak bylo řečeno na začátku kapitoly, najít parametry simulace, aby produktivita populace stagnovala, je v chaotickém systému velmi obtížné. Po určitý simulovaný čas je to však možné. Live expense se nastaví těsně pod horní hranici intervalu talentu jedince, Child expense na hodnotu blízkou polovině intervalu talentu s ohledem na průměrný věk jedince Life length. Ten musí být dostatečně dlouhý, aby jedinci za svůj životní cyklus vytvořili novou generaci. Většina populace se ale tímto nastavením stane neproduktivní. Parametry jsou nastaveny podle obrázku, který následuje.



Obrázek 21 Parametrizace validace B



Obrázek 22 Výsledek validace B – velikost populace

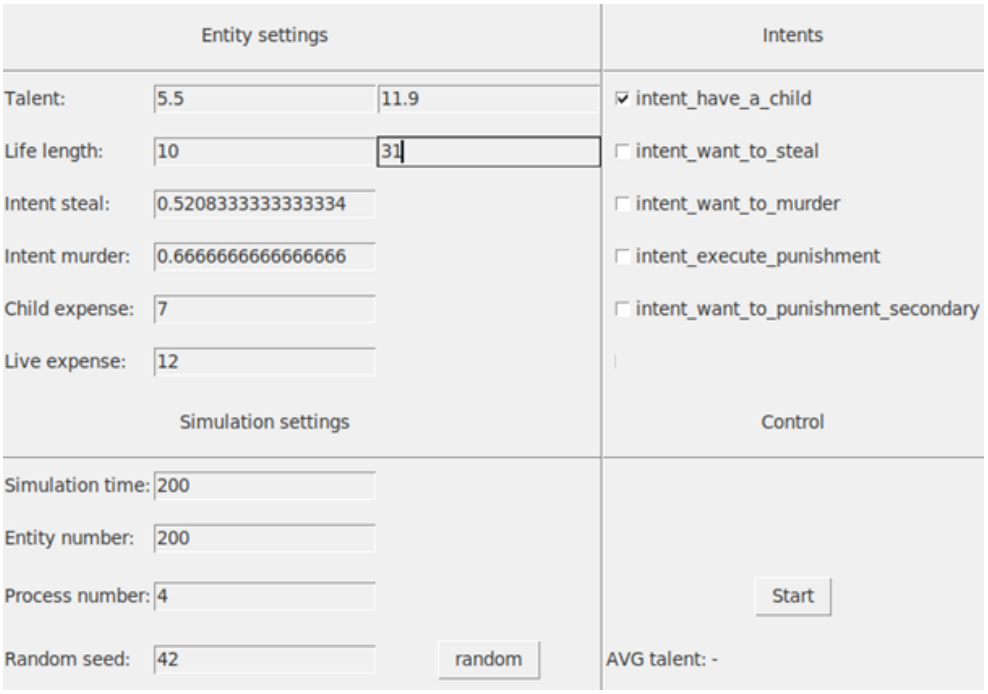


Obrázek 23 Výsledek validace B – produktivita

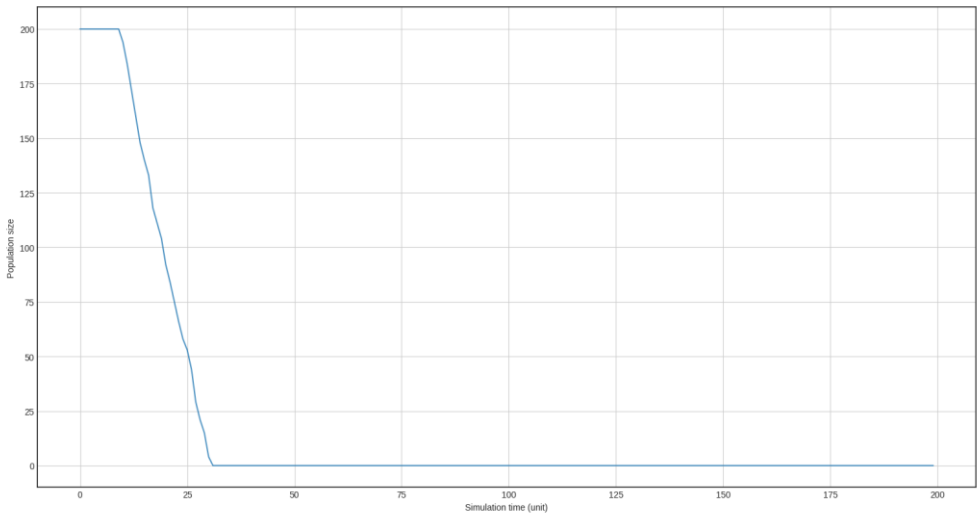
Na grafu produktivity i velikosti populace je vidět velký pokles z počátku simulace. Ten je způsobený zápornou produktivitou a vyhladověním méně talentovaných jedinců. Dále se simulovaná situace vyvíjí kontinuální stagnací. Výsledek validace se shoduje s požadovanou situací.

6.1.3 Situace C, kdy populace nakonec vyhladoví

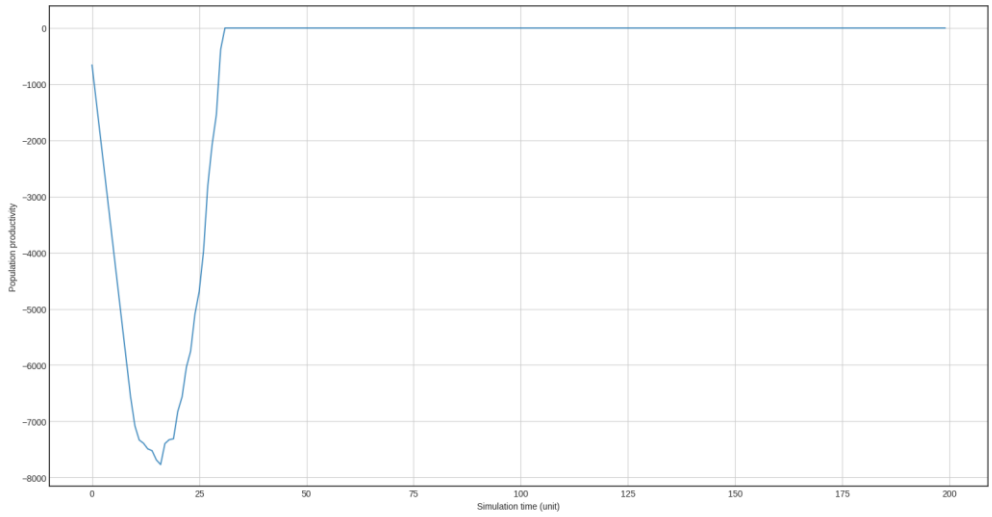
Na obrázku je vidět parametrizace vstupních kritérií a startovních podmínek. V tomto případě bylo zadáním, že všechny entity nakonec zahynou. Parametry jsou nastavené: talent je snížen na relativně malé hodnoty a za zmínku stojí si všimnout i hodnoty living cost, která je těsně nad horní hranicí intervalu talentu. Hodnota child expense je stejně vysoká jako u předchozího scénáře.



Obrázek 24 Parametrizace validace C



Obrázek 25 Výsledek validace C – velikost populace



Obrázek 26 Výsledek validace C - produktivita

Na grafu produkce a velikosti populace je vidět, že entity postupně umírají, až úplně vyhynou a jejich produkce je nulová. Scénář je validní a naplňuje cíle, které byly stanoveny.

6.2 Nastavení vstupních parametrů simulace

Vstupní množina entit je vygenerována pseudonáhodně s nastavením seedu generátoru na 42. Z toho vyplývá, že každá vstupní množina bude vygenerována stejně. Každý experiment bude mít stejné nastavení vstupních parametrů simulátoru podle tabulky níže. Jednotlivé testy se budou lišit pouze v nastavení intentů – jaké jsou povolené. Po splnění ukončovací podmínky a uplynutí simulačního času, se na výstupu objeví výsledky, které budou zakresleny do grafu. Na ose X je simulační čas a na ose Y je současný stav velikosti populace/zločinců + současná produktivita všech jedinců a její popisná statistika. To znamená, že graf ukazuje vývoj populace v čase. Scénář je nastaven tak, aby většina populace měla zvýšenou produktivitu a mít potomka by byla drahá záležitost.

Pro simulátor máme základních šest parametrů, které nastavujeme, což nám dává pro tři nejvýznamnější hodnoty a pro každé z nich 3^6 variant. Z toho důvodu nelze provést všechny možné experimenty. V této kapitole bude pouze několik vybraných případů, které lze do určité míry porovnat s realitou.

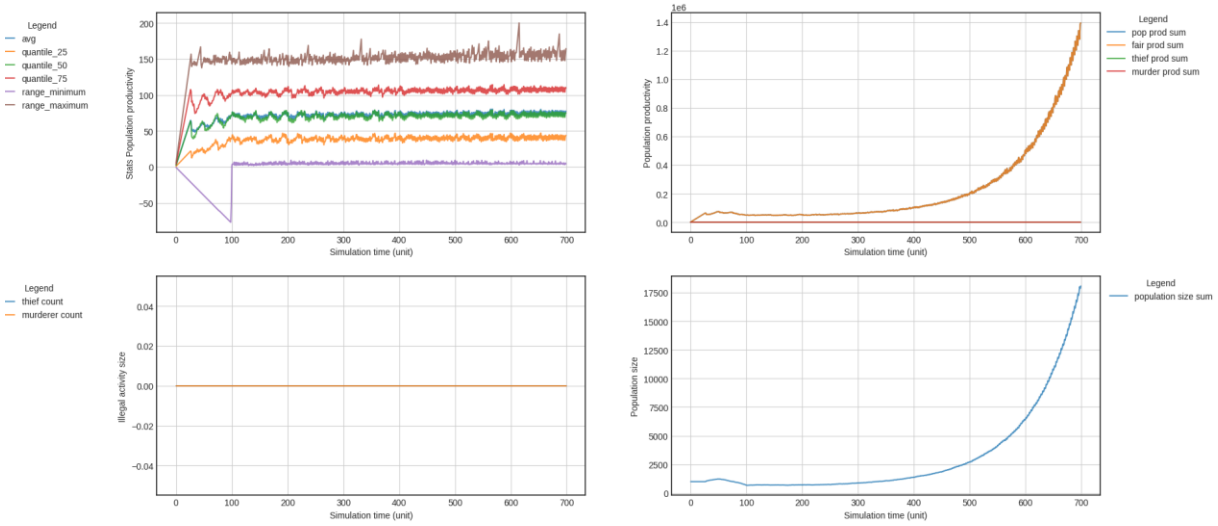
Talent	V intervalu < 5.5; 11.9 >
Multiplikátor krádeže	0.7
Multiplikátor krádeže	0.75
Délka života jedince	V intervalu < 50; 100 >
Životní náklady	5.5
Výdaje na dítě	105
Konec simulace	700
Velikost počáteční množiny	100
Seed generátoru	42

Tabulka 5 Nastavení parametrů pro simulační experimenty

6.3 Experiment 1

Záměry:

- is_on_intent_have_a_child



Obrázek 27 Výsledky simulace

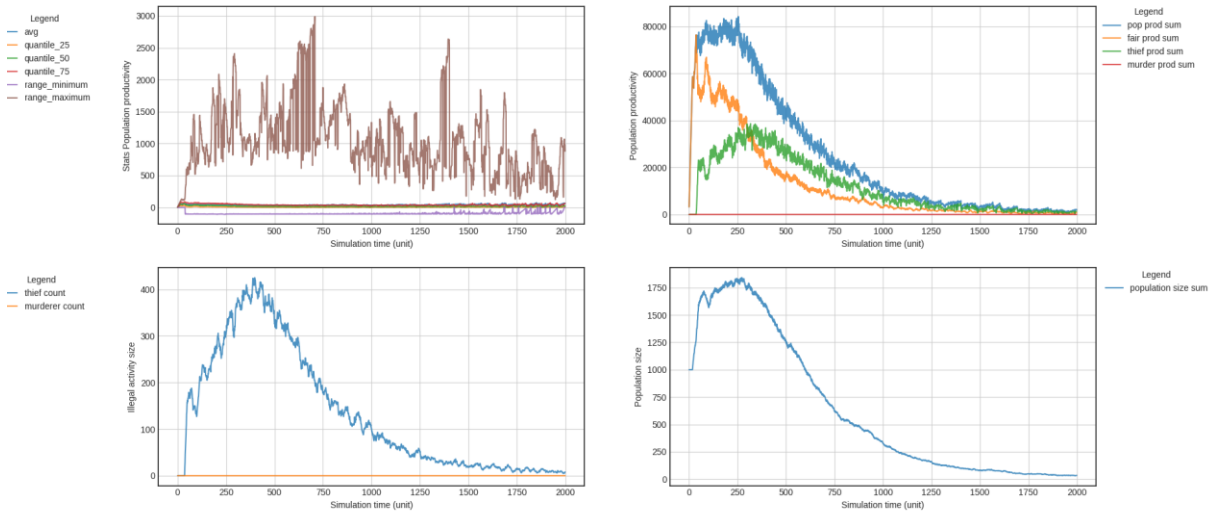
Výsledek simulace nijak nepřekvapí, protože byl podobný situaci A z předcházející kapitoly. Křivka celkové produkce a velikosti populace má exponenciální průběh. Z toho lze usoudit, že

populace prosperuje a pokud by se simulační čas nastavil na více jak 700 jednotek, prosperovala by čím dál tím více. Všechny vstupní parametry byly vybrány tak, aby menšinová část populace finančně nedosáhla na založení si potomstva a proto je hodnota výdaje za jednoho potomka nastavena na 105. To je cena, kterou dokáží talentovanější jedinci za celý svůj životní cyklus zaplatit alespoň jedenkrát, nejtalentovanější dokonce i vícekrát.

6.4 Experiment 2 - krádeže

Záměry:

- is_on_intent_have_a_child
- is_on_intent_want_to_steal



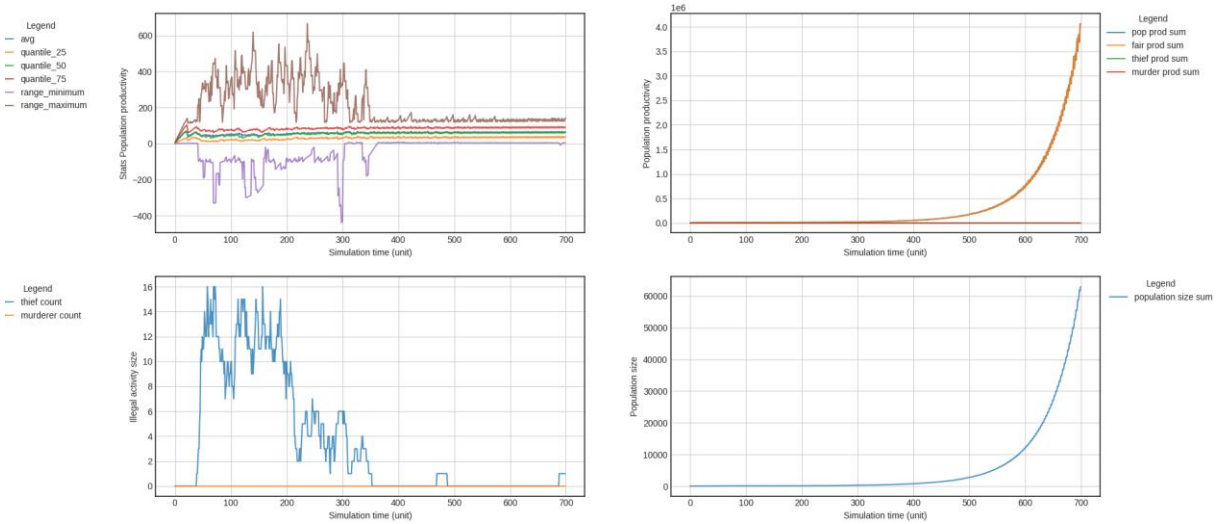
Obrázek 28 Výsledky simulace

Na této situaci je dobře vidět, co se stane, když v prosperující společnosti začnou ti méně talentovaní okrádat produktivní, talentované jedince. Zloději budou prospívat na úkor poctivých a ti začnou pomalu strádat a přestanou mít potomky, kteří by zdědili talent po svých rodičích. Naopak zloději budou mít více potomků s malým talentem a tím pádem populace postupně zdegeneruje, až v ní bude pouze málo jedinců, kteří by generovali produktivně zdroje. Tím je populace odsouzená k záhubě a zanikne.

6.5 Experiment 3 – krádeže s potrestáním

Záměry:

- is_on_intent_have_a_child
- is_on_intent_want_to_steal
- is_on_intent_execute_punishment



Obrázek 29 Výsledky simulace

V experimentu 2 byla popsána situace, ve které se ukazuje, co se může stát, jestliže nikdo nechrání produktivní část populace jedinců před zloději. Výsledek třetího experimentu ukazuje, že jedinci, kteří by normálně využili příležitosti odcizit zdroje jiné entitě, této příležitosti nevyužijí, protože mohou být za své činy potrestáni. Populace je v tomto pokusu prosperující a to při stejném nastavení jako v experimentu 2. Stačí pouze přidat přísné tresty: 50% šance, zabavení všech finančních zdrojů, 45% šance vyhoštění z populace a 5% šance, že bude jedinec vyhoštěn i se svými potomky.

6.6 Ostatní experimenty

Při experimentování s nastavování parametrů se povedlo nasimulovat mnoho situací, které se však nevešly do rozsahu práce nebo se je nepodařilo správně vyhodnotit, protože tento systém je nelineární a zhodnocení či posouzení by vyžadovalo součinnost se sociology a to by bylo nad rámec této práce. Avšak pro inspiraci je dobré si popsat pár zajímavých situací, které si může uživatel zkusit nasimulovat sám.

Například situace, kdy v simulátoru z počátku dominuje specifický počet ilegálních aktivit až nakonec ve druhé až třetí generaci jedinců nastane zvrat a přežije pouze několik vybraných poctivých jedinců, kteří založí prosperující společnost. Vrahové a zloději začnou vyvíjet svoji aktivitu na produktivní společnosti, až nakonec páchají ilegální činnosti jedni na druhých. Z původního scénáře je potřeba nechat vše až na child expense, které se musí nastavit na 130. Záměry, které se mají povolit: have a child, want to steal, want to murder.

Stejný scénář, ale povolené tresty a povolené vyšší tresty (pro již trestané). V tomto případě tresty mají opačný účinek nežli u experimentu 3. Pokud v populaci kde je majoritní podíl zločinců, začneme trestat, ovlivní to i poctivou menšinu. Poctivým došli partneři, se kterými se předtím mohli rozmnožit, a proto jejich úzká talentovaná komunita nyní nepřežila.

Na závěr jiná situace, kdy mírnější tresty pomohou populaci přežít, ale záměr trestat podruhé je zakázán. Koeficient zločinnosti (0.4 intent steal; 0.45 intent murder;) se nastaví na nižší číslo, čímž budou jedinci více konat zločiny. Výdaje za dítě child expense se nastaví 100. Povolí se záměr trestu execute punishment. V tomto případě populace nakonec přežije a začne prosperovat.

Alternativní scénáře k předchozímu: povolením obou záměrů trestu execute punishment, want to punishment secondary populace zanikne. Na druhou stranu nepovolení ani jednoho záměru trestu zapříčiní, že populace rovněž zanikne.

Komentovat z jakých příčin nastávají tyto jednotlivé stavy vývoje populace, přísluší pouze sociologům a podobným odborníkům.

ZÁVĚR

Všechny cíle práce byly splněny. V rámci práce byly představeny v první kapitole základní pojmy a členění evolučního/genetického algoritmu. Ve druhé kapitole se čtenář seznámí s problematikou simulací, která je nedílnou součástí vyvinuté aplikace. Dále byla rozvedena myšlenka o autonomním vývoji algoritmu v řízeném prostředí a kauzalitě v evolučním programování v kapitole třetí. Cílem vytvořené aplikace bylo simulovat autonomní vývoj algoritmů (agentů, jedinců) v řízeném prostředí s podružným úkolem umožnit výzkum sociologických aspektů společnosti. Běh simulátoru ovlivňuje fitness funkce, nastavitelná vstupní množina jedinců a limitní omezení systému. Program zobrazuje vybraná data ze simulace na výstupu.

Implementaci simulačního jádra předcházelo testování několika možnými variantami jejich řešení provázené s úkonem měření výkonosti, za účelem vytipování toho nejvhodnějšího. Pro návrh grafického rozhraní byl vytvořen nákres, který byl následně za pomoci aplikačních nástrojů přetvořen do jazyka Python. Největším programátorským oříškem bylo vyřešit všechny problémy souběhu multiprocesingu paralelní simulačního jádra a komunikaci mezi individui.

V rámci diplomové práce jsem podrobně nastudoval problematiku evolučních algoritmů, simulačních systémů a autonomního vývoje jedince v simulátoru. Aplikaci jsem navrhl tak, že uživatel si nejprve nastaví parametry pro entitu a simulátor a teprve potom si spustí instanci simulace vývoje populací jedinců v čase. Po uplynutí nastaveného simulačního času se všechna data zapíší do souboru a zobrazí se okno s vykresleným grafem.

Diplomová práce mi přinesla mnoho skvělých zkušeností zejména v odvětví evolučních algoritmů, simulátorů, paralelního programování, ošetřování problémů souběhu. Pomohla mi s rozšířením znalostí při navrhování softwarové architektury a designu UI. V poslední řadě nesmím zapomenout, že mě zasvětila do odvětví autonomního vývoje evolučních algoritmů v řízeném prostředí, které může mít využití i v komerční sféře. Pokud bychom měli kvalitně nasbíraná a dobře vyhodnocená sociologická data, mohli bychom nastavit chování entit v simulátoru a následně bychom mohli předpovědět chování člověka na trhu, popřípadě předpovědět vývoj cen nemovitostí nebo vývoj epidemie. Záleží pouze na fantazii.

POUŽITÁ LITERATURA

[1] BEYER, Hans-Georg & SCHWEFEL, Hans-Paul. (2002). *Evolution strategies – A comprehensive introduction. Natural Computing.* 1. 3-52. 10.1023/A:1015059928466.

[2] HOLLAND, John H.: *Adaptation in Natural and Artificial Systems. Ann Arbor*, MI: University of Michigan Press, 1975. - second edition, 1992.

[3] HYNEK, Josef. *Genetické algoritmy a genetické programování*. Praha: Grada, 2008. Průvodce (Grada). ISBN 978-80-247-2695-3.

[4] KVASNIČKA, Vladimír. *Evolučné algoritmy*. Bratislava: Vydavateľstvo STU, 2000. Edícia vysokoškolských učebníc. ISBN 80-227-1377-5.

[5] FRANĚK, Ondřej. *Použití evolučních optimalizačních technik k řešení NP úplných problémů* [online]. Pardubice, 2013 [cit. 2020-02-07]. Dostupné z: https://dk.upce.cz/handle/10195/53637. Bakalářská práce. Univerzita Pardubice, Fakulta elektrotechniky a informatiky. Vedoucí práce Doležel, Petr.

[6] DE JONG, Kenneth & FOGEL, David & SCHWEFEL, Hans-Paul. (1997). *A history of evolutionary computation*.

[7] BORTEL, Martin. *Evoluční algoritmy* [online]. Brno, 2011 [cit. 2020-02-07]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace?zp_id=39817. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Lambertová, Petra.

[8] STUDNIČKA, Vladimír. *Genetické algoritmy – Multi-core CPU implementace* [online]. Brno, 2010 [cit. 2020-02-07]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace?zp_id=33756. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Matoušek, Radomil.

[9] TÁZLAR, Ondřej. *Vliv selekčních metod na tvorbu nové populace a chování genetického algoritmu* [online]. Hradec Králové, 2013 [cit. 2020-02-07]. Dostupné z: https://theses.cz/id/1btx1k/STAG63940.pdf. Bakalářská práce. Univerzita Hradec Králové, Fakulta informatiky a managementu. Vedoucí práce Hynek, Josef.

[10] POHLHEIM, Hartmut. *GEATbx: Introduction: Evolutionary Algorithms: Overview, Methods and Operators* [online]. 3.8. 2006, 95 s. [cit. 2020-02-08]. Dostupné z: http://www.geatbx.com/

[11] KŘIVÝ, Ivan a Evžen KINDLER. 2001. *Simulace a modelování.* [Online] 2001. [Citace: 14. 2. 2020.] Dostupné z: https://vendulka.zcu.cz/Download/Free/SkriptaKindlerMS.pdf

[12] KELTON, W. David, Randall P. SADOWSKI a David T. STURROCK. *Simulation with Arena*. 3rd ed. New York, NY: McGraw-Hill Higher Education, c2004. ISBN isbn0-07-285694-7.

[13] JEŽEK, Ladislav. *Simulační nástroj železniční stanice založený na agentově orientovaném výpočetním jádru* [online]. Pardubice, 2010 [cit. 2020-02-14]. Dostupné z: http://hdl.handle.net/10195/37333. Diplomová práce. Univerzita Pardubice, Fakulta informatiky a elektrotechniky. Vedoucí práce Bažant, Michael.

[14] ADAM, Jiří. *Jádro animace synchronizované s metodou snímání aktivit* [online]. Pardubice, 2013 [cit. 2020-02-14]. Dostupné z: http://hdl.handle.net/10195/52164. Bakalářská práce. Univerzita Pardubice, Fakulta informatiky a elektrotechniky. Vedoucí práce Kavička, Antonín.

[15] KAVIČKA, Antonín, Valent KLIMA a Norbert ADAMKO. *Agentovo orientovaná simulácia dopravných uzlov*. V Žiline: Žilinská univerzita, 2005. ISBN 80-8070-477-5.

[16] RANTOŠ, Daniel. *Simulátory síťového prostředí* [online]. Praha, 2016 [cit. 2020-04-04]. Dostupné z: https://dspace.cvut.cz/handle/10467/65245. Bakalářská práce. České vysoké učení technické v Praze, katedra telekomunikační techniky. Vedoucí práce Bezpalec Pavel.

[17] DORDA, Michal. *Úvod do modelování a simulace systému* [online]. Dostupné z: http://homel.vsb.cz/~dor028/ Aplikace_2.pdf. [cit. 2020-02-14].

[18] PERINGER, Petr. *Modelování a simulace* [online]. Dostupné z: http://www.fit.vutbr.cz/study/courses/IMS/public/prednasky/IMS.pdf. [cit. 2020-02-14].

[19] CHARVÁT, Karel. *Diskrétní simulace v MS Excel* [online]. Praha, 2009 [cit. 2020-03-31]. Dostupné z: <https://theses.cz/id/z98gfo/>. Diplomová práce. Vysoká škola ekonomická v Praze. Vedoucí práce Martina Kuncová.

[20] MORAVČÍK, Štěpán. *Simulační jádro se zaměřením na dopravní systémy* [online]. Pardubice, 2010 [cit. 2020-03-31]. Dostupné z: https://dk.upce.cz/handle/10195/37605. Bakalářská práce. Univerzita Pardubice, Fakulta informatiky a elektrotechniky. Vedoucí práce Bažant, Michael.

[21] POOLE, David & RAFTERY, Adrian. (2000). *Inference for Deterministic Simulation Models: The Bayesian Melding Approach*. Journal of The American Statistical Association – JOURNAL OF THE AMERICAN STATISTICAL ASSOCIATION. 95. 10.1080/01621459.2000.10474324.

[22] KAVIČKA, Antonín. (2017). *Sylaby přednášek předmětu „Pokročilé techniky modelování a simulace“*. Pardubice.

[23] RÁBOVÁ, Z. ČEŠKA, M. ZENDULKA, J. *Modelování a simulace*. [skriptum] Brno: VUT, Praha SNTL, 1982

[24] BARTUŠEK, Martin. *Simulace obchodního domu* [online]. Praha, 2009 [cit. 2020-04-04]. Dostupné z: <https://is.cuni.cz/webapps/zzp/download/130015823/>. Bakalářská práce. Univerzita Karlova v Praze, Matematicko-fyzikální fakulta. Vedoucí práce Černý, Jakub.

[25] ŠILHÁNEK, Jiří. *Simulační metody jako nástroj rozhodování – modelování pomocí programu witness* [online]. Brno, 2007 [cit. 2020-04-04]. Dostupné z: <https://is.muni.cz/th/u9xqq/DP_-_Silhanek_Jiri.pdf>. Diplomová práce. Masarykova univerzita, Ekonomicko-správní fakulta. Vedoucí práce Škapa, Radoslav.

[26] MIKULE, Jindřich. *Mobilní aplikace podporující fitness life style* [online]. Pardubice, 2015 [cit. 2020-05-31]. Dostupné z: <http://hdl.handle.net/10195/64876>. Bakalářská práce. Univerzita Pardubice, Fakulta informatiky a elektrotechniky. Vedoucí práce Brožek, Josef.

[27] WIEGERS, Karl E. *Software Requirements* (2nd ed.). Redmond, WA: Microsoft Press, 2003. ISBN 0-7356-1879-8.

[28] REJNKOVÁ, Petra. *Diagram případů užití*[online]. [cit. 2020-07-07]. Dostupné z: http://uml.czweb.org/pripad_uziti.htm

[29] ČÁPKA, David. *Lekce 2 - UML - Use Case Diagram*[online]. [cit. 2020-07-09]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-use-case-diagram

[30] ČÁPKA, David. *Lekce 3 - UML - Use Case Specifikace*[online]. [cit. 2020-07-19]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-use-case-specifikace-diagram

[31] TIŠNOVSKÝ, Pavel. *Nástroje pro tvorbu UML diagramu.* [online]. [cit. 2020-07-20]. Dostupné z: http://www.root.cz/clanky/nastroje-pro-tvorbu-uml-diagramu/

[32] FÁBERA, Vít. *Konstrukce konečného automatu pomocí gramatické evoluce.* [cit. 2021-05-12]. Dostupné z: https://portal.cvut.cz/wp-content/uploads/2017/04/HP2013-28-Fabera.pdf

[33] KOZA, John R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

PŘÍLOHY

Příloha A – Formát zpráv 80
Příloha B – Nastavení parametrů simulace 82
Příloha C – Použité nástroje a technologie 83
Příloha D – Grafické uživatelské rozhraní 84
Příloha E – Use case specifikace aplikace 87
Příloha F – UML diagram aplikace 91
Příloha G – Algoritmy - metody synchronizace simulačního výpočtu 92
Příloha H – Verze knihoven 96

PŘÍLOHA A – FORMÁT ZPRÁV

Zprávy se posílají přes instanci třídy PostOffice metodami notify_to, notify_all, atd. Zprávy mají určitý formát, ve kterém jsou entitám a jiným objektům (seznamka,…) posílány, aby následně mohly být zpracovány. Kód zprávy, který je uváděn v ukázce, je pouhým odkazem na typ zprávy, tím pádem programátor musí uvést do kódu zprávy celý textový řetězec typu zprávy.

YOU_HAVE_A_CHILD (0)					
index	0	1	2	3	4
význam	Kód zprávy	ID partnera	ID dítěte	Je matkou?	Talent dítěte
ukázka	0	“Entity-8“	“Entity-48“	True	7.4264475

SEND_MONEY (1)			
index	0	1	2
význam	Kód zprávy	Peníze	Odesílatel
příklad	1	5520.8875488	“Thread-8“

YOU_HAVE_BEEN_ROBBED (2)			
index	0	1	2
význam	Kód zprávy	ID zloděje	Byl trestán?
ukázka	2	5.5208875488	19272.83930

YOU_HAVE_BEEN_MURDERED (3)			
index	0	1	2
význam	Kód zprávy	ID vraha	Byl trestán?
příklad	2	5.5208875488	19272.83930

WANT_PUNISHMENT (4)		
index	0	1
význam	Kód zprávy	ID entity
příklad	4	“Thread-8“

WANT_SECONDARY_PUNISHMENT(5)				
index	0	1		
význam	Kód zprávy	ID entity		
příklad	5	“Thread-8“		
EXECUTE_BLOOD_RELATED (6)				
index	0	1		
význam	Kód zprávy	ID entity		
příklad	6	“Thread-8“		
LOOKING_FOR_PARTNER (7)				
index	0	1	2	3
význam	Kód zprávy	ID entity	Talent entity	Peníze entity
ukázka	7	“Entity-48“	5.5898634	7.4264475

Tabulka 6 Formát zpráv

PŘÍLOHA B – NASTAVENÍ PARAMETRŮ SIMULACE

Název	Dat. typ	Default	Vysvětlení
is_on_intent_execute_punishment	boolean	True	Povolení/zakázání trestu
is_on_intent_want_to_punishment_secondary	boolean	False	Povolení/zakázání 2. trestu
is_on_intent_have_a_child	boolean	False	Povolení/zakázání hledání partnera – vždy true
is_on_intent_want_to_steal	boolean	False	Povolení/zakázání krádeží
is_on_intent_want_to_murder	boolean	False	Povolení/zakázání vražd
VAL_INTENT_STEAL	double	0.5208333	Násobitel četnosti krádeží. Čím menší číslo tím četnější jsou krádeže.
VAL_INTENT_MURDER	double	0.6666666	Násobitel četnosti vražd. Čím menší číslo tím četnější jsou vraždy.
VAL_CHILD_EXPENSE	double	6.7	Náklady na jednoho potomka, pokud možno platí oba rodiče
DEAD_T_START	int	50	Začátek intervalu konce života
DEAD_T_END	int	100	Konec intervalu konce života
VAL_LIVE_EXPENSE	double	20.0	Náklady na život jedince. Odečítají se každou simulovanou jednotku času.
TALENT_START	double	5.5	Začátek intervalu talentu entity
TALENT_END	double	11.9	Konec intervalu talentu entity
NUM_ENTITIES	int	100	Velikost počáteční množiny
SIM_TIME	int	10	Ukončovací čas simulace. Počet jednotek času, které musí uplynout, aby simulace skončila
NUM_PROCESSES	int	4	Počet vytvořených simulátorů. Každý simulátor si vytvoří právě jeden proces (maximálně si vytvoří až po počet procesorů v PC/PCs).
RANDOM_SEED	int	42	Nastavení semínka (seed) generátoru náhodných čísel

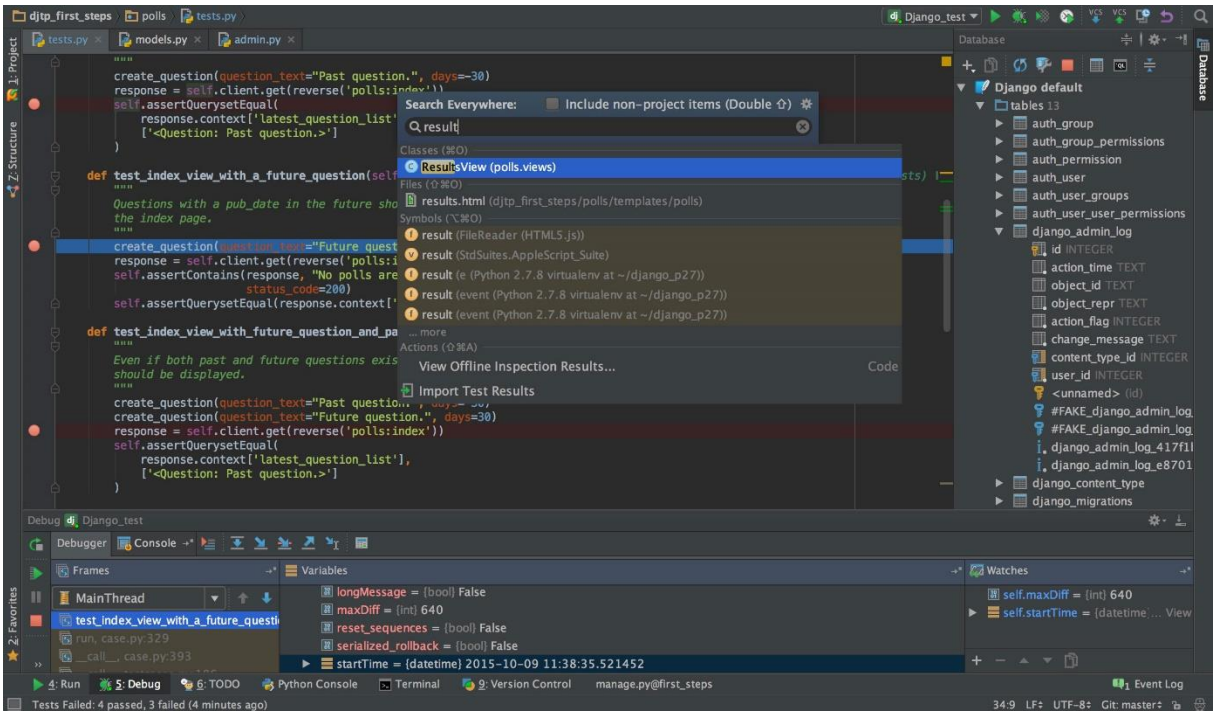
Tabulka 7 Parametry simulátoru

PŘÍLOHA C – POUŽITÉ NÁSTROJE A TECHNOLOGIE

Obsahem této přílohy jsou zejména nástroje a technologie, které byly použity při tvorbě aplikace Simulační jádro pro rozvoj algoritmů.

PyCharm

Komerční vývojové prostředí pro programování v jazyce Python a jeho frameworků. Nejzásadnější výhodou používání IDE, a jiných softwarů je úspora času a energie. Programování v příkazovém řádku nebo v textovém editoru už jsou dávno minulostí. I když papír a tužka je stále nejspolehlivějším nástrojem programátora.



Obrázek 30 PyCharm

Python

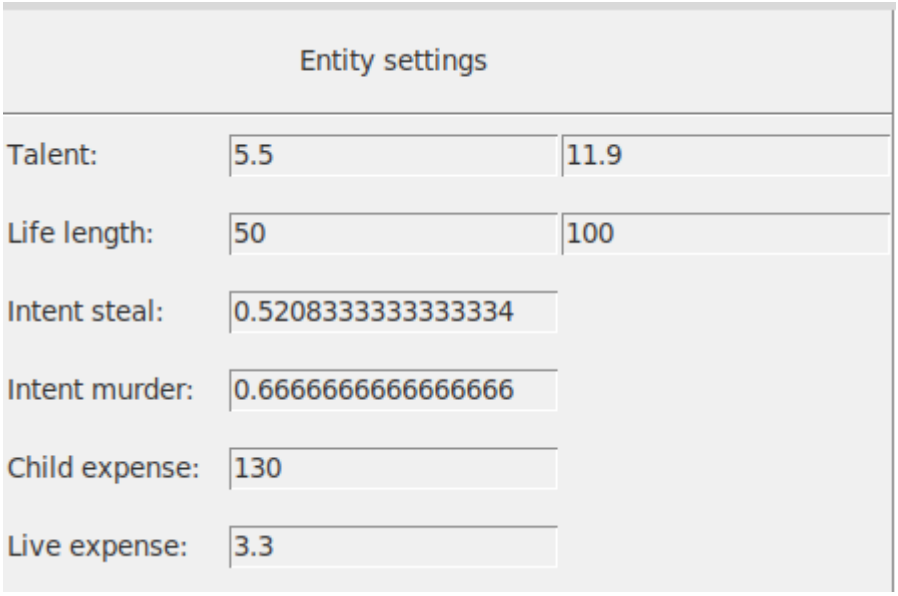
Python je vysokoúrovňový skriptovací programovací jazyk. Nabízí dynamickou kontrolu datových typů a podporuje různá programovací paradigmata, včetně objektově orientovaného, imperativního, procedurálního nebo funkcionálního. Jedná se o jeden z nejpoužívanějších programovacích jazyků na světě a je to také jazyk, který je používaný v této práci. Má obrovskou škálu knihoven, které se dají lehce použít. Obsahuje i knihovnu tkinter, která poskytuje nástroje pro tvorbu a zobrazení grafického rozhraní. Jako hlavní výhodu lze uvést snadný a rychlý vývoj aplikací.

PŘÍLOHA D – GRAFICKÉ UŽIVATELSKÉ ROZHRANÍ

Uživatelské rozhraní umožňuje ovládat aplikaci pomocí grafických ovládacích prvků. Na monitoru počítače jsou zobrazena okna, ve kterých programy zobrazují svůj výstup. Uživatel používá pro interakci klávesnici, myš a grafické vstupní prvky. Grafické rozhraní v této aplikaci je napsáno v anglickém jazyce.

Nastavení parametrů entit

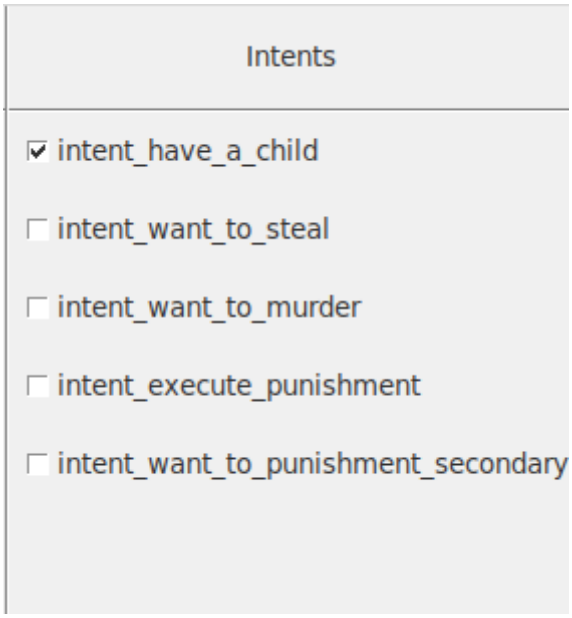
Entita představuje v simulaci jedince, který se snaží být co nejúspěšnější v celé populaci. Každý jedinec se narodí s různými parametry, které jej charakterizují. Tyto parametry jsou nastavitelné v určitém intervalu, který si uživatel vybere. Čísla jsou generována pseudonáhodnými generátory čísel. Čísla ze začátku intervalu (hodnota *From*) musí být větší nežli na konci (hodnota *To*). Tento panel splňuje případ užití UC3.



Obrázek 31 Nastavení entity

Nastavení záměru

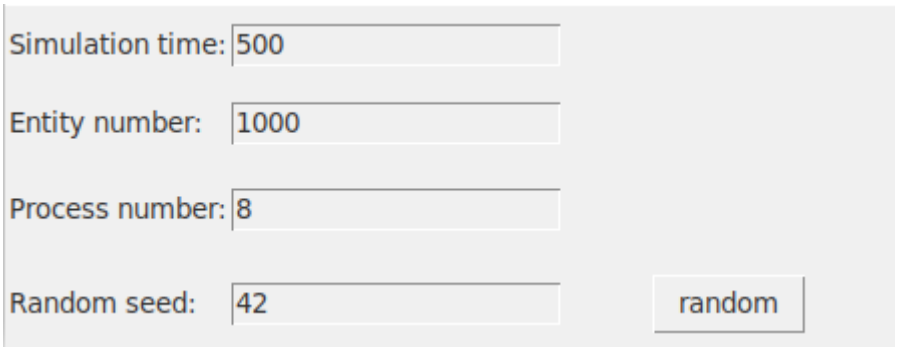
Každá entita má své úmysly, co chce za svůj život dělat. K tomu právě slouží záměry (z anglického slova intent). Uživatel se může rozhodnout jedincům tyto intenty odebrat nebo přidat. Zaškrtnuté pole povoluje entitám záměry a odškrtnuté je odebírá. Tento formulář plní specifikaci případu užití UC1.



Obrázek 32 Nastavení záměrů

Nastavení simulace

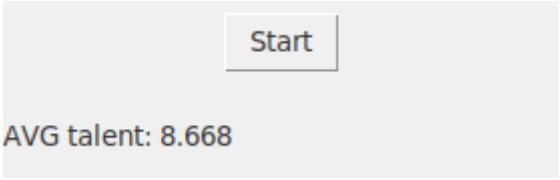
Simulační jádro pro rozvoj evolučních algoritmů je potřeba ještě před spuštěním nastavit. Počáteční podmínky simulace jsou důležité pro budoucí odvíjení simulace, a proto je dobré kontrolovat stav několika počátečních proměnných. Jednou z nich je velikost vstupní množiny jedinců, která určuje počet vygenerovaných entit na úplném začátku simulace. Další důležitý parametr je ukončovací podmínka simulace. Zde je atribut chápán jako jednotka simulačního času do ukončení simulace. Další parametry jsou počet procesů a nastavení semínka, podle kterého se generují pseudonáhodná čísla. Tato pasáž splňuje podmínky z případu užití UC2.



Obrázek 33 Nastavení simulace

Ovládací panel

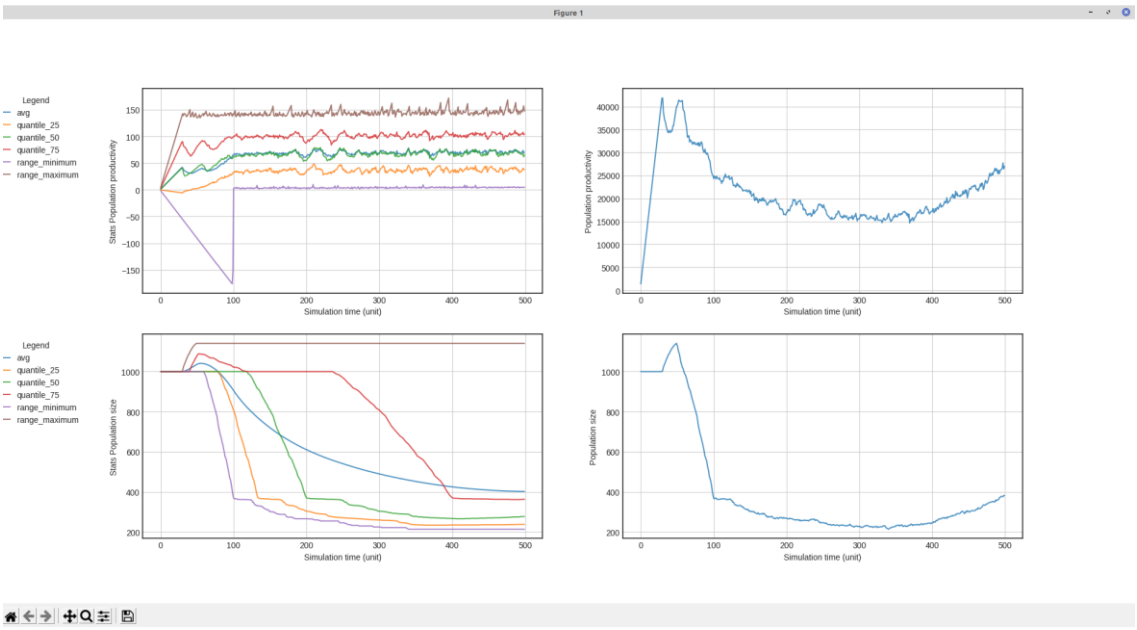
Na ovládacích panelech bývají vždy tlačítka, která ovládají program. Ani v tomto případě to není jiné. Tlačítko *Start* odstartuje celou simulaci vývoje evolučních algoritmů. Těsně před tím, než se pustí, nahlédne do nastavení, které mu bylo předáno ze všech entry a checkbuttonů. Tento ovládací panel byl naprogramována podle specifikace užití UC4.



Obrázek 34 Ovládací panel

Zobrazení výstupu

Aby mohl být vyhodnocen simulační experiment, je nutné výstupy ze simulátoru zaznamenat a následně zobrazit. Výsledky průběhu simulace jsou zapisovány do textového souboru, který je vždy pojmenován po verzi buildu programu. Následně je pak možné s těmito daty dále pracovat v tabulkovém procesoru nebo jiných statisticko-analytických nástrojích. Pro případ, kdy si uživatel zvolí simulaci s validní ukončovací podmínkou, má možnost si nechat zobrazit graf průběhu simulace. Tato funkce plní specifikaci případu užití UC5.



Obrázek 35 Grafický výstup

PŘÍLOHA E – USE CASE SPECIFIKACE APLIKACE

Využití use case specifikace v praxi si lze demonstrovat a rozebrat na příkladu, který vychází z use case diagramu aplikace z kapitoly číslo 4.2.2.

UC01 – set intents

Krátký popis

Use case umožňuje zvolit si nastavení jednotlivých záměrů, které bude entita při spuštění simulace vykonávat.

Aktéři

- Systém
- Uživatel

Podmínky pro spuštění

Uživatel se musí nacházet v nastavení simulace.

Základní tok

1.1 Systém vygeneruje formulář, ve kterém je seznam všech záměrů

1.2 Uživatel vybere ze seznamu záměry v libovolném množství.

1.3 Systém zvaliduje data od uživatele.

1.4 Systém uloží nastavení.

Alternativní tok

Žádný

Podmínky pro dokončení

Nové nastavení bude korektně uloženo v systému.

UC02 – set simulation

Krátký popis

Use case umožňuje zvolit si počáteční podmínky běhu simulace.

Aktéři

- Systém
- Uživatel

Podmínky pro spuštění

Uživatel se musí nacházet v nastavení simulace.

Základní tok

1.1 Systém vygeneruje formulář, ve kterém se nachází všechna textová pole náležící nastavení simulace. Každé pole je náležitě popsáno a jednoznačně identifikováno.

1.2 Uživatel nastaví hodnoty textových polí.

1.3 Systém zvaliduje data od uživatele. **A1, A2**

1.4 Systém uloží nastavení.

Alternativní tok 1

2.1 Pokud nejsou nastaveny žádné hodnoty, uloží si systém defaultní hodnoty.

Alternativní tok 2

3.1 Pokud jsou hodnoty nevalidní, uloží systém pouze defaultní hodnoty.

Podmínky pro dokončení

Nové nastavení bude korektně uloženo v systému.

UC03 – set entity

Krátký popis

Use case umožňuje nastavit všechny parametry, které se týkají entity.

Aktéři

- Systém
- Uživatel

Podmínky pro spuštění

Uživatel se musí nacházet v nastavení simulace.

Základní tok

1.1 Systém vygeneruje formulář, ve kterém se nachází všechna textová pole nastavující počáteční parametry entity. Každé pole je stručně popsáno a jednoznačně identifikováno.

1.2 Uživatel nastaví hodnoty textových polí.

1.3 Systém zvaliduje data od uživatele. **A1, A2**

1.4 Systém uloží nastavení.

Alternativní tok 1

2.1 Pokud nejsou nastaveny žádné hodnoty, uloží si systém defaultní hodnoty.

Alternativní tok 2

3.1 Pokud jsou hodnoty nevalidní, uloží systém pouze defaultní hodnoty.

Podmínky pro dokončení

Nové nastavení bude korektně uloženo v systému.

UC04 – run simulation

Krátký popis

Use case umožňuje systému spustit simulaci.

Aktéři

- Systém
- Uživatel
- Entita

Podmínky pro spuštění

Uživatel se musí nacházet v nastavení simulace. Uživatel musí mít nastavené hodnoty pro entitu, simulaci a mít vybrané záměry entit. CPU a OS musí podporovat multiprocessing.

Základní tok

1.1 Systém vygeneruje tlačítka pro každý typ výstupu.

1.2 Uživatel klikne na tlačítko.

1.3 Systém vygeneruje vstupní množinu entit podle nastavených parametrů.

1.4 Systém spustí simulaci.

1.5 Entita interaguje s ostatními entitami.

1.6 Systém si uloží výsledky průběhu simulace.

1.7 Systém ukončí simulaci. **A1**

Alternativní tok 1

2.1 Pokud není nastavená ukončovací podmínka, systém běží nekonečně dlouho.

Podmínky pro dokončení

Dostatek paměti na RAM a odpovídající výpočetní výkon CPU.

UC05 – show output

Krátký popis

Use case umožňuje zobrazit výstup ze simulace.

Aktéři

- Systém

Podmínky pro spuštění

Uživatel se musí nacházet v nastavení simulace. Uživatel musí mít nastavené hodnoty pro entitu, simulaci a mít vybrané záměry entit.

Základní tok

1.1 Systém čte výsledky průběhu simulace.

1.2 Systém pošle data na textový nebo grafický výstup.

1.3 Systém umožní uložit data. **A1**

Alternativní

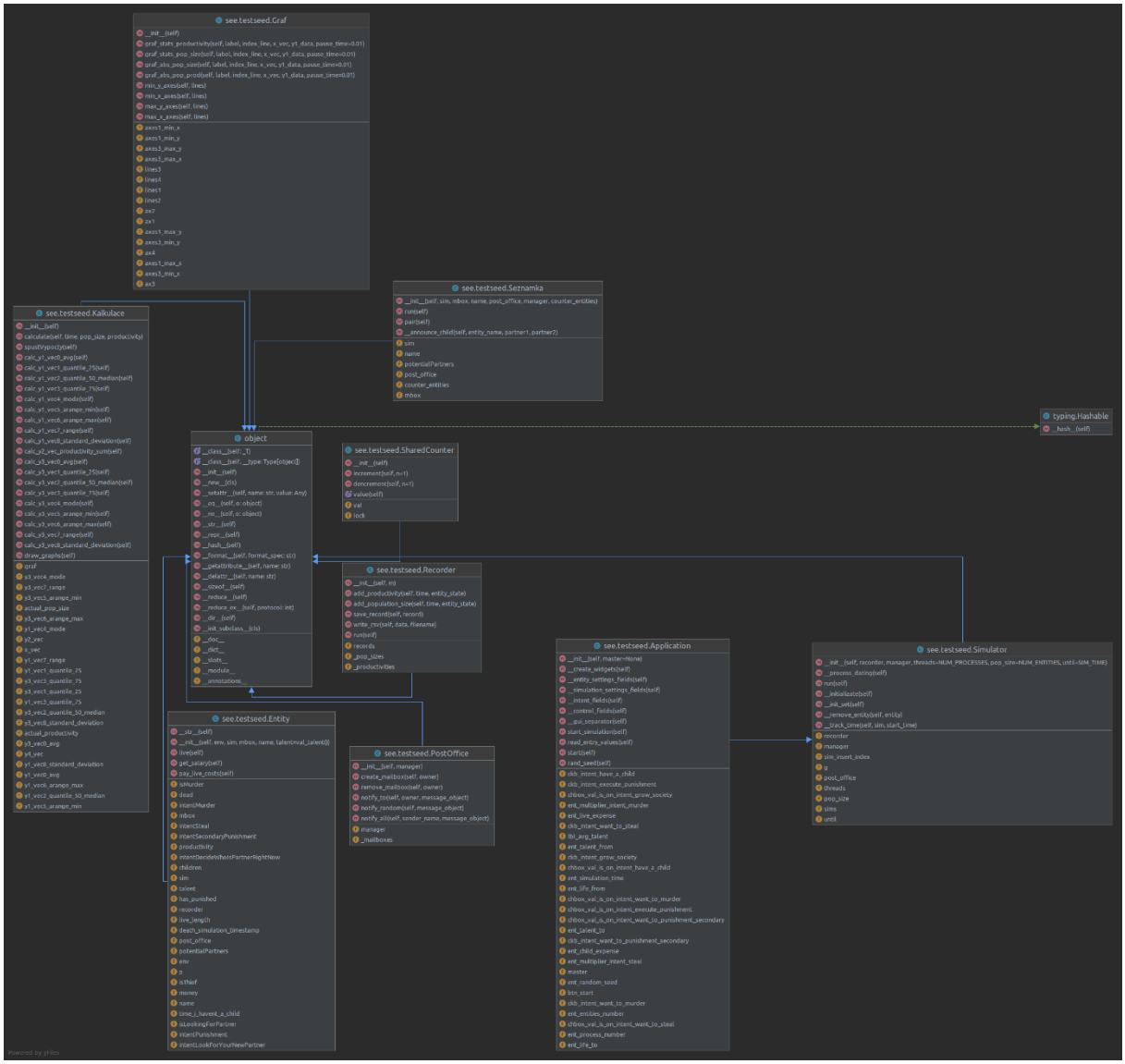
2.1 Pokud systém nemá dostatek volného místa, data nejsou uložena.

Podmínky pro dokončení

Systém musí mít dostatek volného místa na disku.

PŘÍLOHA F – UML DIAGRAM APLIKACE

Z diagramu tříd lze vyčíst, jak v něm budou sestavené všechny třídy, které aplikace bude obsahovat. Třídy budou mít všechny atributy a také metody. Diagram je platformě závislý, tedy specifický pro určitý programovací jazyk a v tomto případě se jedná o jazyk Python. Mimo jiné to znamená, že se v identifikátorech již nevyskytuje diakritika, atributy mají datové typy specifické.



Obrázek 36 UML

PŘÍLOHA G – ALGORITMY - METODY SYNCHRONIZACE SIMULAČNÍHO VÝPOČTU

Krok	Činnost	Ukončení za podmínek
0	Inicializace simulačního času $t_s=0$	
1	Ukončení běhu simulačního programu	Čas určený pro běh simulačního programu byl vyčerpán
2	**1. fáze** Vyhodnocení všech běžících aktivit vzhledem k času t_s a výkon příslušných akcí v případě, že jsou splněny aktivační podmínky	
3	**2. fáze** Aktualizace simulačního času $t_s = t_s + \tau$	
4	Návrat na **krok 1**	

Tabulka 8 Algoritmus snímání aktivit

Krok	Činnost	Ukončení za podmínek
0	Inicializace simulačního času $t_s=0$	
1	Ukončení běhu simulačního programu	Čas určený pro běh simulačního programu byl vyčerpán nebo kalendář neobsahuje žádné události
2	Odebírání první události z kalendáře (tj. událost s nejmenší hodnotou plánovaného času výskytu t_u)	
3	Aktualizace simulačního času $t_s = t_u$	
4	Výkon akce spojené s výskytem události (akce provádí stavové změny a případné naplánování dalších událostí)	
5	Návrat na **krok 1**	

Tabulka 9 Algoritmus plánování aktivit

Krok	Činnost	Ukončení za podmínek
0	Inicializace simulačního času $t_s=0$	
1	Ukončení běhu simulačního programu	Čas určený pro běh simulačního programu byl vyčerpán nebo kalendář neobsahuje žádné události
2	Odebírání první události z kalendáře (tj. událost procesu, která je aktivní a má nejmenší hodnotou plánovaného času výskytu t_u)	
3	Aktualizace simulačního času $t_s = t_u$	
4	Výkon akce spojené s výskytem události (akce provádí stavové změny a případné naplánování dalších událostí)	
5	Návrat na **krok 1**	

Tabulka 10 Algoritmus interakce procesů

Krok	Činnost	Ukončení za podmínek
0	Inicializace simulačního času $t_s=0$ Rozmístění plánovaných aktivit	
1	Ukončení běhu simulačního programu	Čas určený pro běh simulačního programu byl vyčerpán nebo neexistují plánované aktivity
2	Snímací funkce vybere časově nejbližší aktuální plánovaná událost s nejmenší hodnotou času výskytu t_u	
3	Aktualizace simulačního času $t_s= t_u$	
4	Výkon akce spojené s výskytem události (akce provádí stavové změny a případné naplánování dalších událostí)	
5	Testování podmínkových aktivit + jejich vykonávání	
6	Návrat na **krok 1**	

Tabulka 11 Algoritmus třífázové metody

PŘÍLOHA H – VERZE KNIHOVEN

Aplikace používá následující verze knihoven:

cycler	== 0.10.0
greenlet	== 1.1.0
kiwisolver	== 1.3.1
matplotlib	== 3.4.1
numpy	== 1.20.2
pandas	== 1.2.4
Pillow	== 8.2.0
Pyparsing	== 2.4.7
python-dateutil	== 2.8.1
pytz	== 2021.1
scipy	== 1.6.3
simulus	== 1.2.1
singleton	== 0.1.0
six	== 1.16.0
tqdm	== 4.60.0
tkinter	== 8.60.0

Verze Pythonu == 3.8.5