Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

# Vytvoření uživatelského rozhraní k aplikaci pro studium digitální evoluce

***Martin Plávek***

Vedoucí práce: RNDr. Jiřina Scholtzová, Ph.D.

7. ledna 2016

# Poděkování

Poděkování patří všem, kteří mě podporovali během celého studia, stejně tak jako při psaní této práce. Zvláštní poděkování patří přítelkyni, která mi byla oporou v nejtěžších chvílích. Také bych rád poděkoval vedoucí práce, RNDr. Jiřině Scholtzové, Ph.D, za vedení práce a za cenné rady, které vedly k dokončení bakalářské práce.

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 7. ledna 2016 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2016 Martin Plávek. Všechna práva vyhrazena.
*Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.*

### Odkaz na tuto práci

Plávek, Martin. *Vytvoření uživatelského rozhraní k aplikaci pro studium digitální evoluce*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

# Abstrakt

Cílem této práce bylo zanalyzovat, navrhnout a implementovat grafické uživatelské rozhraní pro aplikaci Svet, který slouží k pochopení základů biologické evoluce. Byla provedena počáteční analýza původního řešení - aplikace Svet napsané v programovacím jazyce C. Vzhled uživatelského rozhraní byl navrhnut pomocí technologie Swing a nová verze aplikace Svet 2.3.14 byla implementována v programovacím jazyce Java. Nové řešení umožňuje práci s aplikací Svet bez nutnosti úpravy zdrojového kódu a čtení výstupů bez nutnosti otevírání textových souborů.

**Klíčová slova** digitální evoluce, Java, Swing, grafické uživatelské rozhraní

# Abstract

The objective of this thesis was to analyze, design and implement graphic user interface for the Svet application which is supposed to help us understand the principles of biological evolution. An initial analysis of the original solution (application in the C programming language) was done. The appearance of user interface was designed using Swing technology and a new version of the Svet application – Svet 2.3.14 – was implemented in Java programming language. The new solution gives us opportunity to work with the Svet without

the necessity of updating the application source code and it provides us with an output without opening any text files.

**Keywords** digital evolution, Java, Swing, graphic user interface

# Obsah

# Seznam obrázků

# Seznam tabulek

# Úvod

Biologická evoluce je v přírodě sledovaný proces, při kterém dochází nejen ke vzniku, rozmnožování, vývoji, ale i k zániku organismů. Tento proces je ve většině případů zdlouhavý, například vývoj lidstva takového, jak ho známe dnes, trval tisíce let. Je tedy téměř nemožné vývoj nějakého druhu sledovat vzhledem k době, jakou evoluce vyžaduje. Oproti této době je život člověka, jenž chce evoluci studovat, velice krátký. U organismů s krátkou dobou života se o to sice pokusit můžeme, ale stále budeme potřebovat roky na to, abychom nějaké změny na organismech vypozorovali. Navíc k takovému zkoumání je potřeba mnoho prostoru z důvodu, že je nutné uchovávat tzv. fosilní záznamy. S nástupem počítačů a jejich stále se zvyšujícím výkonem se otevřela nová možnost zkoumání vývoje organismů na Zemi pomocí simulace životních cyklů organismů. Vznikla tak nová oblast zabývající se vývojem organismů – digitální evoluce.

Digitální evoluce je počítačová simulace evoluce biologické. Má především tu výhodu, že oproti biologické evoluci je mnohonásobně rychlejší. Můžeme říci, že během několika minut jsme schopni nasimulovat to, co by v přírodě trvalo desítky let. Systémů, které evoluci zkoumají, je již několik. Můžeme zmínit například systém Tierra [1], Avida [2] nebo Svet, který vytvořil V. Scholtz ve spolupráci se studenty VŠCHT v Praze. Simulace vývoje druhů může nejen biologům, ale i všem lidem, kteří se o evoluci zajímají, pomoci pochopit nahodilost a zákonitost přírody.

Cílem této práce je převzít program pro digitální evoluci Svet, který není pro vlastní uživatele příliš přívětivý, a navrhnout pro něj grafické uživatelské rozhraní. Toto uživatelské rozhraní má zjednodušit práci se Svetem tak, aby v průběhu evoluce bylo možné sledovat její vývoj a snadno analyzovat výsledky, například sledovat jevy jako je speciace, symbióza nebo parazitismus. Zároveň by mělo umožnit snadno nastavit parametry, které ovlivňují vývoj samotné evoluce.

Kapitola 1

# Analýza

V první kapitole si blíže představíme systémy Avida, Tierra a Svet a upřesníme si požadavky, které na nové uživatelské rozhraní pro Svet klient má. Jednou z podstatných částí této analýzy je porozumět současnému stavu systému Svet z důvodu požadavku na zachování původní struktury a funkčnosti. V závěru zanalyzujeme další požadavky a sestavíme si seznam těch, které jsou realizovatelné.

Informace o stávajících systémech jsem čerpal z [1, 2, 3].

## 1.1 Systémy digitální evoluce

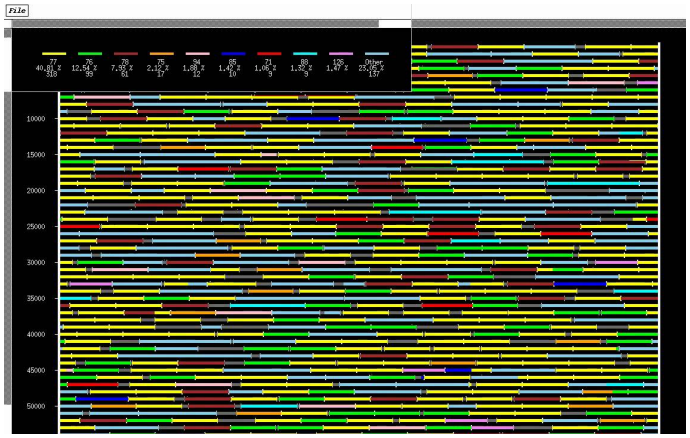
Během vývoje digitální evoluce vzniklo několik systémů simulujících vývoj organismů, ale jen některé z nich byly úspěšné. Ty neúspěšné většinou ztroskotaly na využití genetických algoritmů, které postrádaly některé důležité aspekty evoluce. Částečně se však těmito špatně fungujícími systémy nechali inspirovat vědci a učitelé z předních světových univerzit a vytvořili systémy, které svými principy byly již velice blízko reálné biologické evoluci.

Tato práce se zabývá systémem Svet, ale popíšeme také dva systémy, kterými se autoři Sveta inspirovali.

### 1.1.1 Tierra

Tierra je systém digitální evoluce vyvinutý Thomasem S. Rayem na začátku 90. let. Systém je navržen tak, že pro evoluci je vyhrazena paměť, ve které jsou nakopírované digitální organismy. Ty jsou reprezentované kódem (programem), představující genom organismu, který je uložen v části paměti a chráněn proti přepisu jinými organismy. Tento kód je pak vykonáván v cyklu, a to paralelně pro všechny organismy v systému.

Kód organismu je ekvivalentem genomu živých organismů. Pro vytvoření umělého genomu organismů v systému Tierra byl vytvořen programovací jazyk Terrian složený z 32 pětibitových instrukcí. Jednou z nejdůležitějších vlast-




Obrázek 1.1: Ukázka výstupu systému Tierra

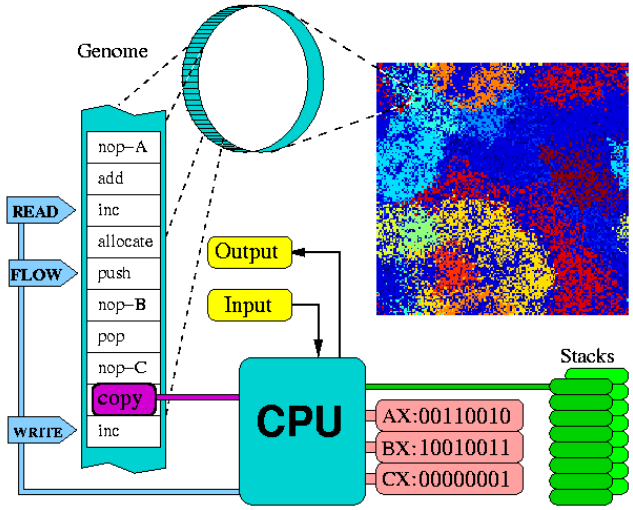
ností systému je pak alokovat a poté chránit proti zápisu dva bloky paměti – mateřskou a dceřinou buňku organismu. Alokací paměti a její ochranou před zápisem se systém Tierra přibližuje buněčnému principu, kde organismus představuje buňku s polopropustnou membránou. Polopropustná proto, že i když ostatní organismy nemohou zapisovat do chráněné paměti jiného organismu, mohou číst nebo dokonce i spouštět kód uložený v této paměti.

Pro organismy se typicky alokuje 64 kB paměti, ze které si každý organismus alokuje část pro sebe. Nejdříve se organismy množí a zaplňují tak přidělenou paměť. Ve chvíli, kdy je paměť asi z 80 % zaplněná, začnou „vadné" organismy vymírat, a tím i uvolňovat jimi alokovanou paměť. Jaký organismus zahyne v určitou chvíli je řízeno frontou, do které se odzadu vkládají nově vytvořené procesy a proces ze začátku fronty umírá jako první. Toto pořadí se může v průběhu evoluce měnit například tím, že při vykonávání instrukce nastane náhodná mutace, která způsobí posun organismu blíže k začátku či konci fronty.

Ke změně v genomu organismu (mutaci) v systému Tierra dochází v nepravidelných intervalech dvěma způsoby. Jednak dochází k prohození náhodného bitu v rámci celé paměti, popřípadě k prohození náhodného bitu při provádění operace zápisu.

Systém je oživován jedním organismem dlouhým 80 instrukcí, který se umí jen rozmnožovat. Ve chvíli, kdy zaplní paměť systému, můžeme pozorovat i poslední podmínku evoluce – kompetici.

Výstupem systému je grafické zobrazení celé paměti, kde jsou jednotlivé organismy barevně rozlišovány. Každá řádka grafického výstupu znázorňuje rozdělení paměti mezi organismy v určitém čase (viz obr. 1.1).




Obrázek 1.2: Schéma virtuálního počítače v Avida

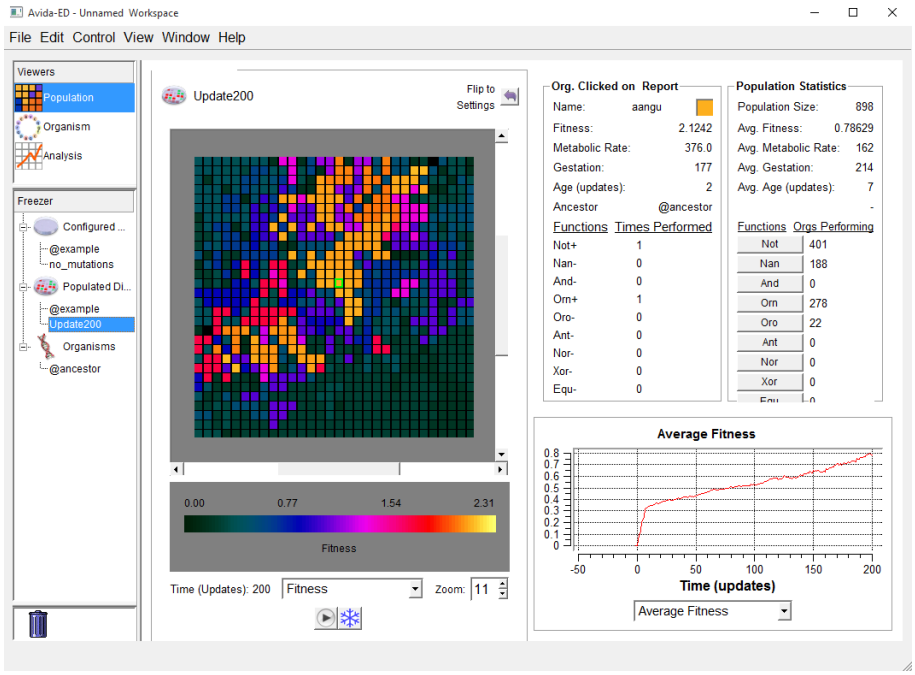
### 1.1.2 Avida

Původní verze systému Avida byla vytvořena v roce 1993 na Michigan State University. Hlavním tvůrcem je Charles Orfia, později se přidali Chris Adami a C. Titus Brown. V roce 2007 byla pro veřejnost zpřístupněna Avida-ED, což je výuková verze systému Avida pro studenty biologie, na kterém si mohou sami vyzkoušet mechanismy evoluce. Na jejím vývoji se podílí odborníci z předních amerických univerzit.

Avida má na rozdíl od systému Tierra trochu jinou architekturu. Každý proces (organismus) představuje vlastní virtuální počítač s vlastním virtuálním procesorem, který vykonává jeho instrukce. Dále je pak tvořen třemi registry a pamětí, ve které je uložen kód (genom) organismu. Procesor každého organismu obsahuje čtyři hlavy ukazující do paměti. Instrukční hlavou je čten kód organismu, čtecí a zapisovací hlava slouží ke čtení a zápisu z a do paměti při procesu kopírování, a řídící hlava je používána při skocích a smyčkách [3].

Životní prostor organismů je tvořen dvourozměrnou, cyklicky uzavřenou mřížkou, kde každý bod mřížky představuje jeden organismus (viz obr. 1.2). Místo aby byl strojový čas na vykonávání svých instrukcí přidělován podle pevně daných pravidel, v Avida se o strojový čas „soutěží". Za úspěšné vyřešení problémů získá organismus čas navíc. Tím je do systému přidán prvek získávání energie a metabolismu organismů.

Mutace v systému Avida může nastat při zápisu, kdy se změní hodnota, a tím je ovlivněn i význam prováděné instrukce, nebo v případě, že se část organismu při rozmnožování nezkopíruje, popřípadě zkopíruje několikrát, a




Obrázek 1.3: Systém digitální evoluce Avida-ED

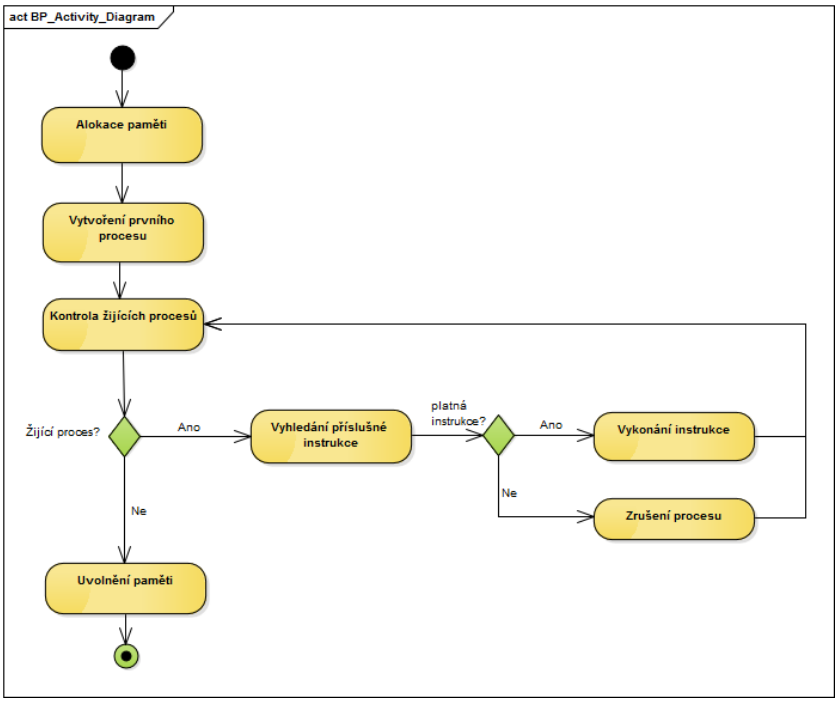
tím dochází ke změně délky potomka.

Novinkou, kterou systém Avida přináší, je získávání prémiového výpočetního času pro organismy, které úspěšně dokončily úkol. Typicky se jedná o logické výpočty. Před tím, než vůbec k výpočtu může dojít, si organismus nejdříve musí příslušnou logickou operaci „vytvořit" pomocí operace NAND (not AND). Několikanásobným použitím této logické operace může organismus vytvořit všechny zbylé, od negace po ekvivalenci.

V původním systému Avida se uživatelé museli spokojit pouze s textovým výstupem s několika základními informacemi o jednotlivých organismech. V případě Avida-ED je již možné sledovat evoluci v reálném čase. Na obrázku 1.3 je možné sledovat probíhající evoluci (vlevo), informace o jednotlivých organismech (uprostřed), údaje o celé evoluci (vpravo) a v tomto případě i graf znázorňující průměrnou kondici ve smyslu úspěchu při rozmnožování.

### 1.1.3 Svet

Autorům systému Svet se na systémech Avida a Tierra nelíbila především snaha násilně se přizpůsobit biologickým systémům. Chtěli systém, který by nebyl založený na dělení buněk, z nichž některé mohou navíc být chráněné. Odstraněním těchto vlastností se Svet ještě více přiblížil realitě, ve které buňky také nejsou ničím chráněny před vnějšími jevy.



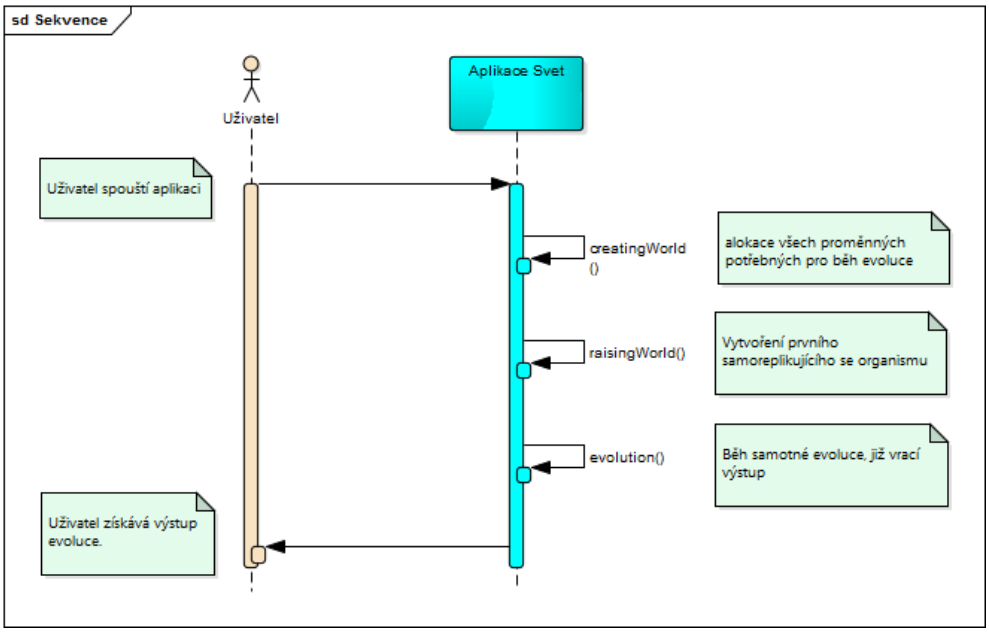

Obrázek 1.4: Diagram aktivit pro původní aplikaci

Systém pracuje jako počítač von Neumannova typu s přímým adresováním přes 32 bitový registr. Kromě sady instrukcí Intel 8080 tvoří instrukční sadu systému Svet i instrukce pro náhodný výběr čísla, sloužící k výběru místa pro vytvoření potomka při rozmnožování.

Na začátku je systém oživen jedním organismem, dlouhým 55 instrukcí, nakopírovaným do paměti. Tato celková paměť je pro všechny organismy, nepatří tedy žádnému z organismů. Proces si podle svého ukazatele do paměti vybere na daném místě instrukci, kterou následně vykoná. Z těchto 128 instrukcí je necelá polovina ponechána nedefinovaná (úplný seznam instrukcí pro Svet najdete v příloze B). V případě, že se nedefinovanou instrukci proces pokusí vykonat, provede tzv. neplatnou instrukci a zahyne. Během zápisu při vykonávání instrukce může dojít k prvnímu typu mutace, kdy se změní náhodná zapisovaná hodnota. V náhodně určených časových intervalech se náhodná změna provede na určité části celé paměti. Celý životní cyklus je znázorněn na obrázku 1.4.

Při experimentech v rámci práce [4] autorka zjistila, že nejdůležitějším aspektem úspěšné evoluce v rámci systému Svet je velikost paměti. Pokud se navyšovala alokovaná paměť, snižovala se úmrtnost organismů. Například pokud velikost pole (paměti) bylo 100 instrukcí, velice brzy došlo k vyhynutí celé populace.

Pro vizualizaci standardních textových výstupů aplikace Svet je zapotřebí




Obrázek 1.5: Sekvenční diagram pro původní aplikaci

použít aplikace třetích stran. Pro vytvoření grafů byl například použit Microsoft Excel.

## 1.2 Původní řešení systému Svet

Původní aplikace je napsaná v programovacím jazyce C. Má výhodu v rychlosti, ovšem je relativně omezený okruh uživatelů, který je schopen s aplikací pracovat. Je totiž potřeba znát alespoň základy programování v jazyce C, a to především proto, že k nastavení parametrů evoluce byl potřeba zásah do zdrojového kódu.

Další prostor na zlepšení aplikace poskytují grafické výstupy, které Svet v původní podobě nevytváří. Uživatel mohl evoluci studovat z textových souborů s uloženým výstupem, popřípadě data vizualizovat pomocí dalších aplikací. Například grafy byly vytvořené pomocí Microsoft Excel. Interakce s uživatelem je tedy minimální, jak je možné vidět na obr. 1.5.

Z výše uvedených nedostatků je tedy možné vypozorovat několik možností pro vylepšení. Kromě vytvoření standardního uživatelského rozhraní existuje prostor pro uživatelsky přívětivé nastavování parametrů, vizualizaci výstupů v podobě grafů, popřípadě generačního stromu.

### 1.2.1 Architektura aplikace

Síla původní aplikace je především v její jednoduchosti a přímočarosti. Všechny důležité funkce jsou implementovány v jednom souboru `svet.c`. V souboru `svet_asm.h` je definovaná instrukční sada pro systém a konečně `svet_td.h`, kde jsou definovány původní struktury aplikace.

Důležitou součástí systému digitální evoluce jsou organismy. Organismus je v systému Svet popsán jako strukturovaná proměnná obsahující důležité informace o organismu. Obsah struktury je vypsán níže.

| **Proměnná** | **Význam** |
|---|---|
| id | jednoznačné identifikační číslo procesu |
| pc | adresa aktuální instrukce v paměti (tzv. program counter) |
| pr | druh procesu (ekvivalent k živočišnému druhu, nevyužito) |
| Stack | zásobník procesu |
| r | registry procesu |
| cf | příznak přetečení (tzv. carry flag) |
| zf | příznak nulového výsledku (tzv. zero flag) |
| dbzf | příznak dělení nulou (tzv. division by zero flag) |
| stlast | ukazatel na poslední prvek zásobníku |

Později byl v rámci práce [4] přidán ještě soubor `svet_dbg.h` pro odladění aplikace. Tato část kódu přinesla do systému především několik výpisů, které pomohly sledovat průběh evoluce.

### 1.2.2 Princip aplikace Svet

Oproti systémům Avida a Tierra je princip Sveta blíže realitě, ale i přesto zůstává jednodušší. Evoluce běží v cyklu, který se ukončí v případě, že všechny procesy zahynou. Jak již bylo zmíněno, proces zahyne v případě, že vykoná jednu z neplatných instrukcí, kterých je přibližně polovina.

Paměť systému Svet je reprezentována jednorozměrným polem. Při spuštění evoluce je na začátek paměti nakopírována první sada 55 instrukcí, dále se obsah mění, pokud dojde k mutaci. Proces si pomocí ukazatele do paměti (program counteru) vybere instrukci, kterou se následně pokusí vykonat. Pokud je daná instrukce definovaná, považuje ji za platnou a vykoná ji. Provedou se modifikace pomocných proměnných, v některých případech nastane mutace, a nakonec se podle instrukce posune ukazatel na další prvek paměti. Pokud instrukce platná není, organismus zahyne a je vymazán ze spojového seznamu.

Mutace během vykonávání instrukce není jediná, která v systému může nastat. Můžeme se setkat z tzv. vnější mutací. V aplikaci Svet nastává v náhodně stanovených časových intervalech. Provede se tím způsobem, že náhodně zvolená část paměti se přepíše náhodně zvolenými hodnotami. Tím se ale také

| Označení | Požadavek |
|---|---|
| FP1 | ovládání evoluce z uživatelského rozhraní |
| FP2 | možnost měnit vstupní parametry a první sadu instrukcí přes uživatelské rozhraní |
| FP3 | výpis a vykreslování výstupů do uživatelského rozhraní |
| FP4 | grafy závislostí a generační strom zobrazený do uživatelského prostředí |
| FP5 | možnost exportu výstupů do externích souborů |
| FP6 | možnost uložení použitých vstupních parametrů a prvních replikátorů |
| NP1 | příjemné a především intuitivní uživatelské rozhraní |
| NP2 | možnost použití aplikace na systémech Windows a Linux (přenositelnost) |
| NP3 | zachování struktury a funkčnosti původního programu |

Tabulka 1.1: Přehledová tabulka požadavků

změní instrukce, které jsou na těchto pozicích uložené a může dojít k další mutaci při vykonávání instrukce, která byla změněna.

## 1.3 Sběr a analýza požadavků

V první fázi vývoje nových aplikací je potřeba provést analýzu požadavků, při které se snažíme vymezit hranice systému nebo funkce softwaru. Požadavkem je zadání požadované služby systému a systémové omezení. Řešíme zde otázku „Co máme dělat?", neřešíme ale jakým způsobem.

Z této analýzy lze také odhadnout složitost a pracnost úkolu. Mimo výše zmíněné může popisovat i speciální hardware nebo databázový model.

Požadavky lze rozdělit do dvou skupin – funkční a nefunkční. Funkční požadavky, které říkají, co software bude dělat (např. export grafického výstupu do určitého grafického formátu), a nefunkční požadavky, které se týkají produktu jako celku. Patří sem například požadavky výkonnostní a požadavky na proces vývoje. Dále můžeme zmínit i legislativní nefunkční požadavky. Mezi nefunkční požadavky dále řadíme například požadavky na rychlost, velikost nebo přenositelnost.

### 1.3.1 Požadavky na systém Svet

Výchozím požadavkem na novou verzi systému Svet je zachování stejné funkcionality původní aplikace a vytvoření uživatelsky přívětivého a intuitivního grafického prostředí, které je navíc přenositelné mezi různými operačními systémy.

Dále má nová verze zjednodušit práci se vstupními parametry. Práce s nimi má probíhat pomocí uživatelského rozhraní bez nutnosti zásahu do zdrojového

kódu. Aplikace by měla mít možnost měnit, nastavovat, popřípadě i ukládat vstupní parametry. U prvního replikátoru pak umět tuto sadu instrukcí externě uložit a také ji zpět do systému načíst.

Další požadavky se týkají výstupů. Nová aplikace by měla uživateli poskytovat textové výstupy přímo v uživatelském rozhraní. Dále pak vytvářet grafické výstupy, konkrétně

- graf závislosti vykonávané instrukce na čase,
- graf závislosti „program counteru" na čase,
- graf žijících procesů v daném čase,
- generační strom procesů v rámci evoluce.

Všechny grafické výstupy by mělo být možné exportovat do externích grafických souborů. Souhrn požadavků je v tabulce 1.3.1.

Kapitola 2

# Návrh

Další částí práce je návrh nové aplikace. Budeme se zabývat výběrem vhodných technologií a postupů podle informací získaných během analýzy. Vytvoříme případy užití nové aplikace a matici pokrytí požadavků jednotlivými případy užití. Zaměříme se na způsob dodání aplikace a navrhneme i první prototyp grafického uživatelského rozhraní.

## 2.1 Výběr programovacího jazyka

Dnes jsou pro vývoj desktopových aplikací nejrozšířenější jazyky Java a C#. Oba dva jazyky jsou objektově orientované, mnohoúčelové a vysokoúrovňové. Jako první vznikla Java. Vycházela přímo z programovacího jazyka C, ze kterého odstranila největší nedostatky, například že alokovaná paměť musí být uvolněna uživatelem, aby bylo zamezeno vzniku tzv. „memory leaků". C# vznikl později a autoři se inspirovali především jazykem Java. I když je tedy potomkem nepřímým, syntaxe má stále své základy v programovacím jazyce C.

Přestože jsou si programovací jazyky velice podobné, není možné zvolit C#, a to z důvodu požadavku na přenositelnost aplikace Svet. Aplikace v něm vytvořené totiž nelze spouštět bez podpůrných aplikací na počítačích s jiným operačním systémem, než je Windows. Sama o sobě ale aplikace přenositelná mezi systémy není.

Java je interpretovaný programovací jazyk. To znamená, že ke spuštění aplikace je potřeba její zdrojový kód a interpret. Interpret je program se speciální funkcí pro provedení (interpretaci) kódu. Tento interpret pro jazyk Java je dostupný jak pro operační systém Windows, ale i pro Linux, popřípadě Mac OS pro zařízení od společnosti Apple. Výběrem programovacího jazyka Java tedy splníme požadavek na přenositelnost aplikace.

| UseCase | Požadavek |
|---|---|
| UC1 – Spuštění evoluce | FP1 |
| UC2 – Pozastavení evoluce | FP1 |
| UC3 – Ukončení evoluce | FP1 |
| UC4 – Změna sledovaného procesu | FP1 |
| UC5 – Změna vstupních parametrů | FP2 |
| UC6 – Změna prvního replikátoru | FP2 |
| UC7 – Uložení prvního replikátoru | FP6 |
| UC8 – Načtení prvního replikátoru | FP6 |
| UC9 – Export grafu | FP5 |

Tabulka 2.1: Tabulka pokrytí funkčních požadavků případy užití

## 2.2 Případy užití

K zobrazení systému tak, jak ho vidí uživatel, slouží diagram případů užití. Popisuje to, co od systému očekáváme a co bude umět. V souvislosti s případy užití mluvíme také o aktérech. Je to role, která přímo komunikuje s případy užití. Může jí být osoba pracující s aplikací nebo i nějaký jiný vnější systém.

Případem užití je myšlen řetězec různých akcí, které slouží k dosažení určitého výsledku. Jedna funkcionalita je popsána právě jedním případem užití. Tím, že aktér inicializuje případ užití vyvolá k případu užití přiřazenou funkcionalitu. Jestliže v sobě funkcionalita obsahuje nějakou další akci, v diagramu ji již nezobrazujeme.

V případě systému Svet zavedeme dva aktéry a to uživatele aplikace a systém GraphViz [5] použitý pro vytváření generačních stromů. Všechny případy užití, se kterými může uživatel interagovat jsou popsány v obrázku 2.1.

### 2.2.1 Pokrytí požadavků případy užití

Pro vlastní kontrolu správného navržení systému je vytvořena matice pokrytí funkčních požadavků jednotlivými případy užití. Funkční požadavek může být pokryt více než jedním případem užití, ale případ užití je vždy „přiřazen" k jednomu funkčnímu požadavku. Matici pokrytí vidíme v tabulce 2.2.1.

Tabulka 2.2.1 nezobrazuje požadavky FP3 a FP4. Jsou to výstupy, které uživatel nezíská inicializací jednoho případu užití, ale v rámci spuštění evoluce se spustí i jejich vypisování či vykreslování.

## 2.3 Metodika vývoje

Souhrn pravidel, podle kterých se řídíme během celého životního cyklu vývoje softwaru, se nazývá metodikou vývoje. Při výběru metodiky je nutné si

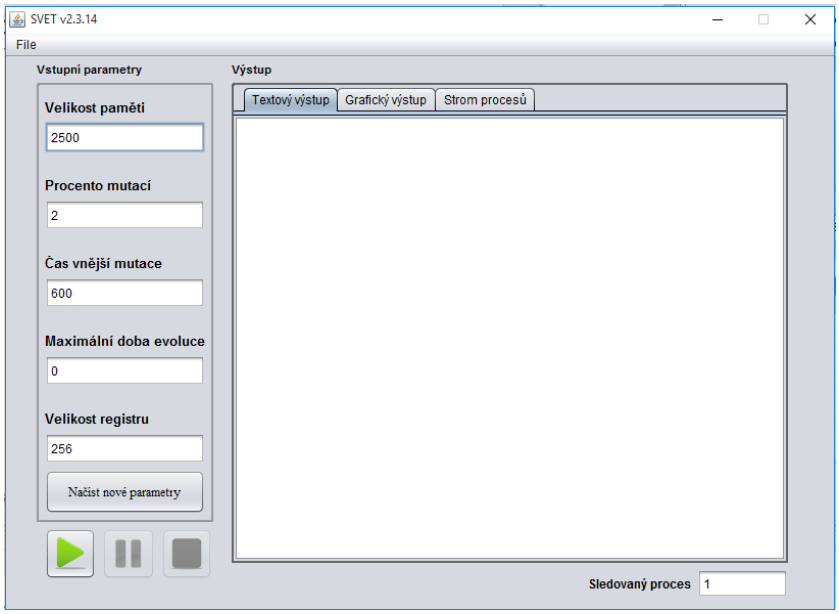

Obrázek 2.1: Diagram případů užití nové aplikace

uvědomit, co z trojice čas, funkcionalita a zdroje je fixní a co je proměnné. V případě, že je fixní funkcionalita, je vhodné zvolit tradiční metodiku (například Vodopádový model). Pokud ale funkcionalita je proměnná a čas a zdroje jsou fixní, je lepší volit ze skupiny agilních metodik, jako je například extrémní programování.

V rámci bakalářské práce jsou zdroje a čas fixní. Z analýzy práce postupně vyplynulo, že funkcionalita aplikace se naopak může v průběhu vývoje ještě lehce změnit a tzv. „trojimperativ" tedy radí k volbě některé z agilních metodik.

Vývoj a především implementační část byla řízena podle prioritního seznamu funkcionalit.

Zvolená agilní metodika pro vývoj Svet je FDD (Feature Driven Development) neboli vývoj řízený užitými vlastnostmi [6]. I přes zachování fáze




Obrázek 2.2: Konečný vzhled uživatelského rozhraní aplikace Svet

modelování systému přináší FDD výhody agilních metodik, ze kterých nejvýznamnější je možnost velice rychle reagovat na změny požadavků nebo na žádosti na změnu v již implementované části aplikace.

Během vývoje podle zvolené metodiky by mělo docházet k častým iteracím v rozmezí přibližně dvou týdnů, vývojový tým by měl být malý, případně jednotlivec a během každé konzultace by mělo být dohodnuto, jaká část úkolu by měla být do další konzultace hotová nebo jaké věci mají být opravené, což bylo splněno, protože na bakalářské práci autor pracoval sám a konzultace v takovýchto intervalech probíhaly.

## 2.4 Grafický návrh

Jedním z úkolů práce bylo inspirovat se (pokud je to možné) při vytváření uživatelského rozhraní a výstupů pro Svet systémy Avida a Tierra. Pro povahu výstupů systému Avida i Tierra, kde je grafickým výstupem zobrazení celé paměti systému, se inspirovat možné není. Z pohledu uživatelského rozhraní nám inspiraci může poskytnout systém Avida-ED, jejíž grafické prostředí se přibližuje představám o rozhraní pro Svet.

Rozložení uživatelského prostředí systému Svet se oproti Avida-ED zjednodušilo a bylo omezeno na tři části. První část je určená pro práci se vstupy, druhá a největší pro sledování výsledků evoluce (výstupy) a třetí část slouží k ovládání celé evoluce.

Z pohledu grafického návrhu prošel největšími změnami panel pro ovládání evoluce, především se vyvíjel vzhled tlačítek pro spuštění, pozastavení a zastavení evoluce. Kromě vzhledu, kdy se tlačítka dostávala z „obyčejného" zevnějšku do tlačítek se standardními znaky pro start, pozastavení a ukončení, se měnila také jejich velikost. Konečný vzhled grafického uživatelského rozhraní je na obrázku 2.2.

Kapitola 3

# Implementace

V následující části práce si popíšeme samotnou implementaci nové verze Svet, konkrétně pak použité nástroje. Detailněji se zaměříme na implementaci složitějších či méně obvyklých částí systému. Předtím si popíšeme vývojové prostředí a architekturu nové aplikace.

## 3.1 Vývojové prostředí

Vývojové prostředí (IDE) je software, který svými nástroji a možnostmi usnadňuje programátorskou činnost. Pro programovací jazyk Java jsou nejrozšířenější NetBeans IDE, Eclipse a IntelliJ IDEA [7, 8, 9].

Programování nové verze aplikace Svet probíhalo za pomoci NetBeans IDE, které naprosto vyhovovalo této práci a autor s ním byl dobře obeznámen.

## 3.2 Vývoj uživatelského rozhraní

Během vytváření grafické podoby nové aplikace bylo využito systému pro rychlý vývoj aplikací (RAD), který je součástí zvoleného vývojového prostředí. V editoru je možné si graficky připravit podobu uživatelského rozhraní, pro který je zároveň i generován zdrojový kód.

Při použití systému RAD si programátor ze škály komponent vybere vyhovující a tu vloží do připravovaného uživatelského rozhraní na místo, kam komponenta patří.

Uživatelské rozhraní nové aplikace Svet bylo vytvořeno z komponent knihovny Swing [10] vycházející ze staršího AWT [11] (Abstract Window Toolkit), které ale nesplňovalo filosofii jazyka Java, protože byl závislý na platformě. Z toho důvodu byl vytvořen právě Swing, který je již plně přenositelný.

### 3.2.1 Swing

Swing je v Javě grafická knihovna, která slouží k vytváření tzv. formulářových aplikací. V jeho standardních knihovnách lze nalézt komponenty od tlačítek po textová pole, které jsou dostačující při vývoji většiny aplikací. Pokud dostupné komponenty při vývoji nestačí, výhodou Swingu je, že si programátor může nový prvek vytvořit, případě upravit stávající komponentu tak, aby odpovídala požadavkům.

### 3.2.2 Alternativy Swingu

V dnešní době je pro vytváření grafického uživatelského rozhraní možné kromě Swingu využít právě i starší AWT. Tato technologie byla však zamítnuta hned zpočátku z důvodu, že jsme chtěli využít jednu z modernějších možností pro implementaci uživatelského rozhraní.

Kromě Swingu se dnes používá Java-FX [12], která má Swing v budoucnosti plně nahradit. Jelikož se ale jedná o relativně novou technologii, množství výukových materiálů a dokumentací nebylo v době, kdy bylo o podobě grafického rozhraní rozhodováno, dostačující.

### 3.2.3 Použité komponenty

Komponenta je základním prvkem uživatelského rozhraní. K tomu, abychom komponenty neumisťovali do prázdna, je nejčastěji používán `JFrame` ze skupiny kontejnerů. Do něj již můžeme přidávat další komponenty použité při implementaci.

K rozdělení hlavního rámce grafického uživatelského rozhraní bylo využito dvou komponent typu `JPanel`. Panel pro výstupy slouží k uložení několika dalších panelů, kde jeden panel bude právě jeden typ výstupu. Panel pro vstupy má sloužit jen ke zvýšení přehlednosti grafického rozhraní aplikace Svet.

Informativní charakter má většina komponent `JTextField`, kromě jednoho pro nastavení sledovaného procesu. Tento typ komponent slouží především pro práci s textem, ať už text zobrazují, nebo do nich lze text programově zapsat pro poskytnutí informací, v tomto případě o aktuálním nastavení vstupních parametrů. Pro přehlednost uživatelů je ke každému `JTextField` přiřazen `JLabel`, který má vždy jen informativní charakter. V aplikaci Svet popisuje, jakou proměnnou právě dané textové pole zobrazuje.

Pro ovládání aplikace slouží komponenty `JButton` a `JMenu`. První zmíněné slouží k interakci aplikace s uživatelem. Menu obsahuje ještě několik komponent `JMenuItem`, které již jsou svým principem velice blízké tlačítkům. Po jejich stisknutí uživatel očekává vykonání slíbené akce. K oběma typům komponent k ovládání aplikace lze přiřadit i klávesové zkratky pro zjednodušení práce s aplikací.

### 3.2.4 Ovládání komponent

Pro ovládání komponent je nutné použít prostředí ActionListener, které jednotlivé prvky oživí. Pokud ActionListener implementujeme a přiřadíme k danému prvku, po interakci prvku s uživatelem se zavolá metoda, ve které je již implementována samotná akce.

V práci byl také použit KeyListener. Funguje téměř na stejném principu jako ActionListener s tím rozdílem, že akce definovaná v KeyListeneru se vyvolává v případě, že byla stisknuta předem určená klávesa nebo kombinace kláves.

## 3.3 Architektura nové aplikace

I když funkcionalita nového Sveta zůstala stejná jako v jeho původní verzi, jeho architektura se stala složitější. Ke změnám došlo i u vstupních parametrů, které již nejsou pevně uložené ve zdrojovém kódu aplikace, ale v souborech typu property, ze kterých jsou proměnné načítány během spuštění evoluce nebo po změně vstupních parametrů.

To, jak spolu jednotlivé třídy interagují, zobrazuje diagram tříd na obrázku 3.1. Obsahuje třídy `Svet`, `Change`, `ChangeR`, `GraphViz`, `Node`, `Organism`, `TreePane`, `loadReplica` a `saveReplica`.

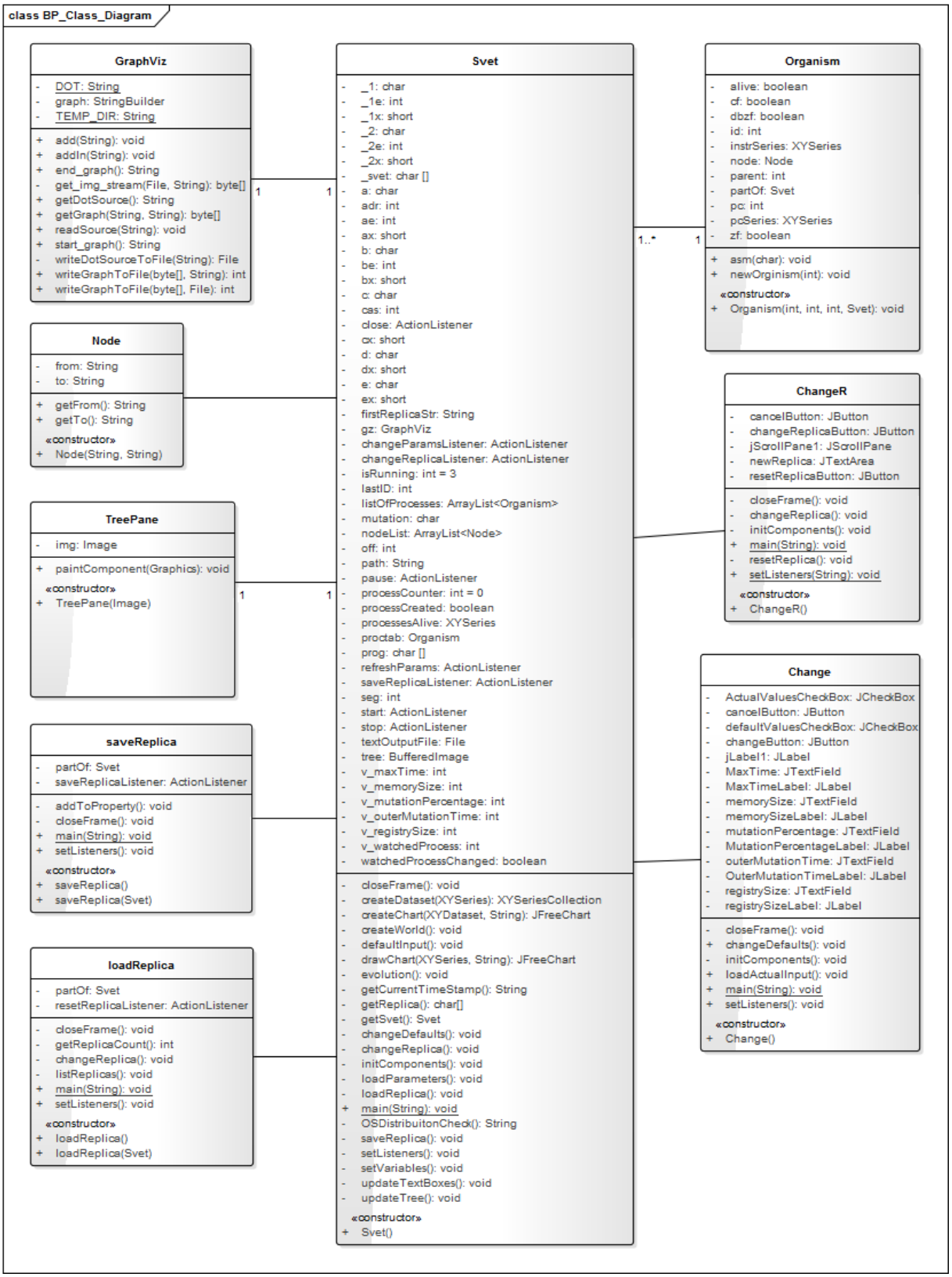
`Svet` je hlavní třídou celé aplikace. Obsahuje všechny důležité metody a proměnné pro běh evoluce. Další důležitou třídou je třída `Organism`, jejíž objekty obsahují důležité proměnné jako například identifikační číslo organismu nebo program counter.

Ostatní třídy jsou určené především k úpravě vstupních proměnných nebo podpoře výstupů. Třída `GraphViz` [13] je využita při vykreslování generačního stromu. K tomu využívá seznam uzlů (třída `Node`), které jsou vytvářeny v případě rozmnožování. Pro zobrazení generačního stromu slouží `TreePane`.

K práci se vstupními parametry byla implementována třída `Change`. Pokud se objekt této třídy volá, vytvoří se nové okno, kde si uživatel může nastavit vybrané parametry. Více možností nabízí práce s první sadou instrukcí. Pokud ji chceme změnit na určité hodnoty, využíváme třídu `ChangeR`. Jestliže se nově nastavená sada instrukcí zdá být úspěšnou, pomocí `saveReplica` si ji můžeme uložit do externího souboru a tím si ji i zachovat do dalšího spuštění aplikace. K jejímu načtení pak využijeme třídy `loadReplica`.

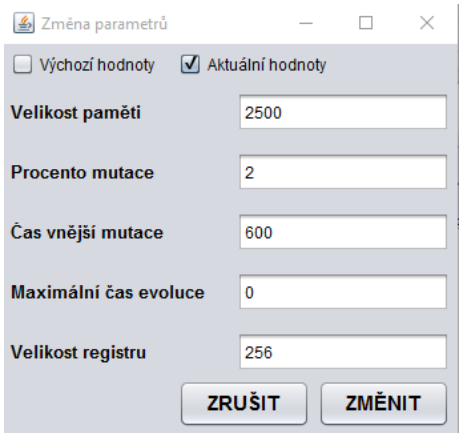
## 3.4 Vstupy a výstupy aplikace

Důležitou částí implementace bylo vytvoření funkcí aplikace pro práci se vstupy a zobrazování výstupů. Popíšeme si implementaci změny vstupních parametrů a možnosti práce s první sadou instrukcí. U výstupů bude vysvětleno, jakým způsobem jsou implementovány grafy závislostí a generační strom.






Obrázek 3.1: Diagram tříd pro novou verzi aplikace Svet




Obrázek 3.2: Formulář pro nastavení vstupních parametrů

### 3.4.1 Změna parametrů

Nastavení vstupních parametrů výrazně ovlivňuje průběh celé evoluce v rámci systému Svet. Vstupní parametry jsou uloženy ve dvou `property` souborech. První obsahuje výchozí hodnoty. Jsou v systému uloženy pro případ, kdyby došlo k chybnému nastavení v druhém souboru (aktuálně používaných hodnot) a pokud by bylo potřeba vrátit se k původnímu nastavení.

K samotné změně slouží formulář vyvolaný přes položku v menu „Změnit parametry", nebo je možné použít klávesovou zkratku `Ctrl+P`. Je zavolán konstruktor formuláře (obr. 3.2), do kterého jsou načteny aktuálně nastavené hodnoty proměnných z aktuálního `property` souboru.

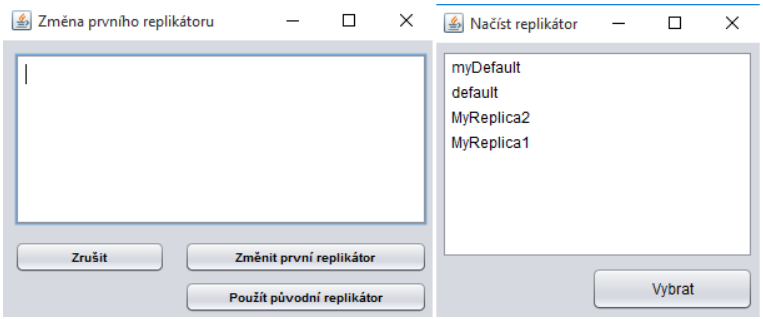
Následně v příslušném textovém poli uživatel provede potřebné úpravy. Ty jsou za pomoci `FileOutputStream` zapsány zpět do `property` souboru, ze kterého jsou po dokončení zápisu nahrány do příslušných proměnných aplikace Svet.

Možnosti změny nastavení proměnných bylo docíleno pomocí Dependency Injection [14]. Do objektu třídy pro změnu parametrů byla přidána reference na objekt typu Svet. Přes tuto referenci můžeme v objektu typu Svet provádět změny parametrů, popřípadě i volat třídní proměnné.

### 3.4.2 Změna první sady instrukcí

Pomocí dalšího formuláře (levá část obrázku 3.3) probíhá i změna první sady instrukcí. Do textového pole jsou vepsány jednotlivé instrukce oddělené čárkou. Při stisknutí tlačítka „Změnit první replikátor" se celý řetězec zapsaný v textovém okně zkopíruje do `property` souboru.

Během vykonávání těla metody `createWorld` se řetězec rozdělí na jednotlivé instrukce (metoda `String.split(",")`), které se pak nahrají na začátek alokované paměti.




Obrázek 3.3: Formulář pro nastavení první sady instrukcí

Při stisknutí tlačítka „Použít původní replikátor" se první sada instrukcí zkopíruje z výchozího `property` souboru do aktuálního.

V případě, kdy by uživatel systému Svet chtěl studovat evoluci pro více sad těchto vstupních instrukcí, je možné si právě používanou sadu instrukcí uložit. Po stisknutí položky v menu „Uložit replikátor" je uživatel dotázán na jméno, pod kterým bude sada instrukcí uložena. Zadané jméno slouží jako identifikátor záznamu v `property` souboru.

Formulář pro načtení replikátoru (pravá část obr. 3.3) zobrazí všechny identifikátory prvních sad instrukcí, které jsou v `property` souborech uloženy. Při výběru se příslušný řetězec instrukcí zkopíruje do proměnné, která je při spuštění nové evoluce rozdělena na jednotlivé instrukce.

### 3.4.3 Grafy závislostí

Stěžejním výstupem pro studium digitální evoluce pomocí systému Svet je možnost pozorovat, jaké instrukce sledovaný organismus vykonává, popřípadě jaké organismy jsou v daném čase živé. K vytvoření těchto grafů bylo během implementace systému Svet použito knihovny třetích stran JFreeChart [15].

Pro to, abychom mohli vytvořit graf, je nutné nejdříve získat data k jeho vykreslení. K uchovávání těchto dat slouží tzv. „XYSeries". Z těchto dat vytvoříme kolekci, ze které se vytvoří „XYDataset". Z toho již můžeme vytvořit samotný graf.

Výsledný graf je ještě nutné zobrazit do uživatelského rozhraní. Nejdříve je vytvořen `ChartPanel`, do kterého je graf vložen. Aby panel byl flexibilní z pohledu délky grafu, pomocí `JScrollPanel` přidáme možnost rozšíření panelu pomocí posuvníků (scrollbarů). Celou komponentu pak vložíme do kontejneru pro zobrazování výstupů. Pak už se jen mohou za běhu přidávat data do „XYSeries" a změna se promítne do výsledného grafu.

### 3.4.4 Generační strom

O vytvoření generačního stromu se stará třída `GraphViz`. Tato třída je volně ke stažení stejně tak jako samotná aplikace, která je třídou externě volána.

Při vytvoření nového procesu se do datové struktury `ArrayList` typu `Node` přidá nový záznam „rodič – potomek". Zavolají se metody třídy `GraphViz`, konkrétně metoda pro vytvoření začátku grafu, metoda pro vytvoření jednotlivých uzlů a metoda pro ukončení grafu. Všechny tyto metody zapisují do zdrojového kódu, který je následně přetransformován pomocí aplikace do výsledného generačního stromu.

Pomocí vstupního proudu pro soubory (`FileInputStream`) si výsledný strom nahrajeme přes `BufferedImage` do nového objektu třídy `treePane`, která implementuje zobrazení obrázku do `JPanel`. Pak už jen stačí přidat tento objekt do kontejneru pro výstupy v uživatelském rozhraní aplikace Svet.

## 3.5 Důležité metody

Implementace několika metod třídy Svet byla stěžejní pro správný běh aplikace. Tyto metody řídí běh evoluce, oživení systému Svet nebo nahrání proměnných z `property` souboru. Tyto důležité metody si popíšeme.

### 3.5.1 loadParameters

Při spuštění systému se do proměnných nejdříve načítají hodnoty. K tomu slouží metoda `loadParameters`. Při spuštění aplikace se vytvoří objekt třídy `Svet`, na který ihned voláme metodu pro načtení příslušných hodnot. Metoda přečte hodnoty z `property` souboru, kde jsou uložené aktuálně užívané hodnoty. Při prvním lokálním spuštění aplikace jsou stejné jako výchozí. Aktuální hodnoty (stejně tak i výchozí) je možné změnit za pomoci třídy `Change` nebo v případě první sady instrukcí (prvního replikátoru) pomocí třídy `ChangeR`.

### 3.5.2 createWorld

Metoda pro inicializaci ostatních proměnných, které mají mít na začátku každé evoluce výchozí hodnoty se volá pokaždé, když uživatel stiskne tlačítko pro spuštění evoluce a v systému není jiná evoluce již spuštěná. Metoda se tedy nevolá, pokud jsme evoluci z nějakého důvodu jen přerušili a chceme v ní pokračovat.

Metoda znovu inicializuje paměť pole, a tím smaže instrukce z předchozí evoluce. Zároveň na začátek paměti nahraje aktuálně nastavenou sadu instrukcí. Kromě paměti je třeba vyčistit i spojový seznam obsahující prvky struktury `Node` pro vytvoření generačního stromu. Vytvoříme první organismus, který je ihned nahrán do seznamu organismů. Tento organismus je třeba

vytvořit aby spuštěná evoluce ihned neskončila z důvodu, že neexistuje žádný žijící organismus a v systému není žádný, který by se mohl vyvíjet.

Vytvoříme také nové `XYSeries`. Smažeme tím stará data o žijících procesech. Dokončením instrukcí v těle metody máme nastavené vše pro spuštění evoluce.

### 3.5.3 evolution

Metoda `evolution` se stará o běh evoluce stejně tak jako o řízení výstupů. První jsou nahrány hodnoty do proměnných, které určují, kde mají být výstupy externě uloženy. Následně se metoda zacyklí a opakovaně vykonává instrukci získanou pomocí program counteru pro každý žijící proces. Před tím, než metoda začne procházet seznam organismů, zkontroluje, zda evoluce nebyla zastavena, pozastavena, nebo zda všechny procesy nevymřely.

Ke zjištění, jestli nějaký organismus ještě žije slouží proměnná uchovávající záznam o počtu žijících procesů. Není možné se pouze podívat, zda je seznam prázdný. Uchováváme totiž i data „mrtvých" organismů, abychom byli schopni sledovat jejich grafy závislostí. Nejdříve se do `XYseries` pro graf zobrazující vykonané instrukce v čase a program counter v čase zapíší nová data, která se také zapíší do textového výstupu ve formátu
*čas – id organismu – program counter organismu – právě vykonávaná instrukce.*
Následně se pro aktuálně vybraný organismus provede instrukce. Ta se volá jako metoda objektu typu `Organism`. Jakým způsobem se instrukce vybírá bude popsáno níže v podkapitole asm.

Ve chvíli kdy jsou všechna data připravená, zobrazí se do uživatelského rozhraní. V případě grafů ještě aplikace kontroluje, zda nezačala nová evoluce nebo jestli nebyl změněn sledovaný proces. Pokud se tak stalo, zabrání duplikaci jednotlivých záložek s grafy tím, že tyto záložky smaže a znovu vloží s daty pro nově sledovaný proces, popřípadě pro první proces vytvořený metodou `createWorld`.

Po skončení cyklu pro procházení organismů se provede kontrola, zda nebylo dosaženo maximálního času evoluce (pokud je nastavený). Pokud limitu bylo dosaženo, evoluce skončí. Všechna data ale zůstávají v aplikaci uložená pro další studium právě ukončené evoluce. Smazána jsou až ve chvíli, kdy se uživatel rozhodne pro spuštění nové evoluce.

### 3.5.4 asm

Pro vykonání instrukce je pro každý proces volána metoda `asm`, která je implementována v třídě `Organism`. Metoda je implementována jako `switch`. Jednotlivé případy jsou reprezentovány číselným ekvivalentem instrukce. Kromě standardních instrukcí se může vykonat i instrukce, která nemá implementovanou žádnou akci (organismus ani paměť, instrukce 0) nebo instrukce pro

vytvoření nového procesu (instrukce 255). Pro vykonání nedefinované (neplatné) instrukce (instrukce 27, 93–127 a 129–254) slouží výchozí operace, která provede nastavení přívlastku, zda je organismus živý, na „false".

V případě vytvoření nového organismu se volá metoda `createProcess`. Ta vytvoří nový objekt typu `Organism`, která jako parametr přebírá program counter rodičovského organismu. Program counter dceřiného organismu je nastaven na stejnou hodnotu jako program counter mateřského, identifikační číslo je poslední ID inkrementováno o 1. Dále si pak uchovává ID mateřského organismu a referenci na objekt `Svet`, do kterého patří.

### 3.5.5 Podpůrné metody

Podpůrné metody slouží k přehlednosti aplikace. Mají za úkol přebírat úlohy, které by zbytečně zdržovaly průběh evoluce, pokud by byly vykonávány v těle metody `evolution`. Většina těchto metod slouží pro překreslení grafických výstupů, které by mohlo zpomalit běh evoluce. Pro zrychlení jsou podpůrné metody implementovány jako vlákna. Pokud je tedy podpůrná metoda zavolána, metoda `evolution` dále pokračuje v cyklickém procházení organismů, zatímco jsou podpůrnými metodami překreslovány grafické výstupy.

Kapitola 4

# Testování

Testování nové aplikace je nedílnou součástí vývoje. Předcházíme tak možným potížím při předávání nového softwaru zákazníkovi. Pro aplikaci Svet byly vytvořeny automatické jednotkové testy, proběhlo manuální funkční testování systému jako celku stejně tak jako akceptační testování na straně klienta. Na závěr si testování vyhodnotíme.

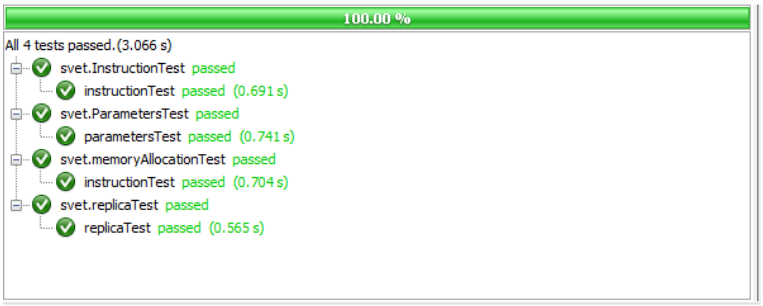
## 4.1 Jednotkové testování

Unit testy slouží k ověřování správného fungování dílčích částí aplikace – jednotek. Za jednotku můžeme považovat celý program, funkci nebo proměnnou, pokud testujeme aplikaci vyvinutou procedurálně. V objektově orientovaném programování je pak jednotkou třída, popřípadě její metoda. Každý test by měl testovat vždy jedinou jednotku, ideálně nezávislou na zbytku aplikace. Z povahy aplikace Svet není možné těmito automatickými testy otestovat celou aplikaci z důvodu častého výskytu náhody, avšak některé části aplikace jsou pevně dané a ty byly otestovány.

Pro jednotkové testy je k většině jazyků dostupný testovací framework. V programovacím jazyce jsou napsány testy, které po vykonání operace testované jednotky kódu kontroluje, zda jsou výsledky shodné s těmi předpokládanými.

### 4.1.1 Testované jednotky

Jednotkové testování aplikace Svet bylo provedeno pro ty části kódu, kde není nutný vnější zásah uživatele a kde se nevyskytuje žádná náhoda. V aplikaci Svet bylo testováno nastavení vstupních parametrů, kde předpokládané hodnoty byly porovnávány s hodnotami načtenými do proměnných z výchozího `property` souboru. Stejným způsobem bylo otestováno i načtení první sady instrukcí, včetně načtení této sady do paměti. Velikost paměti byla také testována a to vůči délce pole instrukcí.




Obrázek 4.1: Výsledky jednotkových testů

Všechny jednotkové testy byly úspěšné a jejich vyhodnocení je na obr. 4.1

## 4.2 Systémové testování

Pro ověřování funkčnosti aplikace jako celku slouží systémové testy, během kterých má vývojář za úkol ověřit systém jako celek z uživatelského hlediska. Systémové testy většinou probíhají, až když je implementace hotová. Může se ale stát, že povaha aplikace umožňuje i testovat systém průběžně. Tímto způsobem byla testována i aplikace Svet. Vždy, když do systému přibyla nová funkcionalita pokrývající nějaký případ užití, jeho funkčnost byla ihned ověřena společně s dalšími funkcionalitami aplikace. Během tohoto testování také probíhá kontrola, zda byly implementovány všechny požadavky na novou aplikaci.

### 4.2.1 Ověření implementace požadavků

Pro ověření byla vytvořena tabulka 4.2.1 na další stránce.

## 4.3 Akceptační testování

Testování ze strany uživatelů je nazýváno jako akceptační testování (UAT). Stejně jako u systémových testů byla aplikace ze strany uživatelů testována průběžně od chvíle, kdy bylo aplikaci možné začít používat. Testování začalo ve chvíli, kdy bylo možné pustit evoluci a sledovat (alespoň některé) výstupy. Uživatel se při vykonávání testů snažil vyzkoušet všechny možné scénáře použití aplikace, které byly již implementované.

Po otestování hotové aplikace z těchto testů vyplynulo, že aplikace byla implementována podle uživatelských požadavků. Přínosem pro novou aplikaci Svet bylo odhalení potíží s přenositelností, které následně byly opraveny.

| Označení | Požadavek |
|---|---|
| FP1 | Systém Svet je plně možné ovládat bez nutnosti zásahu do zdrojového kódu |
| FP2 | Je možné měnit vstupní parametry a první sadu instrukcí přes uživatelské rozhraní |
| FP3 | Všechny výstupy je možné sledovat přes uživatelské rozhraní |
| FP4 | Grafy závislostí i generační strom jsou zobrazovány v uživatelském rozhraní |
| FP5 | Textový výstup a generační strom jsou exportovány automaticky, grafy závislostí lze exportovat pomocí pravého tlačítka myši (možnost „save as -> PNG" ) |
| FP6 | První sadu instrukcí lze ukládat pod vlastním názvem, aktuálně nastavené parametry jsou načteny i po restartu aplikace |
| NP1 | GUI navrženo tak, aby jeho použití bylo co nejintuitivnější |
| NP2 | Systém lze spustit na OS Windows i Linux |
| NP3 | Funkcionalita zachována (porovnání s výstupy původní aplikace) |

Tabulka 4.1: Tabulka kontroly implementace požadavků

## 4.4 Vyhodnocení testů

Ve výsledku byla většina testů úspěšná a v případě, že testování určité funkcionality skončilo záporně, funkcionalita byla opravena a bylo provedeno další testování. Během jednotkového testování byly testovány všechny části, které těmito testy otestované být mohly a systémové testy potvrdily celkovou funkčnost aplikace Svet. Stejně tak bylo otestováno, že implementace pokryla všechny uživatelské požadavky. Nakonec aplikaci otestovali i samotní uživatelé. Pokud to bylo možné, jejich připomínky a návrhy byly také implementovány a následně znovu otestovány.

Výsledná verze aplikace Svet 2.3.14 již splňuje všechny podmínky pro to, abychom mohli implementační část považovat za hotovou a aplikaci za použitelnou při studiu digitální evoluce.

# Závěr

Cílem bakalářské práce bylo navrhnout a implementovat uživatelské rozhraní pro již existující aplikaci Svet. Bylo požadováno zjednodušit práci s aplikací a vizualizovat výstupy. Klient si v uživatelském rozhraní přál tři grafy (graf závoslosti vykonávané instrukce na čase, program counteru na čase a graf žijících procesů v daném čase), z nichž všechny byly implementované. Požadovaný byl také generační strom organismů, který by znázorňoval, jaký proces je kterého rodičem nebo potomkem. Pro splnění požadavku na ukončení nutnosti úpravy zdrojového kódu bylo také nutné implementovat možnost změny vstupních parametrů přes nové uživatelské rozhraní.

Během celého vývoje byl autor s klientem v kontaktu, a i proto bylo možné domlouvat se na dalším postupu práce. Byla implementována možnost měnit vstupní parametry a všechny výstupy. Celkově tedy uživatelské rozhraní splňuje představy klienta a obsahuje všechny požadované části.

Výhledem pro novou verzi programu by mohla být implementace funkcí pro vyhledávání ve velkém objemu dat nebo pro automatickou analýzu vývoje evoluce. Samozřejmě by se dala zlepšit celková struktura programu, která by se promítla do celkové rychlosti evoluce. Bylo by možné generační strom vybavit interaktivními prvky, které by například pouze najetím kurzoru myši na uzel generačního stromu zobrazily informace o organismu.

Příloha A

# Uživatelská příručka

## A.1 Spuštění aplikace

Pro spuštění aplikace Svet byl pro operační systém Windows vytvořen *batch* soubor `Svet2_win.bat`. Jeho ekvivalentem pro spuštění aplikace na operačním systému Linux je `Svet2_l.sh`.

Je nutné si aplikaci z přiloženého CD uložit na lokální úložiště z důvodu automatického ukládání textového výstupu a generačního stromu. Tyto výstupy se ukládají do adresáře, odkud je aplikace spuštěna. Pokud by byla spuštěna z CD, nebylo by možné výstupy uložit a aplikace se zasekne.

## A.2 Ovládání evoluce

K ovládání evoluce slouží sada tří tlačítek pro spuštění, pozastavení a vypnutí evoluce. Vždy jsou aktivní jen tlačítka, jejichž použití je v danou chvíli smysluplné.

Tlačítko „Start" je aktivní, pokud je evoluce vypnuta či pozastavena, tlačítko „Pozastavit" je aktivní pouze při běžící evoluci a konečně tlačítko „Zastavit" je možné použít v případě, že evoluce běží nebo je pozastavená.

## A.3 Změna vstupních parametrů

Formulář pro změnu parametrů vyvoláme přes „Soubor -> Změnit parametry" (Ctrl+P). V příslušných textových polích se zobrazí aktuálně nastavené hodnoty. Změníme požadované hodnoty a tlačítkem „Změnit" uložíme.

Změny se do proměnných provedou okamžitě, avšak pro zobrazení aktuálních hodnot v uživatelském rozhraní je nutné použití tlačítka „Načíst nové parametry".

## A.4 Změna první sady instrukcí

Pomocí volby „Soubor -> Změnit replikátor" (Ctrl+R) můžeme změnit první sadu instrukcí. Do textového pole vepíšeme jednotlivé instrukce oddělené čárkou a tlačítkem „Změnit první replikátor" uložíme. První sadu instrukcí je nutné volit rozumně tak, aby nedošlo k nezastavitelnému množení organismů, popřípadě k jejich okamžitému vyhynutí.

Pokud je nově nastavená sada instrukcí nevyhovující, je možné se vrátit k výchozí použitím tlačítka „Použít původní replikátor" ve formuláři pro změnu první sady instrukcí.

V případě potřeby zachování první sady instrukcí pro další použití, je možné uložit jej pomocí „Soubor -> Uložit replikátor" (Ctrl+S). Uživatel bude dotázán na jméno, pod kterým se první sada instrukcí uloží. Pod tímto jménem je pak možné sadu instrukcí najít ve formuláři vyvolaném použitím „Soubor -> Načíst replikátor" (Ctrl+L) a tlačítkem „Vybrat" zvolenou sadu instrukcí nahrát do aplikace jako aktuálně používanou.

## A.5 Změna sledovaného procesu

Sledovaný proces lze změnit zapsáním identifikačního čísla do příslušného textového pole v pravé spodní části uživatelského rozhraní aplikace Svet. Výběr je nutné potvrdit klávesou Enter.

## A.6 Export výstupů

Export výstupů do externích souborů probíhá u textového výstupu a generačního stromu automaticky. V adresáři s aplikací v podadresáři output jsou výstupy ukládány do dalších dílčích adresářů jejichž název je ve tvaru časového formátu *yyyymmdd.hhmm_ss*.

Grafy závislostí lze pak uložit kliknutím pravého tlačítka myši na graf, který má být uložen, a zvolením možnosti „Uložit jako -> PNG" je uživatel dotázán, kam má být graf ve formátu PNG vyexportován.

Příloha **B**

# Seznam instrukcí

| Kód instrukce | Název instrukce | Popis instrukce | Příznaky |
|---|---|---|---|
| 0 | nop | prázdná instrukce | |
| 1 | mov r,r | 1. argumentu přidá hodnotu 2. argumentu | |
| 2 | mov [r],r | | |
| 3 | mov r,[r] | | |
| 4 | mov [r],[r] | | |
| 5 | mov r,cislo | | |
| 6 | mov rx,cislox | | |
| 7 | mov [re],cislo | | |
| 8 | add r,r | k 1. argumentu připočte hodnotu 2. argumentu 2f | cf |
| 9 | add [r],r | | |
| 10 | add r,[r] | | |
| 11 | add [r],[r] | | |
| 12 | add r,cislo | | |
| 13 | add rx,cislox | | |
| 14 | add [re],cislo | | |
| 15 | sub r,r | od 1. argumentu odečte hodnotu 2. argumentu | cf, zf |
| 16 | sub [r],r | | |

| Kód instrukce | Název instrukce | Popis instrukce | Příznaky |
|---|---|---|---|
| 17 | sub r,[r] | | |
| 18 | sub [r],[r] | | |
| 19 | sub r,cislo | | |
| 20 | sub rx,cislox | | |
| 21 | sub [re],cislo | | |
| 22 | cmp r,r | pouze nastaví příznaky jako funkce sub | cf, zf |
| 23 | cmp [r],r | | |
| 24 | cmp r,[r] | | |
| 25 | cmp [r],[r] | | |
| 26 | cmp r,cislo | | |
| 27 | — | neplatná instrukce | |
| 28 | cmp [re],cislo | | |
| 29 | mul rx,r,r | 1. argumentu přiřadí součin hodnot 2. a 3. argumentu | |
| 30 | mul re,rx,rx | | |
| 31 | mul rx,r,cislo | | |
| 32 | mul re,rx,cislox | | |
| 33 | div r,r,r,r | 1. argumentu přiřadí celočíselný podíl 3. a 4. argumentu, 2. argumentu přiřadí zbytek po dělení 3. a 4. argumentu | dbzf |
| 34 | div rx,rx,rx,rx | | |
| 35 | swp r,r | vymění hodnoty argumentů | |
| 36 | swp [r],r | | |
| 37 | swp r,[r] | | |
| 38 | swp [r],[r] | | |
| 39 | shl r | bitový posun doleva | cf |
| 40 | shl rx | | |
| 41 | shr r | bitový posun doprava | cf |
| 42 | shr rx | | |
| 43 | and r,r | 1. argument vynásobí hodnotou 2. argumentu | zf |
| 44 | and rx,rx | | |
| 45 | and [re],r | | |
| 46 | and [re],rx | | |
| 47 | and r,cislo | | |
| 48 | and [re],cislo | | |

| Kód instrukce | Název instrukce | Popis instrukce | Příznaky |
|---|---|---|---|
| 49 | or r,r | logický součet 1. a 2. argumentu | zf |
| 50 | or rx,rx | | |
| 51 | or [re],r | | |
| 52 | or [re],rx | | |
| 53 | or r,cislo | | |
| 54 | or [re],cislo | | |
| 55 | xor r,r | logický excelentní součet 1. a 2. argumentu | zf |
| 56 | xor rx,rx | | |
| 57 | xor [re],r | | |
| 58 | xor [re],rx | | |
| 59 | xor r,cislo | | |
| 60 | xor [re],cislo | | |
| 61 | jmp re | absolutní skok na adresu v 1. argumentu | |
| 62 | jmp [re] | | |
| 63 | jmp cisloe | | |
| 64 | jmpr rsx | relativní skok na adresu v 1. argumentu | |
| 65 | jmpr [re]s | | |
| 66 | jmpr cislosx | | |
| 67 | jz rsx | podmíněný relativní skok na adresu v 1. argumentu | |
| 68 | jz [re]s | | |
| 69 | jz cislosx | | |
| 70 | jnz rsx | | |
| 71 | jnz [re]s | | |
| 72 | jnz cislosx | | |
| 73 | jc rsx | | |
| 74 | jc [re]s | | |
| 75 | jc cislosx | | |
| 76 | jnc rsx | | |
| 77 | jnc [re]s | | |
| 78 | jnc cislosx | | |
| 79 | jdbz rsx | | |
| 80 | jdbz [re]s | | |
| 81 | jdbz cislosx | | |

| Kód instrukce | Název instrukce | Popis instrukce | Příznaky |
|---|---|---|---|
| 82 | jndbz rsx | | |
| 83 | jndbz [re]s | | |
| 84 | jndbz cislosx | | |
| 82 | jndbz rsx | | |
| 83 | jndbz [re]s | | |
| 84 | jndbz cislosx | | |
| 85 | call re | absolutní odskok do podprogramu na adresu v 1. argumentu | |
| 86 | call [re] | | |
| 87 | call cisloe | | |
| 88 | callr re | relativní odskok do podprogramu na adresu v 1. argumentu | |
| 89 | callr [re] | | |
| 90 | callr cisloe | | |
| 91 | ret | návrat z podprogramu | |
| 92 | mov re,pc | 1. argumentu přiřadí hodnotu registru PC | |
| 93-127 | | neplatné instrukce | |
| 128 | mov r0,random | 1. argumentu přiřadí náhodné 8-bitové číslo | |
| 129-254 | | neplatné instrukce | |
| 255 | nový proces | | |

Příloha C

# Seznam použitých zkratek

**AWT** Abstract Window Toolkit

**DI** Dependency Injection

**FDD** Feature Driven Development

**GUI** Graphical User Interface

**IDE** Integrated Development Environment

**PNG** Portable Network Graphics

**UAT** User Acceptance Testing

**XML** Extensible Markup Language

Příloha D

# Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Martin_Plávek_2016.pdf` ..... text práce ve formátu PDF

# Literatura


[1] *Tiera: Home Page* [online]. Oklahoma: Thomas S. Ray [cit. 2015-12-29]. Dostupné z: http://life.ou.edu/tierra/

[2] *Avida by devosoft* [online]. Michigan: Devosoft, 2011, 2011-5-5 [cit. 2015-12-29]. Dostupné z: http://avida.devosoft.org/

[3] SCHOLTZ, Vladimír. *Evoluce digitálních organismů v počítači.* Praha, 2014.

[4] VLÁČILOVÁ, Lenka. *Digitální evoluce.* Praha, 2009. Diplomová práce. Vysoká Škola Chemicko Technologická. Vedoucí práce Ing. Vladimír Scholtz, Ph.D.

[5] *Graphviz | Graphviz - Graph Visualization Software* [online]. Common public license [cit. 2015-12-30]. Dostupné z: http://www.graphviz.org/

[6] Feature Driven Development. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001, 2015-04-02 [cit. 2015-12-30]. Dostupné z: https://cs.wikipedia.org/wiki/Feature_Driven_Development

[7] *Welcome to NetBeans* [online]. Oracle, 2015, 2015 [cit. 2015-12-30]. Dostupné z: https://netbeans.org/

[8] *Eclipse: The Eclipse Foundation open source community website* [online]. The Eclipse Foundation, 2015 [cit. 2015-12-30]. Dostupné z: https://eclipse.org/home/index.php

[9] *IntelliJ IDEA the Java IDE* [online]. JetBrains, 2015 [cit. 2015-12-30]. Dostupné z: https://www.jetbrains.com/idea/

[10] *Trail: Creating a GUI With JFC/Swing (The Java Tutorial)* [online]. Oracle, 1995 [cit. 2015-12-30]. Dostupné z: http://docs.oracle.com/javase/tutorial/uiswing/

[11] *Java.awt (Java Platform SE 7)* [online]. Oracle, 1993 [cit. 2015-12-30]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html

[12] *Client Technologies: Java Platform, Standard edition (Java SE) 8 Release 8* [online]. Oracle, 2015 [cit. 2015-12-30]. Dostupné z: http://docs.oracle.com/javase/8/javase-clienttechnologies.htm

[13] *Using Graphviz in Java. GraphViz - Graph Visualization Software* [online]. Common public license [cit. 2016-01-07]. Dostupné z: http://www.graphviz.org/content/using-graphviz-java

[14] *Contexts and Dependency Injection in Java EE 6* [online]. Oracle, 2011 [cit. 2016-01-01]. Dostupné z: http://www.oracle.com/technetwork/articles/java/cdi-javaee-bien-225152.html

[15] *JFreeChart* [online]. Andreas Viklund, 2005 [cit. 2015-12-30]. Dostupné z: http://www.jfree.org/jfreechart/