FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Vizualizace procesu řešení vybraných optimalizačních problémů pomocí genetických algoritmů
Student:	Radek Horáček
Vedoucí:	Ing. Mgr. Ladislava Smítková Janků, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Navrhněte interaktivní nástroj pro vizualizaci procesu řešení vybraných optimalizačních problémů pomocí genetických algoritmů.

1. Seznamte se s problematikou genetických algoritmů a s problematikou interaktivní vizualizace, vypracujte rešerši.
2. Vyberte si několik optimalizačních problémů.
3. Vypracujte analytickou studii k vyvíjenému nástroji.
4. Vyberte vhodné technologie a nástroj implementujte.
5. Aplikaci (nástroj) otestujte.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 5. prosince 2022 v Praze.

Bakalářská práce

VIZUALIZACE PROCESU ŘEŠENÍ VYBRANÝCH OPTIMALIZAČNÍCH PROBLÉMŮ POMOCÍ GENETICKÝCH ALGORITMŮ

Radek Horáček

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Mgr. Ladislava Smítková Janků, Ph.D.
11. května 2023

Obsah

Poděkování vii

Prohlášení viii

Abstrakt ix

Seznam zkratek x

Úvod 1

1 Cíl práce 3

2 Teoretická část 5
2.1 Úvod do optimalizačních metod . . . 5
2.2 Evoluční algoritmy . . . 5
2.2.1 Genetické algoritmy . . . 6
2.2.2 Genetické programování . . . 11
2.2.3 Evoluční programování . . . 11
2.3 Typické optimalizační problémy . . . 12
2.3.1 Problém obchodního cestujícího . . . 12
2.3.2 Problém batohu . . . 12
2.3.3 Problém umístění N dam na šachovnici . . . 13

3 Porovnání existujících řešení 15
3.1 Genetic Algorithm Walkers . . . 15
3.2 Genetic Algorithms Demo . . . 16
3.3 Grow Your Own Picture . . . 17
3.4 Blazor.ai . . . 18
3.5 Shrnutí analýzy existujících řešení . . . 19

4 Analýza 21
4.1 Popis požadované aplikace . . . 21
4.2 Model případů užití . . . 21
4.2.1 Funkční požadavky . . . 21
4.2.2 Nefunkční požadavky . . . 22
4.2.3 Případy užití . . . 23
4.3 Analýza vhodných technologií a knihoven . . . 28
4.3.1 Technologie . . . 29
4.3.2 Knihovny . . . 31
4.3.3 Shrnutí analýzy technologií a knihoven . . . 32

5 Návrh 35
5.1 Doménový model a architektura aplikace . . . 35
5.2 Uživatelské prostředí . . . 37

6 Implementace **41**
6.1 Použité knihovny . . . 41
6.1.1 MudBlazor . . . 41
6.1.2 GeneticSharp . . . 42
6.1.3 AKSoftware.Localization.MultiLanguages . . . 42
6.1.4 Blazor.Extensions.Canvas . . . 42
6.1.5 Plotly.Blazor . . . 42
6.1.6 SixLabors.ImageSharp . . . 42
6.1.7 Xunit . . . 43
6.2 Komprese dat aplikace . . . 43
6.3 Implementace optimalizačních problémů . . . 43
6.3.1 Problém obchodního cestujícího . . . 43
6.3.2 Aproximace 2D obrazu . . . 45
6.3.3 Problém batohu . . . 47
6.3.4 Problém umístění N dam na šachovnici . . . 49
6.4 Implementace dalších částí aplikace . . . 51
6.4.1 Úvodní stránka . . . 51
6.4.2 Stránka s teorií genetických algoritmů . . . 51
6.5 Optimalizace na mobilní zařízení . . . 52

7 Testování **53**
7.1 Unit testy . . . 53
7.2 Uživatelské testování . . . 53
7.2.1 Testovací scénář . . . 53
7.2.2 Výsledky uživatelských testů . . . 54
7.2.3 Shrnutí odpovědí respondentů a provedené změny . . . 56

8 Dokumentace **59**
8.1 Doxygen . . . 59

Závěr **61**

Obsah přiloženého archivu **67**

Seznam obrázků

2.1 Diagram aktivity - fungování genetického algoritmu 7
2.2 Princip ruletové selekce [6] 8
2.3 Křížení na principu aritmetického průměru [7] 9
2.4 Jednobodové křížení [7] 9
2.5 Uniformní křížení [7] 10
2.6 Bit-flip mutace [7] 10
2.7 Mutace využívající princip inverze [7] 11
2.8 Problém obchodního cestujícího [10] 12
2.9 Problém batohu [12] 13
2.10 Příklad korektního řešení umístnění čtyř dam na šachovnici [13] 13

3.1 Snímek obrazovky z aplikace Genetic Algorithm Walkers [14] 15
3.2 Snímek obrazovky z aplikace Genetic Algorithms Demo [15] 16
3.3 Snímek obrazovky z aplikace Grow Your Own Picture [16] 17
3.4 Snímek obrazovky z aplikace Blazor.ai [17] 18

4.1 Matice odpovědností 27
4.2 Diagram s případy užití 28

5.1 Snímek obrazovky s doménovým modelem aplikace 36
5.2 Wireframe s návrhem úvodní obrazovky 37
5.3 Wireframe s návrhem stránky optimalizačního problému s postranními panely . . 38
5.4 Wireframe s návrhem stránky optimalizačního problému bez postranních panelů 38
5.5 Wireframy s návrhem rozložení stránky na mobilním telefonu 39
5.6 Wireframe s návrhem stránky optimalizačního problému bez postranních panelů 39

6.1 Snímek obrazovky zobrazující MudBlazor Avatar komponenty [47] 41
6.2 Snímek obrazovky s problémem obchodního cestujícího 44
6.3 Snímek obrazovky s problémem aproximace 2D obrázku 45
6.4 Snímek obrazovky s problémem batohu 47
6.5 Snímek obrazovky s problémem umístění N dam na šachovnici 49
6.6 Snímky obrazovky aplikace s rozvržením pro mobilní telefony 52

Seznam tabulek

3.1 Tabulka se srovnáním existujících řešení 19

4.1 Tabulka se srovnáním potenciálně vhodných technologií 33

Seznam výpisů kódu

6.1 Ukázka práce s komponenty v knihovně MudBlazor [47] 41
6.2 Inicializace knihovny AKSoftware.Localization.MultiLanguages 42
6.3 Výpočet fitness u problému obchodního cestujícího 44
6.4 Výpočet fitness u problému aproximace 2D obrazu 46
6.5 Výpočet fitness u problému batohu 48
6.6 Výpočet fitness u problému umístění N dam na šachovnici 50
8.1 Příklad komentáře ve zdrojovém kódu pro Doxygen dokumentaci 59

Tímto bych rád poděkoval vedoucí mé bakalářské práce, Ing. Mgr. Ladislavě Smítkové Janků, Ph.D., za odborné konzultace a čas, který mi věnovala při tvorbě této práce. Dále bych rád poděkoval mé rodině, přátelům i ostatním, kteří mě po celou dobu podporovali a byli mi oporou.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při 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, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 11. května 2023 .....

Abstrakt

Tato práce se zabývá tvorbou interaktivního nástroje pro vizualizaci řešení optimalizačních úloh pomocí genetických algoritmů. Teoretická část obsahuje popis problematiky evolučních algoritmů a zaměřuje se především na popis genetických algoritmů. Součástí teoretické části je i výčet typických optimalizačních úloh řešených pomocí genetických algoritmů. Praktická část se zabývá popisem všech důležitých částí vývoje softwarového díla. Práce obsahuje výčet existujících řešení i analýzu vhodných technologií a knihoven pro tvorbu daného interaktivního nástroje. Součástí textu je také analytická dokumentace vyvíjené aplikace. V dalších kapitolách je popsána implementace aplikace, ke které byl využit framework Blazor fungující na technologii WebAssembly. Nástroj umožňuje vizualizaci tří typických optimalizačních úloh a kromě toho také generování 2D bitmapové grafiky. V závěru práce jsou popsány metody testování a dokumentace, které byly v projektu využity. Výsledkem práce je funkční webová interaktivní aplikace.

Klíčová slova genetické algoritmy, blazor, webová aplikace, webassembly, optimalizační úlohy, vizualizace, interaktivní nástroj, generování 2D obrazu

Abstract

This thesis describes the process of developing an interactive tool for visualization of the process of solving selected optimization problems using genetic algorithms. The theoretical part provides an introduction to the evolutionary algorithms, more specifically then focuses on genetic algorithms. This part also lists typical optimization problems that are commonly solved using genetic algorithms. The practical part describes all the important steps in creating new functional software. This includes the comparison of existing solutions, analysis of possibly suitable technologies and libraries, and software documentation. The implementation of the application, using the Blazor WebAssembly framework, is also described in this part. The interactive tool provides the visualizations of three typical optimization problems and the problem of 2D bitmap image generation. At the end, the thesis lists the techniques used for testing and documentation of the application. The final result of this thesis is a functional interactive web application.

Keywords genetic algorithms, blazor, web application, webassembly, optimization problems, visualization, interactive tool, generating 2D image

Seznam zkratek

GA	Genetický algoritmus
GP	Genetické programování
SPA	Single-Page Application
TSP	Problém obchodního cestujícího
UI	User Interface
WASM	WebAssembly

Úvod

Genetické algoritmy patří do oblasti evolučních algoritmů a mají široké využití pro řešení praktických problémů. Pochopení jejich fungování ale nemusí být pro zájemce z široké veřejnosti nebo případné studenty této problematiky srozumitelné. Optimalizace problémů pomocí genetických algoritmů je díky jejich iterativní povaze vhodná ke grafické vizualizaci, která může výrazně usnadnit pochopení jejich fungování. Mou motivací je tedy tvorba takové aplikace, která bude na jediném místě obsahovat vizualizaci vhodných optimalizačních úloh i stručný popis teorie genetických algoritmů. Zájemci si budou v aplikaci také moci měnit základní parametry genetického algoritmu, což názornou formou ještě více umožní porozumět jeho chování. Aplikace nabídne možnost rozšířit znalosti laické veřejnosti, ale může být přínosná i pro účely výuky a akademické pracovníky.

V teoretické části shrnu problematiku genetických algoritmů, uvedu je do kontextu ostatních metod spadajících do oblasti evolučních algoritmů a popíšu základní důležité parametry, které ovlivňují jejich fungování. Součástí této rešerše bude i výčet typických optimalizačních problémů, které jsou vhodné k řešení pomocí genetických algoritmů a popis jejich využití v praxi.

V praktické části popíšu kompletní proces vývoje aplikace pomocí frameworku Blazor, využívajícího technologii WebAssembly pro tvorbu webových aplikací. Popíšu zde všechny důležité součásti vývoje softwarového díla. Provedu analýzu problému a případných existujících řešení, srovnám technologie a knihovny, které je možné k implementaci nástroje využít. Na základě tohoto srovnání vyberu vhodnou kombinaci technologie a knihovny pro účely tohoto projektu. Poté se budu zabývat návrhem architektury a uživatelského rozhraní aplikace. V další části popíšu samotný proces a způsob implementace interaktivního nástroje. V závěru práce rozeberu stručně metody, které byly využity k testování a zdokumentování projektu.

Kapitola 1
Cíl práce

Cílem mé práce je tvorba interaktivního nástroje pro vizualizaci řešení optimalizačních problémů pomocí genetických algoritmů.

V teoretické části je mým cílem popsat problematiku genetických algoritmů, jejich fungování a možné praktické využití i výčet několika typických optimalizačních problémů, které pomocí nich lze řešit.

Hlavní úlohou v praktické části práce je implementace samotné interaktivní aplikace, která bude jednoduchým způsobem vizualizovat řešení vybraných optimalizačních problémů. Nástroj bude snadno dostupný a uživatelsky přívětivý, aby umožňoval základní pochopení problematiky odborné, ale i široké veřejnosti. V této praktické části je cílem korektně provést všechny důležité součásti procesu vývoje softwarového díla. Tato část bude obsahovat také popis tvorby programátorské dokumentace a metod testování, které byly během vývoje využity.

Finálním výstupem práce by tak měl být odborný popis teorie dané problematiky a samotný funkční interaktivní nástroj s popisem celé jeho tvorby.

Kapitola 2
Teoretická část

V této kapitole se budu zabývat popisem teorie genetických algoritmů (GA). Rozeberu princip fungování GA a jejich iterativní chování. Popíšu zde také další algoritmy, které spadají do oblasti evolučních algoritmů a stručně shrnu jejich rozdíly. U všech zmíněných témat vhodně zavedu používané pojmy a názvosloví. Součástí této kapitoly bude také rešerše typických optimalizačních úloh, které jsou vhodné k řešení pomocí genetických algoritmů. Zvolené úlohy poté implementuji v praktické části bakalářské práce během vývoje interaktivní aplikace.

2.1 Úvod do optimalizačních metod

Problematika optimalizačních metod je velmi široká oblast. [1, s. 37-40] tvrdí, že seznam metod klasifikovaných jako optimalizační algoritmy se dle různých autorů liší, stejně tak se různí dělení optimalizačních metod do dalších podkategorií. Více zdrojů se ale shoduje, že do kategorie optimalizačních metod patří například Newtonovy metody, dynamické programování, simulované žíhání, hill-climbing algoritmus nebo tabu prohledávání. Mezi optimalizační metody patří i takzvané *evoluční algoritmy*, kam řadíme například genetické programovaní, evoluční programování nebo také právě i *genetické algoritmy* [1, s. 37-40], [2, s. 12364].

2.2 Evoluční algoritmy

V této podkapitole detailně popíšu fungování genetických algoritmů a operátory, které se u těchto algoritmů používají. Zadefinuji základní pojmy, které jsou společné i pro další evoluční algoritmy a i ty stručně popíšu.

Definice klíčových pojmů:

- **Fenotyp** je konkrétní jedinec (individuum) řešící původní problém [3, s. 29].
- **Genotyp** je reprezentací fenotypu pomocí vhodného kódování, evoluční algoritmy pracují při svém fungování právě s genotypy. Je důležité, aby bylo kódování fenotypu invertibilní, tedy aby pro každý genotyp existoval nanejvýš jeden fenotyp, který je daným genotypem reprezentován. Synonymem slova genotyp v kontextu evolučních algoritmů, který se v literatuře často objevuje, je také výraz *chromozom*. [3, s. 29-30]
- **Gen** je základní stavební jednotkou, ze kterých se skládá chromozom [3, s. 29]. Uspořádání genů v chromozomu se liší dle konkrétního typu evolučního algoritmu.
- **Populace** je množina genotypů reprezentujících řešení problému, v množině se stejný genotyp může vícekrát opakovat [3, s. 30].

- **Generace** vyjadřuje, kolikátá iterace běhu evolučního algoritmu proběhla. V každé generaci vyhodnocujeme aktuální populaci jedinců.
- **Fitness funkce** vyjadřuje míru kvality či vhodnosti konkrétního jedince. Fitness funkce by měla nabývat pouze nezáporných hodnot, je také měřítkem reprodukční způsobilosti daného jedince. [4, s. 14, 19] Fitness funkci se snažíme u jedinců maximalizovat, správná volba fitness funkce může výrazně ovlivnit kvalitu řešení, které evolučním algoritmem získáme.
- **Křížení** je jedním z kroků v procesu reprodukce, tedy tvorby potomků vybraných rodičovských jedinců. Existuje více různých způsobů křížení, jejich společným znakem je vzájemná výměna částí rodičovských chromozomů. Většina metod křížení využívá dva rodičovské chromozomy, existují však i výjimky. Konkrétní implementace se různí podle typu evolučního algoritmu, například evoluční programování pak křížení nevyužívá vůbec. [3, s. 32]
- **Selekce** je proces výběru jedinců, kteří budou zvoleni jako rodiče pro tvorbu potomstva další generace. [5, s. 9]
- **Mutace** je unární operace, která se s určitou pravděpodobností aplikuje na vytvořené chromozomy. Mutace by měla být náhodná a její pravděpodobnost je řádově nižší, než pravděpodobnost křížení jedince. [3, s. 31-32], [4, s. 24]

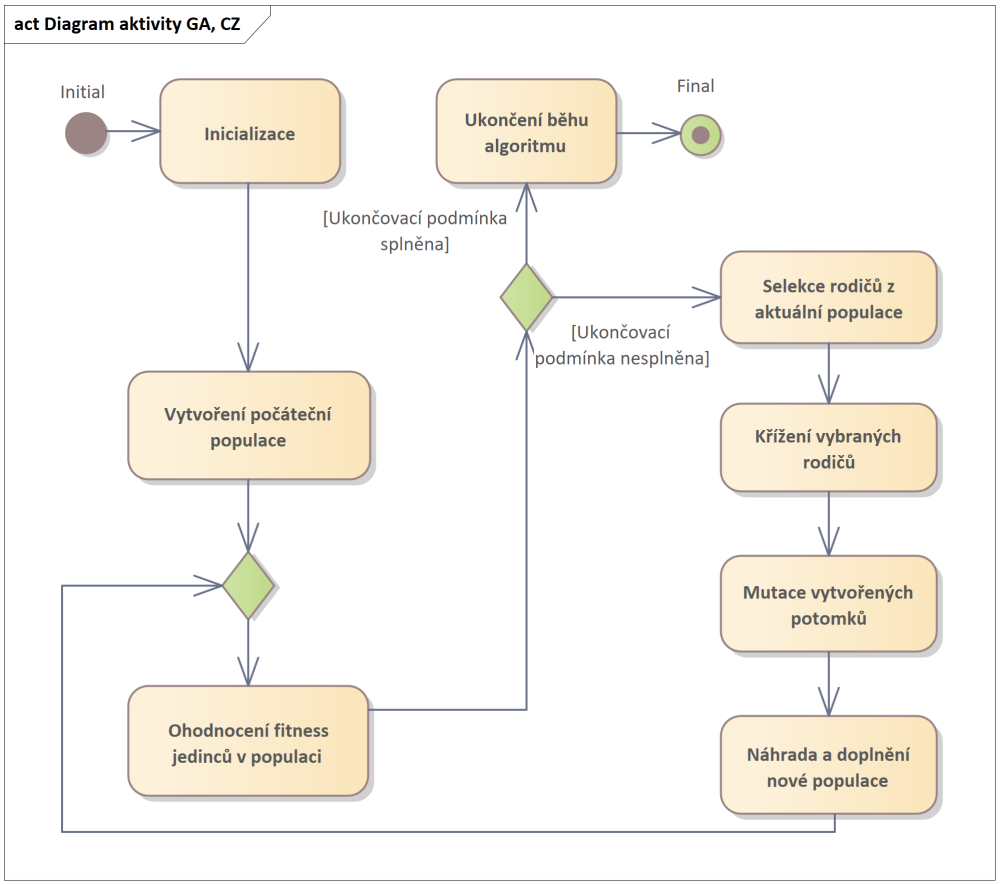
Evoluční algoritmy mají společný rys v tom, že jsou založeny na práci s populací jedinců. Jedinci jsou vždy ohodnoceni pomocí vhodné fitness funkce. Evoluční algoritmy iterativně vytváří nové populace a nahrazují jimi předchozí populace. [1] Konkrétní evoluční algoritmy sdílí i mnoho jiných podobností, často se ale liší v jejich implementaci. Vzhledem k tomu, že cílem této práce je vyvinout aplikaci zaměřující se na vizualizaci procesu řešení úloh pomocí genetických algoritmů, se budu v této kapitole nadále věnovat především jejich problematice. Některá tvrzení v následující podkapitole 2.2.1 je ale možné zobecnit i na ostatní zástupce z oblasti evolučních algoritmů.

2.2.1 Genetické algoritmy

Genetické algoritmy vycházejí z myšlenky Darwinova principu evoluce. Hledání optimálního řešení probíhá formou soutěže. Jednotlivé geny jsou lineárně uspořádány, takže i-tý gen ve dvou různých chromozomech nějakého problému vždy reprezentuje stejnou charakteristiku jedince. Chromozomy genetických algoritmů jsou reprezentovány pomocí vektorů hodnot. Historicky nejstarším způsobem kódování individuí je *binární* kódování. V takovém případě mohou geny chromozomů nabývat pouze hodnot 1 nebo 0. Pro některé úlohy (například pro problém obchodního cestujícího) se však binární kódování nehodí. Chromozomy jsou tedy často reprezentovány i jinými způsoby, například celými čísly. [4, s. 20]

Na počátku se vygeneruje původní populace jedinců. Jedinci v této počáteční generaci jsou nejčastěji vytvářeni náhodně, je možné ale využít i informované generování počáteční generace, v tomto případě však hrozí při chybném návrhu heuristiky nevratné umístění populace do lokálního optima [5, s. 14]. Dalším krokem je vyhodnocení fitness funkce jednotlivých jedinců v této generaci a kontrola případného splnění ukončovací podmínky. Když je ukončovací podmínka splněna, dojde k zastavení běhu genetického algoritmu. Pokud ukončovací podmínka splněna není, provede algoritmus selekci rodičů z aktuální populace. Selekce k výběru zpravidla využívá ohodnocující fitness funkci jedinců. Vybraní jedinci (rodiče) jsou pak křížení pomocí operátoru křížení. Noví potomci jsou s určitou pravděpodobností náhodně zmutováni. Posledním krokem je náhrada předchozí populace novými potomky s případným doplněním jedinci z předchozí generace (způsob náhrady populace závisí na konkrétní implementaci). [4, s. 19-25]

Pro lepší názornost iterativního fungování genetického algoritmu jsem vytvořil diagram aktivity, který tento proces zobrazuje:



Obrázek 2.1 Diagram aktivity - fungování genetického algoritmu

V následující části textu podrobněji popíšu některé důležité parametry a operátory, které významně ovlivňují chování genetického algoritmu:

2.2.1.1 Operátor selekce

Selekce je důležitým krokem během fungování genetického algoritmu. Během procesu selekce vybíráme, které jedince z původní generace zvolit jako rodiče ke křížení a tvorbě potomků. U operátoru selekce lze vidět paralelu s evoluční teorií Charlese Darwina, podle níž přežívají pouze nejlepší jedinci. [1, s. 175] Během selekce se tedy pokoušíme upřednostňovat ke křížení slibné jedince, vhodnost jedinců zpravidla určujeme podle jejich ohodnocení fitness funkcí. Existuje více různých metod selekce, které se svým fungováním liší. Nyní popíšu několik příkladů typických způsobů, jak selekci implementovat.

2.2.1.1.1 Ruletová selekce

Ruletová selekce se řadí mezi nejrozšířenější formy selekce. Název vychází z analogie s klasickou ruletou, která má na každém z polí vypsané jedno z čísel. V případě pomyslné rulety u genetického algoritmu je v každém poli potenciální rodič. U klasické rulety mají všechna pole stejnou pravděpodobnost výběru. U genetického algoritmu mají vetší pravděpodobnost být vybráni jedinci s vyšší hodnotou fitness. Každý jedinec v generaci má tedy přidělenou výseč, která přímo odpovídá ohodnoceni daného individua.

Nechť má populace velikost

$$N > 0$$

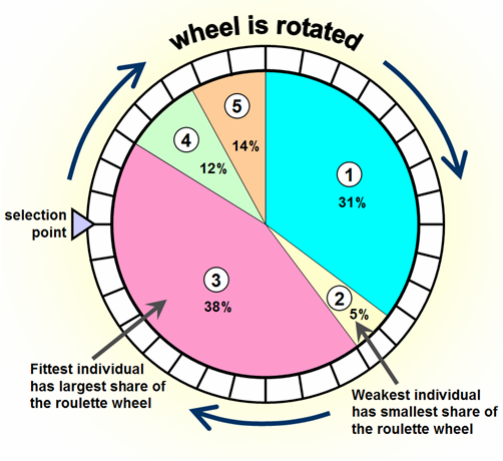
jedinců a ti jsou ohodnoceni fitness funkcí

$$f_i \geq 0 \ (i = 1, ..., N)$$

Pravděpodobnost, se kterou bude jedinec vybrán, je potom definovaná vztahem:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \ i \in \{1, ..., N\}$$

Stejně jako v přírodě, kde se může prosadit i slabší jedinec, i zde může být vybrán jako rodič jedinec s nízkým ohodnocením fitness funkcí, pravděpodobnost tohoto výběru je však nižší, než u jedinců s vysokým fitness. [4, s. 21, 22]



■ **Obrázek 2.2** Princip ruletové selekce [6]

2.2.1.1.2 Turnajová selekce

Turnajová selekce funguje tak, že vždy vybereme definovaný počet k jedinců z dané generace a poté zvolíme jedince, který má v tomto výběru nejvyšší fitness. Výhodou této selekce je její nezávislost na konkretních hodnotách fitness funkce vybraných jedinců. [5, s. 15]

2.2.1.1.3 Pořadová selekce

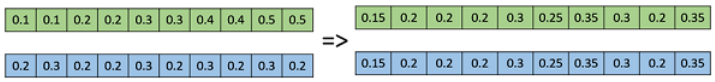
Pořadová selekce je vhodná, pokud jsou mezi fitness ohodnoceními jedinců v dané populaci velké rozdíly (hodnoty mají velký rozptyl). Hodnoty fitness se v takovém případě mezi jedinci hodně liší a například ruletová selekce nemusí fungovat vhodně. Pořadová selekce ohodnotí jedince s nejnižší fitness hodnotou číslem 1, druhému nejhoršímu přiřadí hodnotu 2 - tímto způsobem vzestupně ohodnotí všechny jedince. Toto hodnocení se pak použije místo původních fitness funkcí během ruletové selekce. [1, s. 175] Stejně jako turnajová selekce tedy není závislá na konkrétních hodnotách fitness.

2.2.1.1.4 Elitismus

Selekce využívající elitismus na začátku vybere nejlepšího jedince, kterého okamžitě umístí do nové populace. Poté se teprve provede selekce na ostatních jedincích. V druhém kroku se často využívá například právě ruletová nebo turnajová selekce. Využití elitismu brání ztrátě nejlepšího řešení a zlepšuje tak průběh genetického algoritmu. [1, s. 176]

2.2.1.2 Operátor křížení

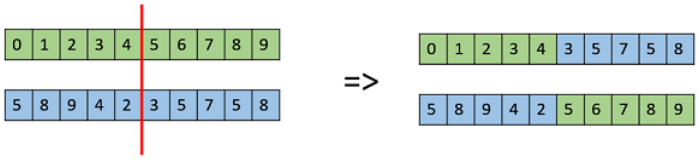
Křížení je první operací procesu reprodukce. Zpravidla se pro křížení vybírají dva rodiče. Rodičem se jedinec může stát vícekrát. U křížení předpokládáme díky využití slibných rodičů vznik vhodného potomka, který povede k nalezení dobrého řešení problému. [1, s. 176, 178] U operace křížení specifikujeme parametr, který určuje pravděpodobnost, že ke křížení dojde. Většinou se volí poměrně vysoká pravděpodobnost křížení (obvykle 75% až 95%). Pokud ke křížení v daném případě zrovna nedojde, stanou se potomky rodičů jejich přímé (nezměněné) kopie. [4, s. 24] Za zmínku stojí také fakt, že různé metody křížení jsou vhodné pro řešení různých problémů a že vhodnost konkrétního křížení souvisí také se způsobem reprezentace fenotypu. Chromozom se může skládat například z binárních hodnot, celých čísel nebo reálných čísel. Například u reálných čísel můžeme využít třeba křížení s využitím aritmetického průměru genů jednotlivých rodičů. [3, s. 49-78] Způsobů křížení existuje opravdu velké množství, v následující části tedy popíšu podrobněji fungování pouze několika typických způsobů křížení.



Obrázek 2.3 Křížení na principu aritmetického průměru [7]

2.2.1.2.1 Jednobodové křížení

U tohoto typu křížení se náhodně zvolí číslo z množiny $\{1, ..., l-1\}$, kde l je délka chromozomů. Geny na indexech před tímto zvoleným číslem se získají z prvního rodiče, geny na indexech větších než zvolený bod se vyjmou z druhého rodiče. [4, s. 23, 24]



Obrázek 2.4 Jednobodové křížení [7]

2.2.1.2.2 Dvoubodové křížení

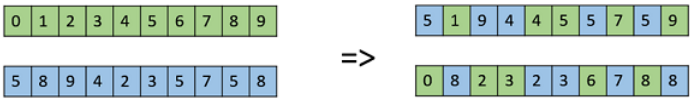
Způsob tohoto křížení se velmi podobá principu jednobodového křížení. Jediným rozdílem je volba dvou čísel, které budou určovat indexy genů chromozomů, kde dojde ke křížení rodičů místo jednoho. První část chromozomu potomka má tak geny prvního rodiče, druhá část obsahuje geny druhého rodiče a třetí část opět geny prvního rodiče. [1, s. 177]

2.2.1.2.3 n-Bodové křížení

Jedná se o zobecněnou verzi jednobodového a dvoubodového křížení. Počet pozic, na kterých rodičovské parametry zkřížíme, je určen parametrem n. [3, s. 53] Například pro dvoubodové křížení bude platit $n = 2$.

2.2.1.2.4 Uniformní křížení

Uniformní křížení nerozděluje chromozomy rodičů na části. Místo toho pracuje s každým genem v chromozomu samostatně a náhodně u každého indexu chromozomu vybírá, jestli se použije gen z prvního nebo druhého rodiče. Většinou je pravděpodobnost použití genu prvního/druhého rodiče stejná (tedy 0.5), ale je teoreticky možné tento parametr i upravit. K rozhodnutí, který z rodičů poskytne konkrétní gen, lze využít generování náhodného čísla v intervalu $(0, 1)$ a zkoumání, zda-li je toto číslo větší, nebo menší než specifikovaný parametr p. [3, s. 53]



Obrázek 2.5 Uniformní křížení [7]

2.2.1.2.5 Křížení se zachováním pořadí

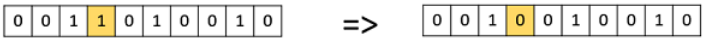
Tento způsob křížení je vhodný například u úloh, kde jsou chromozomy tvořeny na principu permutace. Tato reprezentace se používá například u řešení *problému obchodního cestujícího* (TSP). Důležitým požadavkem je zachování vlastnosti permutace. Vyžadujeme, aby sekvence čísel v chromozomu potomka stále obsahovala každé číslo právě jednou. Křížení se zachováním pořadí vytvoří dva body křížení na rodičovských chromozomech (podobně jako dvoubodové křížení). Do oblasti mezi tyto zvolené indexy se vloží geny prvního rodiče. Od druhého bodu křížení pak vkládáme zatím *nepoužité* geny druhého rodiče, přejdeme během toho přes poslední gen potomka a pokračujeme od začátku po první bod křížení. [3, s. 73] Mezi další způsoby křížení pro zachování vlastnosti permutace chromozomu patří například *operátor křížení s částečným přiřazením* (PMX), *cyklický operátor křížení* (CX) a *operátor křížení s rekombinací hran* (ERX). [4, s. 116]

2.2.1.3 Operátor mutace

Mutace je zpravidla unární operace, která náhodným způsobem mění strukturu daného chromozomu [3, s. 31, 32]. Na rozdíl od operátoru křížení dochází k mutaci s řádově mnohem menší pravděpodobností (většinou 0,5-1%). Mutace slouží jako nástroj proti uváznutí řešení v lokálním maximu. Díky své náhodnosti se mohou řešení přesunout z oblasti lokálního optima pryč. Mutace by neměla být příliš intenzivní, jinak by mohl mít GA rysy náhodného prohledávání. [1, s. 178] Opět je důležité zmínit, že stejně jako u operátoru křížení jsou pro konkrétní optimalizační problémy a reprezentace chromozomů vhodné různé způsoby mutace.

2.2.1.3.1 Flip-bit mutace

Tento typ mutace je použitelný pro binární chromozomy. U každého genu se s malou pravděpodobností změní hodnota v genu na opačnou binární hodnotu. [3, s. 52] Konkrétní implementace se ale v různých zdrojích liší, například [8, s. 4] uvádí, že pravděpodobnost určuje, zda-li je k mutaci vybrán konkrétní chromozom a v něm je pak vždy mutován jeden náhodný gen.



Obrázek 2.6 Bit-flip mutace [7]

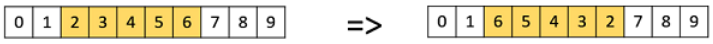
2.2.1.3.2 Uniformní mutace

Jedná se o mutaci, která je svým chování analogická k bit-flip mutaci, používá se ale pro chromozomy, které se skládají z reálných čísel. Každý gen je opět individuálně mutován s malou pravděpodobností. V případě celých čísel by byl postup mutace podobný, jen bychom vybírali v případě mutace náhodné celé číslo z daného intervalu. [3, s. 55, 57]

2.2.1.3.3 Inverze

Tato mutace se často využívá pro problémy, kde je nutné zachovat u chromozomů vlastnost permutace a kde záleží na pořadí hodnot v chromozomu. Kvůli tomu není možné uvažovat mutaci pro každý gen chromozomu zvlášť. Podstatou je tedy změna pořadí genů v rámci chromozomu, hodnoty genů však neměníme. Operátor inverze zvolí 2 body (indexy) v chromozomu a mezi těmito body provede inverzi uspořádání hodnot genů. Geny mimo tuto část jsou zachovány.

Mezi další mutace, které jsou vhodné pro chromozomy s vlastností permutace se řadí například *mutace prohozením* nebo *mutace promícháním*. [3, s. 67-70]



Obrázek 2.7 Mutace využívající princip inverze [7]

2.2.1.4 Náhrada a doplnění nové populace

Po vytvoření potomků křížením vybraných rodičů, a jejich případné mutaci, je nutné určit, kteří jedinci postoupí do nové generace, která je zpravidla omezena svou velikostí. Metod výběru přeživších jedinců existuje více. Hlavním rozdílem mezi metodami náhrady populace je výběr podle věku jedince, nebo výběr založený na ohodnocení fitness funkcí jedince. Některé metody mohou také využívat principu elitismu - pokud by nejlepší potomek z předchozí generace nebyl vybrán do další generace a v nové generaci nemá žádný jedinec stejné nebo vyšší fitness ohodnocení, bude přidán tento nejlepší jedinec na úkor jednoho z ostatních potomků. [3, s. 87-90]

2.2.1.5 Ukončovací podmínka

Ukončovací podmínka určuje, kdy má dojít k přerušení iterativního cyklování genetického algoritmu. Bez ukončovací podmínky by genetický algoritmus nikdy neskončil. Jako ukončovací podmínka je často voleno dosažení maximálního limitu počtu generací, stagnace vývoje fitness funkce jedinců v posledních generacích nebo dosažení limitu výpočetního času. [3, s. 34]

2.2.2 Genetické programování

Genetické programování (GP) se také řadí do kategorie evolučních algoritmů. Jedná se o rozšíření genetických algoritmů představené autorem Johnem Kozou. Populace u genetického programování neobsahuje čísla, ale samotné symbolické objekty. Vzhledem k faktu, že se jedná o rozšíření GA, je názvosloví stejné - například každý parametr jedince označujeme také jako *gen*. [1, s. 254-256] Genotyp v případě GP zobrazujeme jako syntaktický strom. Vnitřní listy stromové struktury (neterminály) reprezentují funkce. Počet hran vycházejících z vrcholu funkce určuje její aritu. Listy stromu (terminály) reprezentují konstanty a proměnné. Samozřejmostí je nutnost jiných implementací křížení a mutace, než u genetických algoritmů. Ke křížení se využívá například výměna podstromů rodičů a mezi používané mutace se řadí například *uzlová mutace*, která nahrazuje neterminál jiným neterminálem se stejným počet argumentů nebo terminál jiným terminálem. Genetické programování lze využít například pro symbolickou regresi (hledání funkce v symbolickém tvaru matematického výrazu). [4, s. 123-131, 145]

2.2.3 Evoluční programování

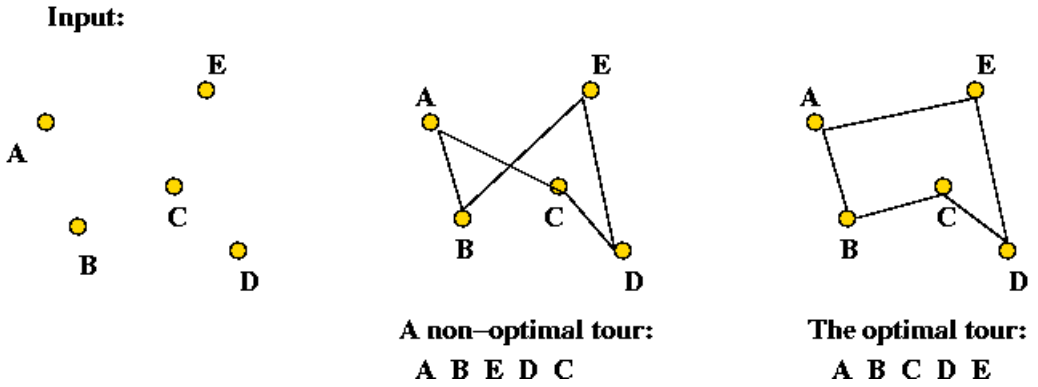
Evoluční programování využívá pro reprezentaci jedinců konečné stavové automaty. Dalším specifikem této metody je fakt, že se zde nevyužívá operátor křížení. Evoluční programování bylo navrženo Lawrencem Foglem s původním cílem simulovat evoluci a vytvořit umělou inteligenci. Dnes evoluční programování často využívá reprezentaci pomocí reálných čísel a téměř splývá s principem fungování *evolučních strategií*, které také patří mezi metody evolučních algoritmů. Rozdíly mezi evolučním programováním a evolučními strategiemi jsou hlavně v použití operátoru křížení a také v procesu selekce. [3, s. 103]

2.3 Typické optimalizační problémy

V této podkapitole teoretické části se budu zabývat typickými optimalizačními problémy, k jejichž řešení se často využívá fungování genetických algoritmů. Všechny zde popsané klasické problémy bych rád implementoval ve své aplikaci v praktické části této práce.

2.3.1 Problém obchodního cestujícího

Cílem obchodního cestujícího v této úloze je navštívit množinu všech zadaných měst, přičemž každé město (kromě výchozího) musí být navštíveno právě jednou. Cesta obchodního cestujícího začíná i končí ve výchozím (prvním) městě, a proto ho obchodní cestující jako jediné navštíví dvakrát. Výsledná cesta tak tvoří cyklus. V úloze obchodního cestujícího (TSP) hledáme takové uspořádání (permutaci) měst, aby byla celková cesta, kterou obchodní cestující urazí co nejkratší. Tato úloha se řadí mezi *NP-těžké* problémy. Pro nalezení nejlepšího řešení je nutné pro n měst vyzkoušet všech $n!$ permutací. Se zvětšujícím se parametrem n tak výpočetní náročnost extrémně narůstá. Pro přibližné řešení problému existují tradiční algoritmy, které se úlohu snaží řešit. Negarantují však nalezení nejlepšího řešení. Mezi takové tradiční algoritmy pro řešení TSP se řadí například *hladový algoritmus* nebo algoritmy založené na lokálním prohledávání (například *2-opt algoritmus*, který začíná vygenerováním náhodné cesty a tu se poté snaží vylepšovat vzájemnou výměnou dvou nepřímo navazujících hran cesty). [4, s. 111-113] V praxi lze problém obchodního cestujícího uplatnit například při hledání nevhodnější cesty pro vozidla rozvážkových služeb, vyzvedávání balíčků ve skladech nebo pro návrh tištěných spojů. [9, s. 2-5]

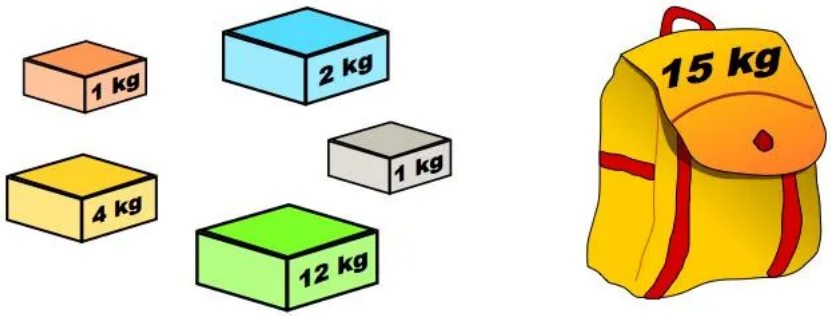


Obrázek 2.8 Problém obchodního cestujícího [10]

2.3.2 Problém batohu

U problému batohu uvažujeme situaci, kdy máme množinu n předmětů, které mají určené své váhy c_i a hodnoty v_i. K dispozici máme batoh s určitou nosností C_{max}. Cílem je nalézt takovou podmnožinu předmětů, u které součet hodnot zvolených předmětů bude co nejvyšší a součet vah nebude větší, než nosnost batohu. K reprezentaci problému se nabízí využít binární kódování, kde hodnota 1 znamená, že daný předmět umístíme do batohu a hodnota 0 znamená, že předmět do batohu nevložíme. Každý předmět je tak reprezentován jedním genem chromozomu na určitém indexu. [3, s. 39, 40]. Problém batohu lze řešit například otestováním všech 2^n kombinací umístění předmětů, pro velké hodnoty n je ale tento přístup výpočetně neúnosný.

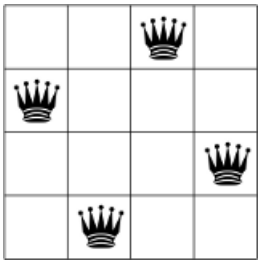
Problém batohu má více různých variací, všechny z nich jsou klasifikované jako *NP-těžké* problémy. V praxi můžeme varianty problému batohu využít například v oblasti nákladní dopravy a její organizace, optimalizace využití hotelů a dalších služeb nebo například nebo u správy produkce a inventáře firem. [11, s. 1, 2]



Obrázek 2.9 Problém batohu [12]

2.3.3 Problém umístění N dam na šachovnici

Při řešení problému umístění N dam na šachovnici je naším cílem, jak již z názvu vyplývá, umístit právě N figur šachové dámy na šachovnici o velikosti $N \times N$. Podmínkou správného řešení je ale požadavek, aby se žádné dvě z dam neohrožovaly (tedy nemohly jedna druhou v pomyslném aktuálním tahu sebrat). Řešení tohoto problému existuje pro každé přirozené číslo $N \geq 4$. V případě šachovnice o velikosti $N = 1$ pak existuje jediné triviální řešení. Například pro klasickou šachovnici o velikosti 8×8 polí je možné nalézt právě 96 různých řešení, která splňují zadanou podmínku. Úloha umístění N dam na šachovnici je klasickým kombinatorickým problémem z oblasti splňování podmínek. Mezi tradiční způsoby řešení tohoto problému se řadí například kombinatorický algoritmus využívající backtracking. Ten se postupně pokouší umístit jednotlivé dámy na políčka, aby nedošlo ke kolizi. Pokud k ní dojde, vrátí se algoritmus o krok zpět a změní pozici dámy umístěné v daném kroku. Toto řešení sice najde vhodné řešení (bez kolizí figur), ale vzhledem ke kombinatorickému nárůstu výpočtů je nevhodný pro velké hodnoty N. Dále existují různé stochastické přístupy k řešení problému. Ty většinou generují náhodné rozestavění figur a poté se pokouší provádět výměny dvojic dam, aby se snížil počet vzájemných kolizí. [4, s. 91-93].



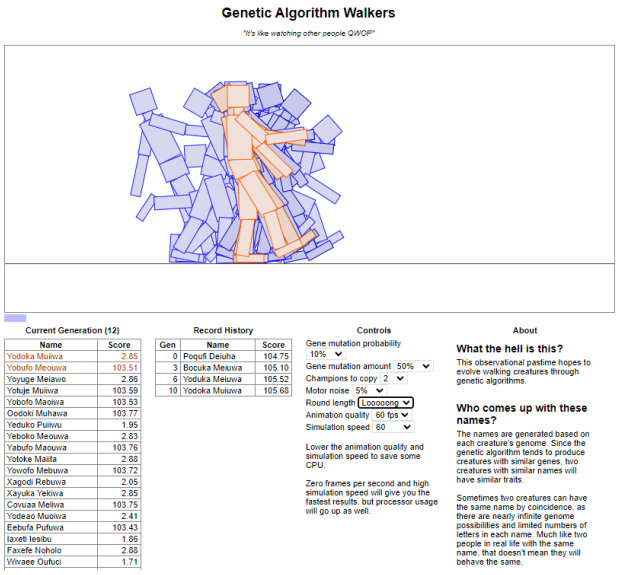
Obrázek 2.10 Příklad korektního řešení umístnění čtyř dam na šachovnici [13]

Kapitola 3
Porovnání existujících řešení

V této kapitole analyzuji již existující podobná řešení, která se zabývají podobnou problematikou. U každého shrnu případné zajímavé funkce, které nástroje nabízí, a také jejich nedostatky, které se pokusím ve své práci odstranit. Během analýzy existujících řešení jsem bral v potaz pouze aplikace, které jsou dostupné na webu a jsou tedy snadno přístupné pro případného uživatele.

3.1 Genetic Algorithm Walkers

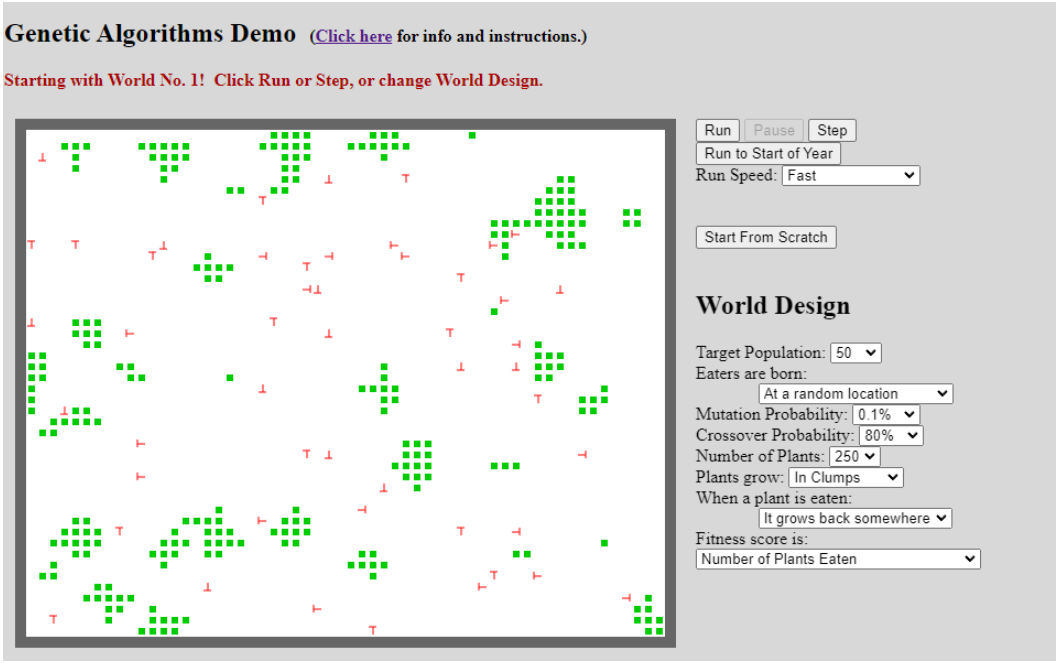
Webová aplikace Genetic Algorithm Walkers [14] nabízí vizualizaci pouze jediného problému - učení 2D humanoidních postav chůzi. Na stránce není k dispozici popis fungování genetických algoritmů a případný zájemce by proto musel hledat stručný úvod do problematiky jinde. Uživatelské prostředí (UI) aplikace není optimalizované pro telefony a na stránce není vysvětlený popis jednotlivých parametrů, které lze měnit. Stránka neumožňuje pokročilé změny parametrů jako například typ mutace a křížení. Stránka může být vhodná pro uživatele s předchozí znalostí genetických algoritmů, který by hledal tuto konkrétní vizualizaci. Dle mého názoru je UI aplikace poněkud zastaralé a nezaujme na první pohled případného zájemce.



■ **Obrázek 3.1** Snímek obrazovky z aplikace Genetic Algorithm Walkers [14]

3.2 Genetic Algorithms Demo

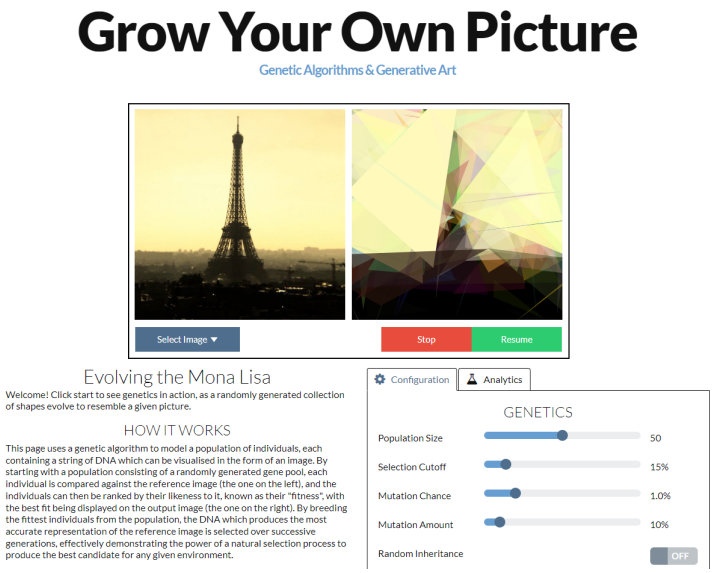
Webová aplikace Genetic Algorithms Demo [15] také nabízí vizualizaci pouze jediného optimalizačního problému. Na rozdíl od předchozí aplikace obsahuje i vysvětlení fungování genetických algoritmů, jejich parametrů i detailní popis vizualizovaného problému. Stránka ale nedisponuje responzivním UI, které by se jakkoliv přizpůsobilo obrazovce mobilního telefonu. Myslím si, že problém, který aplikace vizualizuje, je poměrně komplikovaný a vyžaduje důkladné přečtení zmíněné sekce, která vysvětluje jeho podstatu. To může případného zájemce, společně s nemoderním designem stránky, odradit.



Obrázek 3.2 Snímek obrazovky z aplikace Genetic Algorithms Demo [15]

3.3 Grow Your Own Picture

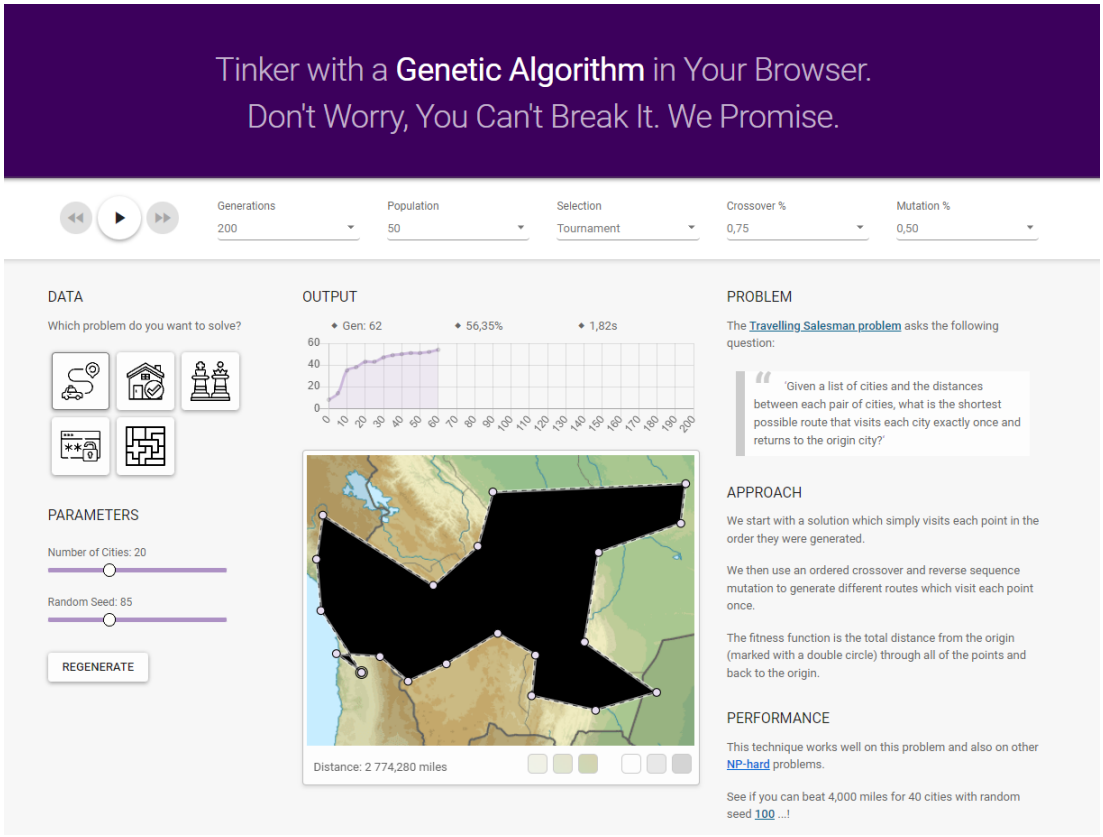
Webová aplikace Grow Your Own Picture [16] je také aplikací zaměřující se na vizualizaci jednoho konkrétního problému. Aplikace umožňuje uživateli generovat obraz pomocí barevných polygonů. Jedná se o jedinou webovou aplikaci, která nějakým způsobem aproximuje bitmapovou grafiku, kterou jsem během analýzy existujících řešení našel. Vizualizace 2D grafiky může být dle mého názoru skvělým způsobem jak názorně ukázat iterativní chování genetických algoritmů nezkušenému uživateli. Aplikace disponuje responzivním uživatelským prostředím, na stránce je také stručné vysvětlení fungovaní aplikace.



■ **Obrázek 3.3** Snímek obrazovky z aplikace Grow Your Own Picture [16]

3.4 Blazor.ai

Webová aplikace Blazor.ai [17] nabízí možnost vizualizace pěti různých optimalizačních úloh. U každé úlohy je dostupný její krátký popis a na stránce je i stručně vysvětlená obecná myšlenka a princip genetických algoritmů. Stránka má responzivní UI a nabízí možnost měnit parametry genetického algoritmu. Aplikace nabízí pouze omezený počet zadání pro každou úlohu, neumožňuje jednotlivé krokování po generacích. Nenabízí ani možnost měnit typ mutace a křížení.



Obrázek 3.4 Snímek obrazovky z aplikace Blazor.ai [17]

3.5 Shrnutí analýzy existujících řešení

Z mé provedené analýzy vyplývá, že na webu již existují veřejně dostupné aplikace, které umožňují vizualizaci řešení optimalizačních úloh pomocí genetických algoritmů. Většina existujících nástrojů ale vizualizuje pouze jediný konkrétní problém, nenabízí přívětivé UI, vysvětlení teorie genetických algoritmů a implementované úlohy nebo neumožňuje pokročilou modifikaci parametrů. Z mého pohledu řeší problematiku nejlépe již zmíněný nástroj Blazor.ai [17]. I tento nástroj má ale své nedostatky. Ve své práci se tedy pokusím využít poznatků získaných během této analýzy a vyvinout nástroj, který bude pro případného zájemce svými nabízenými funkcemi i uživatelským prostředím nejvhodnější a nejpřínosnější volbou.

Název aplikace	Vhodné UI	Více vizualizací	Popis teorie	Modifikovatelnost	Krokování
Genetic Algorithm Walkers	Ne	Ne	Ne	Částečná	Ne
Genetic Algorithms Demo	Ne	Ne	Ano	Částečná	Ano
Grow Your Own Picture	Ano	Ne	Stručný	Částečná	Ano
Blazor.ai	Ano	Ano	Stručný	Částečná	Ne

Tabulka 3.1 Tabulka se srovnáním existujících řešení

Kapitola 4
Analýza

V této kapitole popisuji analýzu - jednu z důležitých částí životního cyklu vývoje softwarového díla. Součástí této kapitoly je popis funkčních a nefunkčních požadavků kladených na výslednou aplikaci. Naplnění všech těchto požadavků pak bude mým cílem v implementační části této práce.

4.1 Popis požadované aplikace

Cílem mé bakalářské práce je vyvinout funkční veřejně dostupnou aplikaci, která uvede odborné zájemce i širokou veřejnost do problematiky řešení optimalizačních úloh pomocí genetických algoritmů. Aplikace by tedy měla být jednoduše dostupná a nabízet vizualizaci hned několika optimalizačních problémů pro lepší názornost. V kapitole 3 jsem totiž svou analýzou existujících řešení zjistil, že žádný dostupných nástrojů nenabízí současně možnost vizualizace více úloh pro lepší názornost, pokročilou modifikaci parametrů, vysvětlení problematiky a vhodné UI. Nyní tedy popíšu zvolené požadavky na aplikaci, která by měla tyto nedostatky jiných nástrojů řešit.

4.2 Model případů užití

V této podkapitole popíšu funkční a nefunkční požadavky kladené na aplikaci a případy užití, které je budou naplňovat.

4.2.1 Funkční požadavky

Vzhledem k faktu, že aplikace nenabízí více uživatelských rolí ani nedisponuje sekcí pro administrátory, je jediným aktérem vystupujícím v seznamu funkčních požadavků standardní *uživatel*. Uživatelem je kdokoliv, kdo aplikaci spustí.

- FP1 – Volba jazykové preference aplikace

  Aplikace bude umožňovat volbu jazyka mezi češtinou a angličtinou, aby byla použitelná pro širší okruh uživatelů.

- FP2 – Možnost navigace v aplikaci na stránky problémů a stránku s teorií problematiky

  Aplikace bude obsahovat navigační menu, ve kterém si uživatel může vybrat, který optimalizační problém chce zobrazit. Z navigačního menu se lze také vrátit zpět na úvodní stránku a na stránku se zjednodušeným shrnutím teorie problematiky genetických algoritmů. Aplikace také bude obsahovat velmi stručný popis každé implementované optimalizační úlohy.

- FP3 – Spuštění vizualizace řešení optimalizačního problému

  Aplikace bude mít viditelný panel pro ovládání vizualizace řešení daného optimalizačního problému. Ovládací panel bude umožňovat automatické spouštění dalších generací, pozastavení tohoto automatického běhu programu, krokování řešení problému po jedné generaci a resetování řešení problému.

- FP4 – Změna parametru optimalizačního problému

  Uživateli bude v aplikaci umožněno měnit základní parametr daného optimalizačního problému pomocí interaktivního komponentu. Takovým parametrem může být například délka chromozomu reprezentujícího řešení úlohy. Každá úloha umožňuje nějakou vhodnou modifikaci tohoto typu.

- FP5 – Změna parametrů genetického algoritmu

  Aplikace bude nabízet jednoduché menu pro změnu parametrů genetického algoritmu na stránkách s optimalizačními problémy. Uživatel bude moci měnit cílový počet generací (ukončovací podmínku), typ křížení, typ selekce, typ mutace a pravděpodobnost mutace.

- FP6 – Zobrazení výsledků řešení optimalizační úlohy

  Uživatel v aplikaci uvidí po každé uběhlé generaci aktuální vizualizovaný výsledek řešení úlohy a vývoj hodnoty fitness funkce, která ohodnocuje kvalitu řešení. Vizualizace problému i graf fitness funkce se automaticky aktualizuje po skončení každé iterace genetického algoritmu.

4.2.2 Nefunkční požadavky

- NP1 – Vzhled uživatelského prostředí

  Aplikace bude nabízet moderní a přehledné uživatelské prostředí pro jednoduchou navigaci. Uživatelské prostředí se přizpůsobí i menším obrazovkám, aby se aplikace dobře používala i na mobilních telefonech a dalších podobných zařízeních.

- NP2 – Dostupnost aplikace

  Aplikace bude spustitelná ve webovém prohlížeči bez nutnosti manuálního stažení dalšího softwaru. Aplikace bude optimalizována pro řádné fungování v prohlížečích Google Chrome, Mozilla Firefox a Safari. Nástroj bude spustitelný na počítačích i na přenosných zařízeních (notebooky, tablety, mobilní telefony).

- NP3 - Náročnost na výpočetní výkon

  Běh výpočetně náročných algoritmů bude prováděn na straně klienta, dle závislosti na velikosti řešené úlohy a výkonu použitého zařízení se může lišit doba vykreslení vizualizace následující generace řešení úlohy. Aplikace by i přes pomalejší běh měla být stále spustitelná a plnit svůj účel i na mobilních telefonech, zařízeních s pomalým připojením k internetu a ostatních zařízeních s nižším výpočetním výkonem. Uživatelský zážitek na těchto zařízeních tím ale může být ovlivněn.

- NP4 – Nasazení aplikace

  Aplikaci bude možné jednoduše nasadit na webový server jako statickou webovou stránku. Aplikace bude vyžadovat minimum nastavení na straně webhostingu.

- NP5 - Rozšířitelnost a udržovatelnost aplikace

  Aplikace bude řádně zdokumentovaná, aby bylo možné ji v budoucnu případně snadno rozšířit nebo upravit. To zahrnuje především dokumentaci zdrojových kódů. Aplikace by svou architekturou měla umožnit přidání i dalších optimalizačních problémů.

4.2.3 Případy užití

- UC1 – Změna jazyku, ve kterém se aplikace zobrazuje

  Umožňuje změnit preferovaný jazyk aplikace.

  - Hlavní scénář:
    * Uživatel si otevře navigační menu pomocí tlačítka na horním panelu.
    * Uživatel si v navigačním menu zvolí preferovaný jazyk kliknutím na příslušné tlačítko ve spodní části panelu.
    * Aplikace změní jazyk celého uživatelského prostředí bez nutnosti aktualizovat stránku.

- UC2 – Zobrazení shrnutí teoretického základu genetických algoritmů

  Umožňuje zobrazit stránku se zjednodušeným úvodem do teorie genetických algoritmů.

  - Hlavní scénář:
    * Uživatel si otevře stránku s úvodem do teorie genetických algoritmů pomocí hypertextového odkazu na úvodní stránce.
    * Aplikace zobrazí požadovanou stránku, bere v potaz zvolenou jazykovou preferenci.
  - Alternativní scénář:
    * Uživatel klikne na tlačítko stránky s teorií genetických algoritmů na navigačním postranním panelu. Ten bude dostupný na všech stránkách aplikace.
    * Aplikace zobrazí požadovanou stránku, bere v potaz zvolenou jazykovou preferenci.

- UC3 – Zobrazení popisu nabízených optimalizačních úloh

  Umožňuje zobrazit stručný popis každé optimalizační úlohy.

  - Hlavní scénář:
    * Uživatel si pomocí levého postranního navigačního menu otevře úvodní stránku, pokud se nachází na jiné stránce. Případně se úvodní stránka otevírá jako výchozí, pokud uživatel poprvé spouští webovou aplikaci.
    * Uživateli se zobrazí tyto informace ihned po otevření úvodní stránky.
  - Alternativní scénář:
    * Uživatel si pomocí postranního navigačního panelu otevře konkrétní optimalizační problém, který ho zajímá pomocí příslušného tlačítka.
    * Uživateli se otevře stránka s možností interaktivní vizualizace daného problému.
    * Uživatel se posune na spodní část stránky, pod vizualizační komponenty, kde je stručně popsána podstata dané úlohy. Obsah popisu je identický jako na úvodní stránce.

- UC4 – Zobrazení stránky s vizualizací optimalizační úlohy

  Umožňuje uživateli navštívit stránku s interaktivní vizualizací dané optimalizační úlohy.

  - Hlavní scénář:
    * Uživatel si pomocí navigačního panelu zvolí konkrétní problém kliknutím na příslušné tlačítko.
    * Aplikace uživateli zobrazí stránku s vizualizací daného problému.
  - Alternativní scénář:
    * Uživatel klikne na hypertextový link v nadpisu daného problému na úvodní stránce aplikace.
    * Aplikace uživateli zobrazí stránku s vizualizací daného problému.

- UC5 – Ovládání vizualizačních součástí aplikace

  Umožňuje uživateli spustit vizualizaci implementovaných optimalizačních problémů.

  - Hlavní scénář:
    * Případ začíná otevřením příslušné stránky v aplikaci s danou optimalizační úlohou (*include UC4*).
    * Uživatel spustí automatický běh vizualizace pomocí tlačítka „Spustit“ s ikonou natočeného trojúhelníku na ovládacím panelu v levé horní části stránky.
    * Aplikace automaticky provede vizualizaci řešení až do ukončovací podmínky, kterou je v tomto případě parametr maximálního počtu generací. Po vytvoření a ohodnocení každé generace se aktualizuje komponent zobrazující nejlepší řešení v dané generaci.
    * Uživatel může běh programu pozastavit tlačítkem „Pauza“ s ikonou dvou svislých rovnoběžných čar. Běh lze potom obnovit znovu v automatickém režimu (hlavní scénář) nebo v krokovacím režimu (alternativní scénář).
    * Aplikace ukončí běh po dosažení ukončující podmínky a čeká na další vstup uživatele. Uživatel si může nechat pomocí tlačítka „Reset”(ikona dvou zatočených šipek ve tvaru kruhu) nechat vygenerovat nové řešení a vizualizaci opakovat s případně jinými parametry genetického algoritmu.
  - Alternativní scénář:
    * Případ začíná otevřením příslušné stránky v aplikaci s danou optimalizační úlohou (*include UC4*).
    * Uživatel spustí vizualizaci v režimu krokování pomocí tlačítka „Krok“ s ikonou jednoduché šipky směřující doprava.
    * Aplikace provede vytvoření a vyhodnocení pouze jediné generace po každém kliknutí. Vizualizace výsledku probíhá stejně jako v hlavním scénáři.
    * Uživatel může pomocí tlačítka s ikonou natočeného trojúhelníku přejít i do režimu automatického krokování vizualizace, který je popsaný v hlavním scénáři.

- UC6 – Zobrazení vizualizace problému obchodního cestujícího

  Umožňuje uživateli zobrazit vizuální reprezentaci řešení problému obchodního cestujícího.

  - Hlavní scénář:
    * Případ začíná spuštěním vizualizačního komponentu ovládacím panelem (*include UC5*).
    * Uživateli se po každé uplynulé generaci (funguje v automatickém i krokovacím režimu) zobrazí reprezentace nejlepšího řešení problému obchodního cestujícího v dané generaci. Na obrazovce se vždy znovu vykreslí 2D plátno s kruhy, které představují města, která má obchodní cestující navštívit. Tyto body jsou spojené úsečkami, které reprezentují trasu a pořadí, ve kterém obchodní cestující města navštíví.

- UC7 – Zobrazení vizualizace problému generování 2D bitmapové grafiky

  Umožňuje uživateli zobrazit vizuální reprezentaci řešení problému generování 2D bitmapového obrázku.

  - Hlavní scénář:
    * Případ začíná spuštěním vizualizačního komponentu ovládacím panelem (*include UC5*).
    * Uživateli se po každé uplynulé generaci (funguje v automatickém i krokovacím režimu) zobrazí reprezentace nejlepšího řešení problému generování 2D bitmapové grafiky. Na stránce bude vidět vždy barevná mřížka zobrazující aktuální řešení generované genetickým algoritmem, poté ideální řešení pro daný obrázek (tedy obrázek s rozlišením sníženým na stejnou velikost) a také původní, originální, obrázek v plném rozlišení.

- UC8 – Zobrazení vizualizace problému batohu

  Umožňuje uživateli zobrazit vizuální reprezentaci řešení problému batohu.

  - Hlavní scénář:
    * Případ začíná spuštěním vizualizačního komponentu ovládacím panelem (*include UC5*).
    * Uživateli se po každé uplynulé generaci (funguje v automatickém i krokovacím režimu) zobrazí reprezentace nejlepšího řešení problému batohu. Na obrazovce se předměty rozdělí do dvou sloupců, které představují množinu předmětů uvnitř batohu a předměty, které genetický algoritmus do batohu neumístí. Umístění předmětů bude jednoznačně graficky rozděleno.

- UC9 – Zobrazení vizualizace problému umístění N dam na šachovnici

  Umožňuje uživateli vidět vizuální reprezentaci řešení problému umístění N dam na šachovnici.

  - Hlavní scénář:
    * Případ začíná spuštěním vizualizačního komponentu ovládacím panelem (*include UC5*).
    * Uživateli se po každé uplynulé generaci (funguje v automatickém i krokovacím režimu) zobrazí reprezentace nejlepšího řešení problému umístění N dam na šachovnici. Na obrazovce je vidět tabulka s buňkami reprezentujícími pole klasické šachovnice o velikosti 8×8 polí. Po uplynutí každé generace je vidět nejlepší řešení, jak rozmístit dámy na šachovnici, aby se minimalizoval počet vzájemných kolizí.

- UC10 – Změna základního parametru optimalizačního problému

  Umožňuje uživateli u každého problému změnit základní parametr optimalizačního problému (například délku chromozomu nebo výběr originálního obrázku, který se algoritmus pokusí aproximovat). Tento parametr výrazně ovlivňuje výsledné řešení i proces optimalizace a umožňuje tak uživateli lépe pochopit daný problém.

  - Hlavní scénář:
    * Uživatel si v horní části obrazovky u konkrétního optimalizačního problému pozmění parametr, který byl k danému problému vybrán. Pokud se jedná o výběr délky chromozomu, lze ji vybrat pomocí interaktivního posuvníku nebo vstupního pole. Pokud se jedná o výběr obrázku k aproximaci, jedná se o výběrové menu nabízející jednotlivé možnosti.
    * Provedená změna se ihned projeví překreslením vizualizačního komponentu.

- UC11 – Změna parametrů genetického algoritmu

  Umožňuje uživateli změnit parametry genetického algoritmu. Změnu lze provést vždy před spuštěním vizualizace.

  - Hlavní scénář:
    * Uživatel si na stránce s příslušným optimalizačním problémem otevře pravý postranní panel s nastavením parametrů genetického algoritmu pomocí tlačítka s ikonou ozubeného kola na pravé straně horní lišty. Uživatel z následujících kroků hlavního scénáře může změnit všechna nastavení nebo jejich libovolnou podmnožinu.
    * Uživatel pomocí interaktivního posuvníku nebo číselného vstupu zvolí limit maximálního počtu generací, který slouží jako ukončující podmínka genetického algoritmu.
    * Uživatel pomocí výběrového menu zvolí preferovaný typ selekce, který má být použit pro řešení optimalizační úlohy.
    * Uživatel pomocí výběrového menu zvolí preferovaný typ křížení, který má být použit pro řešení optimalizační úlohy.

- Uživatel pomocí výběrového menu zvolí preferovaný typ mutace, který má být použit pro řešení optimalizační úlohy.
- Uživatel pomocí interaktivního posuvníku nebo číselného vstupu zvolí pravděpodobnost mutace při řešení optimalizační úlohy.

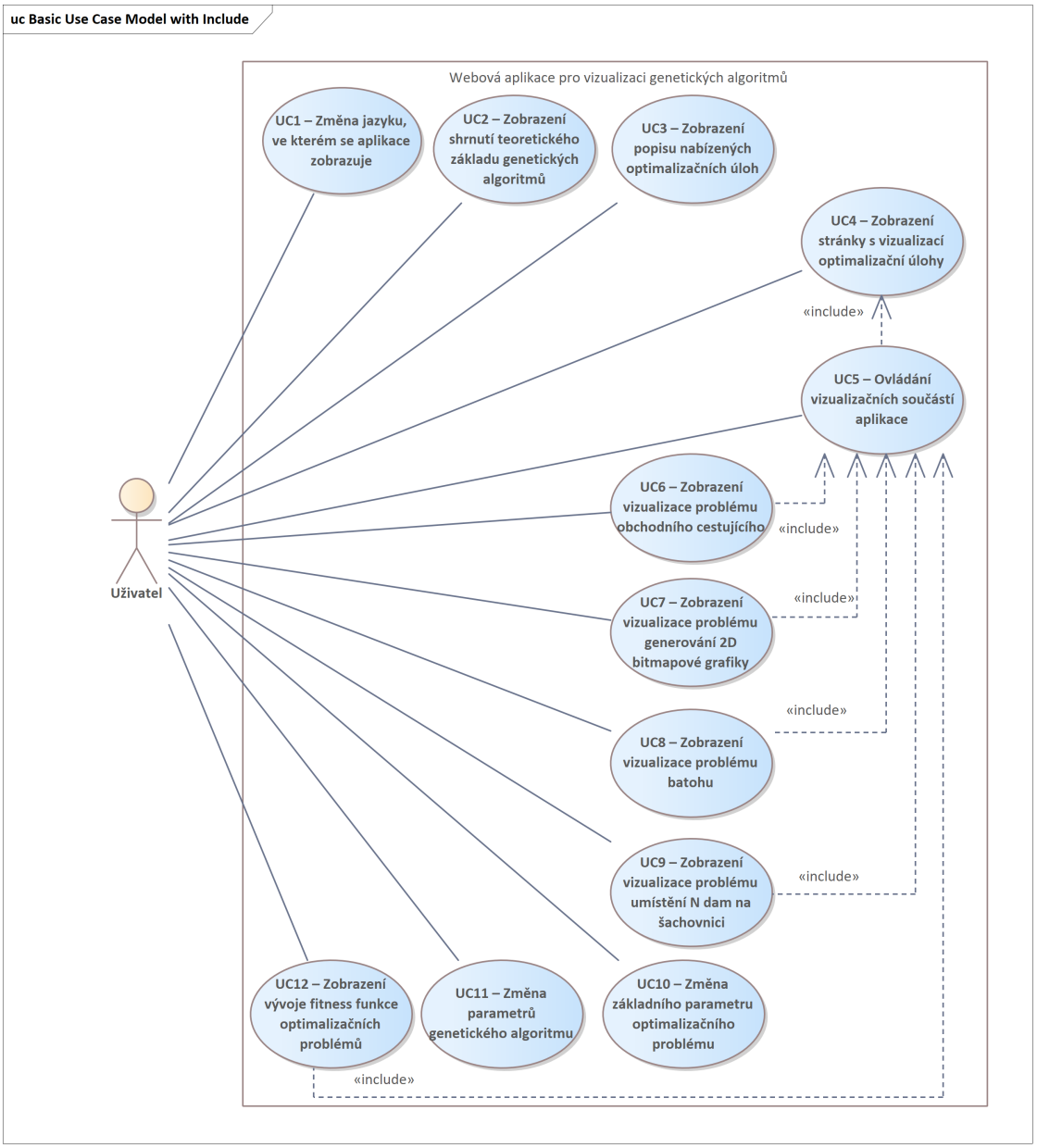
- UC12 – Zobrazení vývoje fitness funkce optimalizačních problémů

  Umožňuje uživateli zobrazit a dále pracovat s ohodnocením řešení optimalizační úlohy v jednotlivých generacích. Data jsou zobrazena v interaktivním grafu.

  - Hlavní scénář:
    - Případ začíná spuštěním vizualizačního komponentu ovládacím panelem (*include UC5*).
    - Aplikace graf fitness funkce automaticky aktualizuje po každé uplynulé generaci.
    - Uživatel pomocí pohybu myši může analyzovat konkrétní části grafu, který zobrazuje vývoj ohodnocení řešení pomocí fitness funkce.
    - Uživatel pomocí navigačních tlačítek v horní části grafu změní rozsah grafu, případně může výsledek exportovat jako obrázek a uložit jej.

	Functional requirements::FP1 – Volba jazykové preference aplikace	Functional requirements::FP2 – Možnost navigace v aplikaci na stránky problémů a stránku s teorií problematiky	Functional requirements::FP3 – Spuštění vizualizace řešení optimalizačního problému	Functional requirements::FP4 – Změna parametru optimalizačního problému	Functional requirements::FP5 – Změna parametrů genetického algoritmu	Functional requirements::FP6 – Zobrazení výsledků řešení optimalizační úlohy
Use Cases::UC1 – Změna jazyku, ve kterém se aplikace zobrazuje	⇑					
Use Cases::UC10 – Změna základního parametru optimalizačního problému				⇑		
Use Cases::UC11 – Změna parametrů genetického algoritmu					⇑	
Use Cases::UC12 – Zobrazení vývoje fitness funkce optimalizačních problémů						⇑
Use Cases::UC2 – Zobrazení shrnutí teoretického základu genetických algoritmů		⇑				
Use Cases::UC3 – Zobrazení popisu nabízených optimalizačních úloh		⇑				
Use Cases::UC4 – Zobrazení stránky s vizualizací optimalizační úlohy		⇑				
Use Cases::UC5 – Ovládání vizualizačních součástí aplikace			⇑			
Use Cases::UC6 – Zobrazení vizualizace problému obchodního cestujícího			⇑			
Use Cases::UC7 – Zobrazení vizualizace problému generování 2D bitmapové grafiky			⇑			
Use Cases::UC8 – Zobrazení vizualizace problému batohu			⇑			
Use Cases::UC9 – Zobrazení vizualizace problému umístění N dam na šachovnici			⇑			

Obrázek 4.1 Matice odpovědností



■ **Obrázek 4.2** Diagram s případy užití

4.3 Analýza vhodných technologií a knihoven

Po provedení analýzy existujících řešení a požadavků na aplikaci je nutné zvolit vhodnou technologii a nástroje, pomocí kterých se bude aplikace jednoduše vyvíjet a umožní naplnění všech požadovaných funkcí. Právě touto volbou se budu zabývat v této podkapitole.

4.3.1 Technologie

Aby byla dostupnost aplikace co nejsnazší, požadujeme po aplikaci dostupnost ve webovém prohlížeči. Využití technologie pro vývoj lokálně spustitelných aplikací by totiž výrazně omezilo potenciál dosahu vyvíjené aplikace. [18] uvádí, že důležitým kritériem, které zohlednit při výběru vhodné technologie pro tvorbu nového projektu, je předchozí zkušenost vývojáře s danou technologií - i toto kritérium tedy budu brát během své analýzy v potaz.

4.3.1.1 JavaScript

JavaScript se v roce 2022 už desátý rok v řade umístil na první místo v žebříčku nejpopulárnějších programovacích jazyků [19]. Díky velké komunitě aktivních vývojářů je jednoduché nalézt materiály a odpovědi na otázky, které mohou během vývoje nastat. U JavaScriptu se často využívá externích frameworků, které usnadňují vývojářům práci a šetří čas implementací obecných funkcionalit aplikace. Mezi populární frameworky jazyku JavaScript se řadí například React, Angular nebo Vue.js [20]. Subjektivní nevýhodou JavaScriptu pro mě je má malá zkušenost s tímto jazykem. Pokud pro JavaScript existuje vhodná knihovna, bylo by využití tohoto jazyka s vhodným frameworkem jistě dobrou volbou.

4.3.1.2 Java

Java se v roce 2022 umístila na šestém místě v žebříčku nejpopulárnějších programovacích jazyků [19]. Javu jsem se v této analýze rozhodl zahrnout, protože je to jazyk, se kterým mám nejvíce zkušeností. Java se ale zcela nehodí pro vývoj frontendu a grafických webových aplikací. I přes pokus ji zavést jako jednu z možností vývoje webů pomocí takzvaných *appletů* už v roce 1995 [21], se jako jazyk pro tvorbu webových stránek neuchytila a podpora appletů v prohlížečích se začala od roku 2013 ukončovat. Dnes již nejsou applety podporovány v moderních prohlížečích vůbec [22]. Java sice nachází své uplatnění v mnoha oblastech vývoje, pro účely tvorby webové aplikace je však nevhodná.

4.3.1.3 Python

Python se také zařadil v roce 2022 mezi pět nejoblíbenějších jazyků [19]. Python má ve vývoji velmi široké uplatnění, lze ho využít mimo jiné pro datovou analýzu a strojové učení, automatizaci úloh, tvorbu her nebo právě i pro vývoj webových aplikací [23]. Mezi populární frameworky pro tvorbu webových aplikací se řadí například Django[24]. Django je však framework pro tvorbu backendové částí webových aplikací. Během analýzy se mi nepodařilo objevit populární frontendový framework pro jazyk Python, který by byl vhodný pro potřeby vyvíjené aplikace.

4.3.1.4 C# - Unity Engine

Vzhledem k mým zkušenostem s herním enginem Unity jsem uvažoval o jeho využití. Unity engine se nejčastěji používá pro tvorbu 2D a 3D her a grafických nástrojů, které se exportují jako lokálně spustitelné aplikace. Vzhledem k tomu, že se používá pro tvorbu grafických aplikací, jsem předpokládal, že by mohl být dobrou volbou pro tvorbu mého nástroje, protože by mi při vývoji mohl hodně usnadnit práci s tvorbou UI a grafických komponentů. Během analýzy jsem také zjistil, že Unity podporuje od roku 2018 možnost exportovat projekt do WebGL s technologií WebAssembly (WASM) a aplikace by tak mohla být spustitelná přímo v prohlížeči [25]. Vzhledem k mé předchozí zkušenosti s jazykem Java, C# a C++ jsem usoudil, že pokud by vše fungovalo dle očekávání, mohlo by být využití Unity vhodnou volbou pro tvorbu projektu. V rámci analýzy jsem vytvořil několik testovacích projektů, abych ověřil vhodnost nástroje. Bohužel jsem ale narazil na problémy při spouštění exportované webové aplikace na mobilním telefonu a na problémy při snaze využívat více vláken. Unity dokumentace potvrzuje, že podpora více vláken při psaní C#

kódu není při exportu do WebAssembly podporována [26]. V kombinaci s velikostí výsledného projektu, který by musel uživatel stáhnout se tak Unity v závěru nejevilo jako dobrá volba.

4.3.1.5 C# - Blazor

V kapitole 3 s analýzou existujících řešení byl zmíněn nástroj Blazor.ai. Ten využívá framework Blazor, který je vyvíjen společností Microsoft[27]. Blazor mě během analýzy možných technologií zaujal, protože v něm již byl vytvořen podobný nástroj. Výhodou by pro mě byla předchozí zkušenost s jazyky využívajícími podobnou syntaxi. Při vývoji aplikací pomocí frameworku Blazor je možné vytvořit webové aplikace téměř bez využití HTML, CSS a JavaScriptu, namísto toho se využívají pro vývoj aplikací takzvané *Razor* komponenty. Framework Blazor lze využít třemi způsoby, které se zásadně liší:

- **Blazor WebAssembly** je jednostránková aplikace (Single Page Application), která běží na straně klienta pomocí .NET kódu, který se spouští ve webových prohlížečích právě pomocí technologie WebAssembly. Razor komponenty a C# kód jsou spouštěny v klientském prohlížeči. Technologie WASM je podporována ve všech moderních prohlížečích a funguje i na mobilních telefonech. S Blazor WASM lze tvořit interaktivní webové aplikace. Blazor WASM projekty lze hostovat jako statické webové aplikace díky tomu, že se obsah klientovi stahuje jako sada statických souborů. [28], [29]

  Za zmínku stojí také fakt, že WebAssembly podle [30] u reálných aplikací dosáhlo o 17.24% lepšího výkonu ve srovnání s JavaScriptem. Podle [31] dosahuje WebAssembly až 11.71x lepšího výkonu než JavaScript v prohlížeči Firefox na stolních počítačích a až 16.11x lepšího výkonu ve stejném prohlížeči na mobilním telefonu Moto G5 Plus. Vzhledem k výpočetní náročnosti na straně klienta při zpracování a vyhodnocování generací genetického algoritmu, považuji tento fakt za velkou výhodu umožňující plynulejší běh webové aplikace.

  Výhody:

  - Aplikace běží kompletně na straně klienta a využívá výkonu klientského zařízení, běh aplikace je velmi rychlý s krátkou odezvou.
  - Aplikace má velmi nízké nároky na výkon hostingu, kde je aplikace uložena. Ten totiž nevykonává žádné výpočetní operace, které aplikace během svého běhu vykonává.
  - Aplikace může fungovat i offline, nevyžaduje neustálé připojení k serveru.
  - Nasazení aplikace je velmi jednoduché, stačí přesunout soubory vygenerované sestavením projektu do složky webového serveru. Aplikaci je možné nasadit i bez serveru (například pomocí sítě pro doručování obsahu).
    [29], [32]

  Nevýhody:

  - Aplikace může na nevýkonných zařízeních fungovat pomaleji.
  - Aplikace potřebuje pro běh stáhnout větší objem souborů, spuštění tak může trvat déle na zařízeních s pomalým internetovým připojením. Další spuštění už však díky cacheování dat probíhají rychleji.
  - Některé staré prohlížeče nepodporují technologii WebAssembly.
    [29]

- **Blazor Server** hostuje Razor komponenty na straně serveru v aplikaci ASP.NET Core. Server zajišťuje spuštění C# kódu aplikace a aktualizace uživatelského rozhraní. Změny se zpracovávají pomocí SignalIR připojení. Server vždy odesílá klientovi rozdíl změn UI v binárním formátu a prohlížeč uživatele tyto změny zobrazí. Server uchovává stav stav přidružený ke každému konkrétnímu klientovi (tzv. okruh).[28], [29]

  Výhody:

- Velikost stažených souborů pro spuštění aplikace je výrazně menší.
- Aplikace využívá výpočetního výkonu serveru.
- Aplikace funguje i v prohlížečích nepodporujících WASM.
  [29]

Nevýhody:

- Aplikace má obvykle vyšší latenci, každá interakce vyžaduje komunikaci se serverem.
- Aplikace nefunguje bez připojení k serveru, je nutné stálé připojení.
- Aplikace se složitěji škáluje, pro více souběžných uživatelů je potřeba vyšší výkon serveru.
- Aplikace vyžaduje ASP.NET Core Server, bezserverová řešení nejsou možná.
  [29]

- **Blazor Hybrid** slouží k tvorbě nativních klientských aplikací, které je možné spouštět na klientských zařízeních. Nevýhodou je, že Blazor Hybrid aplikace nelze jednoduše spustit pomocí internetového prohlížeče. [29] Z tohoto důvodu se více touto možností nebudu zabývat, protože v mém případě by výrazně omezila dostupnost aplikace.

4.3.2 Knihovny

Vzhledem k rozsahu samotné aplikace, riziku chyb a faktu, že implementaci GA nabízí mnoho veřejně dostupných knihoven, jsem se rozhodl využít nějakého dostupného nástroje, který by automatizoval celý proces iterativní optimalizace pomocí genetických algoritmů. Vzhledem k rozdílné kvalitně dostupných knihoven bylo nutné provést jejich analýzu. Jako požadavky na knihovnu jsem si zvolil následující kritéria:

- Dostupnost knihovny - knihovna by měla mít veřejně dostupný zdrojový kód k nahlédnutí pro případ potřeby analyzovat interní fungování nějaké části knihovny.
- Spolehlivost knihovny - knihovna by měla být řádně otestovaná, abych se během vývoje vyhnul neočekávanému chování ze strany knihovny, které by bylo složité odladit a opravit.
- Komplexnost knihovny - knihovna by měla zajišťovat fungování celého iterativního procesu. To zahrnuje například implementaci vytvoření a vyhodnocení jednotlivých generací nebo implementaci základních metod selekce, křížení a mutace.

Ve své analýze knihoven budu porovnávat pouze knihovny pro jazyk JavaScript, Java, Python a C#, které jsem popisoval v předchozí části této kapitoly se srovnáním potenciálních technologií 4.3.1. Pokud existuje pro nějaký jazyk větší množství knihoven, popíšu zde maximálně dvě z nich, které jsem dle kritérií zmíněných výše vyhodnotil jako nejlepší. Během porovnávání popularity knihoven jsem vycházel především z počtu hvězdiček (stars) v repozitáři knihovny na stránce GitHub.

4.3.2.1 genetic-js (JavaScript)

Knihovna *genetic-js* je nejpopulárnější JavaScript knihovnou pracující s genetickými algoritmy, kterou jsem během analýzy našel. Knihovna ale nemá implementované žádné metody křížení a mutace - musely by tedy být implementovány manuálně. Obsah knihovny je otestován. [33]

4.3.2.2 genome.js (JavaScript)

Knihovna *genome.js* je znatelně méně populární než knihovna genetic-js. Knihovna neobsahuje testy kódu ani implementaci standardních metod mutace nebo křížení. [34]

4.3.2.3 Jenetics (Java)

Knihovna *Jenetics* je velmi populární knihovnou pro jazyk Java. Součástí je výborná dokumentace knihovny i uživatelská příručka, která velmi odborně popisuje problematiku GA. Knihovna je řádně otestovaná. [35]

4.3.2.4 deap (Python)

Knihovna *deap* pro jazyk Python pracuje nejen s genetickými algoritmy, ale například také poskytuje logiku algoritmů pro genetické programování a pro další algoritmy z oblasti evolučních algoritmů. Na stránce GitHub je velmi populární, je vhodně otestovaná a implementuje v sobě několik typů selekce, křížení i mutace. [36]

4.3.2.5 PyGAD (Python)

Knihovna *PyGAD: Genetic Algorithm in Python* pro jazyk Python je knihovna usnadňující využití genetických algoritmů a pro optimalizaci algoritmů z oblasti strojového učení. Aplikace má aktivní komunitu a je velmi populární na stránce GitHub. Aplikace je vhodně otestovaná a implementuje v sobě několik typů selekce, křížení i mutace. [37], [38]

4.3.2.6 GeneticSharp (C#)

Knihovna *GeneticSharp* pro jazyk C# usnadňuje práci s genetickými algoritmy. Zajišťuje obecné fungování genetického algoritmu, implementuje velké množství standardních metod křížení, selekce i mutace. Nastavení lze do velké míry přizpůsobit konkrétním potřebám a celá knihovna je řádně otestovaná. [39]

4.3.2.7 AForge.NET (C#)

Knihovna *AForge.NET* pro jazyk C# obsahuje kód nejen pro genetické algoritmy, ale také pro další odvětví strojového učení, zpracování obrazu a další. Knihovna je vhodně otestovaná, ale na rozdíl od knihovny GeneticSharp neimplementuje velkou množinu typů křížení a mutace. [40]

4.3.3 Shrnutí analýzy technologií a knihoven

Z analýzy potenciálně využitelných technologií a knihoven vyplývají následující závěry:

Pro jazyk JavaScript neexistuje knihovna, která by splňovala všechny stanovené požadavky. V kombinaci s mou menší zkušeností s jazykem JavaScript z toho plyne závěr, že tento jazyk by nebyl vhodný pro implementaci tohoto projektu.

Pro jazyky Python, Java i C# existuje alespoň jedna knihovna, která splňuje všechna požadovaná kritéria. V případě využití kterékoliv z těchto technologií lze použít spolehlivou knihovnu, která výrazně usnadní práci s genetickými algoritmy.

Java ani Python však nejsou pro tvorbu webové interaktivní aplikace v porovnání s technologií Blazor dobrou volbou, a to z důvodu absence vhodného frameworku nebo jiné možnosti pro vývoj grafických webových aplikací.

Na základě poznatků z kapitoly 3, kde jsem zjistil, že frameworkem Blazor byla vytvořena plně funkční podobná aplikace, a vzhledem ke své předchozí znalosti podobných jazyků jako C#, jsem se tedy rozhodl využít právě jazyk C# a framework Blazor.

Využiji Blazor WebAssembly, protože aplikaci bude snadné nasadit na jakýkoliv webhostingový server. Navíc nebude nutné se starat o případné škálování aplikace, protože potřebný výpočetní výkon bude směrován na zařízení klienta. S pomocí Razor komponentů předpokládám efektivní vývoj aplikace s minimem práce v JavaScriptu a kaskádování pomocí CSS.

Technologie	Vhodnost	Dostupné knihovny	Předchozí zkušenost
JavaScript	Výborná	Nedostatečné	Nízká
Java	Velmi špatná	Výborné	Velmi dobrá
Python	Špatná	Výborné	Nízká
C# - Unity	Přijatelná	Výborné	Průměrná
C# - Blazor	Výborná	Výborné	Průměrná

Tabulka 4.1 Tabulka se srovnáním potenciálně vhodných technologií

Kapitola 5
Návrh

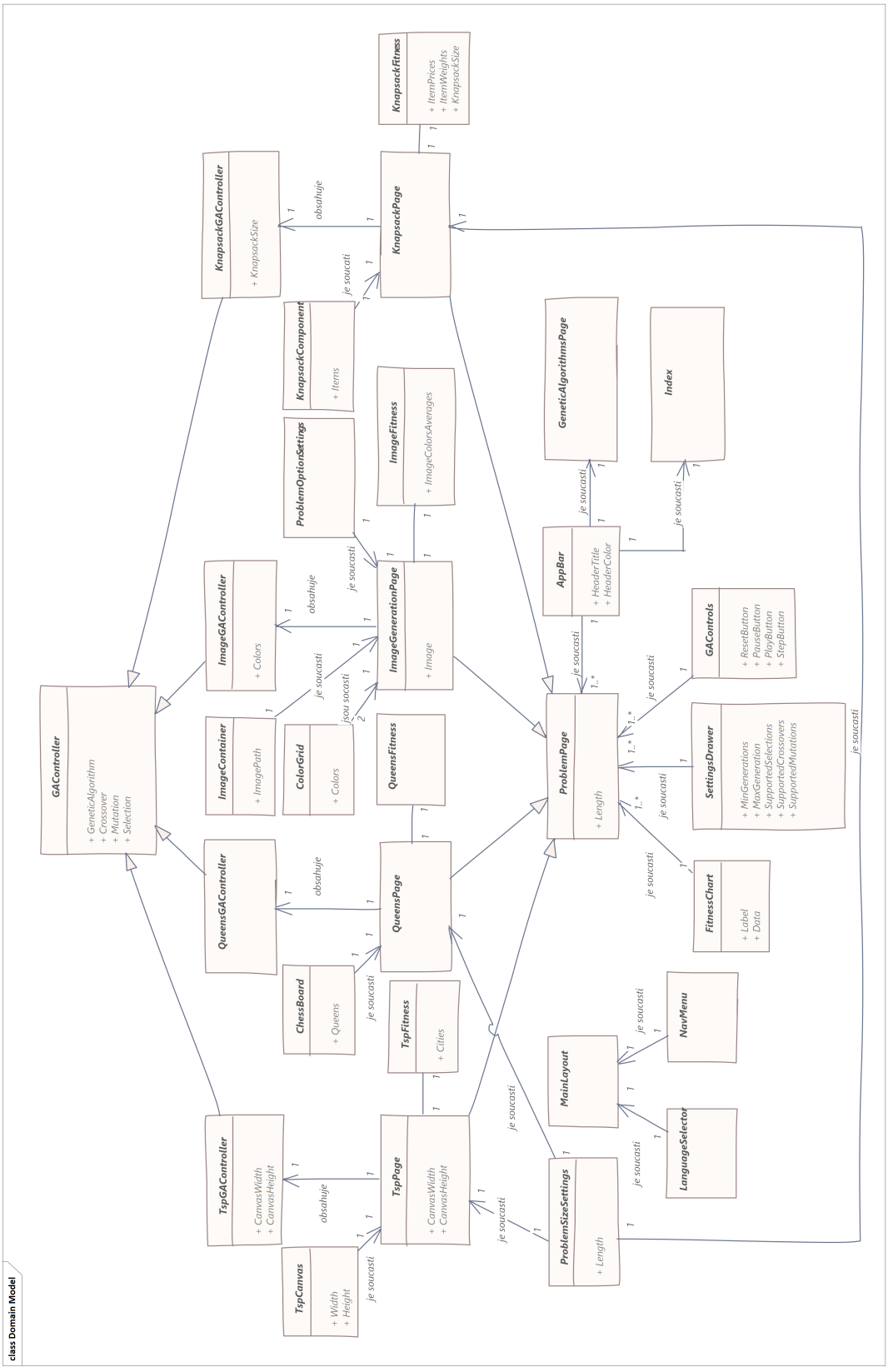
V této kapitole se budu věnovat popisu návrhu webové aplikace. To bude zahrnovat doménový model, který bude zobrazovat základní strukturu architektury aplikace, a také návrh uživatelského rozhraní. Vhodně navržená architektura aplikace může výrazně usnadnit další vývoj a rozšíření aplikace, stejně jako návrh uživatelského prostředí pomocí wireframů umožňuje vývojářům mít lepší představu o finální podobě aplikace ještě před její implementací.

5.1 Doménový model a architektura aplikace

Pro usnadnění následné implementace webové aplikace a dodržení vhodného návrhu architektury jsem vytvořil základní doménový model webové aplikace.

Doménový model se vytváří v počáteční fázi vývoje softwaru. Na rozdíl od návrhového diagramu tříd je platformově *nezávislý* a třídy v doménovém modelu jsou popsány zjednodušeně. Jednotlivé třídy nemají popsané své metody a zahrnuty jsou pouze důležité atributy. Doménový model slouží pro identifikaci důležitých entit a vzájemných vztahů mezi nimi. [41]

V doménovém modelu jsem zohlednil podobné vlastnosti a součásti stránek pro jednotlivé vizualizace a controllerů pro konkrétní problémy - vzhledem k inkrementálním změnám u jednotlivých konkrétních problémů se zde nabízelo využít principu dědičnosti. Díky tomu je možné případně aplikaci poměrně jednoduše rozšířit o další optimalizační problém bez nutnosti psaní velkého množství opakujícího se kódu.



■ **Obrázek 5.1** Snímek obrazovky s doménovým modelem aplikace

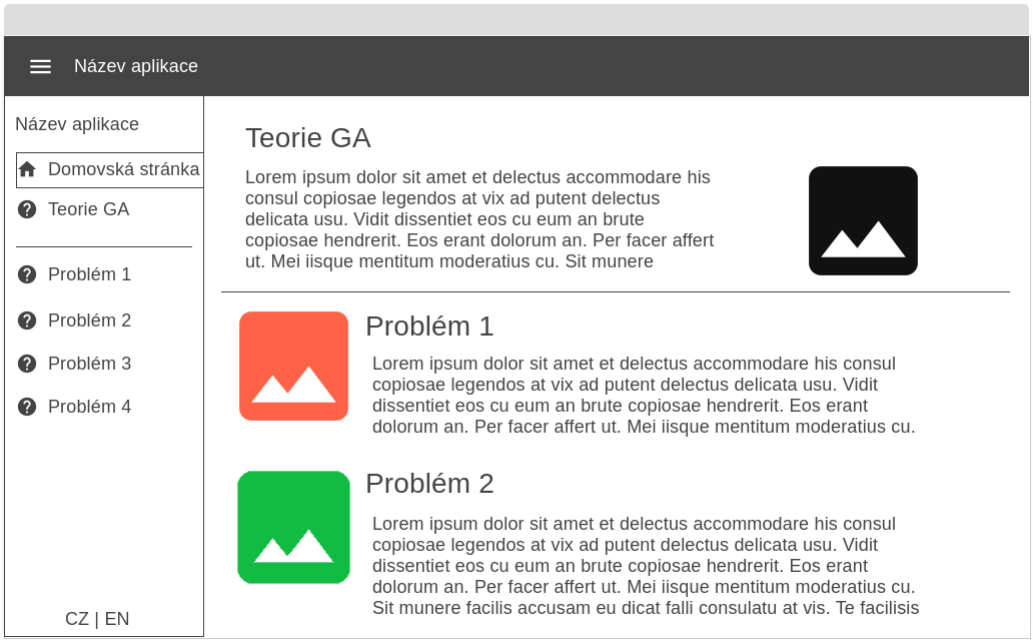
5.2 Uživatelské prostředí

Uživatelské prostředí jsem navrhoval pomocí takzvaných wireframů.

Wireframy umožňují designovat rozvržení komponentů uživatelského prostředí na klíčových stránkách. Jedná se o jednoduché grafické náčrty, které jsou u velkých projektů vhodné pro sjednocení představy o vzhledu aplikace pro ostatní členy týmu nebo pro představení konceptu UI klientovi. [42]

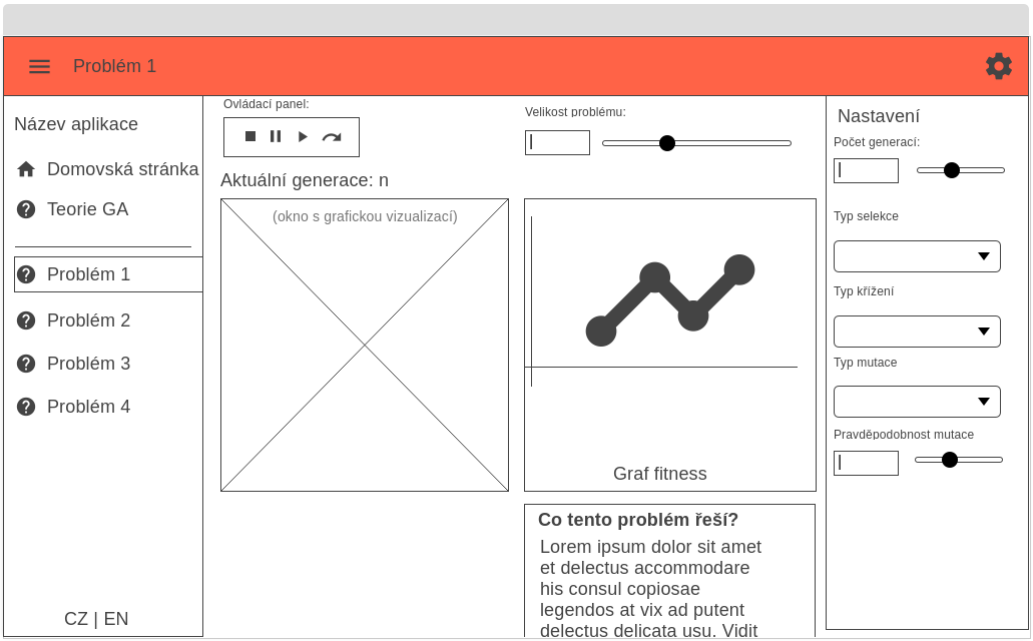
Hlavním cílem při návrhu grafického rozhraní pro mě byla jednoduchost, intuitivnost a přehlednost. Dále aby stránka působila hravým dojmem, který vyzývá uživatele k vyzkoušení všech různých problémů a ke zkoumání změny chování při změně parametrů problému a genetického algoritmu. Ke každému optimalizačnímu problému jsem přiřadil jednu barvu, aby stránka nepůsobila příliš chladně a akademicky a neodradila tak případné laické zájemce. Touto barvou bude vždy vybarvena ikona problému na úvodní stránce a také vrchní panel na stránce s optimalizačním problémem.

Každá stránka webové aplikace má k dispozici hlavní navigační menu - levý postranní panel, kterým lze měnit zobrazovanou úlohu a další stránky. V tomto hlavním panelu je možné také změnit jazyk aplikace.

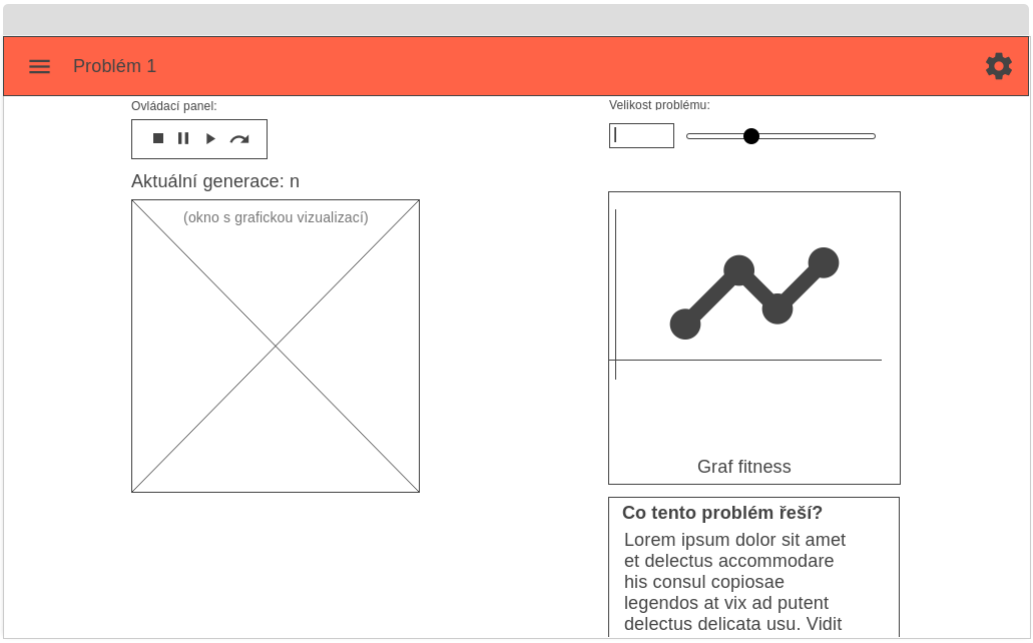


Obrázek 5.2 Wireframe s návrhem úvodní obrazovky

Stránku jsem navrhoval tak, aby byl nejdůležitější obsah na stránce vždy co nejvýše. U optimalizačních problémů je tedy nahoře komponent umožňující ovládání vizualizačního problému a změna parametru optimalizačního problému. Hned pod tímto ovládacím panelem a základním nastavením je společně s počítadlem generací umístěn i samotný vizualizační komponent, na němž je zobrazen stav optimalizačního problému. Vedle komponentu optimalizačního problému je vidět graf s vývojem fitness funkce dané úlohy. Na stránkách s optimalizačními problémy je navíc dostupný také pravý postranní panel, který slouží ke změně nastavení parametrů genetického algoritmu. Hlavní navigační panel i panel s nastavením lze skrýt pomocí tlačítek na horním panelu aplikace.

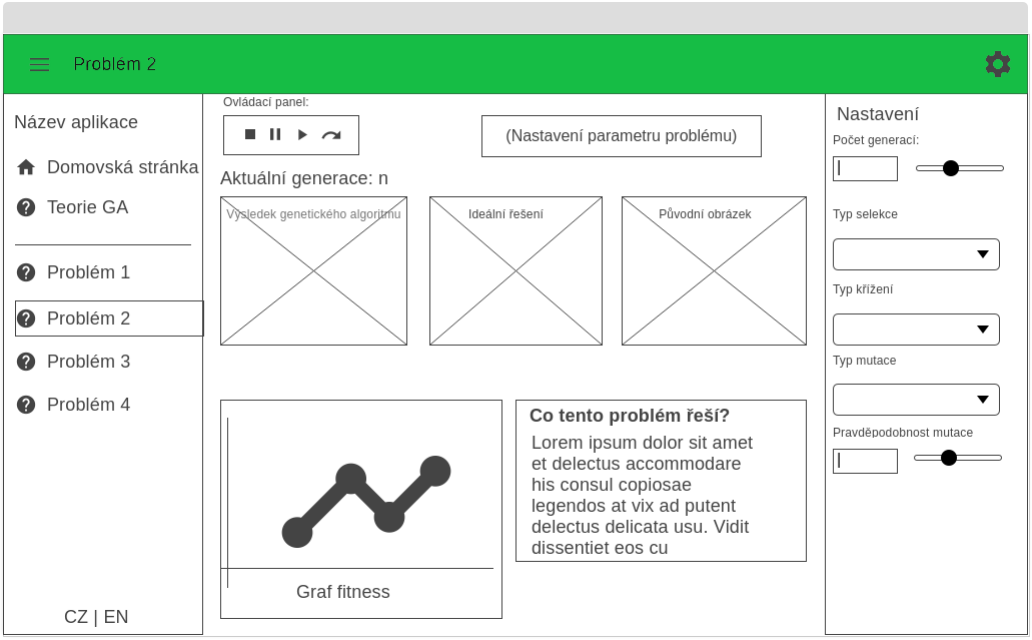


Obrázek 5.3 Wireframe s návrhem stránky optimalizačního problému s postranními panely



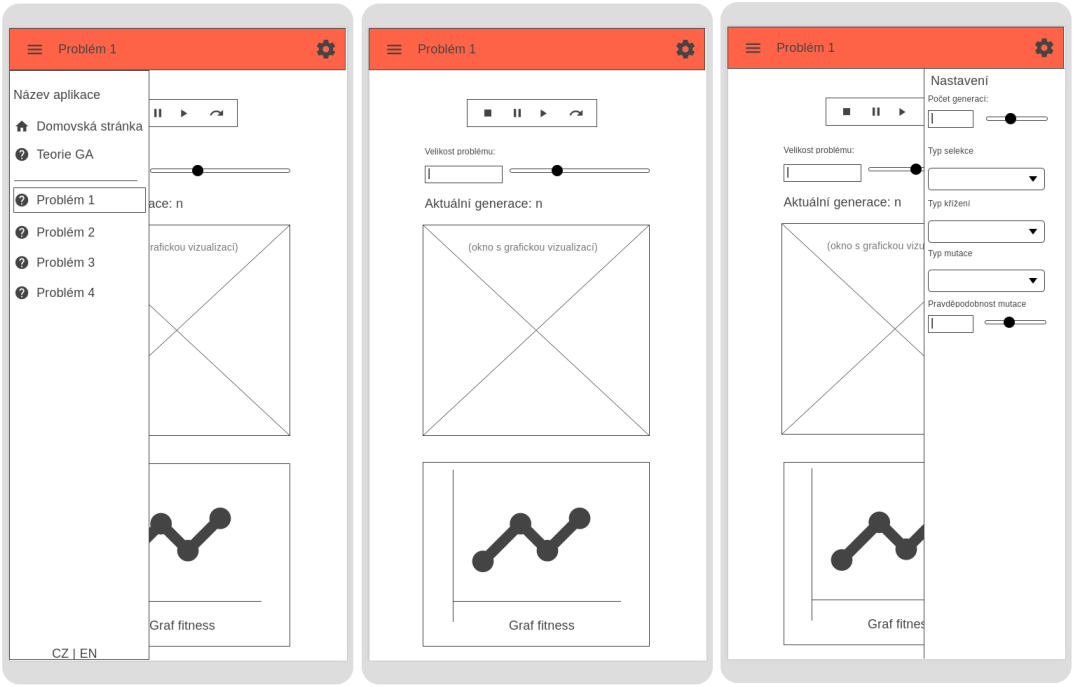
Obrázek 5.4 Wireframe s návrhem stránky optimalizačního problému bez postranních panelů

Vizualizace generování 2D bitmapové grafiky bude vyžadovat pro lepší názornost jiné rozložení, než ostatní úlohy. U této úlohy by nebyl vhodný jediný vizualizační komponent s aktuálním řešením. Místo toho bude stránka obsahovat navíc ukázku nejlepšího možného řešení hned vedle aktuálního řešení (pro lepší názornost rozdílu oproti požadovanému optimu) a původní obrázek.



Obrázek 5.5 Wireframy s návrhem rozložení stránky na mobilním telefonu

Uživatelské prostředí aplikace by se mělo přizpůsobovat vertikálně natočeným obrazovkám (například na mobilních telefonech). V takovém případě nebudou standardně vidět postranní vysouvací panely a rozložení stránky se bude řadit vertikálně pod sebe, aby se obrazovka co nejlépe využila.



Obrázek 5.6 Wireframe s návrhem stránky optimalizačního problému bez postranních panelů

Kapitola 6
Implementace

V této kapitole své práce se budu věnovat implementační části vývojového cyklu. Popíšu zde tedy pomocné knihovny, které jsem během vývoje využil k vytvoření všech důležitých součástí aplikace. Budu se zabývat také konkrétními implementacemi některých důležitých součásti projektu (například metoda výpočtu fitness ohodnocení u jednotlivých optimalizačních problémů).

Jak bylo zmíněno v podkapitole 4.3.3 *Shrnutí analýzy technologií a knihoven*, zvolil jsem pro implementaci tohoto projektu jazyk C# s frameworkem Blazor pro tvorbu single page aplikací, využívajícím technologii WebAssembly. Pro Blazor existuje velké množství knihoven, které mi pomohou s implementací důležitých součástí aplikace a s tvorbou uživatelského prostředí.

6.1 Použité knihovny

6.1.1 MudBlazor

MudBlazor [43] je knihovna pro tvorbu materiálního uživatelského prostředí. MudBlazor se řadí mezi nejpopulárnější Blazor knihovny pro tvorbu UI. Mezi další knihovny nabízející stejnou funkcionalitu se řadí například knihovna AntDesign [44] a Blazorise [45]. MudBlazor jsem mezi těmito alternativami zvolil především díky vzhledu komponentů a aktivní komunitě. MudBlazor výrazně usnadnil vývoj mé aplikace díky velké sadě předpřipravených UI komponentů, které je dále možné modifikovat pro konkrétní použití. MudBlazor má k dispozici velice názornou dokumentaci na svém webu, kde je možné jednotlivé komponenty vyzkoušet a případně modifikovat v jejich online editoru kódu. MudBlazor je dostupný pod licencí MIT [46].

Výpis kódu 6.1 Ukázka práce s komponenty v knihovně MudBlazor [47]

```
<MudAvatar Color="Color.Primary">M</MudAvatar>
<MudAvatar Color="Color.Secondary">U</MudAvatar>
<MudAvatar Color="Color.Tertiary">D</MudAvatar>
```

Obrázek 6.1 Snímek obrazovky zobrazující MudBlazor Avatar komponenty [47]

6.1.2 GeneticSharp

Pro zajištění fungování genetického algoritmu u všech optimalizačních úloh v aplikaci jsem využil knihovnu GeneticSharp [39]. Její autor má dokonce na své osobní stránce ukázkový projekt řešící problém obchodního cestujícího s technologií Blazor [48]. Tento ukázkový projekt mi v dalším vývoji pomohl lépe pochopit fungování knihovny GeneticSharp i technologie Blazor. Knihovna mi svou robustností výrazně ulehčila práci s genetickými algoritmy, naprostou většinu potřebné logiky totiž knihovna obsahuje. Knihovna je dostupná pod MIT licencí [49].

6.1.3 AKSoftware.Localization.MultiLanguages

Tato knihovna [50] umožňuje v projektu využít lokalizaci. Díky tomu lze pomocí tlačítka přepínat mezi jazyky, které aplikace podporuje, a změna se ihned dynamicky projeví. V případě prvního spuštění se aplikace pokusí zvolit nejvhodnější jazyk podle jazyku systému. Ke správnému fungování s technologií Blazor je navíc potřeba použít ještě přídavný rozšiřující balík *AKSoftware.Localization.MultiLanguages.Blazor* [51]. V případě Blazoru je pak použití této knihovny velmi snadné. Je potřeba si pouze připravit lokalizační soubory ve formátu *.yaml* a v C# kódu každé Razor stránky inicializovat komponent této knihovny, který využívá dependency injection. Knihovna je dostupná pod MIT licencí [52].

Výpis kódu 6.2 Inicializace knihovny AKSoftware.Localization.MultiLanguages

```
[Inject] protected ILanguageContainerService LanguageContainer { get; set; }
protected override void OnInitialized() {
        LanguageContainer.InitLocalizedComponent(this);
    }
```

6.1.4 Blazor.Extensions.Canvas

Jedná se o knihovnu implementující API pro HTML5 Canvas v technologii Blazor, umožňuje práci s 2D plátnem i WebGL [53]. Tato knihovna usnadňuje práci s 2D HTML5 plátnem tím, že lze využívat C# kód místo využívání JavaScriptu. Tuto knihovnu jsem použil pro vizualizaci problému obchodního cestujícího. Knihovna je dostupná pod MIT licencí [54].

6.1.5 Plotly.Blazor

Plotly.Blazor [55] je port velice populární knihovny *Plotly.js* [56]. Díky této knihovně lze vytvořit Razor komponent, který bude obsahovat požadovaný graf a měnit jeho vlastnosti v sekci s C# kódem dané stránky. Knihovnu jsem použil pro vykreslování grafu zobrazujícího vývoj fitness funkce v závislosti na dané generaci populace genetického algoritmu. Knihovna je dostupná pod MIT licencí [57].

6.1.6 SixLabors.ImageSharp

Tuto knihovnu [58] jsem použil pro zpracování bitmapové grafiky u problému *aproximace 2D obrazu*. Externí knihovnu bylo nutné použít z důvodu, že balík *System.Drawing.Common* je od uvedení .NET6 podporován pouze na platformě Windows. Microsoft přímo mezi doporučenými alternativami uvádí knihovnu *ImageSharp*. Pomocí této knihovny snižuji rozlišení výchozího obrázku pro výpočet fitness funkce jedinců řešící výše zmíněný problém. Knihovna je za splnění určitých podmínek použitelná pod Apache 2.0 licencí, v opačném případě (především pro velké komerční projekty) je nutné se řídit licencí *Six Labors Commercial License*[59].

6.1.7 Xunit

Knihovna *Xunit* je open-source nástroj pro unit testování .NET aplikací. Aplikaci využívám k otestování důležitých součástí kódu, především funkcí pro výpočet fitness jednotlivých problémů. Knihovna je dostupná pod Apache 2.0 licencí, některé součásti knihovny byly zveřejněny pod MIT licencí [57].

6.2 Komprese dat aplikace

Nevýhodou technologie WebAssembly pro tvorbu SPA je velký objem dat, které klient musí při prvním spuštění stáhnout. V případě mé aplikace se tento objem pohyboval okolo 38 MB. Takový objem dat už může způsobit pomalejší načítání aplikace u klientů s pomalým připojením k internetu. K řešení tohoto problému jsem využil možnost komprese zdrojových souborů, které klient stahuje. Podle oficiální dokumentace [60] lze využít *Brotli* komprese pro výrazné snížení objemu dat, která klient bude stahovat. Použití doporučeného skriptu [61] v Blazor projektu bylo velmi jednoduché a jeho implementace snížila v mém případě objem stažených dat na méně než 10 MB, což je již akceptovatelná hodnota.

6.3 Implementace optimalizačních problémů

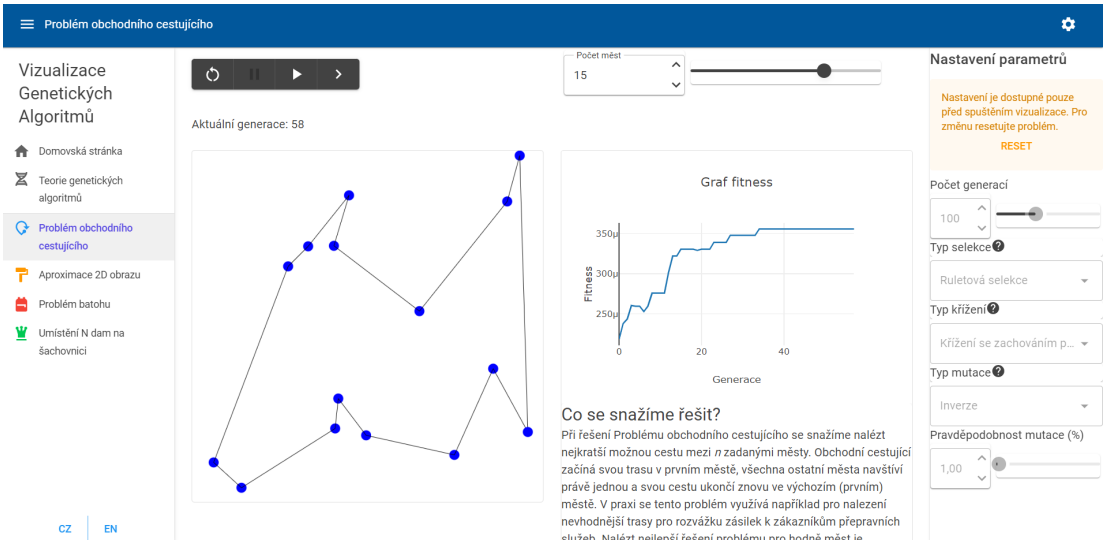
V této podkapitole rozeberu postup a metody, které jsem použil k implementaci jednotlivých optimalizačních úloh v interaktivním nástroji. U každého problému také popíšu algoritmus, kterým ohodnocuji jedince a určuji jejich fitness. Všechny stránky využívají komponent *MudGrid*, který umožňuje měnit rozvržení stránky podle velikosti a dimenzí obrazovky uživatelského zařízení. Díky tomu vypadají stránky vhodně i na mobilních zařízeních - jednotlivé komponenty na stránce se začnou řadit pod sebe místo vedle sebe, aby byly dobře čitelné na menší obrazovce.

Snímky obrazovky zachycující stránky aplikace v této kapitole jsou pořízené na nejnovější verzi aplikace v době odevzdání textu této práce.

6.3.1 Problém obchodního cestujícího

K vizualizaci problému obchodního cestujícího jsem využil 2D plátno, kde jsou pomocí modrých kruhů reprezentována města k navštívení. Čára, která je spojuje, pak představuje cestu, kterou obchodní cestující podnikne. Uživatel má u tohoto problému možnost měnit počet měst, která se na plátně zobrazí. Lze tak pozorovat, jak rychle se řešení optimalizuje v závislosti na počtu měst. U problému obchodního cestujícího je automaticky nastavena ruletová selekce, křížení se zachováním pořadí a mutace pomocí inverze. Nastavením těchto parametrů lze zpravidla dosáhnout dobrých výsledků. Uživatel si však může vyzkoušet i například kombinaci uniformního křížení a mutace, která většinou k dobrému řešení nekonverguje. Díky tomu je možné dobře vidět důsledky volby nevhodných parametrů GA na řešení problému.

K výpočtu fitness využívám algoritmus, který sečte všechny vzdálenosti mezi městy na zvolené trase obchodního cestujícího a tento součet umocní na mínus prvou. Se zvyšující celkovou vzdáleností tak klesá hodnota fitness ohodnocení. Chromozom je v tomto případě reprezentován celými kladnými čísly, která vyjadřují, které město ze seznamu bude navštíveno jako i-té podle indexu i. Čísla tvoří permutaci. Řešení, ve kterém se stejné číslo objevuje dvakrát, je nepřípustné a je penalizováno nulovým fitness ohodnocením. Po každém resetu problému se vygenerují nové náhodné pozice měst.



Obrázek 6.2 Snímek obrazovky s problémem obchodního cestujícího

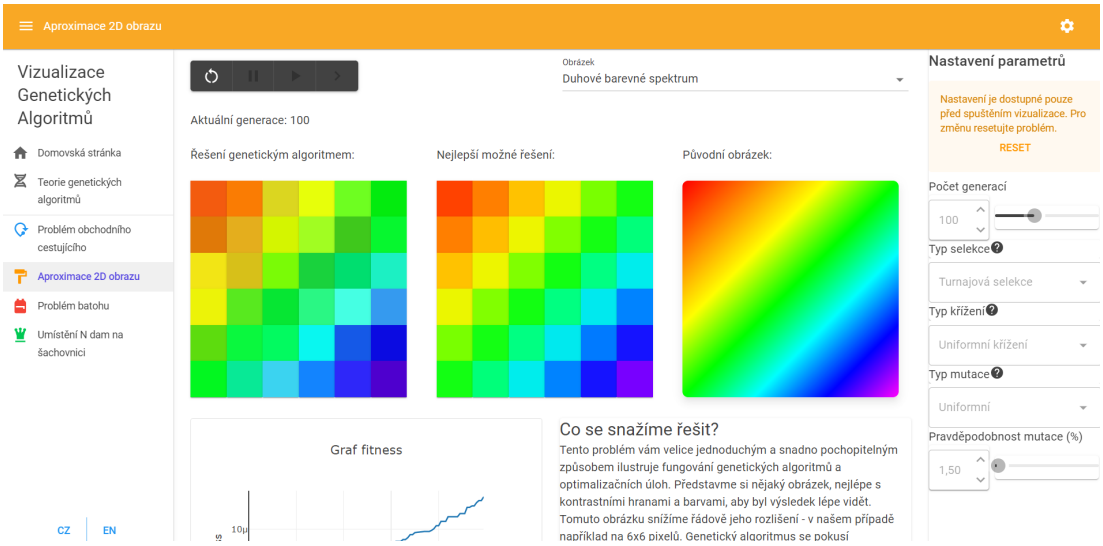
Výpis kódu 6.3 Výpočet fitness u problému obchodního cestujícího

```
public double Evaluate(IChromosome chromosome) {
        double fitnnes = 0;
        TspChromosome tspChromosome = (TspChromosome)chromosome;
        double totalDistance = 0;
        //previous city is set to the last city in the traverse sequence
        TspCity prevCity =
            Cities[(int)tspChromosome.GetGene(tspChromosome.Length -
            1).Value];
        for (int i = 0; i < tspChromosome.GetGenes().Length; i++) {
            TspCity currentCity =
                Cities[(int)tspChromosome.GetGene(i).Value];
            //calculate distance between the current and previous city
            totalDistance += currentCity.Distance(prevCity);
            //set the previous city to the current city for the next
                iteration
            prevCity = currentCity;
        }
        //chromosome with any duplicate numbers is invalid and its fitness
            equals to zero
        if (tspChromosome.DuplicateGenesCount() > 0) return 0;
        //edge case if all cities are on the same coordinates
        if (totalDistance == 0) return 1;
        fitnnes = 1 / totalDistance;
        return fitnnes;
    }
```

6.3.2 Aproximace 2D obrazu

Tato úloha se neřadí mezi typické optimalizační úlohy. Během průzkumu existujících řešení mě však zaujala vizualizace zmíněná v podkapitole 3.3 na stránce *Grow Your Own Picture*. Vizualizace obrázku se mi zdála vhodná, protože je na ní velmi jednoduše vidět postupná optimalizace řešení. Ve svém nástroji jsem se ale rozhodl pro ještě pochopitelnější přístup k podobné problematice. U této úlohy tedy dojde ke snížení rozlišení původního obrázku na velikost 6×6 pixelů. Tento rozměr mi přišel vhodný, protože chromozom, který bude obraz představovat, bude stále přijatelně dlouhý a zároveň při tomto rozlišení lze okem rozpoznat původní rysy originálního obrázku. Genetický algoritmus se u této úlohy snaží vytvořit obrázek stejného rozlišení (6×6 pixelů), který se bude původnímu obrázku co nejvíce podobat. Jedná se o implementaci, kterou jsem nikde na webu ani v žádné literatuře neobjevil. Jen jedna stránka, kterou jsem našel, se zabývala podobnou tématikou [62]. V tomto případě šlo ale o generování obrázku o velkém rozlišení, kde bylo nutné iterovat tisícemi generací pro dosažení přijatelného řešení. Musel jsem tedy pokusným testováním zjistit, jaká bude vhodná metoda výpočtu fitness a především, které parametry genetického algoritmu zajistí dobré řešení. Pro výpočet fitness jsem považoval za nejlepší sčítat rozdíl RGB barevných složek jednotlivých pixelů mezi řešením GA a původním obrázkem ve sníženém rozlišení. Abych penalizoval velké barevné rozdíly, využil jsem rozdíl kvadrátů. Tento součet jsem na konci umocnil na mínus prvou, s rostoucím barevným rozdílem tedy fitness klesá.

Pro dobrou reprezentaci jednotlivých pixelů, jsem musel do chromozomu zahrnout všechny tři barevné složky. Pro velikost obrázku $n \times n$ má tedy chromozom délku $n \times n \times 3$. V mém konkretním případě je tedy chromozom reprezentován 108 celými kladnými čísly z intervalu $[0; 255]$. Ke snížení rozlišení obrázku jsem využil knihovnu *ImageSharp*. Testováním jsem zjistil, že vhodná řešení nalézá kombinace turnajové selekce, uniformního křížení a uniformní mutace s pravděpodobností mutace kolem 1-1,5%. Výsledná vizualizace mě překvapila výborným fungováním a myslím, že je nejnázornější ukázkou využití GA v celém nástroji. Žádný jiný online nástroj dle mé analýzy takovou vizualizaci nenabízí. Uživatel si navíc jako parametr problému může zvolit jeden ze dvou různých obrázků k aproximaci. Na prvním z obrázků je fotka přírody s kontrastní zelenou loukou a stromem. Druhý obrázek je abstraktnější - jedná se o duhové barevné spektrum. Myslím ale že právě díky širokému spektru barev je na druhém obrázku vidět optimalizace řešení nejlépe.



Obrázek 6.3 Snímek obrazovky s problémem aproximace 2D obrázku

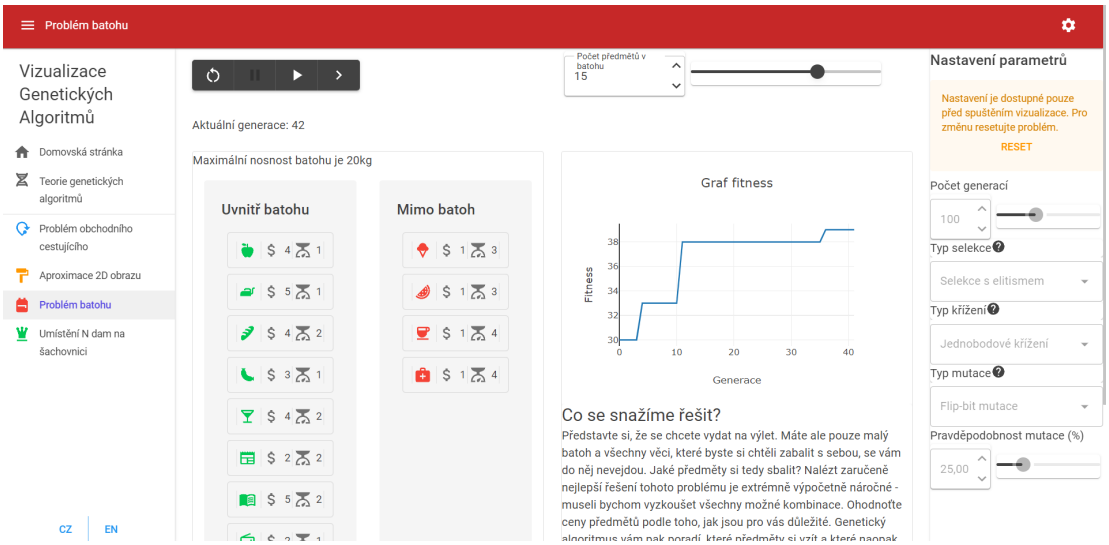
Výpis kódu 6.4 Výpočet fitness u problému aproximace 2D obrazu

```
public double Evaluate(IChromosome chromosome) {
    if (_imageColorsAverages.Length * 3 != chromosome.Length)
        throw new ConstraintException(
            "The average colors array must have the same length as the
                chromosome to be evaluated!");
    double sum = 0.0;
    for (int i = 0; i < chromosome.Length; i++) {
        int currentColor = (int)chromosome.GetGene(i).Value;
        sum += CalculateColorDifference(currentColor,
            _imageColorsAverages[i / 3], i);
    }

    if (sum == 0) return 1;
    double fitness = 1.0 / sum;
    return fitness;
}

private double CalculateColorDifference(int color1, Color color2, int
    index) {
    if (index % 3 == 0) return Math.Pow(color1 - color2.R, 2);
    if (index % 3 == 1) return Math.Pow(color1 - color2.G, 2);
    return Math.Pow(color1 - color2.B, 2);
}
```

6.3.3 Problém batohu

U vizualizace problému batohu bylo nutné graficky interpretovat předměty, které se nachází uvnitř a které vně batohu. Využil jsem databázi ikon knihovny MudBlazor a připravil jsem velký seznam ikon běžných předmětů, které by bylo logické si za určité situace sbalit do batohu. Vizualizační komponent má dvě části - v jedné jsou předměty mimo batoh a v druhé jsou předměty uvnitř batohu. Umístění předmětů je i graficky rozlišené barvou ikon. Velikost batohu je v případě mé implementace 20 kilogramů. Při generování váhy předmětů se zohledňuje jejich počet, tedy délka chromozomu. Fitness funkce je rovna součtu cen (hodnot) předmětů, které jsou umístěny do batohu. Pokud je ale součet vah předmětů u konkrétního řešení vyšší, než jaká je kapacita batohu, je řešení ohodnoceno nulovým fitness. Řešení, která by totiž přeplnila batoh považujeme za nevalidní.

U problému batohu jsem využil klasické reprezentace řešení pomocí binárního vektoru. Každý gen představuje stav (umístění) jednoho předmětu. Pokud je hodnota rovna jedné, je předmět umístěn do batohu, pokud je hodnota rovna nule, předmět je mimo batoh. U binárního chromozomu je možné využít i *flip-bit* mutaci. Tu jsem u tohoto problému tedy nastavil jako výchozí možnost. Dále je zde ve výchozím nastavení připravena selekce s elitismem a jednobodové křížení.



Obrázek 6.4 Snímek obrazovky s problémem batohu

Výpis kódu 6.5 Výpočet fitness u problému batohu

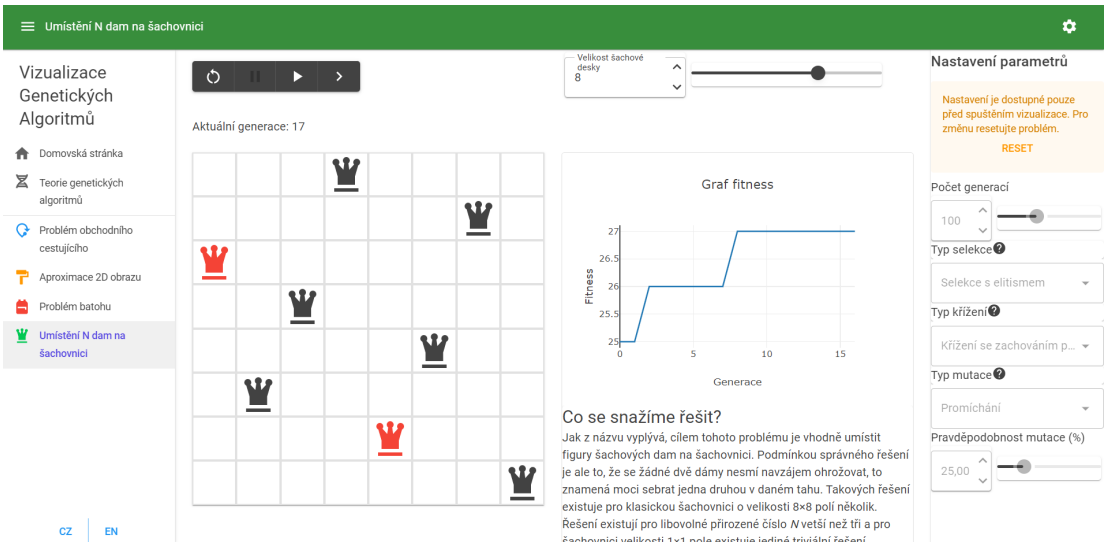
```
        public double Evaluate(IChromosome chromosome) {
        //check that weights and prices have the same length as chromosome
        if (ItemPrices.Length != chromosome.Length || ItemWeights.Length !=
            chromosome.Length)
            throw new ConstraintException(
                "The item prices/weights array must have the same length as
                    the evaluated chromosome!");
        double fitness = 0;
        double pricesSum = 0;
        double weightsSum = 0;
        //iterate over all items
        //sum the weights of items to be added
        //sum the prices of itmes to be added
        for (int i = 0; i < chromosome.Length; i++) {
            int current = (int)chromosome.GetGene(i).Value;
            if (current != 0  current != 1)
                throw new ConstraintException("The binary chromosome can
                    only contain 1 or 0 values!");
            if (current == 1) {
                pricesSum += ItemPrices[i];
                weightsSum += ItemWeights[i];
            }
        }
        if (weightsSum > KnapsackSize) fitness = 0;
        else fitness = pricesSum;
        return fitness;
    }
```

6.3.4 Problém umístění N dam na šachovnici

U problému umístění N dam na šachovnici bylo nutné vytvořit komponent, který bude reprezentovat právě šachovou desku. Tento komponent, je hlavním prvkem celé vizualizace. Pro zobrazení figur dam jsem využil jednu z dostupných ikon, které jsou součástí knihovny MudBlazor. U tohoto problému jsem chtěl umožnit uživateli měnit velikost šachovnice, aby bylo možné si vyzkoušet, že i u šachovnic o větším rozměru $n \times n$ je vždy opravdu možné n různých dam umístit se splněním podmínky nulových kolizí. Problémem ale byla dynamická změna vzhledu šachovnice podle nastavení jejího rozměru. Ani ukázkové řešení šachovnice pomocí *MudDropZone*[63] komponentu na oficiální stránce s dokumentací knihovny se nebylo schopné přizpůsobit užším obrazovkám a šachovnice byla deformována. Nakonec se mi ale podařilo tento problém překonat a šachovnice se tak dynamicky překresluje a přizpůsobuje obrazovce.

Chromozom pro tento problém je reprezentován celými kladnými čísly. Každý gen vyjadřuje pozici i-té dámy (konkrétně specifikuje v jakém sloupci bude dáma umístěna). Index genu pak vyjadřuje, na jakém řádku bude dáma umístěna. Díky této reprezentaci zajistíme, že žádné dvě šachové figury nebudou na stejném řádku. Pokud navíc použije uživatel metodu křížení a mutace, která zachová u chromozomů vlastnost permutace, nebudou nikdy 2 šachové figury ani ve stejném sloupci. Při náhodném generování chromozomu se totiž generují permutace čísel a žádné číslo se v chromozomu nevyskytuje dvakrát.

První důležitou částí algoritmu pro výpočet fitness bylo zjišťování počtu kolizí mezi dámami. Pro zjištění, zda dochází ke kolizi dvou dam, jsem využil faktu, že ke kolizi vzhledem k chování šachové dámy dochází právě v případě, když jsou dvě dámy na stejném řádku, ve stejném sloupci nebo je rozdíl jejich pozic v řádcích stejný jako rozdíl pozic ve sloupcích (ohrožení diagonálně). Tento algoritmus jsem využil ve funkci pro výpočet fitness ohodnocení. U problému dam nezáleží na tom, jestli každou kolizi započítáme dvakrát (z pohledu každé figury), nebo pouze jednou, já ve své implementaci počítám kolize pouze jednou. Finální hodnota fitness ohodnocení je pak dána rozdílem maximálního počtu kolizí, které mohou nastat, a skutečným počtem kolizí. Pro šachovnici velikosti n může nastat až $\sum_{i=1}^{n-1} i$ kolizí.



Obrázek 6.5 Snímek obrazovky s problémem umístění N dam na šachovnici

Výpis kódu 6.6 Výpočet fitness u problému umístění N dam na šachovnici

```
public double Evaluate(IChromosome chromosome) {
    Gene[] genes = chromosome.GetGenes();
    int[] queensPositions = new int[chromosome.Length];
    int collisions = 0;
    int maxPossibleCollisions = 0;
    //get values from the genes of the chromosome
    for (int i = 0; i < genes.Length; i++) {
        queensPositions[i] = (int) genes[i].Value;
        maxPossibleCollisions += i;
    }
    //check every combination of queen pairs
    for (int i = 0; i < queensPositions.Length; i++) {
        for (int j = i + 1; j < queensPositions.Length; j++) {
            int firstRow = i;
            int firstCol = queensPositions[i];
            int secondRow = j;
            int secondCol = queensPositions[j];
            //add the collision to the counter if found
            if (firstRow == secondRow) collisions++;
            if (firstCol == secondCol) collisions++;
            if (Math.Abs(firstRow - secondRow) == Math.Abs(firstCol -
                secondCol)) collisions++;
        }
    }
    return maxPossibleCollisions - collisions;
}
```

6.4 Implementace dalších částí aplikace

Kromě vizualizací jednotlivých optimalizačních úloh je na stránce dostupná také úvodní stránka a stránka s popisem fungování genetických algoritmů.

6.4.1 Úvodní stránka

Tato stránka se uživateli zobrazí hned po otevření aplikace. Je na ní dostupné navigační menu a stručný popis všech problémů s hypertextovými odkazy na dané stránky. Popisy jsou záměrně psané srozumitelnou formou, aby byl princip každého problému pochopitelný i neodbornému uživateli. Každý problém má u svého popisu velkou ikonu ve stejné barvě, jako horní panel na stránce konkrétního problému.

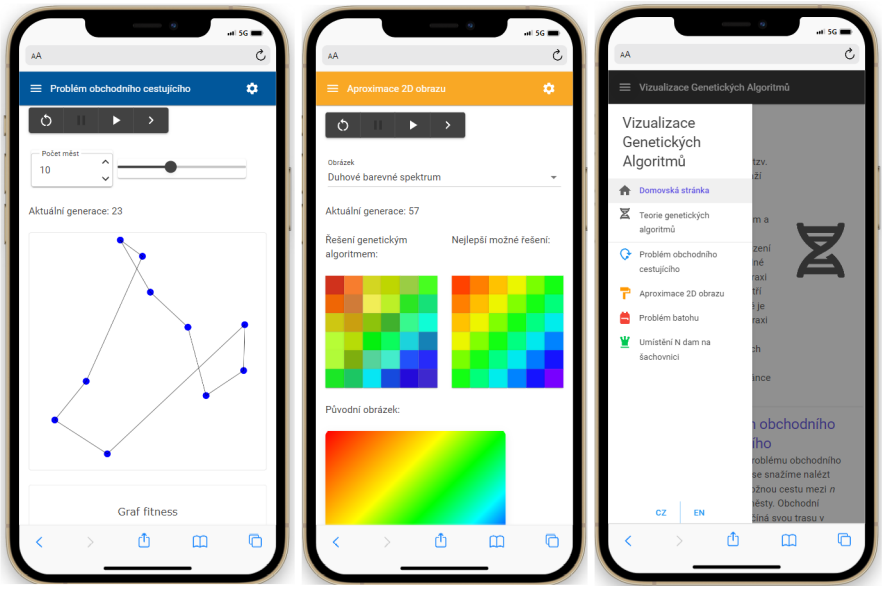
K rozložení prvků na stránce byl opět využit komponent *MudGrid* pro optimalizaci na různé velikosti obrazovek. Stránka neobsahuje žádné složité dynamické prvky, jedná se o standardní statickou stránku.

6.4.2 Stránka s teorií genetických algoritmů

Poslední důležitou součástí aplikace je stránka s delším popisem fungování genetických algoritmů. Popis je psán zjednodušeně, je spíše cílený na neodborné uživatele, kteří problematice nerozumí. Odborných textů pro akademické uživatele existuje mnoho a jsou dobře dostupné. Předpokládám, že popis obecného fungování na této stránce bude vyhledávat právě spíš neodborný uživatel a tomu je celý text přizpůsoben. Uživatel je na konci textu upozorněn, že se nejedná o odborný text, popisující genetické algoritmy exaktně, aby nedošlo k případnému zmatení. Uživatel je vyzván k dalšímu prostudování problematiky na stránce Wikipedia, která může být dalším, odbornějším, krokem pro studium GA pro laického uživatele.

6.5 Optimalizace na mobilní zařízení

Aplikace byla navrhována s myšlenkou dostupnosti i na mobilních zařízeních. Design nástroje je tedy přizpůsoben malým vertikálním obrazovkám. Uspořádání a velikost grafických komponentů se na mobilních zařízeních uzpůsobí rozměrům obrazovky. MudBlazor knihovna pro tyto účely umožňuje měnit rozvržení komponentů stránky dle velikosti obrazovky v pixelech.



■ **Obrázek 6.6** Snímky obrazovky aplikace s rozvržením pro mobilní telefony

Kapitola 7
Testování

Vzhledem k většímu rozsahu aplikace bylo nutné provést její testování. Aplikace obsahuje sadu automatických testů a byla otestována také uživateli.

7.1 Unit testy

Část aplikace byla otestována sadou unit testů. Testovací sada obsahuje desítky testů, které kontrolují standardní scénáře, ale i případy s mezními hodnotami nebo neplatnými vstupy. Pomocí unit testů jsem otestoval především třídy zajišťující logiku chromozomů, fitness, modelů a částečně controllerů. Pro Razor komponenty existuje podle [64] také nástroj (*bUnit*) na jejich testování. Komponenty a stránky jsem ale do automatického testování nezahrnul. Místo toho byly tyto součásti aplikace otestovány během uživatelského testování. Vyhodnocení unit testů je součástí GitLab skriptu, který se spustí při každém nahrání projektu do vzdáleného repositáře. Stejně tak dojde v rámci skriptu průběžné integrace k sestavení celého projektu.

7.2 Uživatelské testování

Pro otestování funkčnosti aplikace jako celku, chování jednotlivých interaktivních vizualizací a uživatelské přívětivosti jsem využil metody uživatelského testování. Pro tento účel jsem připravil důležité body testovacího scénáře:

7.2.1 Testovací scénář

Uživatel by měl během testování provést následující úkony:

- Otevřít stránku na preferovaném zařízení (telefon / počítač).
- Pročíst si text na úvodní stránce, zhodnotit jeho srozumitelnost.
- Seznámit se s problematikou genetických algoritmů (pokud uživatel nemá předchozí zkušenost) na stránce s úvodem do jejich teorie, ohodnotit přínos tohoto textu.
- Vyzkoušet všechny optimalizační úlohy, seznámit se s jejich ovládáním a nastavením.
- Změnit u některých úloh jejich parametr a parametry genetického algoritmu.

Časový limit testu nebyl nijak omezen, protože cílem bylo, aby si uživatel opravdu v klidu vyzkoušel měnit nastavení parametrů a s nástrojem se beze spěchu důkladně seznámil. Během testování mohl uživatel popisovat případné nedostatky, které ovlivňovaly jeho uživatelský zážitek. Mimo to bylo cílem do testování z mé strany co nejméně zasahovat. Uživatelé byli ujištěni, že žádné odpovědi ani názory nejsou špatné.

Cílem bylo mít mezi respondenty co nejširší zastoupení různých skupin - věkových i zájmových. Cílovou skupinou aplikace jsou totiž odborní i neodborní uživatelé všech věkových kategorií. Vybral jsem respondenty ze svého okolí, kteří reprezentovali co nejpestřejší vzorek. Po dokončení uživatelského testování následovala krátká diskuze s každým uživatelem, kde jsme probrali případné připomínky a další zpětnou vazbu k aplikaci.

Na konci uživatelského testování jsem položil uživateli několik doplňujících otázek:

- Měl jste pocit, že vás popis implementovaných problémů dostatečně uvedl do jejich problematiky?
- Objasnila vám stránka se stručným shrnutím teorie genetických algoritmů jejich fungování?
- Porozuměl jste vizualizaci jednotlivých problémů? Byla dostatečně názorná?
- Bylo pro vás uživatelské prostředí přehledné? Dokázal jste se v aplikaci snadno orientovat?
- Myslíte, že je obsah aplikace užitečný pro studium genetických algoritmů?
- Co se vám v aplikaci nejvíce líbilo, která součást vám přišla užitečná?
- Máte k aplikaci nějaké další připomínky?

Respondenti reagovali na první čtyři otázky ohodnocením na stupnici od 1 do 10 (s případným slovním rozvinutím a odůvodněním). Díky tomu jsem získal měřitelné odpovědi, uživatelé pak své hodnocení dále slovně rozvedli.

7.2.2 Výsledky uživatelských testů

Uživatelského testování se zúčastnilo celkem pět respondentů, v následujícím výčtu uvedu odpovědi jednotlivých dobrovolných respondentů.

7.2.2.1 Respondent 1

- Věková kategorie: méně než 30 let.
- Zaměstnání: student.
- Předchozí znalost GA: Ano.
- Platforma: počítač.
- Hodnocení úvodní stránky a popisu úloh: 10/10.
- Hodnocení stránky s teorií GA: *nehodnoceno.*
- Názornost vizualizačních úloh: 9/10.
- Přehlednost uživatelského prostředí: 9/10.
- Užitečnost, použitelnost aplikace: 10/10.

Respondent byl s prototypem stránky spokojený, ocenil popis optimalizačních úloh na úvodní stránce a možnost pokročilé modifikace parametrů genetického algoritmu. Uživatel postřehl několik překlepů v textech a doporučil graficky zdůraznit nemožnost měnit parametry GA během procesu vizualizace, případně přidat vysvětlení jednotlivých typů operátorů v panelu nastavení.

7.2.2.2 Respondent 2

- Věková kategorie: méně než 30 let.
- Zaměstnání: student.
- Předchozí znalost GA: Ne.
- Platforma: počítač.
- Hodnocení úvodní stránky a popisu úloh: 9/10.
- Hodnocení stránky s teorií GA: 7/10.
- Názornost vizualizačních úloh: 10/10.
- Přehlednost uživatelského prostředí: 10/10.
- Užitečnost, použitelnost aplikace: 10/10.

Respondent ocenil popis úloh na úvodní obrazovce, jako nejzajímavější považoval vizualizaci problému obchodního cestujícího. Také všechny ostatní úlohy považoval za velmi názorné. Uživatel postřehl drobnou chybu v lokalizaci, doporučil přeformulovat některé části popisu teorie GA.

7.2.2.3 Respondent 3

- Věková kategorie: více než 30 let.
- Zaměstnání: pracující.
- Předchozí znalost GA: Ne.
- Platforma: počítač.
- Hodnocení úvodní stránky a popisu úloh: 10/10.
- Hodnocení stránky s teorií GA: 7/10.
- Názornost vizualizačních úloh: 10/10.
- Přehlednost uživatelského prostředí: 8/10.
- Užitečnost, použitelnost aplikace: 10/10.

Třetí respondent velmi chválil popis úloh na úvodní stránce, stejně tak podle něj byly všechny vizualizační úlohy velmi názorné a pochopitelné. Uživatel ocenil vzhled a chování komponentu s grafem pro vývoj fitness funkce. Uživatel navrhl přidat formátování textu s teorií GA, zdůraznit nemožnost měnit parametry GA před resetováním vizualizace a zvětšit tlačítka ovládacího panelu.

7.2.2.4 Respondent 4

- Věková kategorie: méně než 30 let.
- Zaměstnání: pracující.
- Předchozí znalost GA: Ne.
- Platforma: mobilní telefon.
- Hodnocení úvodní stránky a popisu úloh: 8/10.

- Hodnocení stránky s teorií GA: 7/10.
- Názornost vizualizačních úloh: 10/10.
- Přehlednost uživatelského prostředí: 10/10.
- Užitečnost, použitelnost aplikace: 9/10.

Respondent ohodnotil digram na stránce s teorií GA jako velmi přínosný, obsah stránky označil jako dobrý pro úvod do problematiky. Stejně tak hodnotil kladně texty na úvodní stránce. Vytkl pouze několik překlepů a navrhl při případném dalším vývoji aplikace umožnit změnu ceny předmětů u problému batohu. Uživatel zhodnotil uživatelský zážitek na mobilním telefonu jako velmi dobrý.

7.2.2.5 Respondent 5

- Věková kategorie: více než 30 let.
- Zaměstnání: pracující.
- Předchozí znalost GA: Ne.
- Platforma: počítač.
- Hodnocení úvodní stránky a popisu úloh: 9/10.
- Hodnocení stránky s teorií GA: 7/10.
- Názornost vizualizačních úloh: 10/10.
- Přehlednost uživatelského prostředí: 8/10.
- Užitečnost, použitelnost aplikace: 10/10.

Respondent ocenil pestrý barevný design aplikace. Jako snadno pochopitelné vizualizace označil problém obchodního cestujícího a problém batohu. Uživatel postřehl drobné stylistické vady textů a doporučil zdůraznit nemožnost měnit parametry GA během vizualizace.

7.2.3 Shrnutí odpovědí respondentů a provedené změny

Z výsledků uživatelského testování plyne, že úvodní stránka s popisem optimalizačních úloh plní svůj účel velice dobře. Průměrné hodnocení této stránky bylo 9,2/10. Všichni uživatelé si všimli několika překlepů v textech na této stránce. Vzhledem k prezentaci stejné verze aplikace všem uživatelům se tato výtka opakovala. Překlepy na úvodní stránce jsem opravil a provedl jsem důslednou kontrolu gramatiky a stylistiky nejen na této stránce, ale ve všech textech aplikace. Uživatelé byli jinak s úvodní stránkou velmi spokojení.

Stránka s detailnějším popisem teorie GA dosáhla během uživatelského testování nejnižšího hodnocení - 7/10. Uživatelé vytkli horší srozumitelnost některých částí a drobné překlepy. Text na této stránce jsem na základě zpětné vazby přepracoval. Zavedl jsem členění textu na odstavce, zvýraznil jsem klíčová slova a přeformuloval jsem problematické části. Některé části textu jsem více rozvedl, u jiných jsem naopak výklad zkrátil.

Samotné vizualizační komponenty dosáhly průměrného hodnocení 9,8/10. Uživatelé je hodnotili jako velmi srozumitelné a názorné. Vizualizace optimalizačních úloh jsou stěžejní součástí aplikace. Jejich jednoznačně kladné hodnocení vypovídá o vhodném návrhu těchto částí.

Uživatelské prostředí bylo průměrně hodnoceno známkou 9/10. Mezi doporučeními pro lepší uživatelský zážitek byl zmíněn návrh na zvětšení ovládacího panelu optimalizačních úloh. Tuto

změnu jsem provedl a panel jsem přesunul více do levého horního roku. Při přesunu myši na ovládací panel se zobrazí popis funkce jednotlivých tlačítek pro lepší názornost. Uživatelé také zmínili, že není zcela zřejmé, kdy je možné provádět změny parametrů GA na pravém postranním panelu. Přidal jsem proto výrazné upozornění do horní části panelu, které obsahuje i tlačítko pro reset úlohy. Na panel s nastavením jsem přidal možnost zobrazit nápovědu se stručným popisem jednotlivých implementovaných metod selekce, křížení a mutace.

Všichni uživatelé se shodli, že aplikace má dobrý potenciál pro potřeby výuky a lepšího pochopení fungování GA. Použitelnost aplikace pro edukační účely byla ohodnocena průměrnou známkou 9,8/10. Pro lepší použitelnost a pochopitelnost byly na pravý panel s nastavením přidány nápovědy se stručným popisem operátorů GA. Uživatel si tak může připomenout rozdíl mezi jednotlivými typy operátorů.

Obecně hodnotím výsledky uživatelského testování jako velmi pozitivní. Dokladem dobrého návrhu a robustnosti aplikace byl fakt, že většina připomínek se týkala pouze drobných překlepů, formulace odbornější části textu a připomínek týkajících se lepšího uživatelského zážitku a intuitivnosti aplikace. Všechny tyto připomínky jsem zapracoval do finální verze aplikace. Implementace těchto vylepšení nebyla příliš náročná, protože se jednalo pouze o drobné grafické změny. Vzhledem k pozitivním uživatelským ohlasům považuji aplikaci po implementaci těchto drobných zlepšení za hotovou a způsobilou k poskytnutí všem uživatelům.

Kapitola 8
Dokumentace

Jednou ze závěrečných fází vývojového cyklu je i dokumentace hotového projektu. V této kapitole se budu věnovat právě způsobu, kterým jsem projekt zdokumentoval. Dokumentace rozsáhlého projektu může výrazně usnadnit pochopení jeho fungování a umožnit tak například na projekt navázat dalším vývojářům.

8.1 Doxygen

Doxygen [65] je nástroj pro automatické generování programátorské dokumentace z komentářů zdrojového kódu. Dokumentaci lze vyexportovat jako HTML stránky nebo jako PDF dokument. Právě dokumentace v jednom PDF souboru je vhodnou volbou v mém případě. Pro správné vygenerování plnohodnotné dokumentace je nutné kód okomentovat.

Výpis kódu 8.1 Příklad komentáře ve zdrojovém kódu pro Doxygen dokumentaci

```
/// <summary>
/// Adds new value to the chart
/// </summary>
/// <param name="value">Specifies the value to be added</param>
public void AddValue(double value) {
    _scatter.Y.Add(value);
    _chart.Update();
}
```

V dokumentaci jsem vygeneroval i grafické znázornění závislostí a dědičností - dokumentaci tak lze použít případně i jako diagram tříd. Vygenerovaná dokumentace je přiložena jako příloha této práce.

Závěr

Cílem této práce bylo vytvořit funkční webovou interaktivní aplikaci pro vizualizaci fungování genetických algoritmů.

Prvním krokem, který jsem provedl, bylo seznámení se s problematikou genetických algoritmů a evolučních algoritmů. Tuto problematiku jsem popsal v první kapitole, která obsahovala teoretický úvod do dané problematiky. Během tvorby teoretické kapitoly jsem popsal i některé klasické optimalizační úlohy, které se pomocí genetických algoritmů typicky řeší.

Dále jsem srovnal existující dostupná řešení. Z analýzy těchto řešení jsem zjistil, jaké nedostatky se u stávajících aplikací objevují. Tyto poznatky jsem využil při analýze požadavků na mnou vyvíjený nástroj. Součástí mé práce byl i proces výběru vhodné technologie pro implementaci nástroje. Důležitým požadavkem pro výběr technologie byla i dostupnost vhodné knihovny pro práci s genetickými algoritmy.

Po výběru technologie jsem navrhl architekturu aplikace a vytvořil wireframy zobrazující návrh uživatelského prostředí. Aplikaci jsem implementoval ve frameworku Blazor s využitím technologie WebAssembly. Tuto volbu jsem učinil na základě předem provedené analýzy. K hotové aplikaci jsem vygeneroval programátorskou dokumentaci a aplikaci jsem otestoval kombinací automaticky spustitelných unit testů a uživatelského testování. Výsledkem mé práce tedy je hotový softwarový produkt připravený k použití uživateli.

Myslím, že použití technologie Blazor bylo vhodnou volbou. Implementaci usnadnila široká nabídka dostupných knihoven. Za nejnázornější z optimalizačních problému v aplikaci považuji problém generování 2D obrazu. Jedná se totiž o problém, který jiné nástroje neimplementují, ačkoliv je podle mého názoru jeho vizualizace velmi názorná a pro uživatele velmi přínosná.

Bibliografie

1. ZELINKA, Ivan; OPLATKOVÁ, Zuzana; ŠEDA, Miloš; OŠMERA, Pavel; VČELAŘ, František. *Evoluční výpočetní techniky - principy a aplikace*. First. BEN - technická literatura, 2009. ISBN 80-7300-218-3.
2. SLOWIK, Adam; KWASNICKA, Halina. Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications*. 2020, roč. 32. ISSN 1433-3058. Dostupné z DOI: 10.1007/s00521-020-04832-8.
3. EIBEN, A.E.; SMITH, J.E. Introduction to Evolutionary Computing. In: Springer Berlin, Heidelberg, 2015. ISBN 978-3-662-44874-8. Dostupné z DOI: 10.1007/978-3-662-44874-8.
4. HYNEK, Josef. *Genetické algoritmy a genetické programování*. First. Grada Publishing, 2008. ISBN 978-80-247-2695-3.
5. ŘEHOŘEK, Tomáš. *Evoluční výpočetní techniky, Genetický algoritmus, Genetické programování* [online]. [B.r.]. [cit. 2023-04-20]. Dostupné z: https://courses.fit.cvut.cz/BI-ZUM/media/lectures/04-evolution-v5.0.pdf.
6. DALTON, John. *January 2007: Genetic Algorithms (GAs)* [online]. [cit. 2023-05-05]. Dostupné z: http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php.
7. TUTORIALSPOINT. *Genetic Algorithms - Crossover* [online]. [cit. 2023-05-05]. Dostupné z: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm.
8. RIFKI, Omar; ONO, Hirotaka. A survey of computational approaches to portfolio optimization by genetic algorithms. In: 2012. Dostupné z DOI: 10.13140/RG.2.1.5192.8561.
9. MATAI, Rajesh; SINGH, Surya; MITTAL, Murari Lal. Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches. In: DAVENDRA, Donald (ed.). *Traveling Salesman Problem*. Rijeka: IntechOpen, 2010, kap. 1. Dostupné z DOI: 10.5772/12909.
10. *The Traveling Salesman Problem (TSP)* [online]. [cit. 2023-05-01]. Dostupné z: https://www2.seas.gwu.edu/~simhaweb/champalg/tsp/tsp.html.
11. BADIRU, Kayode. *Knapsack Problems; Methods, Models and Applications* [online]. 2009. [cit. 2023-05-04]. Dostupné z: https://shareok.org/handle/11244/319274.
12. TERH, Fabian. *How to solve the Knapsack Problem with dynamic programming* [online]. [cit. 2023-05-01]. Dostupné z: https://medium.com/@fabianterh/how-to-solve-the-knapsack-problem-with-dynamic-programming-eb88c706d3cf.
13. *The N-queens Problem* [online]. [cit. 2023-05-01]. Dostupné z: https://developers.google.com/optimization/cp/queens.

14. MATSUNAGA, Rafael; CATTO, Erin; JOHNSON, Leif. *Genetic Algorithm Walkers* [online]. [cit. 2023-04-20]. Dostupné z: https://rednuht.org/genetic_walkers/.
15. ECK, David J. *Genetic Algorithm Walkers* [online]. [cit. 2023-04-20]. Dostupné z: https://math.hws.edu/eck/js/genetic-algorithm/GA.html.
16. CUMMINS, Chris. *Grow Your Own Picture* [online]. [cit. 2023-04-21]. Dostupné z: https://chriscummins.cc/s/genetics/.
17. HALSEY, Lee. *blazor.ai* [online]. [cit. 2023-04-21]. Dostupné z: https://www.blazor.ai/.
18. UPADHYAY, Anu. *How to Choose a Programming Language For a Project?* [Online]. [cit. 2023-04-22]. Dostupné z: https://www.geeksforgeeks.org/how-to-choose-a-programming-language-for-a-project/.
19. STACK OVERFLOW. *Stack Overflow Developer Survey 2022* [online]. [cit. 2023-04-22]. Dostupné z: https://survey.stackoverflow.co/2022/#technology-most-popular-technologies.
20. *JavaScript Frameworks: What Are They and How Do They Work?* [Online]. [cit. 2023-04-22]. Dostupné z: https://www.simplilearn.com/javascript-frameworks-what-are-they-how-do-they-work-article.
21. BYRNE, Michael. *The Rise and Fall of the Java Applet: Creative Coding's Awkward Little Square* [online]. [cit. 2023-05-05]. Dostupné z: https://www.vice.com/en/article/8q8n3k/a-brief-history-of-the-java-applet.
22. *What is a Java applet?* [Online]. [cit. 2023-05-05]. Dostupné z: https://www.educative.io/answers/what-is-a-java-applet.
23. TOAL, Rory. *What is Python used for? 7 Practical Uses* [online]. [cit. 2023-04-22]. Dostupné z: https://codeinstitute.net/global/blog/what-is-python-used-for/.
24. DJANGO SOFTWARE FOUNDATION. *Django*. 2023. Ver. 2.2. Dostupné také z: https://djangoproject.com.
25. TRIVELLATO, Marco. *WebAssembly is here!* [Online]. [cit. 2023-04-23]. Dostupné z: https://blog.unity.com/technology/webassembly-is-here.
26. UNITY TECHNOLOGIES. *PlayerSettings.WebGL.threadsSupport* [online]. [cit. 2023-04-23]. Dostupné z: https://docs.unity3d.com/ScriptReference/PlayerSettings.WebGL-threadsSupport.html.
27. LATHAM, Luke; ANDERSON, Rick et al. *ASP.NET Core Blazor* [online]. [cit. 2023-04-22]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-7.0.
28. LATHAM, Luke; SCOPEL, Fabio; ANDERSON, Rick et al. *ASP.NET Core Blazor* [online]. [cit. 2023-04-23]. Dostupné z: https://learn.microsoft.com/cs-cz/aspnet/core/blazor/?view=aspnetcore-7.0.
29. LATHAM, Luke; ANDERSON, Rick; CHINGONO, Alfero et al. *Modely hostování ASP.NET Core Blazor* [online]. [cit. 2023-04-23]. Dostupné z: https://learn.microsoft.com/cs-cz/aspnet/core/blazor/hosting-models?view=aspnetcore-7.0#static-site-hosting.
30. DE MACEDO, João; ABREU, Rui; PEREIRA, Rui; SARAIVA, João. WebAssembly versus JavaScript: Energy and Runtime Performance. In: *2022 International Conference on ICT for Sustainability (ICT4S)*. 2022, s. 24–34. Dostupné z DOI: 10.1109/ICT4S55073.2022.00014.
31. *How fast and efficient is Wasm?* [Online]. [cit. 2023-04-24]. Dostupné z: https://www.adservio.fr/post/how-fast-and-efficient-is-wasm.
32. SPASOJEVIC, Marinko. *Blazor Server vs Blazor WebAssembly, Pros and Cons* [online]. [cit. 2023-04-23]. Dostupné z: https://code-maze.com/blazor-webassembly-introduction/.

33. *genetic-js* [online]. [cit. 2023-04-24]. Dostupné z: https://github.com/subprotocol/genetic-js.
34. RIVA, Vincent. *genome.js* [online]. [cit. 2023-04-24]. Dostupné z: https://github.com/Treast/genome.js.
35. *Jenetics* [online]. [cit. 2023-06-05]. Dostupné z: https://jenetics.io/.
36. DISTRIBUTED EVOLUTIONARY ALGORITHMS IN PYTHON. *deap* [online]. [cit. 2023-04-24]. Dostupné z: https://github.com/DEAP/deap.
37. GAD, Ahmed Fawzy. *PyGAD: Genetic Algorithm in Python* [online]. [cit. 2023-04-24]. Dostupné z: https://github.com/ahmedfgad/GeneticAlgorithmPython.
38. GAD, Ahmed Fawzy. *PyGAD: An Intuitive Genetic Algorithm Python Library*. 2021. Dostupné z arXiv: 2106.06158 [cs.NE].
39. GIACOMELLI, Diego. *GeneticSharp* [online]. [cit. 2023-04-24]. Dostupné z: https://github.com/giacomelli/GeneticSharp.
40. KIRILLOV, Andrew. *AForge.NET* [online]. [cit. 2023-04-24]. Dostupné z: https://github.com/andrewkirillov/AForge.NET.
41. ČÁPKA, David. *Lekce 4 - UML - Doménový model* [online]. [cit. 2023-04-27]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-domenovy-model-diagram.
42. *What is wireframing?* [Online]. [cit. 2023-04-27]. Dostupné z: https://www.experienceux.co.uk/faqs/what-is-wireframing/.
43. *MudBlazor - Blazor Component Library* [online]. [cit. 2023-05-03]. Dostupné z: https://mudblazor.com/.
44. *Ant Design Blazor* [online]. [cit. 2023-05-03]. Dostupné z: https://antblazor.com/en-US/.
45. *Blazorise Component Library* [online]. [cit. 2023-05-03]. Dostupné z: https://blazorise.com/.
46. LARSSON, Jonny; RECHEIS, Meinrad. *MudBlazor/LICENSE* [online]. [cit. 2023-05-03]. Dostupné z: https://github.com/MudBlazor/MudBlazor/blob/dev/LICENSE.
47. *Avatar - MudBlazor* [online]. [cit. 2023-05-03]. Dostupné z: https://mudblazor.com/components/avatar#usage.
48. GIACOMELLI, Diego. *TSP with GeneticSharp and Blazor* [online]. [cit. 2023-04-24]. Dostupné z: https://diegogiacomelli.com.br/tsp-with-geneticsharp-and-blazor/.
49. GIACOMELLI, Diego. *GeneticSharp/LICENSE* [online]. [cit. 2023-05-03]. Dostupné z: https://github.com/giacomelli/GeneticSharp/blob/master/LICENSE.
50. MOZAFFAR, Ahmad. *AK MULTILANGUAGES* [online]. [cit. 2023-05-03]. Dostupné z: https://akmultilanguages.azurewebsites.net/.
51. MOZAFFAR, Ahmad. *multilanguages* [online]. [cit. 2023-05-03]. Dostupné z: https://github.com/aksoftware98/multilanguages.
52. MOZAFFAR, Ahmad. *multilanguages/LICENSE* [online]. [cit. 2023-05-03]. Dostupné z: https://github.com/aksoftware98/multilanguages/blob/master/LICENSE.
53. *Blazor Extensions Canvas* [online]. [cit. 2023-05-03]. Dostupné z: https://github.com/BlazorExtensions/Canvas.
54. RIBEIRO, Gutemberg; HAJDRIK, Attila. *Canvas/LICENSE* [online]. [cit. 2023-05-03]. Dostupné z: https://github.com/BlazorExtensions/Canvas/blob/master/LICENSE.
55. *Plotly.Blazor* [online]. [cit. 2023-05-03]. Dostupné z: https://github.com/LayTec-AG/Plotly.Blazor.
56. *Plotly.js* [online]. [cit. 2023-05-03]. Dostupné z: https://github.com/plotly/plotly.js/.

57. MCLEISH, Sean. *Plotly.Blazor/LICENSEE* [online]. [cit. 2023-05-03]. Dostupné z: https://github.com/LayTec-AG/Plotly.Blazor/blob/main/LICENSE.
58. SIX LABORS. *ImageSharp* [online]. [cit. 2023-05-03]. Dostupné z: https://github.com/SixLabors/ImageSharp.
59. JACKSON-SOUTH, James; WILLIAMS, Scott. *ImageSharp/LICENSE* [online]. [cit. 2023-05-03]. Dostupné z: https://github.com/SixLabors/ImageSharp/blob/main/LICENSE.
60. LATHAM, Luke; ANDERSON, Rick et al. *Host and deploy ASP.NET Core Blazor WebAssembly* [online]. [cit. 2023-05-05]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/blazor/host-and-deploy/webassembly?view=aspnetcore-7.0#compression.
61. GOOGLE. *brotli* [online]. [cit. 2023-05-05]. Dostupné z: https://github.com/google/brotli.
62. GAD, Ahmed. *Reproducing Images using a Genetic Algorithm with Python* [online]. [cit. 2023-05-03]. Dostupné z: https://www.linkedin.com/pulse/reproducing-images-using-genetic-algorithm-python-ahmed-gad.
63. *Drop Zone - MudBlazor* [online]. [cit. 2023-05-04]. Dostupné z: https://mudblazor.com/components/dropzone#mics-chess-board.
64. LATHAM, Luke; ANDERSON, Rick et al. *Test Razor components in ASP.NET Core Blazor* [online]. [cit. 2023-05-04]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/blazor/test?view=aspnetcore-7.0.
65. HEESCH, Dimitri van. *Doxygen* [online]. [cit. 2023-05-04]. Dostupné z: https://www.doxygen.nl/index.html.

Obsah přiloženého archivu

```
readme.txt ..... stručný popis obsahu média
src
    impl ..... zdrojové kódy vytvořené aplikace
    thesis ..... zdrojová forma práce ve formátu LaTeX
    installationmanual ..... instalační příručka v vytvořené aplikaci
    documentation ..... programátorská dokumentace k vytvořené aplikaci
text
    thesis.pdf ..... text práce ve formátu PDF
```