

# Projekt 2

## Hľadanie pokladu (2b)

**Laura Fulajtárová**

Fakulta informatiky a informačných technológií STU

`xfulajtarova@stuba.sk`

ID: 120782

## Contents

Contents .....	2
1 Parametre môjho počítača .....	3
2 Implementačné prostredie .....	3
3 Riešený problém .....	3
4 Evolučný algoritmus .....	3
5 Reprezentácia údajov .....	3
5.1 Populácia .....	3
5.2 Chromozóm .....	4
5.3 Gén .....	4
6 Virtuálny stroj .....	4
7 Tvorba populácie .....	5
7.1 Prvá populácia .....	5
7.2 Ostatné populácie .....	5
7.3 Selekcia .....	6
7.3.1 Ruleta .....	6
7.3.2 Turnament .....	6
7.4 Reprodukcia .....	7
7.4.1 Crossover .....	7
7.4.2 Mutácia .....	7
8 Používateľské rozhranie .....	8
8.1 Mriežka zo zadania .....	8
8.2 Používateľom vytvorená mriežka .....	8
8.3 Animácia .....	9
9 Zhodnotenie výsledkov .....	9
10 Možné vylepšenia .....	10
11 Použitá literatúra .....	12

## 1 Parametre môjho počítača

- Processor AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz
- Installed RAM 16.0 GB (13.9 GB usable)
- System type 64-bit operating system, x64-based processor

## 2 Implementačné prostredie

Rozhodla som sa použiť Python ako implementačné prostredie kvôli jeho jednoduchosti, prirodzenému čitateľnému kódu, rozsiahlemu ekosystému knižníc a funkcií, čo umožňuje rýchle a efektívne vývojové práce. Python je ideálny pre implementáciu rôznych algoritmov vrátane evolučných algoritmov, pričom minimalizuje komplikácie spojené s programovaním, takže môžem sústrediť svoju pozornosť na samotný algoritmus a jeho správne fungovanie.

## 3 Riešený problém

Našou hlavnou úlohou bolo vytvoriť hru, v ktorej sme hľadali poklady na dvojrozmernej mriežke. Hľadač sa mohol pohybovať v štyroch základných smeroch: hore, dole, vpravo a vľavo. Naša úloha spočívala v tom, že sme mali obmedzený počet krokov, v ktorých sme sa snažili nájsť čo najviac pokladov. Túto úlohu sme riešili pomocou evolučného programovania nad virtuálnym strojom.

Virtuálny stroj mal za úlohu generovať postupnosť náhodných krokov, ktoré jednotlivci v populácii vykonal. Títo jedinci boli zoskupení do populácie, ktorá sa vyvíjala prostredníctvom selekcie, kríženia, mutácie a elitizmu. Každého jedinca sme ohodnotili pomocou fitness hodnoty, ktorá nám indikuje, kvalitu jednotlivca.

Program má byť ukončený v prípade, že hľadač našiel všetky poklady alebo keď sme dosiahli maximálny počet generácií.

## 4 Evolučný algoritmus

Evolučný algoritmus predstavuje fascinujúci prístup k riešeniu problémov, ktorý čerpá inšpiráciu z prírody, konkrétne z Darwinovej evolúcie a Mendelovej genetiky. Jeho cieľom je nájsť a zlepšiť najlepšie možné riešenie pre daný problém. Tento proces sa uskutočňuje v niekoľkých krokoch:

1. Inicializácia: Začíname vytvorením prvej populácie náhodných jedincov. Každý jedinec v populácii má svoj chromozóm, ktorý je zložený z génov.
2. Evaluácia: Pre každého jedinca v populácii určíme jeho "fitness" alebo schopnosť riešiť daný problém. Táto hodnota ovplyvňuje, ktorí jedinci budú vybraní pre ďalší vývoj.
3. Možné ukončenie: Po určitom čase alebo po dosiahnutí požadovanej úrovne fitness posúdime, či sme spokojní s populáciou a jedincami. Ak áno, ukončíme algoritmus. V opačnom prípade pokračujeme v ďalšom vývoji populácie.
4. Selekcia: Najlepší jedinci s najvyššou "fitness" prežijú a budú základom pre budúcu generáciu.
5. Variácia: Na vytvorenie novej generácie použijeme týchto lepších jedincov a kombinujeme ich genetický materiál napríklad cez procesy ako crossover a mutácia, čo nám umožňuje objavovať nové možnosti.
6. Návrat k evaluácii: S novou populáciou, ktorá má potenciálne vyššiu "fitness", sa vrátíme k evaluácii. Tento krok sa opakuje, kým nedosiahneme požadované výsledky alebo nedosiahneme stanovený maximálny počet generácií.

Evolučný algoritmus nám teda umožňuje postupne zlepšovať riešenia v rámci populácie, pričom sa inšpiruje prírodnými procesmi evolúcie a genetiky, a to všetko s cieľom nájsť čo najlepšie riešenie daného problému.

## 5 Reprezentácia údajov

### 5.1 Populácia

V rámci evolučného algoritmu je populácia reprezentovaná ako kolekcia jednotlivcov, ktorí sú v tomto kontexte známi ako chromozómy.

## 5.2 Chromozóm

Chromozóm je jedinečným zástupcom v populácii, a preto mu venujem osobitný objekt, v ktorom uchovávam všetky relevantné informácie. To zahŕňa hodnoty, fitness, počet nájdených pokladov, počet krokov, ktoré vykonal, a informácie o jeho pohybe v rámci mriežky po vykonaní pohybů.

## 5.3 Gén

Chromozóm je rozdelený na 64 bytov, pričom každých 8 bitov tvorí jeden gén. Tieto gény obsahujú informácie, ktoré určujú vlastnosti a správanie jednotlivca. Gény sú základnými stavebnými jednotkami chromozómov a ovplyvňujú vývoj a výsledky jednotlivcov v populácii.

## 6 Virtuálny stroj

Náš virtuálny stroj disponuje 64 pamäťovými bunkami, pričom každá má veľkosť 1 byte, a rozumie štyrom základným inštrukciám: zvýšenie hodnoty bunky, zníženie hodnoty bunky, skok na konkrétnu adresu a výpis na základe hodnoty bunky. Inštrukcie majú tvar 00XXXXXX (increment), 01XXXXXX (decrement), 10XXXXXX (jump) a 11XXXXXX (write), kde XXXXXX reprezentuje adresu bunky. Výstup inštrukcií závisí od hodnoty bunky: P pre 1-2, H pre 3-4, D pre 5-6 a L pre 7-8. Program okrem výstupu poskytuje aj informácie o pohyboch, ktoré vykonal. Program sa ukončí, ak nájde všetky poklady, vyjde z mriežky alebo vykona 500 krokov. Vytvorí objekt pre daný chromozóm s údajmi ako je počet nájdených pokladov, gény, fitness, kroky a mriežka. Fitness funkcia závisí od počtu nájdených pokladov a vykonaných krokov, je reprezentovaná vzorcom:  $f = 2 / (\text{počet krokov} + 1) + 5 * \text{počet nájdených pokladov}$ . Program bol zjednodušený na **pseudo program**, pre jeho úplné znenie pozrite súbor *game.py*. Pre jednotlivé pohyby na mriežke pozrite súbor *moves.py*.

<b>Adresa:</b>	000000	000001	000010	000011	000100	000101	000110	...
<b>Hodnota:</b>	00000000	00011111	00010000	01010000	00000101	11000000	10000100	...

```
def virtual_machine(individual, board, board_size, treasure_count):
    board_copy = board.copy()
    moves_list = []
    individual_copy = individual.copy()
    out_of_bounds = False
    treasure_found_num = 0
    register_index = 0
    for i in range(500):
        if register_index <= 63:
            opcode = individual[register_index][:2]
            register_value = individual[register_index][2:]

            if opcode == "00":
                increment

            elif opcode == "01":
                decrement

            elif opcode == "10":
                jump

            elif opcode == "11":
                new_register_index = int(register_value, 2)
                new_register_value = individual[new_register_index]
                ones_count = new_register_value.count("1")
                if ones_count <= 2:
                    up
                elif ones_count <= 4:
                    down
                elif ones_count <= 6:
                    left
                else:
                    right
                if treasure_found:
                    treasure_found_num += 1
                register_index += 1
            else:
                register_index = 0
```

```

    if out_of_bounds or treasure_found_num == treasure_count:
        break

    fitness = 2 / (len(moves_list) + 1) + 5 * treasure_found_num
    individual_object = Individual(
        individual_copy, fitness, moves_list, treasure_found_num, board_copy)

    if treasure_found_num == treasure_count:
        global solution_individual
        solution_individual = individual_object

    return individual_object

```

## 7 Tvorba populácie

### 7.1 Prvá populácia

Prvú populáciu vytvárame vytvorením náhodných génov pre určený počet jedincov v populácii. Každý gén je reprezentovaný 64 bytmi. Následne týchto jedincov vkladáme do virtuálneho stroja, ktorý vytvára objekty s relevantnými údajmi pre každého jedinca. Tieto objekty potom ukladáme do spoločného zoznamu, ktorý vytvorí danú populáciu.

```

def make_first_generation(
    board, individual_count, random_values_for_individuals, board_size, treasure_count
):
    generation_list_object = []
    global solution_individual

    for i in range(individual_count):
        if solution_individual is None:
            board_copy = board.copy()
            individual_values = vm_create_random_values(random_values_for_individuals)
            individual_object = virtual_machine(
                individual_values, board_copy, board_size, treasure_count
            )
            generation_list_object.append(individual_object)

    return generation_list_object

```

### 7.2 Ostatné populácie

Začínáme tým, že zoradíme jedincov v populácii na základe ich fitness hodnôt. Potom z tejto zoradenej populácie vyberáme elitných jedincov, ktorých automaticky preniesieme do novej populácie.

Pre vytvorenie ďalších jedincov využívame selekciu, či už pomocou metódy rulety alebo turnaja. Z týchto vybraných rodičov potom vytvárame nových potomkov pomocou procesu kríženia, kde kombinujeme genetický materiál od oboch rodičov, čím vytvárame nových jedincov.

Títo noví jedinci sú následne podrobení mutáciám s určitou pravdepodobnosťou, čo pridáva variabilitu do populácie. Ak je populácia plná, presunieme ju do ďalšieho kola, v ktorom opakujeme tento proces.

Celý tento cyklus pokračuje, kým nenájdeme optimálne riešenie, alebo nedosiahneme maximálny počet generácií. Týmto spôsobom evolučný algoritmus postupne zlepšuje populáciu a hľadá najlepšie možné riešenia pre daný problém.

```

while solution_individual is None and generation_num < max_generations:
    generation_num += 1

    generation_list_object.sort(key=lambda x: x.fitness, reverse=True)

    best_fitness_individuals.append(generation_list_object[0].fitness)

    elite_individuals = []
    elite_individuals = copy.deepcopy(
        generation_list_object[:elite_individual_count]
    )

    subelite_individuals = []
    if selection_type == 1:
        subelite_individuals = roulette_wheel(

```

```

        copy.deepcopy(generation_list_object),
        individual_count,
        elite_individual_count,
    )
else:
    subelite_individuals = tournament(
        copy.deepcopy(generation_list_object),
        individual_count,
        elite_individual_count,
    )

mutation_list = []
mutation_list = mutation(subelite_individuals, mutation_probability)

subelite_list_objects = make_other_generations(
    mutation_list, board_copy, board_size, treasure_count
)

population_list_object = []
population_list_object = copy.deepcopy(elite_individuals)
population_list_object.extend(subelite_list_objects)

generation_list_object = population_list_object

```

### 7.3 Selekcia

Selekcia je výber lepších jedincov na základe ich úspešnosti, čím zvyšuje ich šancu na prežitie a ďalšiu generáciu. Pomáha vylepšovať celkovú kvalitu populácie.

#### 7.3.1 Ruleta

Metóda rulety vyberá jedincov do podpopulácie (subelite). Rozpočítame fitness jedincov, vytvoríme pravdepodobnosti výberu, a na základe nich vyberáme rodičov a tvoríme potomkov. Potomkovia sú pridaní do subpopulácie.

```

def roulette_wheel(generation_list_object, individual_count, elite_individual_count):
    subelite_individuals = []

    total_fitness = sum(individual.fitness for individual in generation_list_object)
    selection_probabilities = [
        individual.fitness / total_fitness for individual in generation_list_object
    ]

    while len(subelite_individuals) != (individual_count - elite_individual_count):
        parent1 = random.choices(generation_list_object, selection_probabilities)[0]
        parent2 = random.choices(generation_list_object, selection_probabilities)[0]

        first_child, second_child = crossover(parent1.value, parent2.value)

        if first_child not in subelite_individuals and len(
            subelite_individuals
        ) + 1 <= (individual_count - elite_individual_count):
            subelite_individuals.append(first_child)
        if second_child not in subelite_individuals and len(
            subelite_individuals
        ) + 1 <= (individual_count - elite_individual_count):
            subelite_individuals.append(second_child)

    return subelite_individuals

```

#### 7.3.2 Turnament

V prípade funkcie tournament\_winner vytvárame náhodný turnajný zoznam a vyberáme víťaza na základe fitness.

Funkcia tournament vytvára náhodne veľké turnaje, kým nedosiahneme požadovaný počet jedincov pre subpopuláciu. V každom turnaji vyberáme rodičov na základe ich fitness pomocou funkcie tournament\_winner, a títo rodičia sú následne využití na vytvorenie potomkov. Potomkovia sú pridaní do subelite podľa stanovených kritérií.

```

def tournament_winner(generation_list_object, size_of_tournament):
    tournament_list = []

    for _ in range(size_of_tournament):
        tournament_list.append(random.choice(generation_list_object))

```

```

tournament_list.sort(key=lambda x: x.fitness, reverse=True)
winner = tournament_list[0]
return winner

def tournament(generation_list_object, individual_count, elite_individual_count):
    subelite_individuals = []

    size_of_tournament = random.randint(2, 5)
    while len(subelite_individuals) != (individual_count - elite_individual_count):
        parent_1 = tournament_winner(generation_list_object, size_of_tournament)
        parent_2 = tournament_winner(generation_list_object, size_of_tournament)

        while parent_1 == parent_2:
            parent_2 = tournament_winner(generation_list_object, size_of_tournament)

        parent_1_values = parent_1.value
        parent_2_values = parent_2.value

        first_child, second_child = crossover(parent_1_values, parent_2_values)

        if first_child not in subelite_individuals and len(
            subelite_individuals
        ) + 1 <= (individual_count - elite_individual_count):
            subelite_individuals.append(first_child)
        if second_child not in subelite_individuals and len(
            subelite_individuals
        ) + 1 <= (individual_count - elite_individual_count):
            subelite_individuals.append(second_child)

    return subelite_individuals

```

## 7.4 Reprodukcia

Reprodukcia v evolučných algoritmoch slúži na vytvorenie nových jedincov v populácii pomocou kríženia (crossover) a mutácie existujúcich jedincov. Týmto spôsobom sa kombinujú genetické vlastnosti a vytvárajú sa potomkovia, čím sa postupne zlepšuje celková kvalita populácie a hľadá najlepšie riešenia pre daný problém.

### 7.4.1 Crossover

Táto funkcia dostáva dvoch rodičov ako vstupné parametre a vykonáva kríženie tým spôsobom, že náhodne vyberie index, kde rozdelí genetický materiál oboch rodičov, a potom spojí tieto časti, čím vytvorí nových potomkov.

```

def crossover(parent1, parent2):
    r_num = random.randint(1, 64)
    first_child = parent1[:r_num] + parent2[r_num:]
    second_child = parent2[:r_num] + parent1[r_num:]
    return first_child, second_child

```

### 7.4.2 Mutácia

Mutácia prebieha s pravdepodobnosťou, ktorú určuje používateľ, a ovplyvňuje zmenu genov a ich počet. Začíname výberom náhodných indexov génov na mutáciu podľa stanovenej pravdepodobnosti. Potom vybrané gény mutujeme pomocou XOR operácie na 3 bitoch. Mutovaných jedincov postupne pridávame do zoznamu a následne ho vrátime.

```

def mutation(other_generation_list, mutation_probability):
    mutation_list = []

    for individual in other_generation_list:
        for j in range(mutation_probability):
            cell_index = random.randint(0, 63)
            cell = individual[cell_index]
            dec_byte = int(cell, 2)
            for k in range(3):
                mask = 1 << random.randint(0, 7)
                cell = dec_byte ^ mask
                dec_byte = cell
            individual[cell_index] = bin(cell)[2:].zfill(8)
            mutation_list.append(individual)

    return mutation_list

```

## 8 Používateľské rozhranie

Upravila som používateľské rozhranie vytvorením menu v termináli. Používateľ má teraz možnosť vybrať si, či chce použiť vopred definovanú mriežku zo zadania, vytvoriť vlastnú mriežku alebo ukončiť program. V prípade, že neukončí program, používateľ môže zadať rôzne údaje, ako je počet jedincov v populácii, počet náhodne inicializovaných génov pre jednotlivca, typ selekcie, elitizmus, mutáciu, maximálny počet generácií a aj to, či chce zobrazíť výstupy a jednoduchú animáciu simulácie pohybu hľadača na mriežke. Po vykonaní nám program taktiež zobrazí graf, ktorý opisuje fitness hodnotu najlepšieho jedinca v jednotlivej populácii. Týmto spôsobom sme pridali viac interakcie a prispôbili program podľa potrieb používateľa.

### 8.1 Mriežka zo zadania

```
Welcome to the Treasure Hunt Game!

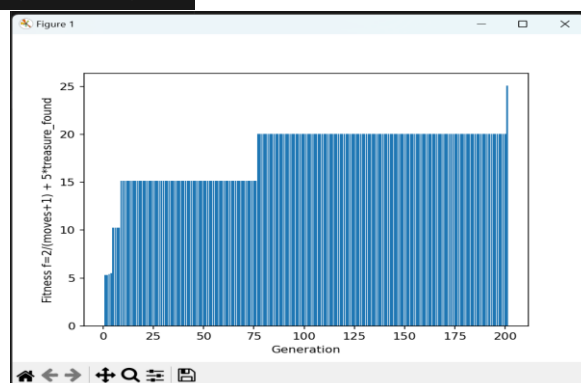
1. Generate a board from assignment
2. Generate a custom board
3. Exit
1

+-----+
| | | | | | | |
+-----+
| | | | | T | |
+-----+
| | | T | | | |
+-----+
| | | | | | | T
+-----+
| | | | | | | |
+-----+
| | T | | | T | |
+-----+
| | | | S | | |
+-----+

Enter the number of individuals (40): 40
Enter number of random values for individual in virtual machine (60): 60
Enter the selection type (1 - Roulette Wheel, 2 - Tournament): 2
Enter the mutation probability percentage (10): 10
Enter the elitism percentage in a population (10): 10
Enter the maximum number of generations (1000): 1000
Do you want to see the animation if solution is found? (1 - Yes, 2 - No): 2
```

```
Solution found.
Generation number: 201
Treasures found: 5
Solution path:
['up', 'right', 'up', 'up', 'right', 'right', 'up', 'up', 'left', 'left', 'up', 'down', 'left',
 'down', 'left', 'up', 'down', 'left', 'right', 'down', 'down', 'left']
Solution board after moves:

+-----+
| | | | | | | |
+-----+
| | | | | | | |
+-----+
| | | | | | | |
+-----+
| | | | | | | |
+-----+
| | | | | | | |
+-----+
| | S | | | | |
+-----+
| | | | | | | |
+-----+
```



### 8.2 Používateľom vytvorená mriežka

```
1. Generate a board from assignment
2. Generate a custom board
3. Exit
2

Enter the board size: 3
+-----+
| 1 | 2 | 3 |
+-----+
| 4 | 5 | 6 |
+-----+
| 7 | 8 | 9 |
+-----+

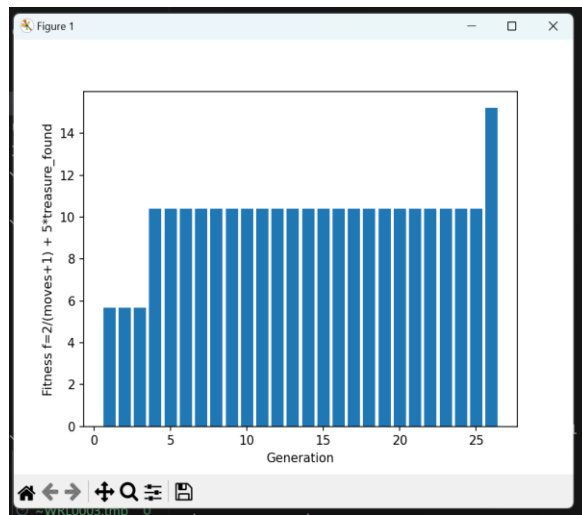
Enter number of the starting point: 7
Enter the treasure indexes (separated by spaces): 1 3 5
+-----+
| T | | T |
+-----+
| | T | |
+-----+
| S | | |
+-----+

Enter the number of individuals (40): 20
Enter number of random values for individual in virtual machine (60): 50
Enter the selection type (1 - Roulette Wheel, 2 - Tournament): 1
Enter the mutation probability percentage (10): 5
Enter the elitism percentage in a population (10): 10
Enter the maximum number of generations (1000): 100
Do you want to see the animation if solution is found? (1 - Yes, 2 - No): 2
```

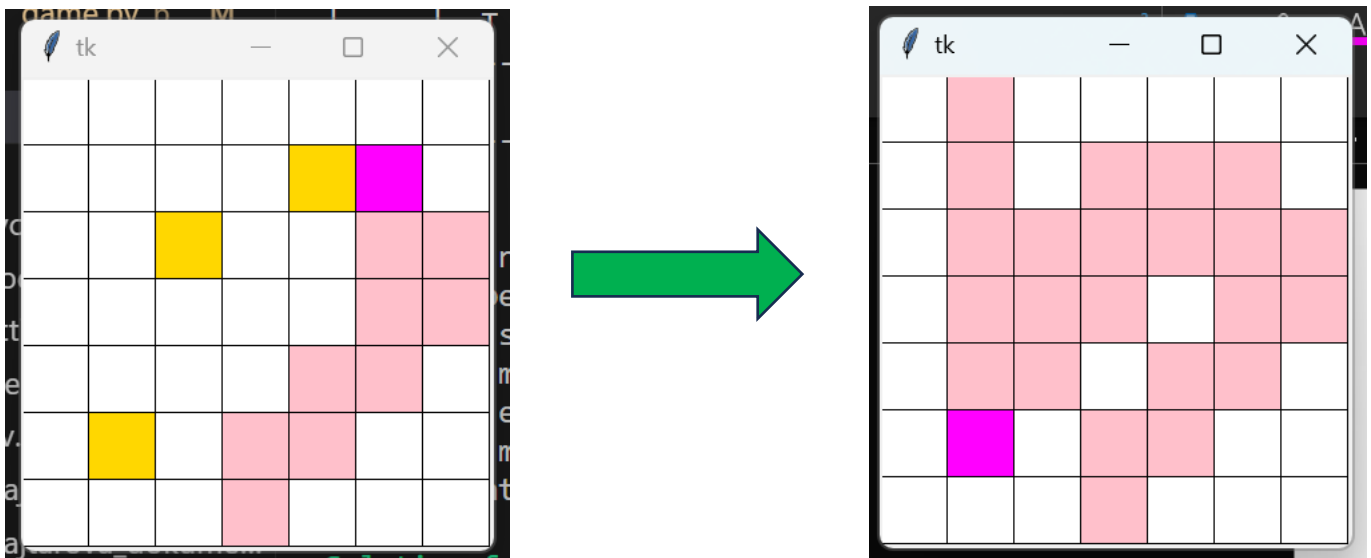
```
Solution found.
Generation number: 26
Treasures found: 3
Solution path:
['up', 'right', 'left', 'up', 'right', 'down', 'up', 'right']
Solution board after moves:

+-----+
| | | S |
+-----+
| | | |
+-----+
| | | |
+-----+
| | | |
+-----+
```





### 8.3 Animácia

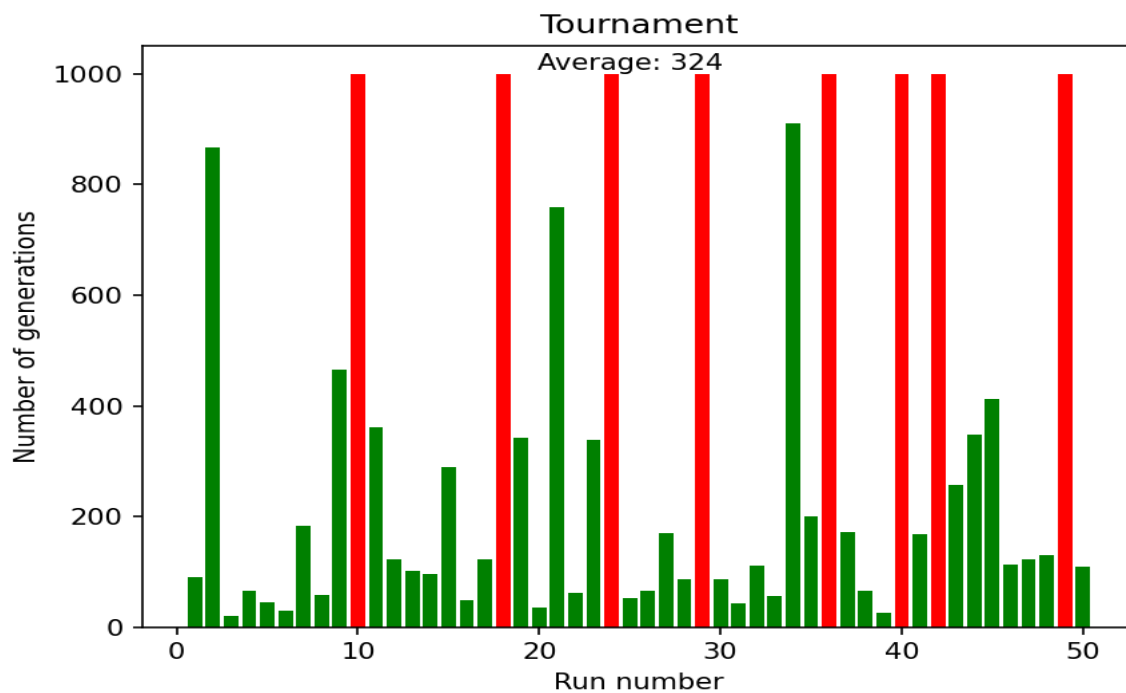
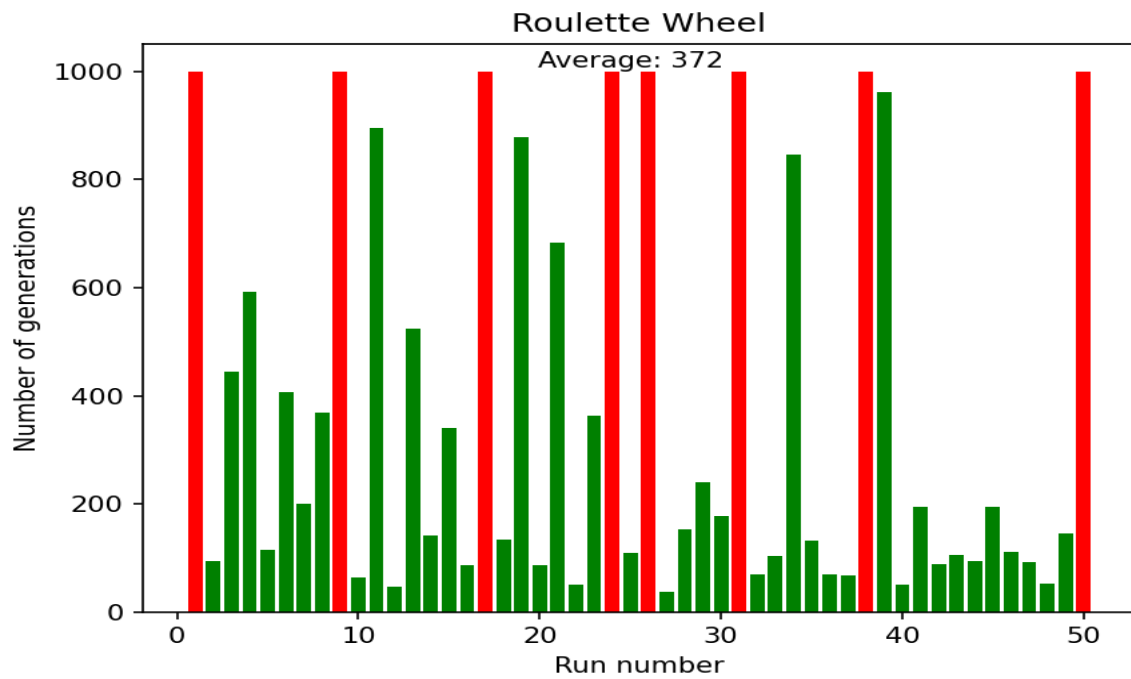


## 9 Zhodnotenie výsledkov

Vykonal som selekciu v populácii dvoma spôsobmi: pomocou metódy rulety a turnaja. Pre každý z týchto dvoch typov selekcie som spustila program 50-krát a získala priemerné výsledky. Zistila som, že tieto dva typy selekcie sa podobajú, a výsledné rozdiely sú skoro zanedbateľné.

Pokiaľ ide o počet nenájdenných riešení, oba typy boli na tom rovnako. Avšak, ak sa pozrieme na nájdené riešenia a v akej populácii boli nájdené, vidíme, že turnajová selekcia dosahuje lepšie výsledky. Čo sa týka časovej náročnosti, ruleta trvala o 10 sekúnd dlhšie.

Na grafe sú na osi X uvedené poradové čísla jednotlivých testov a na osi Y je zobrazené, v ktorej generácii bolo riešenie nájdené alebo nenájdené. Nenájdenné riešenia sú označené červenou farbou a nájdené zelenou. Pri výpočte priemeru som zahrnula všetky riešenia.



```
Roulette Wheel
Time: 66.01692008972168 s

Tournament
Time: 79.63804221153259 s
```

## 10 Možné vylepšenia

V snahe zlepšiť môj kód a optimalizovať ho, začnem identifikovaním miest, kde môžem zvýšiť jeho efektívnosť. To znamená preskúmanie existujúceho algoritmu a použitie efektívnejších dátových štruktúr alebo algoritmických postupov. Zároveň zredukujem nepotrebný kód a odstránim zbytočné časti, ktoré neprinášajú hodnotu.

Rovnako dôležité je zvýšenie priestorovej efektivity kódu. Preskúmam, ako sa využíva pamäť, a pokúsim sa znížiť pamäťovú náročnosť tam, kde to bude možné.

Na záver, možnosti rozšírenia programu zahrňujú pridanie ďalších typov selekcií a mutácií. Môžem implementovať rôzne stratégie selekcie, a experimentovať s rôznymi spôsobmi mutácií genetického algoritmu. Týmto spôsobom by sme mohli dosiahnuť lepšie výsledky pri riešení problémov pomocou genetických algoritmov.

## 11 Použitá literatura

1. [https://www.youtube.com/watch?v=L--lxUH4fac&t=112s&ab\\_channel=Dr.ShahinRostami](https://www.youtube.com/watch?v=L--lxUH4fac&t=112s&ab_channel=Dr.ShahinRostami)