

STU FIIT
Umela Inteligencia

“Zenová záhrada”

Zadanie č. 1a

Made by: Meredov Nazar

Cvičiaci: Bc. Jakub František Setnický

AIS ID: 122092

Anotácia:

Cieľom môjho projektu je pomocou evolučného algoritmu vypočítať najlepšiu cestu pre mnícha, ktorý chce zorať záhradu s najlepším výsledkom.

Záhrada je pokrytá pieskom a má pevné objekty, ako sú kamene a sochy, ktoré slúžia ako prekážky.

Mojou úlohou bolo nájsť optimálnu cestu pre mnícha, aby čo najefektívnejšie pohrabal celú záhradu.

Evolučný algoritmus mal dôležitú rol pri optimalizácii tohto procesu tým, že vypočítal a vybral najlepšiu cestu.

Efektívnosť riešenia sa vyhodnocuje pomocou fitness funkcie, ktorá počíta počet úspešne zhrabaných plôch.

Na dosiahnutie výkonnosti sa v algoritme použili rôzne mutácie na zabezpečenie rozmanitosti populácie.

Ako najlepšia možnosť bol na výber zvolený turnajový algoritmus a konečným výstupom algoritmu bude matica vizualizujúca pohyb mnícha a zhrabane plochy záhrady.

Opis úlohy:

Mních môže vstúpiť do záhrady z ľubovoľného miesta a kráčať len smerom pred sebou, a keď narazí na prekážku pred sebou, môže sa podľa vlastného výberu otočiť doľava alebo doprava a pokračovať tak v ceste, ale ak otočenie nie je možné, mních končí svoju cestu a nemôže sa ďalej pohybovať.

Ak mních opustí záhradu, môže do nej opäť vstúpiť z ľubovoľnej pozície, a tak pokračovať v algoritme.

Úlohou je pomocou evolučných algoritmov optimalizovať pohyb mnícha tak, aby dokázal pohrabať čo najväčšiu časť záhrady a v ideálnom prípade celú záhradu. Hlavnými výzvami bude optimalizácia výpočtového času, správna reprezentácia pohybov hráča v matrici záhrady a vytvorenie logiky pohybu, ktorá zohľadňuje všetky špeciálne prípady, ako sú obraty, výstup z hraníc a ukončenie hry.

Vývoj metód výberu populácie, ako aj metódy fitness na hodnotenie jednotlivcov.

Najlepším riešením je plne rozoraná záhrada pre čo najväčší počet generácií a populácií v každej generácii

Prečo, ako, a načo to robime?

Projekt rieši špecifický problém optimalizácie pohybu v priestore s prekážkami. V tomto prípade ide o simuláciu mnícha, ktorý má čo najefektívnejšie zorať piesok v záhrade.

Takéto úlohy môžu byť užitočné pri modelovaní dopravy, plánovaní trasy a iných úlohách, pri ktorých je dôležité efektívne prekonávať prekážky.

Použitie evolučných algoritmov je spôsob, ako takéto úlohy riešiť efektívne a automaticky a ktorý napodobňuje proces prirodzeného výberu.

V tomto kontexte to znamená generovanie rôznych možných ciest (nazývaných osobnosti mnícha/cesty vývoja), ktoré môže skutočný mních použiť na pohyb po záhrade.

Tieto cesty sa potom vyhodnocujú pomocou funkcie fitness, ktorá meria, koľko piesku bolo preoraného.

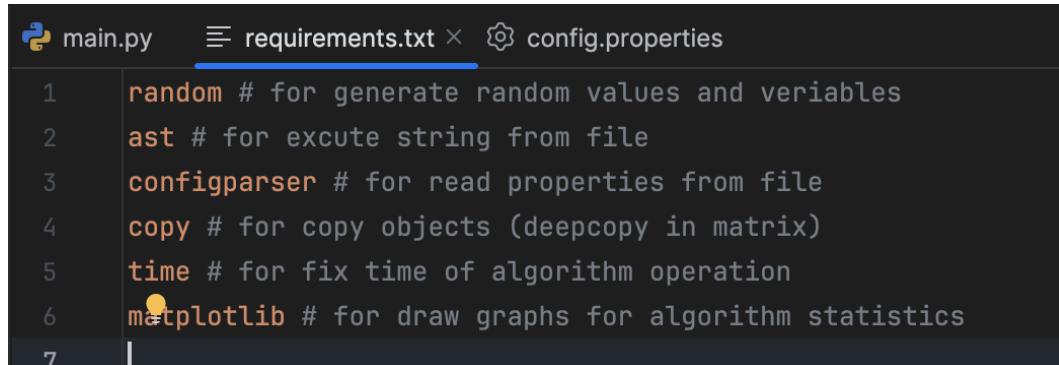
Najlepšie cesty sa potom krížia“ prostredníctvom „crossover“ a mutujú pri rôznych pomeroch „mutation“, aby v ďalšej generácii vznikli ešte lepšie riešenia.

Týmto spôsobom algoritmus dôsledne nachádza optimálne riešenie

V tomto projekte evolučný algoritmus výrazne zjednodušuje hľadanie efektívnych riešení a zároveň poskytuje presnosť a flexibilitu na riešenie rôznych scenárov v záhrade.

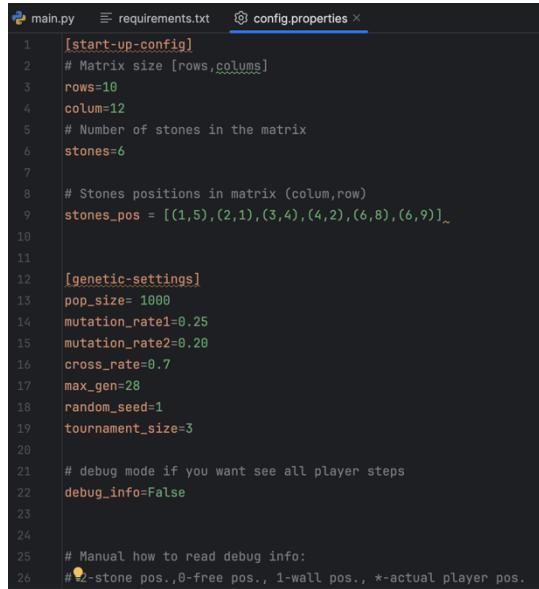
Ako spustiť program, ako funguje súbor .conf?:

1. Potrebné knižnice na stiahnutie pre fungovanie programu:



```
1 random # for generate random values and variables
2 ast # for execute string from file
3 configparser # for read properties from file
4 copy # for copy objects (deepcopy in matrix)
5 time # for fix time of algorithm operation
6 matplotlib # for draw graphs for algorithm statistics
7 |
```

2. Ako funguje config súbor na nastavenie počiatočných hodnôt programu;



```
1 [start-up-config]
2 # Matrix size [rows,columns]
3 rows=10
4 colum=12
5 # Number of stones in the matrix
6 stones=6
7
8 # Stones positions in matrix (colum,row)
9 stones_pos = [(1,5),(2,1),(3,4),(4,2),(6,8),(6,9)]
10
11 [genetic-settings]
12 pop_size= 1000
13 mutation_rate1=0.25
14 mutation_rate2=0.20
15 cross_rate=0.7
16 max_gen=28
17 random_seed=1
18 tournament_size=3
19
20 # debug mode if you want see all player steps
21 debug_info=False
22
23
24
25 # Manual how to read debug info:
26 # 2-stone pos., 0-free pos., 1-wall pos., *-actual player pos.
```

Konfiguračný súbor, definuje nastavenia programu v dvoch hlavných častiach: [start-up-config] a [genetic-settings].

[start-up-config] - V tejto časti sa nastavuje prostredie, v ktorom program pracuje, napríklad veľkosť matice a pozície kameňov

rows=10: počet riadkov v matici.

colum=12: počet stĺpcov v matici.

stones=6: počet kameňov

stones_pos = [(1,5),(2,1),(3,4),(4,2),(6,8),(6,9)]: Zoznam pozícíí kameňov v matici.

(Pozície sú tuple, kde každý predstavuje (riadok, stĺpec) kameňa.

[genetic-settings] - Táto časť definuje nastavenia týkajúce sa genetického algoritmu:

pop_size=1000: veľkosť populácie v každej generácii.

mutation_rate1=0,25: Mutačná rýchlosť pre prvý typ mutácie (mutácia génovej cesty).

mutation_rate2=0,20: Mutačná rýchlosť pre druhý typ mutácie (mutácia polohy).

cross_rate=0,7: Miera kríženia používaná počas reprodukcie na kombinovanie jedincov.

max_gen=28: Maximálny počet generácií

random_seed=1: Seed na generovanie náhodných čísel.

tournament_size=3: Veľkosť turnajového výberového procesu.

debug_info=False: Prepínač na zapnutie alebo vypnutie režimu ladenia, ktorý poskytuje podrobný výstup o každom kroku hráčov počas ich pohybu maticou.

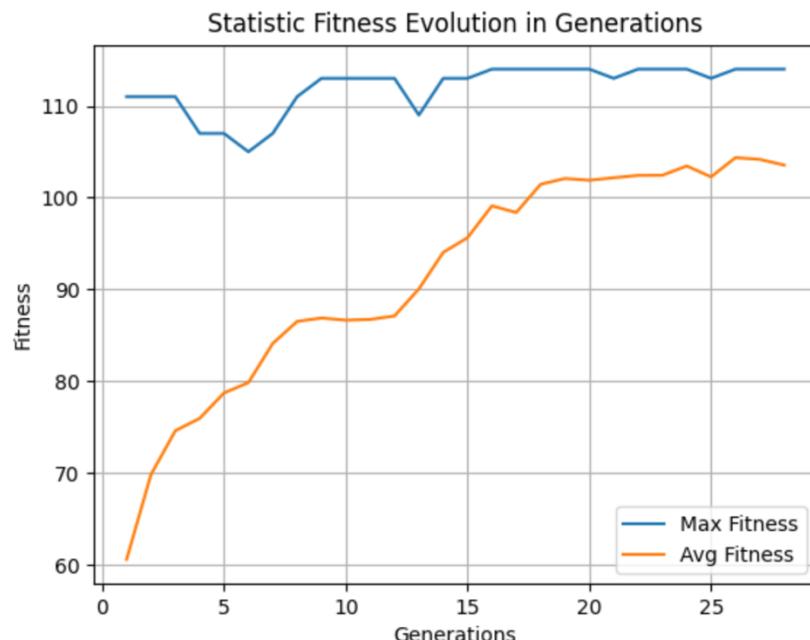
Podrobný opis fungovania programu:

- 1) Rozbalenie a inicializácia premenných z konfiguračného súboru pre ďalšiu prácu s premennými
- 2) Vytvorenie záhrady (matice) a východiskovej populácie na základe počiatočných údajov
- 3) Spustenie cyklu na výcvik a vytvorenie novej populácie
 - a. Vyvolanie metódy fitness na vyhodnotenie každého jedinca v populácii každej generácie
 - b. Zozbierajte a spočítajte skóre každého jednotlivca v každej generácii na vytvorenie štatistických grafov.
 - c. Usporiadanie turnaja na výber najlepších jednotlivcov budúcej generácie
 - d. Kríženie vybraných jedincov a šľachtenie ich potomstva pre budúcu generáciu
 - e. Aplikácia mutácií na populáciu
 - f. Nahradenie starej populácie novou
- 4) Stanovenie času stráveného spracovaním a výpočtom a zobrazenie týchto informácií na console
- 5) Výstup najlepšieho jedinca v poslednej generácii (úplná cesta mnícha so všetkými akciami) s konečnými výsledkami a štatistikami výpočtu algoritmu

P.s Kliknuv na krok, ktorý vás zaujíma, si budete môcť pozrieť popis a foto kódu tejto časti.

Foto výstupu programu v normálnom režime:

V tomto režime na výpise vidíme štruktúru a skóre každého indivídua v každej generácii, a na konci vidíme čas výpočtu a najlepšieho indivídua, ako aj graf jeho zlepšenia.



Debug režime:

Cez debug mode vidíme postup chôdze hráča a jeho rozhodnutia s náhľadom výpisu matice.

```
#####
Individual 191:
+Start Position: [(0, 6), 'S'], ((9, 10), 'N'), ((9, 11), 'W'), ((9, 0), 'N'), ((0, 7), 'S'), ((0, 8), 'E')
+Gene Path: [0, 0, 1, 1, 1]
I can move in (0, 6), continue.

 0 1 2 3 4 5 6 7 8 9 10 11
+-----+
0| 0 0 0 0 0 0 1 0 0 0 0 0 |
1| 0 0 0 0 0 0 0 0 0 0 0 0 |
2| 0 2 0 0 0 0 0 0 0 0 0 0 |
3| 0 0 0 0 0 2 0 0 0 0 0 0 |
4| 0 0 2 0 0 0 0 0 0 0 0 0 |
5| 0 0 0 0 0 0 0 0 0 0 0 0 |
6| 0 0 0 0 0 0 0 0 0 2 0 0 |
7| 0 0 0 0 0 0 0 0 0 0 0 0 |
8| 0 0 0 0 0 0 0 0 0 0 0 0 |
9| 0 0 0 0 0 0 0 0 0 0 0 0 |

I can move in (1, 6), continue.

 0 1 2 3 4 5 6 7 8 9 10 11
+-----+
0| 0 0 0 0 0 0 1 0 0 0 0 0 |
1| 0 0 0 0 0 0 2 1 0 0 0 0 0 |
2| 0 2 0 0 0 0 0 0 0 0 0 0 |
3| 0 0 0 0 0 2 0 0 0 0 0 0 |
4| 0 0 2 0 0 0 0 0 0 0 0 0 |
5| 0 0 0 0 0 0 0 0 0 0 0 0 |
6| 0 0 0 0 0 0 0 0 0 2 0 0 |
7| 0 0 0 0 0 0 0 0 0 0 0 0 |
8| 0 0 0 0 0 0 0 0 0 0 0 0 |
9| 0 0 0 0 0 0 0 0 0 0 0 0 |

I can move in (2, 6), continue.

 0 1 2 3 4 5 6 7 8 9 10 11
+-----+
0| 0 0 0 0 0 0 1 0 0 0 0 0 |
1| 0 0 0 0 0 0 2 1 0 0 0 0 0 |

I can move in (9, 5), continue.

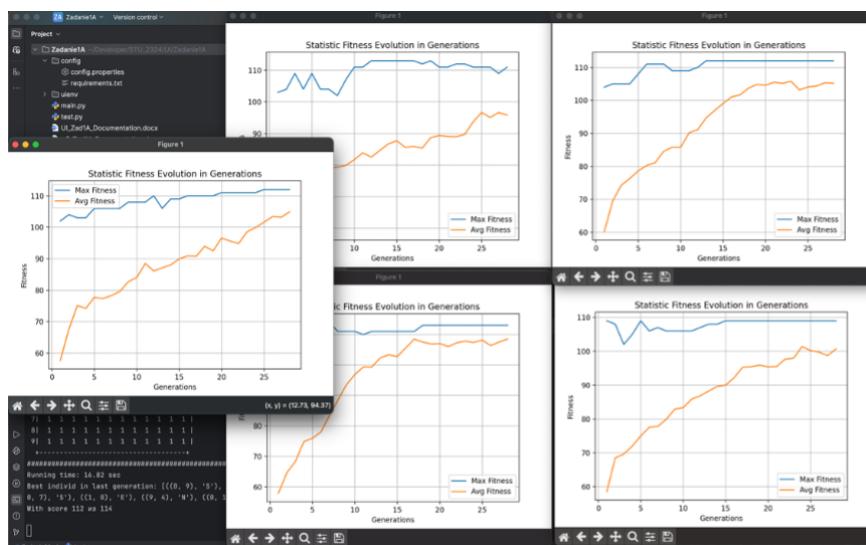
 0 1 2 3 4 5 6 7 8 9 10 11
+-----+
0| 1 1 1 1 1 1 1 1 1 1 1 1 |
1| 1 1 1 1 1 1 2 1 1 1 1 1 1 |
2| 1 2 1 1 1 0 1 1 1 1 1 1 1 |
3| 1 0 1 1 2 0 1 1 1 1 1 1 1 |
4| 1 0 2 1 1 1 1 1 1 1 1 1 1 |
5| 1 1 1 1 1 1 1 1 1 1 1 1 1 |
6| 1 1 1 1 1 1 1 1 1 2 2 1 1 1 |
7| 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |
8| 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |
9| 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 |

Out of garden. Go to another start position.
Position (9, 4) is blocked. Try another.
Position (9, 1) is blocked. Try another.
All start positions are blocked. GAME OVER
```

```
#####
Running time: 161.45 sec
Best individ in last generation: [(0, 6), 'S'], ((9, 10), 'N'), ((9, 11), 'W'), ((9, 0), 'N'), ((0, 7), 'S'), ((0, 8), 'E'), ((7, 11), 'W'), ((8, 0), 'E'), ((0, 5), 'S'), ((9, 2), 'N'), ((1, 0), 'E')
With score 114 vs 114
```

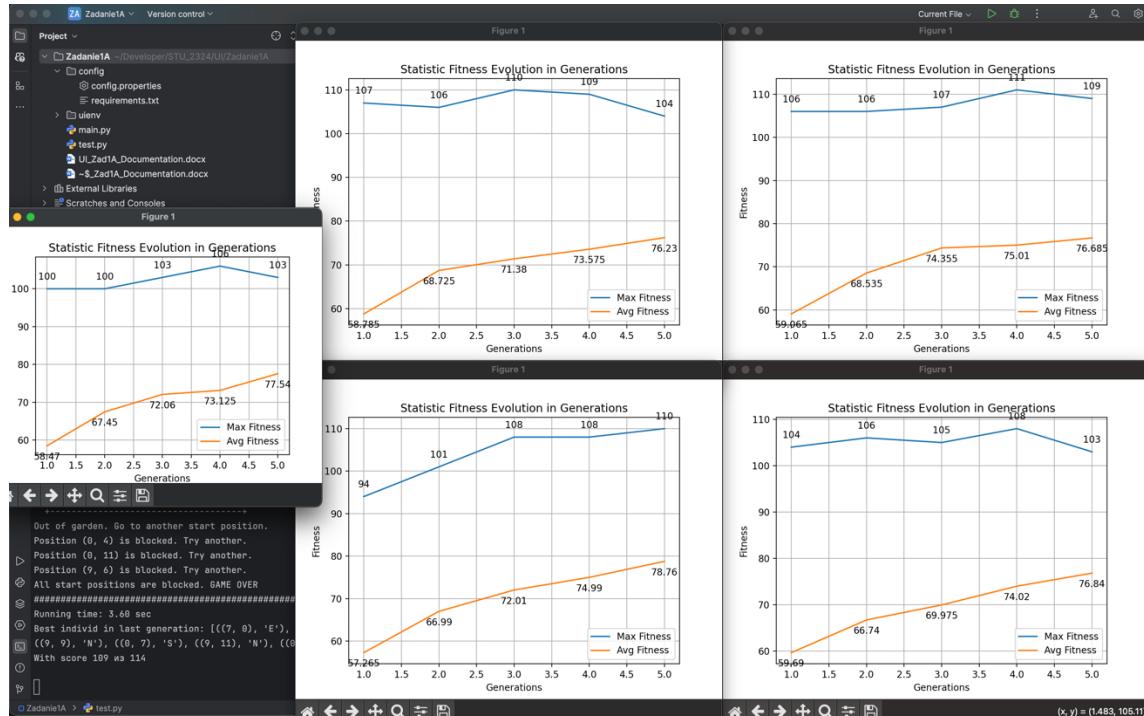
Testing režim:

Spustenie súboru test.py, ktorý spustí main.py 5-krát s rôznymi parametrami seed, a následne budeme mať 5 grafov a výpisov.

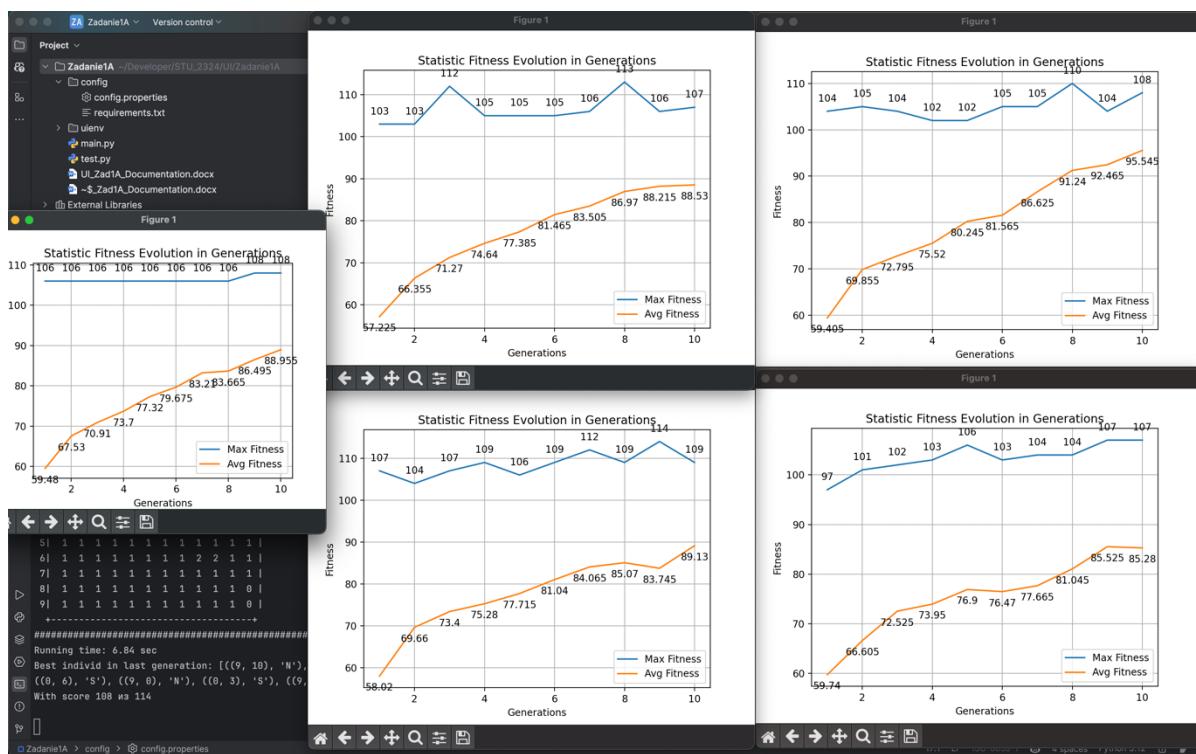


Testing Examples:

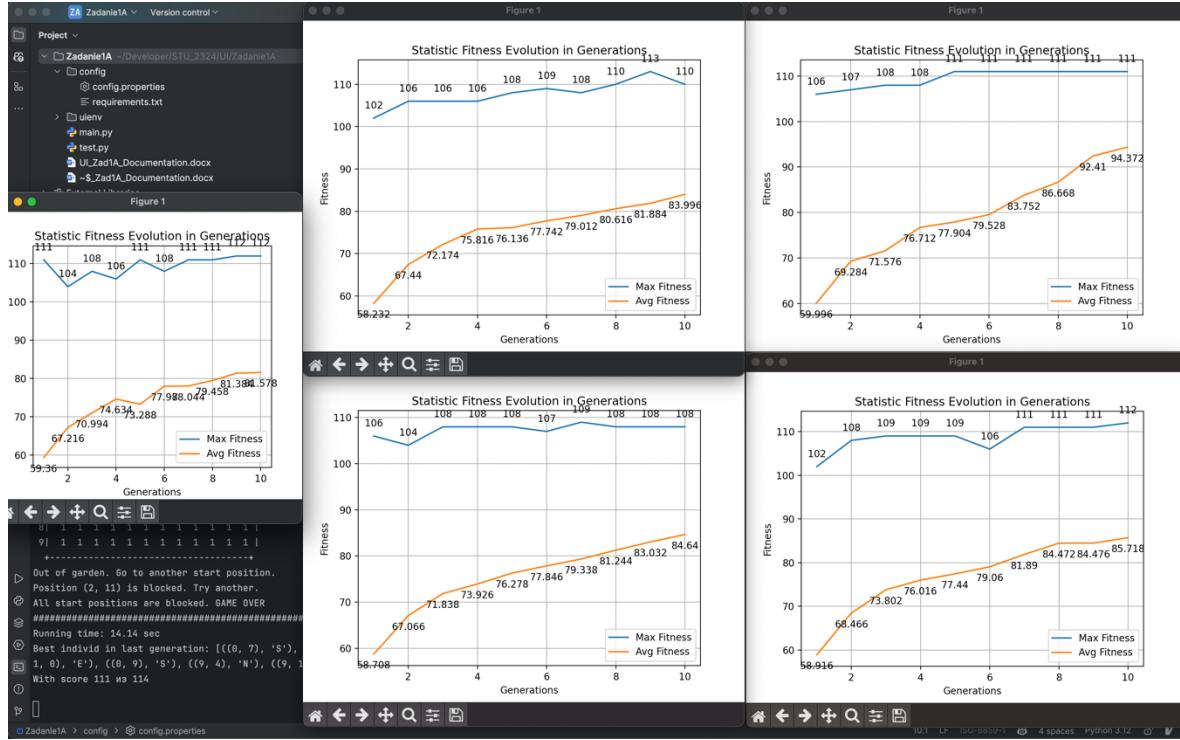
5 generacii pre 200 populacii



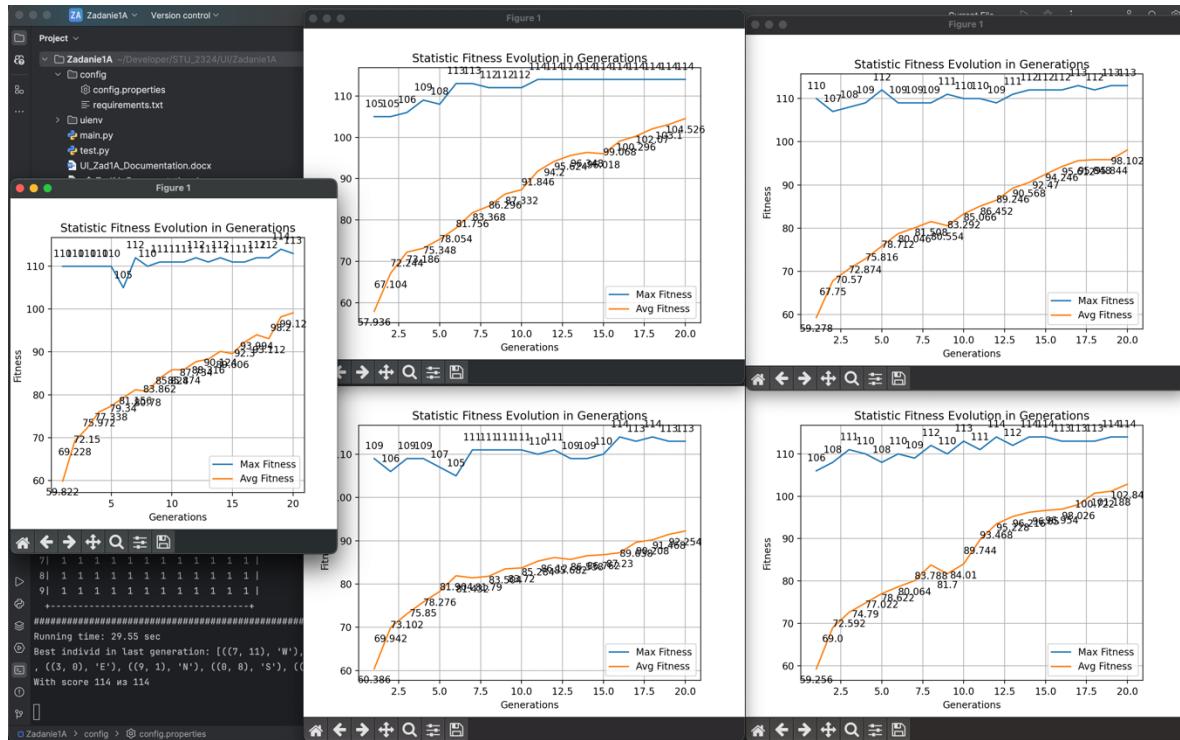
10 generacii pre 200 populacii



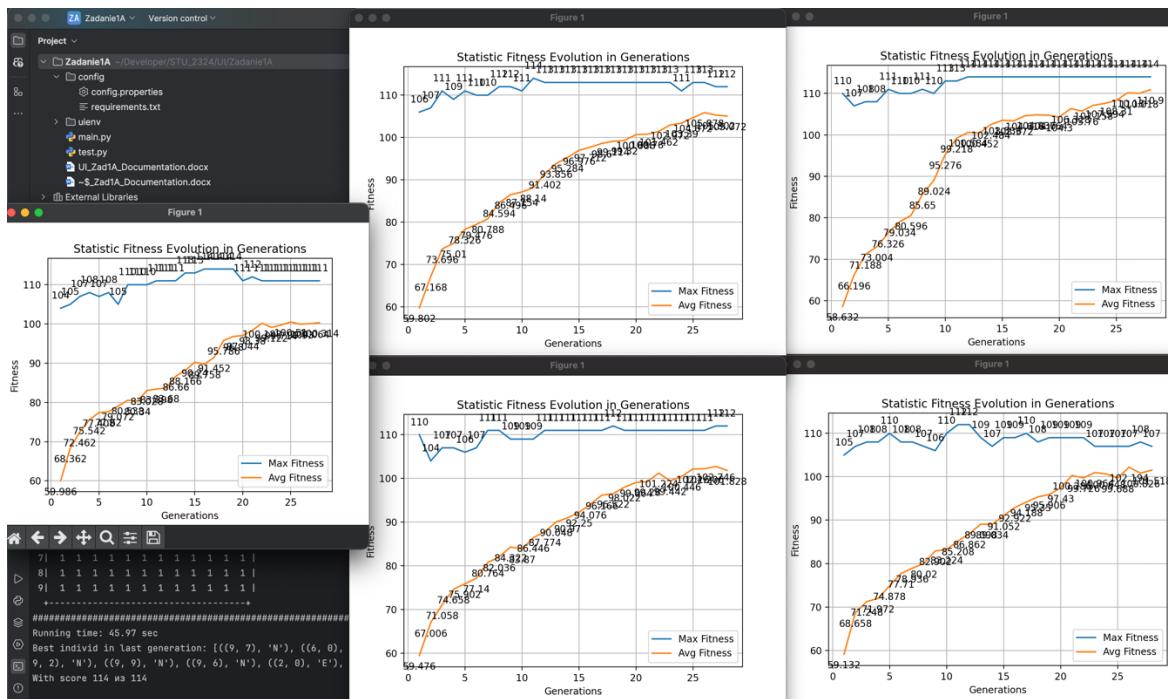
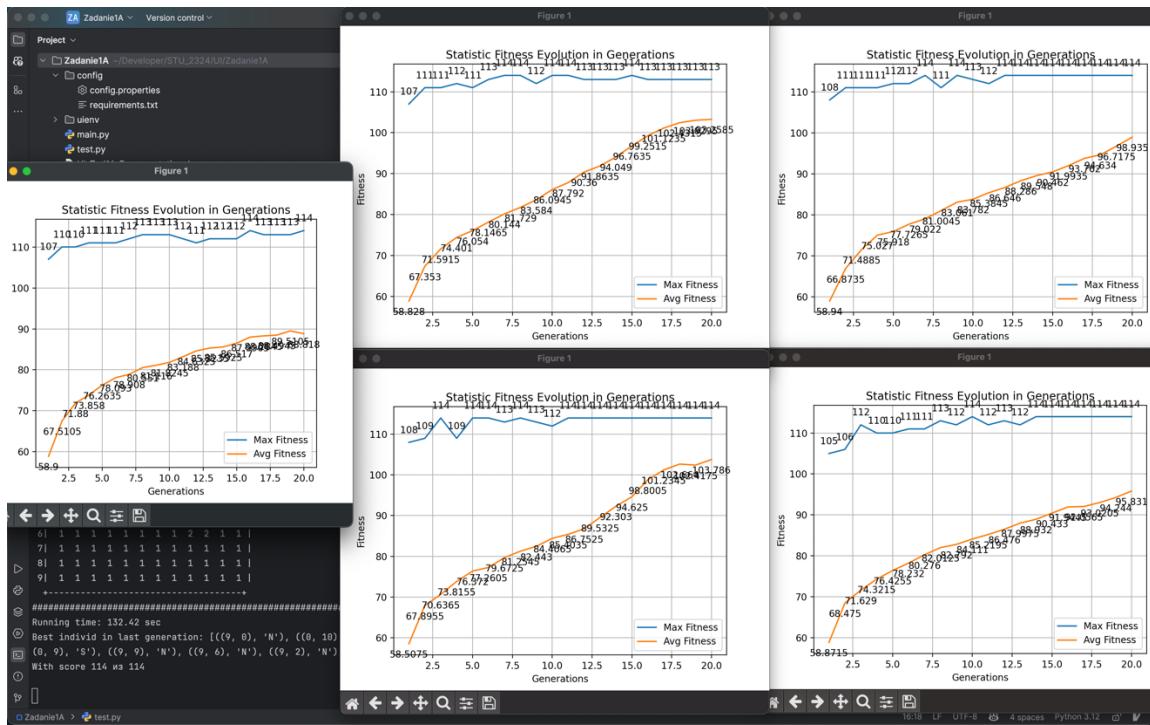
10 generacii pre 500 populacii



20 generacii pre 500 populacii



20 generacii pre 2000 populacii



28 generacii pre 500 populacii

Vysledky testov:

Najlepším a najefektívnejším variantom riešenia tohto problému bude použitie 20 generácií po 400-500 jedincoch, keďže na základe grafov dosiahneme maximálnu hodnotu efektivity a tiež sa nám zlepší priemerná hodnota v porovnaní s ostatnými variantmi testovania, za zmienku stojí aj to, že variant s testovaním 20 generácií a 2000 jedincov je tiež priateľný, ale časová náročnosť je oveľa vyššia.

Čo sa nedá povedať o variante s 28 generáciami a 500, 5/10 generáciami s 200 jedincami, kde výsledky nedosiahli ani maximálne hodnoty, pričom v prvom prípade nadbytku generácií sa naše konečné výsledky zhoršili a v druhom prípade s malým počtom jedincov algoritmus nemal s čím pracovať.

Priemerné výsledky sú 10 generácií s 500 jedincami, kde sme nedostali ideálne výsledky, ale priateľný rast priemernej hodnoty a uspokojivý graf max fitness.

Code Examples & Explanations:

- Fungovanie metody **main** možete prečítať tu:
 - inicializácia premenných z konfiguračného súboru

```
def main():
    config = configparser.ConfigParser()
    config_path = 'config/config.properties'
    config.read(config_path)

    height = config.getint('start-up-config', 'rows')
    width = config.getint('start-up-config', 'columns')
    stones = config.getint('start-up-config', 'stones')
    stones_position = ast.literal_eval(config.get('start-up-config', 'stones_pos'))

    population_size = config.getint('genetic-settings', 'pop_size')
    mutation_rate1 = config.getfloat('genetic-settings', 'mutation_rate1')
    mutation_rate2 = config.getfloat('genetic-settings', 'mutation_rate2')
    crossover_rate = config.getfloat('genetic-settings', 'cross_rate')
    max_generations = config.getint('genetic-settings', 'max_gen')
    tournament_size = config.getint('genetic-settings', 'tournament_size')
    debug_info = config.getboolean('genetic-settings', 'debug_info')

    max_score = (height * width) - stones
    genes_for_position = int((height + width))
    genes_for_path = int(stones)

[genetic-settings]
random_seed = 8148
pop_size = 500
mutation_rate1 = 0.25
mutation_rate2 = 0.20
cross_rate = 0.8
max_gen = 28
tournament_size = 3
debug_info = False

[start-up-config]
rows = 10
columns = 12
stones = 6
stones_pos = [(1,5),(2,1),(3,4),(4,2),(6,8),(6,9)]
```

```
# init garden
garden = init_matrix(width, height, stones_position)

# init population
population = init_population(population_size, genes_for_position, genes_for_path, garden)

# init statistic parameters
max_fitnesses = []
avg_fitnesses = []

💡 # začneme počítať čas
start_time = time.time()
```

- trenning loop alhoritmu

```
# Training loop
for generation in range(max_generations):
    print(f"-----Generation {generation + 1} -----")

    population_scores = [(individual, fitness_metod(individual, garden, debug_info)) for individual in population]

    scores = [score[1][0] for score in population_scores]
    max_fitness = max(scores)
    avg_fitness = sum(scores) / len(scores)
    max_fitnesses.append(max_fitness)
    avg_fitnesses.append(avg_fitness)

    # tournament for population
    selected_population = tournament_selection(population, garden, tournament_size, debug_info)

    # new population
    new_population = []
    for i in range(0, len(selected_population), 2):
        parent1 = selected_population[i]
        parent2 = selected_population[i + 1] if i + 1 < len(selected_population) else selected_population[0]

        child1 = crossover(parent1, parent2, crossover_rate)
        child2 = crossover(parent2, parent1, crossover_rate)

        child1 = mutation_gene_path(child1, mutation_rate1)
        child2 = mutation_gene_path(child2, mutation_rate2)

        new_population.extend([child1, child2])

    # replace old pop. to new one
    population = new_population[:population_size]
    print_population(population, garden, debug_info)
```

- spočítame čas a analiticke data

```
end_time = time.time()
elapsed_time = end_time - start_time
best_individual = max(population_scores, key=lambda x: x[1])
```

- vypíšeme results na obrazovku podla rezimu

```
# print results
fitness_metod(best_individual[0], garden, debug=True)
print("#####")
print(f"Running time: {elapsed_time:.2f} sec")
print(f"Best individual in last generation: {best_individual[0][0]}")
print(f"With score {best_individual[1][0]} vs {max_score}")
print()
plot_fitness_evolution(max_fitnesses, avg_fitnesses)
```

• Fungovanie vypisov v consolu a plot grafy:

○ Metoda plot_fitness_evolution:

1. Najprv vygenerujeme zoznam generácií na základe dĺžky vstupných dát max_fitnesses
2. Potom vykreslime dve krivky: jednu pre max fitness hodnoty a druhú pre avg fitness hodnoty
3. Následne pridáme anotácie k jednotlivým bodom v grafe, aby boli viditeľné konkrétné hodnoty max a avg fitness pre každú gen
4. Pridáme os x (Generations), os y (Fitness), a ostatne info veci
5. Zobrazime aj samotný graf

```
def plot_fitness_evolution(max_fitnesses, avg_fitnesses): 1 usage
    generations = range(1, len(max_fitnesses) + 1)
    plt.plot(*args: generations, max_fitnesses, label='Max Fitness')
    plt.plot(*args: generations, avg_fitnesses, label='Avg Fitness')

    for i, (max_fitness, avg_fitness) in enumerate(zip(max_fitnesses, avg_fitnesses)):
        plt.annotate(text=f'{max_fitness}', xy=(generations[i], max_fitness), textcoords="offset points", xytext=(0,10), ha='center')
        plt.annotate(text=f'{avg_fitness}', xy=(generations[i], avg_fitness), textcoords="offset points", xytext=(0,-15), ha='center')

    plt.xlabel('Generations')
    plt.ylabel('Fitness')
    plt.title('Statistic Fitness Evolution in Generations')
    plt.legend()
    plt.grid(True)
    plt.show()
```

○ Metoda print_garden:

1. Vypočítame počet stĺpcov v matici
2. Zobrazíme hlavičku s číslami stĺpcov a ohraničujúce čiary
3. Vypiše celu ,maticu.
Ak je v bunke aktuálna pozícia,
zobrazí namiesto hodnoty hviezdičku
("*)
4. Do konca vypíšeme spodné
ohraničenie matici.

```
def print_garden(garden, current_position=None): 5 usages
    cols = len(garden[0])

    print()
    print(" " + " ".join(f"{i:2}" for i in range(cols)))
    print(" " + "---" * cols + "+")

    for y, row in enumerate(garden):
        row_display = []
        for x, value in enumerate(row):
            if current_position == (y, x):
                row_display.append("*")
            else:
                row_display.append(f"{value:2}")
        print(f"{y:2}| " + " ".join(row_display) + " |")
    print(" " + "---" * cols + "+")
```

○ Metoda print_population:

1. Iterujeme kazdeho individa v populácií.
2. Pre každého individa vypíšeme info:
 1. Počiatočnú pozíciu
 2. Cestu génu
 3. Vypočítame score individa pomocou funkcie “fitness_metod”
4. Nakoniec vypíše score individa a oddelí všetko znakom “#“ pre ďalšieho indovida

```
def print_population(population, garden, debug): 1 usage
    for idx, individ in enumerate(population):
        start_position, gene_path = individ
        print(f"Individual {idx + 1}:")
        print(f" *-Start Position: {start_position}")
        print(f" *-Gene Path: {gene_path}")

        score = fitness_metod(individ, garden, debug)
        print(f" *-Score: {score}")
        print("#####")
```

- **Metody pre inicializaciu startovych hodnot:**

- Metoda init_matrix:

- Jednoducho vytvoríme maticu s aktuálnym počtom stĺpcov a riadkov s predvolenou hodnotou ‘0’.
 - Spoločne tam, kde má byť kameň, napíšeme ‘2’
 - Vrátime maticu pomocou funkcie return

```
def init_matrix(columns, rows, stones_pos): 1 usage
    garden_matrix = [[0 for _ in range(columns)] for _ in range(rows)]
    for stone in stones_pos:
        row, col = stone
        garden_matrix[row][col] = 2
    return garden_matrix
```

- Metoda generate_genes_position:

Vytvorí nabor počiatočných génov poyicii na hraniciach matice.

De najprv vypočítame počet riadkov a stĺpcov v matice.

Potom sa vytvorime štyri zoznamy počiatočných pozícii:

1. Horná (prvé riadky) so smermi „S“ (juh),
2. Dolná so smermi „N“ (sever),
3. Lavá (stĺpce) so smermi „E“ (východ),
4. Pravá so smermi „W“ (západ).

Tieto pozície sú spojeme do jedného zoznamu ‘start_garden_pos’ ktorý obsahuje súradnice hranice záhrady a smery pre každý prvok.

Z tohto zoznamu sa náhodne vyberieme ‘n’ počiatočných pozícii pomocou funkcie random

Výsledkom metódy je vrátenie TUPLE vybraných pozícií (jedna pre každý gén).

```
def generate_genes_position(garden, n):| 2 usages
    rows = len(garden)
    cols = len(garden[0])

    start_garden_pos = [((0, col), 'S') for col in range(cols)] #up
    start_garden_pos += [((rows - 1, col), 'N') for col in range(cols)] #down
    start_garden_pos += [((row, 0), 'E') for row in range(1, rows - 1)] #left
    start_garden_pos += [((row, cols - 1), 'W') for row in range(1, rows - 1)] #right

    random_positions = random.sample(start_garden_pos, n)
    return tuple(random_positions for _ in range(n))
```

- Metoda generate_gene_path:

Generuje cestu zloženú zo sekvencie náhodných smerov

Pre každý z N génov sa náhodne vyberie jeden z dvoch smerov:

1. 0 - pohyb doprava
2. 1 - pohyb doľava

Pre každý gén sa vygeneruje a vráti TUPLE náhodných smerov.

```
def generate_gene_path(n): # 0 - right, 1 - left 1 usage
    return tuple(random.choice([0, 1]) for _ in range(n))
```

- Metoda init_individual:

Táto metóda vytvorí individua so zadanými génmi pre počiatočnú pozíciu (start_path) a cestu (gene_path)

- Náhodná štartovacia pozícia sa vyberie pomocou metódy generate_genes_position(garden, genes_for_position). Metóda vráti zoznam možných štartovacích pozícií na hraniciach matice. Z neho sa náhodne vyberie jedna pozícia so smerom.
- Pomocou metódy generate_gene_path(genes_for_path) sa vygeneruje náhodná cesta s dĺžkou genes_for_path. Táto cesta určuje smery pohybu (doprava alebo doľava).

Vráti sa tuple pozostávajúci z počiatočnej pozície a cesty.

Vrátená hodnota bude TUPLE v tvare (start_position, genes_path),

```
def init_individual(genes_for_position, genes_for_path, garden): 1 usage
    start_position = random.choice(generate_genes_position(garden, genes_for_position))
    gene_path = generate_gene_path(genes_for_path)
    return start_position, gene_path
```

○ Metoda init_population:

Táto metóda vytvára počiatočnú populáciu viacerých individov.

Algoritmus:

- Na vytvorenie každého individua sa použije metóda init_individual().
- Pre každého individua sa vygeneruje náhodná počiatočná pozícia a cesta
- Vytvorí zoznam všetkých individov (s dĺžkou population_size).

Zoznam individov, kde každý prvok je tuple (start_position, gene_path).

```
def init_population(population_size, genes_for_position, genes_for_path, garden): 1 usage
    individuals = [init_individual(genes_for_position, genes_for_path, garden) for _ in range(population_size)]
    return individuals
```

• Metody Evolučného algoritmu:

Pri písaní algoritmu, konkrétnie metódy fitness, som potreboval ďalšie metódy na vytvorenie a implementáciu logiky presunu jedincov v matici podľa logiky programu.

○ Metoda move:

Táto metóda je zodpovedná za pohyb objektu v určitom smere

- x, y: aktuálne súradnice objektu v matici.
- direct: aktuálny smer pohybu

Logika je nasledovná

- Ak je smer „N“ (sever), objekt sa pohybuje smerom nahor (súradnica y sa zmenšuje).
- Ak je smer „E“ (východ), objekt sa pohybuje doprava (súradnica x sa zväčšuje).
- Ak je smer „S“ (juh), objekt sa posunie nadol (súradnica y sa zväčší).
- Ak je smer „W“ (západ), objekt sa posunie doľava (súradnica x sa zmenší).

Zrozumiteľný príklad:

Ak sa objekt nachádza na súradničiach (2, 2) a smeruje na sever („N“), po volaní move(2, 2, „N“) sa presunie na súradnice (1, 2).

```
direction = ['N', 'E', 'S', 'W']
def move(x, y, direct):
    if direct == 'N':
        return y - 1, x # up
    elif direct == 'E':
        return y, x + 1 # right
    elif direct == 'S':
        return y + 1, x # down
    elif direct == 'W':
        return y, x - 1 # left
```

- Metoda turn_right\left:

Táto metóda zmení aktuálny smer otáčania objektu na otáčanie vpravo/ vľavo

Logika je nasledovná

- Zistí index aktuálneho smeru v zozname direction = ['N', 'E', 'S', 'W'].
- Vráti nasledujúci smer v smere hodinových ručičiek. Prechod sa vykoná výpočtom indexu $(idx +/- 1) \% 4$.
- Index sa zvýši/zmenší o 1 a berie sa modulo 4, aby cyklus pokračoval v kruhu!

Zrozumiteľný príklad:

Ak je aktuálny smer 'N', po volaní turn_right('N') je výsledkom 'E'.

Ak je aktuálny smer 'W', po volaní turn_left('W') je výsledkom 'S'.

```
def turn_right(current_direction): 1 usage
    idx = direction.index(current_direction)
    return direction[(idx + 1) % 4]
def turn_left(current_direction): 1 usage
    idx = direction.index(current_direction)
    return direction[(idx - 1) % 4]
```

○ Metoda fitness_metod:

Metóda vyhodnocuje výkon jednotlivca v kontexte danej matice a vracia počet prejdených pozícií a stav matice po akciách jednotlivca.

Logika je nasledovná:

- Vytvorí sa deep kópia matice záhrady, aby sa počas behu nemenil originál.
- Získajú sa počiatočné pozície a cesta (gény) jedinca.
- Premenné pre inicializáciu:
 1. index_pos a index_gene - na sledovanie aktuálnych indexov počiatočných pozícií a génov.
 2. (y, x), D - na získanie aktuálnej východiskovej polohy a smeru.
 3. path_history - na sledovanie pohybov.
 4. current_direction - na uloženie aktuálneho smeru.
 5. score - na výpočet skóre.
 6. all_directions_blocked - na sledovanie, či sú všetky smery zablokované.
 7. direction_attempts - na sledovanie pokusov o pohyb do jednotlivých smerov.

```
def fitness_metod(person, matrix, debug): 4 usages
    garden = copy.deepcopy(matrix)
    start_positions, gene_path = person
    index_pos = 0
    index_gene = 0
    (y, x), D = start_positions[index_pos]
    path_history = []
    current_direction = D
    score = 0
    all_directions_blocked = True
    direction_attempts = {directions: 0 for directions in range(len(gene_path))}
```

- Kontrola východiskovej pozície:
 1. Ak je východisková pozícia zablokovaná (hodnota v matici je 1 alebo 2), presunie sa na ďalšiu pozíciu.

```
if garden[y][x] in (1, 2):
    debug_print("I cant start in this position: " + str((y, x)))
    index_pos += 1
(y, x), D = start_positions[index_pos]
```

- Hlavný cyklus:

Pohyb: kým individ nevyčerpá východiskové pozície:

Skontroluje, či jedinec dosiahol koniec génov (index_gene >= len(gene_path)). Ak áno, vynuluje index génu na 0.

Ak je aktuálna pozícia voľná (hodnota v matici je 0):

Aktualizujte stav matice, pridajte skóre a presuňte sa na novú pozíciu.

```
if garden[y][x] == 0:
    debug_print("I can move in " + str((y, x)) + ", continue.")
    garden[y][x] = 1
    score += 1
    path_history.append((y, x), current_direction)
    (y, x) = move(x, y, current_direction)
    if debug:
        print_garden(garden)
    direction_attempts[index_gene] = 0
```

Ak aktuálna pozícia nie je voľná:

Vykoná sa cyklus 4 pokusov (MAX_CYCLES) na nájdenie iného smeru.

V závislosti od aktuálneho génu sa otočí doľava alebo doprava a pokúsi sa presunúť.

```
while cycle_count < MAX_CYCLES:
    if index_gene >= len(gene_path):
        index_gene = 0

        if gene_path[index_gene] == 0: # направо
            current_direction = turn_right(current_direction)
            debug_print("Turn right: " + str(current_direction))
        elif gene_path[index_gene] == 1: # налево
            current_direction = turn_left(current_direction)
            debug_print("Turn left: " + str(current_direction))
        else:
            print("Error: invalid gene")
            break

    (temp_y, temp_x) = move(prev_x, prev_y, current_direction)
```

Ak je presun úspešný, aktualizujú sa súradnice a história presunov.

Ak sú všetky smery zablokované, zobrazí sa správa o ukončení hry.

```
if temp_y < 0 or temp_y >= len(garden) or temp_x < 0 or temp_x >= len(garden[0]):  
    index_pos += 1  
    break  
  
if garden[temp_y][temp_x] == 0: # Успешный шаг  
    y, x = temp_y, temp_x  
    path_history.append((y, x), current_direction)  
    index_gene += 1  
    debug_print("Go to " + str((y, x)) + " position.")  
    all_directions_blocked = False  
    direction_attempts[index_gene - 1] = 0 if index_gene > 0 else 0  
    if debug:  
        print_garden(garden)  
    break  
else:  
    debug_print(f"Hit in ({temp_y}, {temp_x}), return to ({prev_y}, {prev_x}).")  
    y, x = prev_y, prev_x  
    current_direction = prev_direction  
    cycle_count += 1  
    index_gene += 1
```

Kontrola matíc mimo hraníc:

Ak sa jedinec pohybuje mimo hraníc matice, vykoná sa presun na ďalšiu východiskovú pozíciu.

Ak sú všetky východiskové pozície zablokované, hra sa skončí.

```
if all_directions_blocked:  
    debug_print(f"All direction around ({prev_y}, {prev_x}) blocked. GAME OVER")  
    if debug:  
        print_garden(garden)  
    break  
  
if index_gene > 0:  
    direction_attempts[index_gene - 1] += 1  
  
if index_gene > 0 and direction_attempts[index_gene - 1] > len(gene_path):  
    debug_print("Alot of try to go to similar position. GAME OVER")  
    if debug:  
        print_garden(garden)  
    break
```

Výsledok vykonania je:

skóre: počet bodov, ktoré jednotlivec získal

garden: aktuálny stav matice po všetkých ťahoch.

- Metoda tournament_selection:

Táto metóda vykonáva výber jedincov z populácie pomocou turnajového vzoru.

Logika je nasledovná

- Na uloženie vybraných jedincov sa inicializuje prázdný zoznam vybraných jedincov.
- Pre každého jedinca v populácii:
 1. Turnaj_veľkosť jedinci z populácie sú náhodne vybraní.
 2. Títo jedinci sú zoradení podľa hodnoty fitness, ktorá sa vypočíta pomocou funkcie fitness_metod (pozri metódu vyššie).
 3. Jedinec s najvyššou hodnotou fitness sa vyberie ako víťaz turnaja.
- Najlepší jedinci sa pridajú do zoznamu vybraných jedincov.

Výstupom je zoznam vybraných jedincov, z ktorých každý má najlepšiu fitness vo svojej skupine.

```
def tournament_selection(population, garden ,tournament_size,debug): 1 usage
    selected = []
    for _ in range(len(population)):
        tournament = random.sample(population, tournament_size)
        tournament_sorted = sorted(tournament, key=lambda individual: fitness_metod(individual, garden,debug), reverse=True)
        best_individual = tournament_sorted[0]
        selected.append(best_individual)
    return selected
```

○ Metoda crossover:

Táto metóda vykonáva kríženie medzi dvoma rodičmi s cieľom vytvoriť potomstvo (budúcu populáciu).

Logika je nasledovná

- Náhodne sa skontroluje, či sa vykoná vzor kríženia. Ak je náhodné číslo väčšie ako miera, rodičovský jedinec sa vráti nezmenený.
- Vyberú sa počiatočné pozície a cesty oboch rodičov.
- Vygenerujú sa náhodné body pre kríženie:
 1. crossover_point_pos pre počiatočné pozície.
 2. crossover_point_path pre cestu.

Nové počiatočné pozície a cesty pre potomka sa vytvoria kombináciou častí z oboch rodičov s použitím vygenerovaných bodov.

Výsledkom je tuple počiatočných pozícíí a ciest pre potomka.

```
def crossover(mother, father, rate): 2 usages
    if random.random() > rate:
        return mother

    mother_start_pos, mother_gene_path = mother
    father_start_pos, father_gene_path = father

    crossover_point_pos = random.randint( a: 1, len(mother_start_pos) - 1)
    crossover_point_path = random.randint( a: 1, len(mother_gene_path) - 1)

    child_start_pos = mother_start_pos[:crossover_point_pos] + father_start_pos[crossover_point_pos:]
    child_gene_path = mother_gene_path[:crossover_point_path] + father_gene_path[crossover_point_path:]

    return child_start_pos, child_gene_path
```

Mutácie pre génovú rozmanitosť v populácii

- Metoda mutation_gene_path a mutation_position_path:

Tieto metódy vykonávajú mutáciu cesty a počiatočných pozícií jedincov

Logika je nasledovná

- Cesta alebo počiatočné pozície sa prevedú na zoznam mutated_gene_path / mutated_start_positions.
- Pre každý gén v ceste alebo počiatočných pozíciách sa náhodne skontroluje, či sa zmení podľa zadanej miery. Ak áno, gén sa náhodne vyberie z dvoch možných 0 alebo 1 a v druhom prípade nahradíme štartovaciu pozíciu novou pomocou metódy generate_genes_position

Výsledkom je tuple počiatočných pozícií a zmutovanej cesty.

```
# Mutation
def mutation_gene_path(person, rate):  2 usages
    start_positions, gene_path = person

    mutated_gene_path = list(gene_path)
    for i in range(len(mutated_gene_path)):
        if random.random() < rate:
            mutated_gene_path[i] = random.choice([0, 1])

    return start_positions, tuple(mutated_gene_path)
def mutation_position_path(person, rate, matrix):
    start_positions, gene_path = person

    mutated_start_positions = list(gene_path)
    for i in range(len(mutated_start_positions)):
        if random.random() < rate:
            mutated_start_positions[i] = random.choice(generate_genes_position(matrix, n: 1))

    return tuple(mutated_start_positions), gene_path
```

Zhodnotenie:

V tejto úlohe som sa dozvedel o evolučnom algoritme, jeho prístupoch a metódach. Návrh tohto algoritmu od fáz vytvorenia populácie, testovania, výberu a mutácií. Dozvedel som sa o rôznych metódach výberu populácie, ako je turnajový výber, ruleta a zoradenie.

Pri testovaní riešenia tohto problému som použil metódu turnajovej selekcie a dve rôzne mutácie a zistil som, že pre uspokojivé výsledky by sme mali určiť aspoň 10 generácií a minimálnu populáciu 200 jedincov.