Programowanie Genetyczne – nauka chodzenia

Celem projektu jest stworzenie systemu, w którym osobniki uczą się poruszać w prawo w środowisku fizycznym.

Do symulacji fizyki zostanie wykorzystana biblioteka **Pymunk**, która umożliwia dokładne odwzorowanie zasad dynamiki, takich jak grawitacja, tarcie czy kolizje. Wizualizacja działania systemu zostanie zrealizowana przy użyciu biblioteki **Pygame**, co pozwoli na intuicyjne przedstawienie wyników symulacji oraz analizy ewolucji ruchu osobników.

Celem projektu jest stworzenie populacji osobników sterowanych przez programy genetyczne, które uczą się wykonywać ruch w prawo. Populacja będzie ewoluować poprzez iteracyjne dopasowywanie funkcji sterującej. Projekt ten stanowi praktyczne zastosowanie programowania genetycznego w środowisku symulacyjnym.

Każdy osobnik skała się z korpusu oraz 2 nóg. Na nogę przypadają 2 stawy. Nogi łączą się z korpusem w jego dolnych rogach.

Generalny opis kodu

brain – drzewo Node-ów, które służą do sterowania osobnikiem (Individual)

Node składa się funkcji, którą wykona oraz lewego i prawego dziecka. Domyślną funkcją jest zwrócenie lewego dziecka.

Sposób użycia:

float(ind.brain)

__create_random_tree – za pomocą techniki full generuje drzewo Node-ów

Najniższe liście, to podanie zmiennej albo stałej. Do zmiennych należą wysokość korpusu osobnika oraz (pozycja w osi X lewego dolnego stawu - pozycja w osi X prawego dolnego stawu)

Od korzenia do prawie najniższego poziomu drzewa występują takie funkcje, jak:

- **rotateAcLeft(x: Node, y: Node) -> float**
    - Ustawia prędkość obrotową lewego górnego stawu (AC), łącząc korpus z pierwszą częścią lewej nogi.
- **rotateBaLeft(x: Node, y: Node) -> float**
    - Steruje prędkością obrotową lewego dolnego stawu (BA), łącząc pierwszą i drugą część lewej nogi.
- **rotateAcRight(x: Node, y: Node) -> float**
    - Kontroluje prędkość obrotową prawego górnego stawu (AC), łącząc korpus z pierwszą częścią prawej nogi.
- **rotateBaRight(x: Node, y: Node) -> float**
    - Steruje prędkością obrotową prawego dolnego stawu (BA), łącząc pierwszą i drugą część prawej nogi.
- **addDegree(x: Node, y: Node) -> float**
    - Sumuje kąty z dwóch węzłów, zwiększając kąt obrotu stawu.
- **substractDegree(x: Node, y: Node) -> float**
    - Odejmuje jeden kąt od drugiego, zmniejszając kąt obrotu stawu.
- **condition(x: Node, y: Node) -> float**
    - Wykonuje węzeł x, jeśli zwrócił wartość dodatnio zwraca ją. W przeciwnym wypadku zwraca y.

Funkcje odpowiedzialne za obrót stawów liczą średnią z inputów, a potem ograniczają wartość średnie, aby mieściła się w ramach określanych jako zmienna statyczna

Gramatyka

Program sterujący jest drzewem obiektów klasy Node, która składa się z lewego, prawego dziecka oraz funkcji, przez którą je przepuści. Aby skorzystać ze zmiennej używamy funkcji, która ją zwróci. Dla stałych istnieje funkcja default, która zwraca lewe dziecko, więc wystarczy tylko przypisać jemu wartość.

W postaci gramatyki formalnej to <{A,F},{zbiór funkcji,(,),X},P,S>, gdzie

X - zbiór stałych

P = {

S → A,

A → F(A, A),

A → default(X, 0.0),

A → getHeight(0, 0),

A → getSpread(0, 0),

F → rotateAcLeft,

F → rotateBaLeft,

F → rotateAcRight,

F → rotateBaRight,

F → addDegree,

F → substractDegree,

F → condition

}

Proces ewolucji:

- Przez turniej wybieramy liczba_osobników_w_populacji * crossover_rate rodziców i wyjmujemy je z populacji.
- Przeprowadzamy crossover.
- Przeprowadzamy negatywny turniej z negative_tournament_rate, przez który wyłaniamy osobniki do odrzucenia i je wyrzucamy z populacji.
- Przeprowadzamy mutacje na liczba_osobników_w_populacji * crossover_rate.
- Dodajemy do populacji rodziców, dzieci i osobniki po mutacji.
- Całość sortujemy rosnąco po (ilości liście, maksymalnej głębokości), jeśli aktualna aktualna ilość osobników się nie zgada, to usuwamy z końca lub generujemy nowego, aby wyrównać, to.

Funkcja przystosowania:

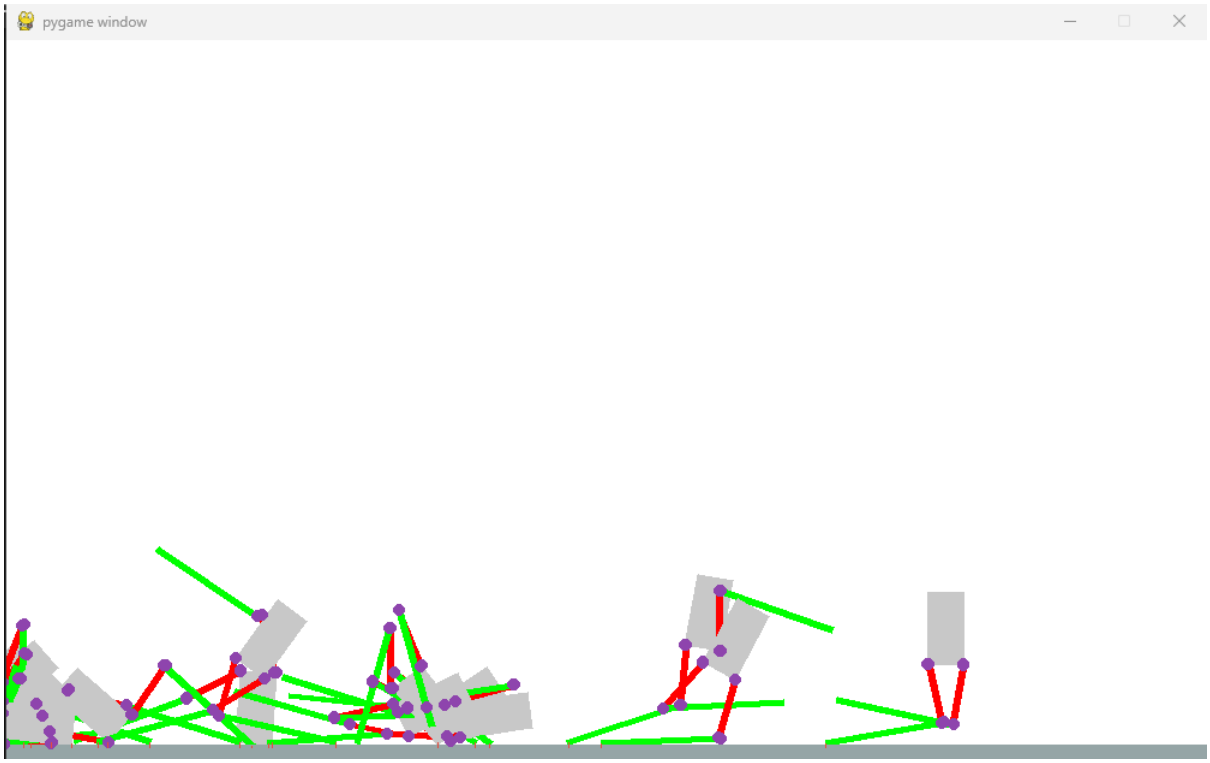
Bazowo jest to odległość w osi X pokonana od punktu startowego w prawą stronne.

Warunek końca epoki:

Limit czasu określany w sekundach (max_TTL).

Problem 1

Płaska powierzchnia



Podeszliśmy do tego problemu na 2 sposoby

W różnice w parametrach

Parametr	Sposób 1	Sposób 2
max_depth	8	8
max_TTL (w sekundach)	50	30
generation	50	50
mutation_rate	0.25	0.25
crossover_rate	0.25	0.25
mutation_rate_critic	-	0.5
crossover_rate_critic	-	0.0
negative_tournament_rate	0.25	0.25

Ale najbardziej znaczące różnice pomiędzy są w podejściu do problemu stagnacji.

Jako stagnacje rozumiemy sytuacje, w której przez określoną liczbę generacji (w obu przypadkach 5) pod rząd

fitness najlepszego osobnika z pokolenia – fitness najlepszego osobnika z całego procesu < epsilon

W pierwszym przypadku, gdy napotykaliśmy stagnacje usuwaliśmy 2 najgorsze oraz 2 najlepsze osobniki.

W drugim sposobie zmienialiśmy mutation oraz crossover rate na ich wersje critic na 1 ewolucje.

Crossover

Wykonujemy głęboką kopie rodziców, po czym w każdej z kopii wybieramy po punkcie.

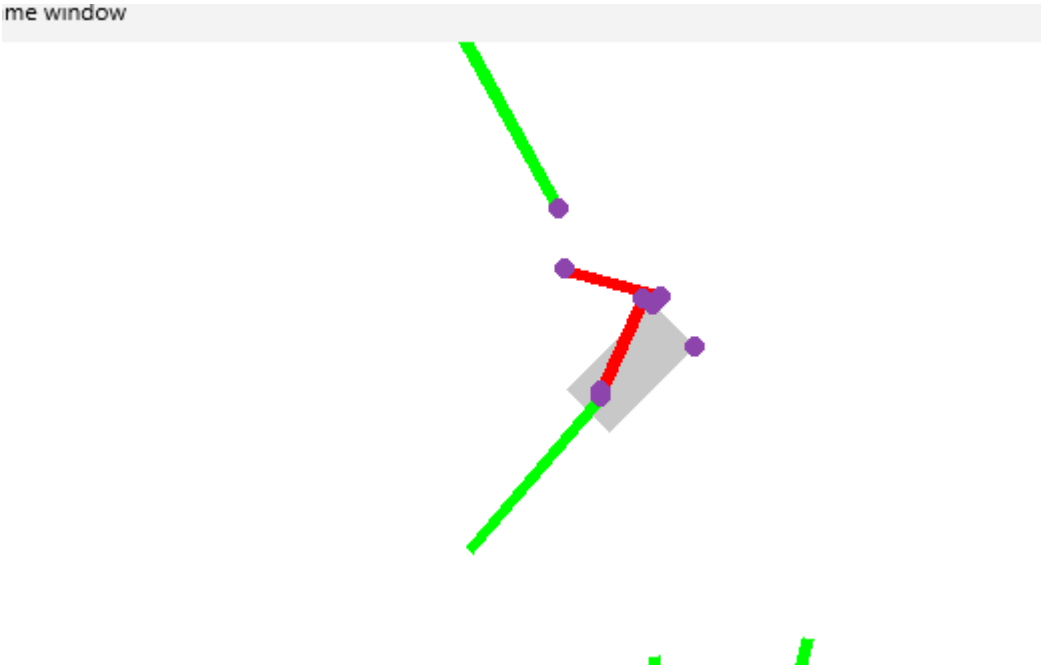
W wybranych punktach dokonujemy podmiany gałęzi drzew.

Punkty wymiany to losowe Node na głębokość od crossover_min_depth (4) do crossover_max_depth (6)

Mutacja

Wybieramy losowy Node na głębokości od mutation_min_depth (2) do mutation_max_depth (6), po czym wybieramy 1 z jego dzieci, które potem podmieniamy wygenerowany przez __create_random_tree drzewem o głębokości wylosowane z pomiędzy 1 i mutation_max_depth (6).

Anomalie



Rozumiemy, za tym słowem przypadki, w którym osobnik bez jasnej przyczyny zrobił coś niezrozumianego.

Oprócz szukania potencjalnych przyczyn oraz ich naprawiania (np. dodanie limitów prędkości obrotu, napisanie własnej funkcji do kopiowania głębokiego drzewa sterującego), staraliśmy się wyłapywać osobniki, które mogą powodować takie anomalie.

W pierwszym sposobie jedynie powstrzymywaliśmy, aby osobnik przez anomalie nie został najlepszym. Robiliśmy to przez sprawdzanie, przy szukaniu najlepszego osobnika, czy fitness potencjalnego osobnika nie jest większy od fitnessu aktualnego najlepszego o ponad ustalony próg (równy 2000)

W drugim przypadku w funkcji, która usuwała osobniki wyrzucające błędy, dodatkowo usuwaliśmy osobniki, który spełniały warunek w pierwszym sposobie.

Wyniki są zapisane w plikach:

best_ind_base_1.txt

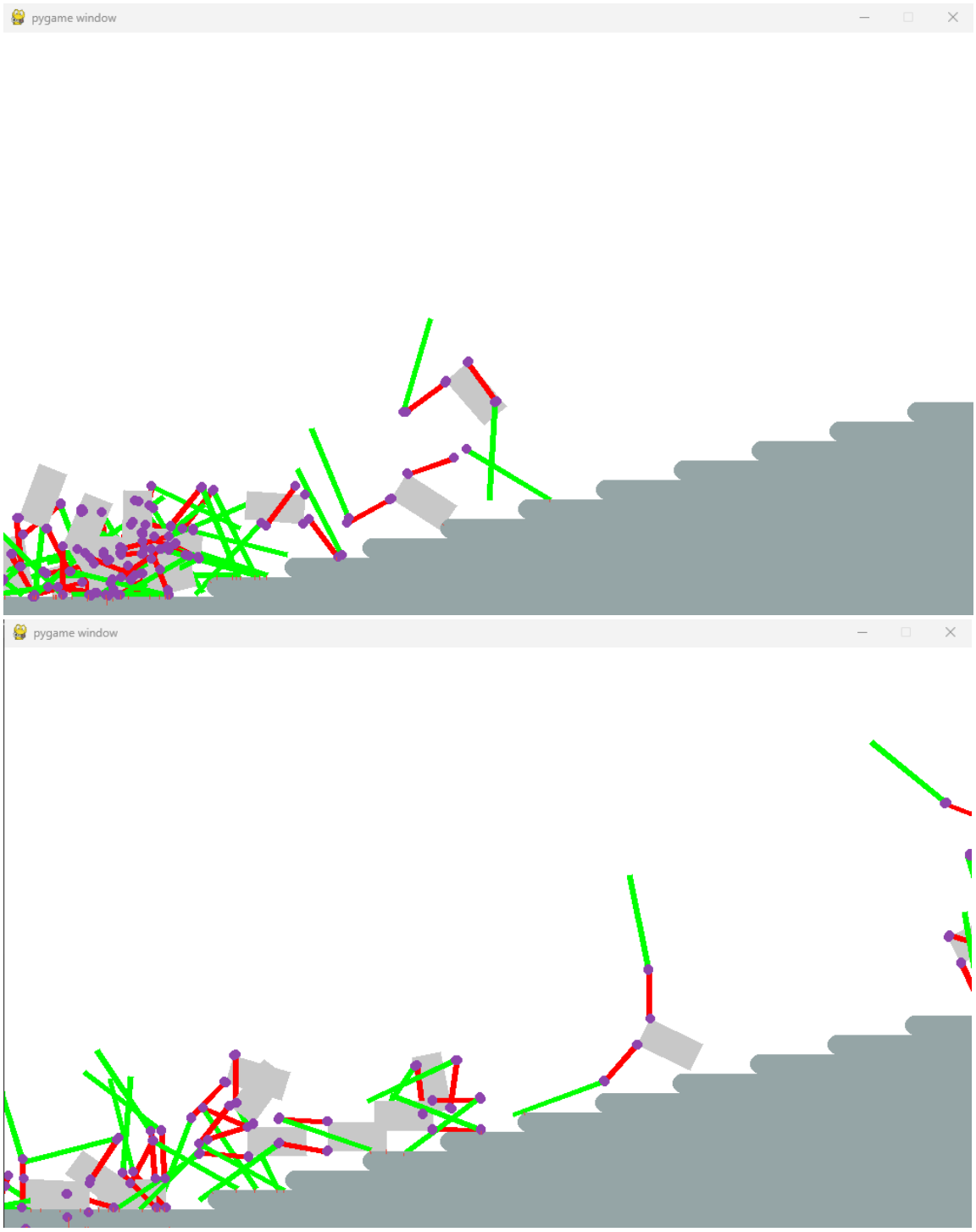
population_base_1.txt

best_ind_base_2.txt

population_base_2.txt

Problem 2

Łagodne schody i inne zmiany środowiska (15/8 razy większa siła grawitacji)



Pierwszą generacją są osobniki wczytane z plików best_ind_base_2.txt oraz population_base_2.txt, czyli pierwsze pokolenia zawiera 51 osobników.

Różnice między sposobem 2 w problemie 1, a rozwiązanie problemu 2.

Rożnice w parametrach:

Parametr	Poprzednio	Teraz
max_depth	8	10
max_TTL (w sekundach)	30	50
generation	50	50
mutation_rate	0.25	0.25
crossover_rate	0.25	0.25
mutation_rate_critic	0.5	0.3
crossover_rate_critic	0.0	0.0
liczba generacji do stagnacji	5	7
negative_tournament_rate	0.25	0.25
negative_tournament_rate (w wypadku stagnacji)	0.5	0.3
mutation_min_depth	2	4
mutation_max_depth	6	6
crossover_min_depth	4	6
crossover_max_depth	6	8
rotation_rate_up_limit	15	30
rotation_rate_down_limit	-15	-30
Próg anomalii	2000	750

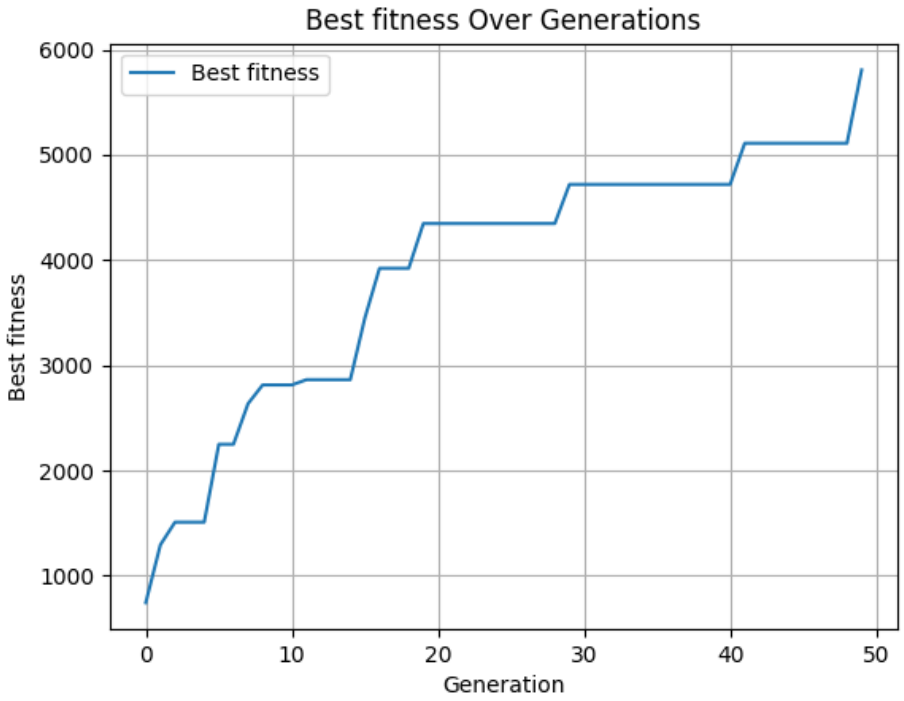
Funkcja przystosowania:

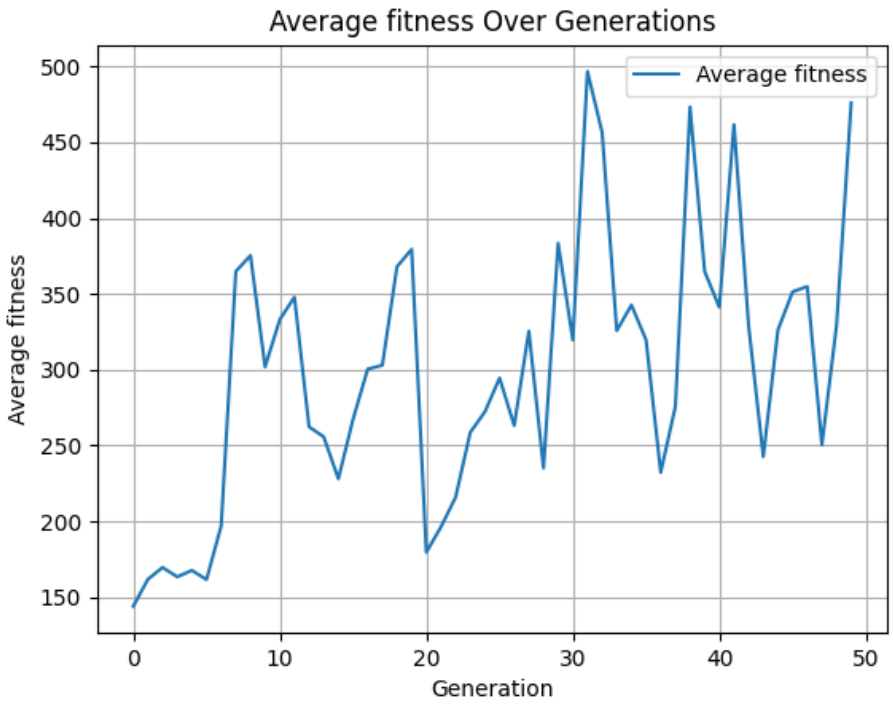
Teraz jest to maksymalna odległość w osi X pokonana od punktu startowego w prawą stronne. Zmiana ma na celu nie eliminowanie osobników, którym dobrze poszło, ale w pewnym momencie odbił się od schodka i poleciał w drugą stronne.

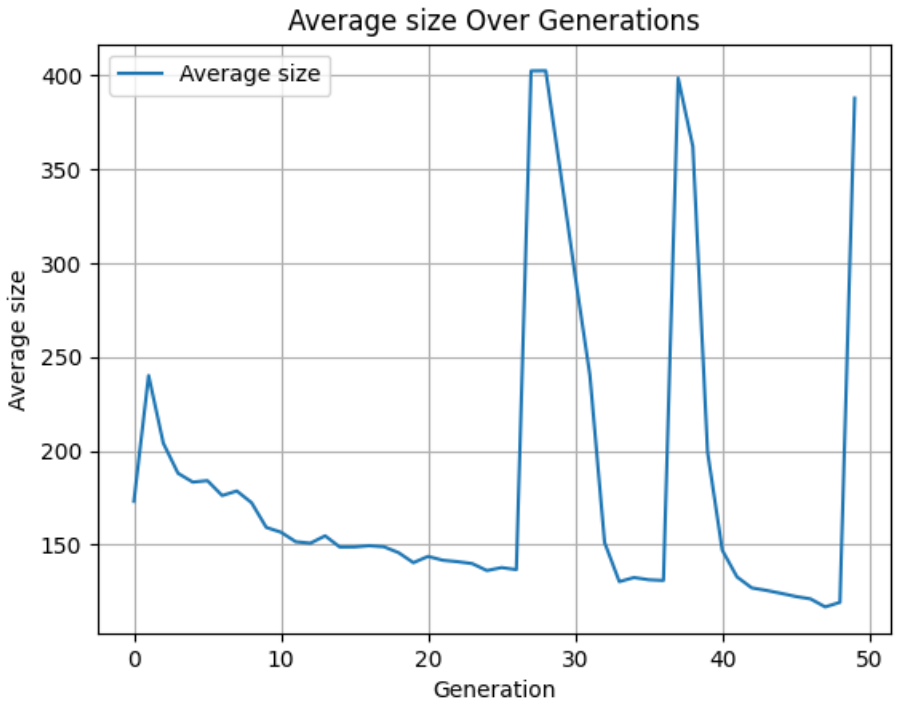
Mutacja

Wybieramy losowy Node na głębokości od mutation_min_depth (4) do mutation_max_depth (6), po czym wybieramy 1 z jego dzieci, które potem podmieniamy wygenerowany przez __create_random_tree drzewem o głębokości wylosowane z pomiędzy 3 i mutation_min_depth (6).

Wyniki:


Best fitness Over Generations
Best fitness
Best fitness
Generation
6000
5000
4000
3000
2000
1000
0
10
20
30
40
50


Average fitness Over Generations
Average fitness
Average fitness
Generation
500
450
400
350
300
250
200
150
0
10
20
30
40
50



oraz

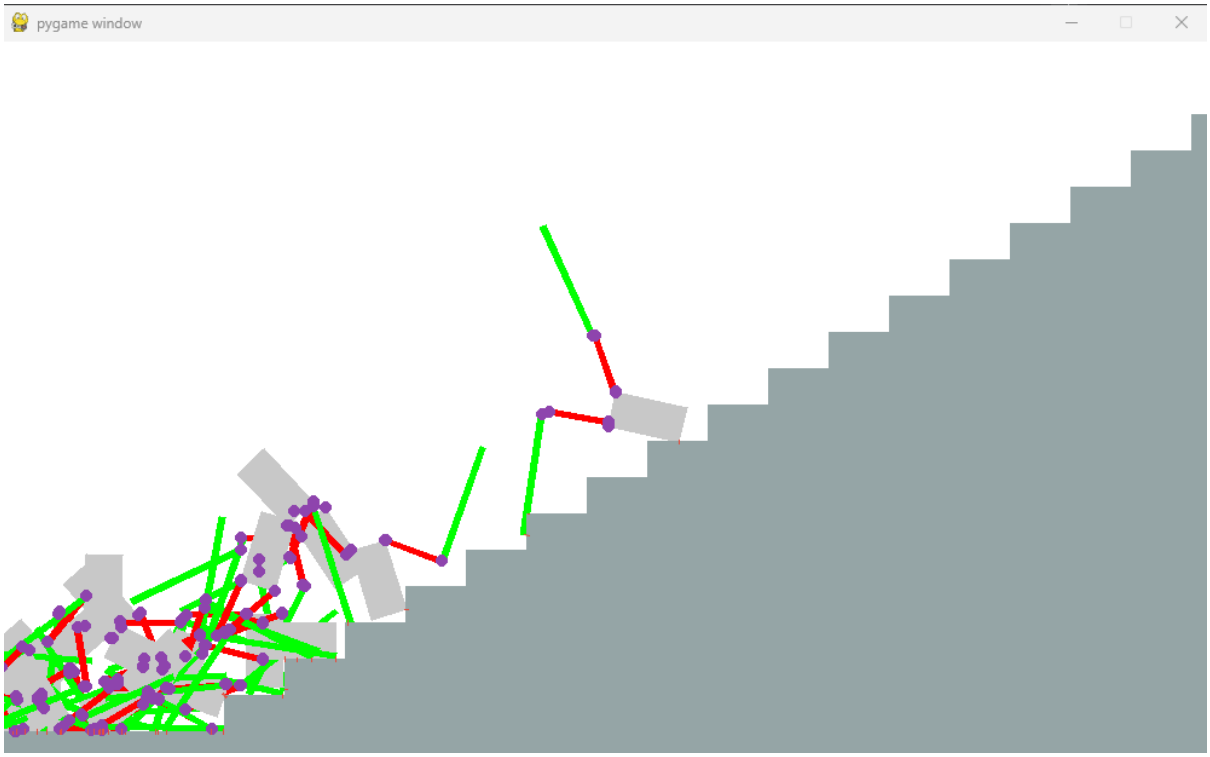
best_ind_problem_2.txt

population_problem_2.txt

Problem 3

Strome schody. Tak samo, jak w problemie 2 weźmiemy ostatnie.

pokolenie z problemu 1.



Rożnice w parametrach:

Parametr	Poprzednio	Teraz
max_depth	10	12
mutation_rate	0.25	0.3
mutation_rate_critic	0.3	0.5
mutation_max_depth	6	8
negative_tournament_rate	0.25	0.25
negative_tournament_rate (w wypadku stagnacji)	0.3	0.25
rotation_rate_up_limit	30	15
rotation_rate_down_limit	-30	-15
liczba generacji do stagnacji	7	5
epsilon stagnacji	50	100

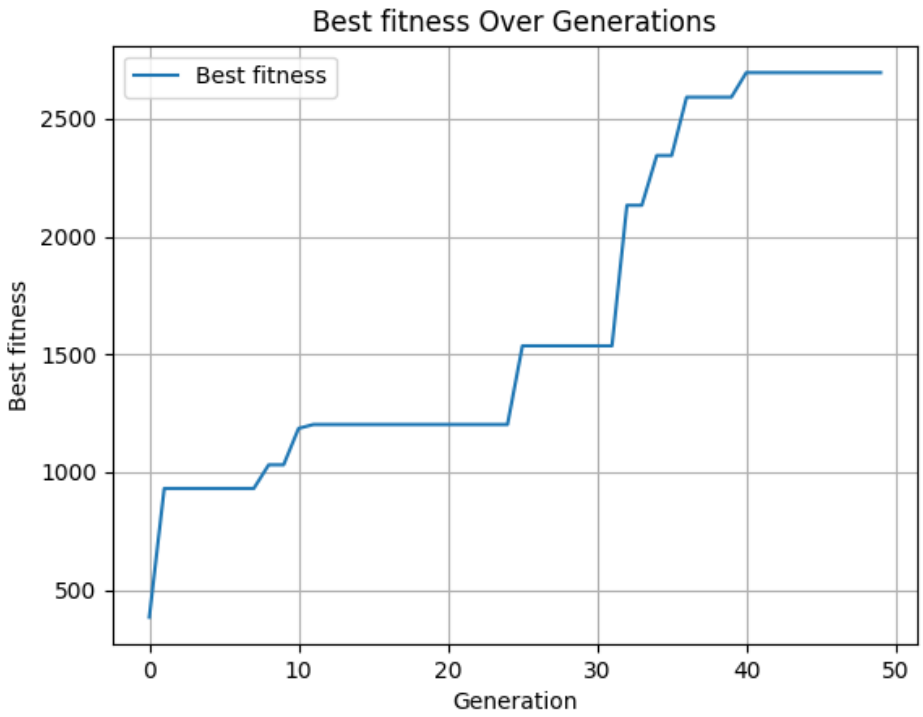
__create_random_tree – teraz działa tak, na podstawie tego czy wylosuje funkcje,stało,zmienna decyduje, czy drzewo dalej rośnie. Oczywiście jest nadal ograniczone przez max_depth.

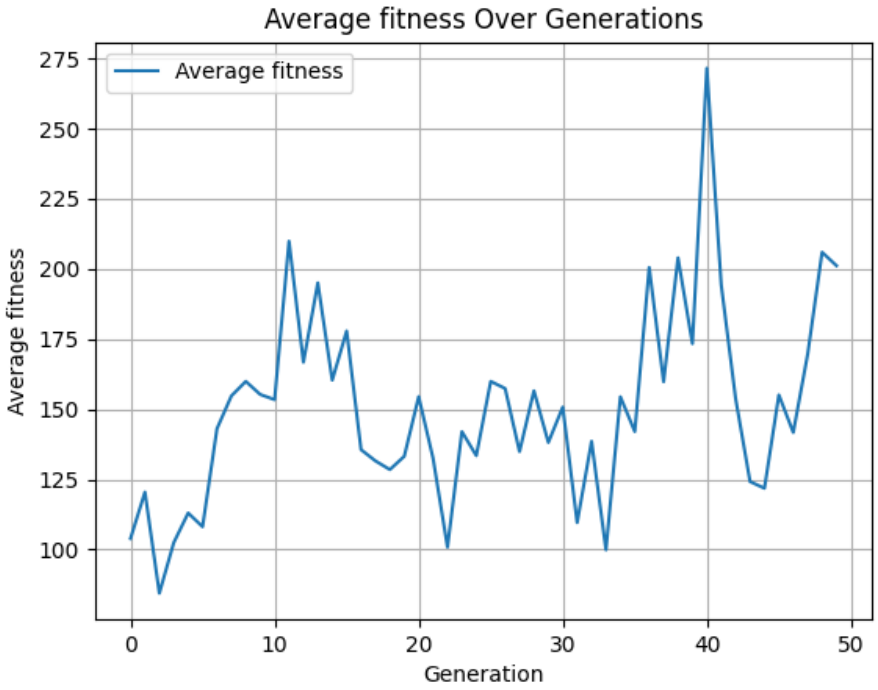
Mutacja

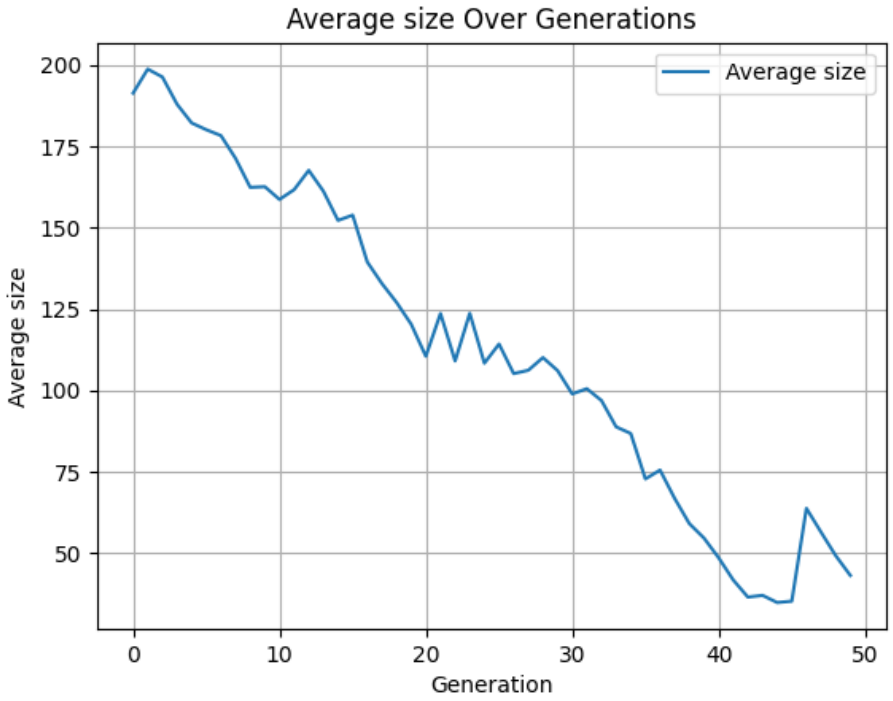
Wybieramy losowy Node na głębokości od mutation_min_depth (4) do mutation_max_depth (8).

Wygenerowany przez __create_random_tree drzewem mają co najwyżej głębokości wylosowaną spomiędzy mutation_min_depth (4) i mutation_max_depth (8).

Wyniki:







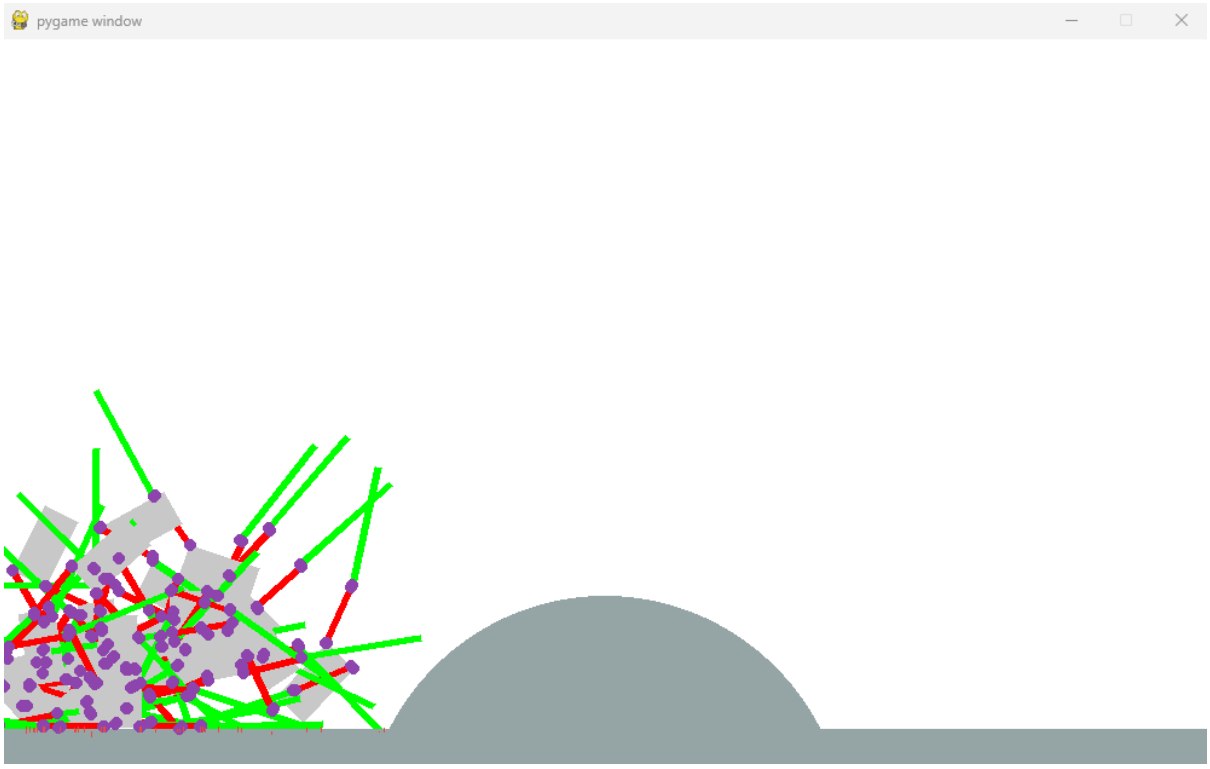
oraz

best_ind_problem_3.txt

population_problem_3.txt

Problem 4

Wzgórze (półkole w podłożu) oraz zmniejszone tarcie do 10.

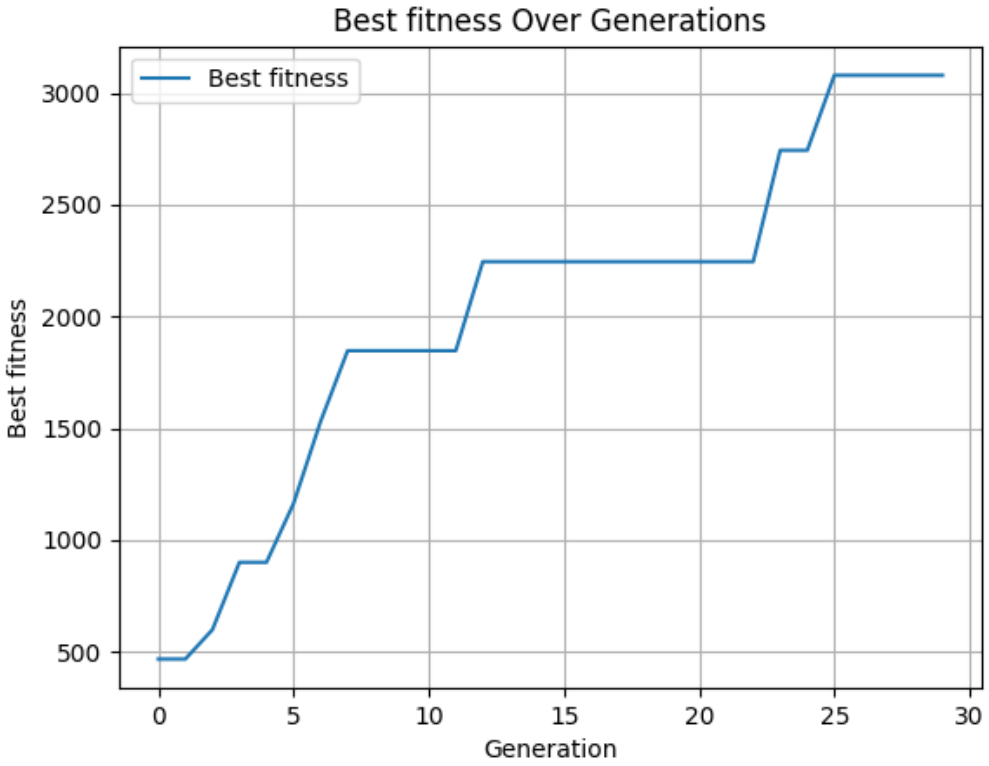


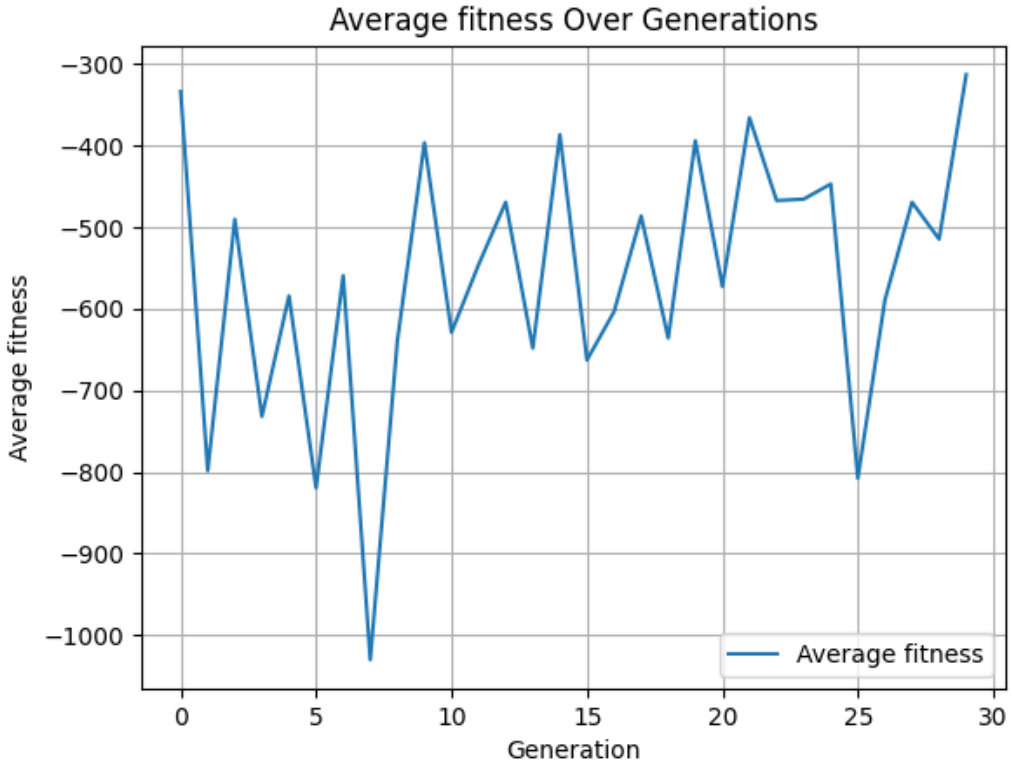
Powrót do bazowej funkcji przystosowania oraz do __create_random_tree w formie full grow.

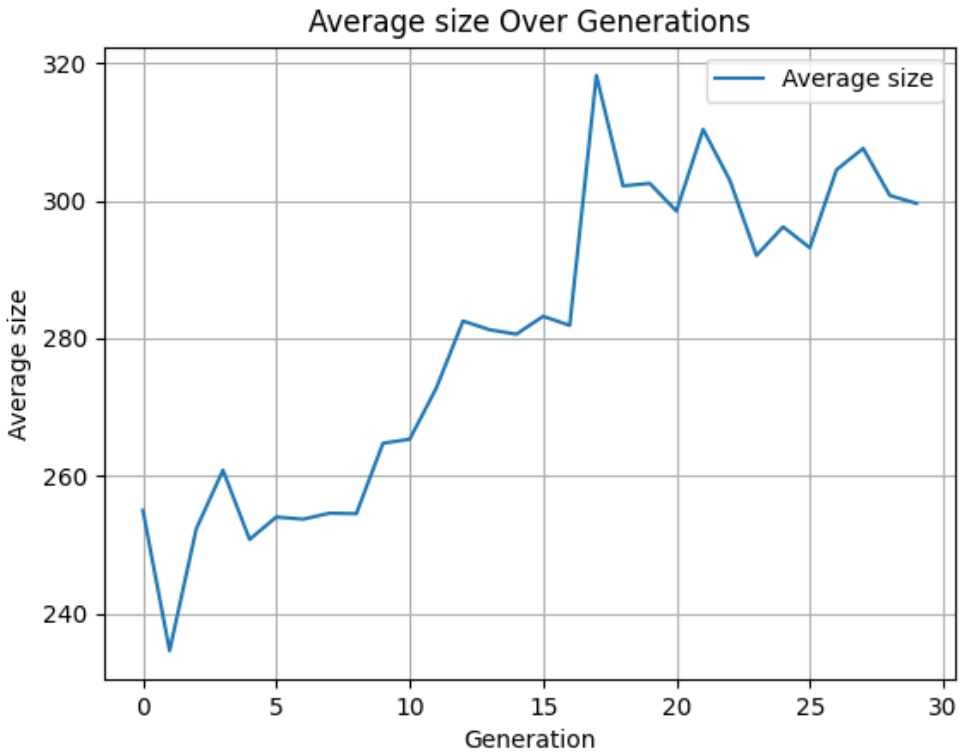
Rożnice w parametrach:

Parametr	Poprzednio	Teraz
max_depth	12	8
max_TTL	0.3	0.3
generation	50	30
mutation_min_depth	4	4
mutation_max_depth	8	6
crossover_min_depth	6	6
crossover_max_depth	8	8
próg anomali	750	500

Wyniki:


Best fitness Over Generations
Best fitness
3000
2500
2000
1500
1000
500
Best fitness
0
5
10
15
20
25
30
Generation


Average fitness Over Generations
−300
−400
−500
−600
−700
−800
−900
−1000
Average fitness
Average fitness
0
5
10
15
20
25
30
Generation



oraz

best_ind_problem_4.txt

population_problem_4.txt

Wnioski:

W problemie 1 manipulacja prawdopodobieństwem lepiej się sprawdziła przeciwko stagnacji.

Ustawienie jako funkcji przystosowania odległości maksymalnej w problemie 2 i 3 mogło spowodować, że promowane były próby wskoczenia na schody przez anomalie.

Anomalie zaczęły pojawiać się częściej, gdy zmieniliśmy otoczenie - szczególnie w momentach, gdy nogi się klinowały.

W problemie3 nie udało się nadrobić utraty rozmiaru spowodowane przez zmienione __create_random_tree.

Link do repozytorium https://github.com/lsdre/gp_project

Autorzy: Gilbert Guszcza, Tomasz Madeja