
	<p>Instytut Informatyki Politechniki Śląskiej</p> <p>Zespół Mikroinformatyki i Teorii Automatów Cyfrowych</p>		
<p>Rok akademicki:</p>	<p>Rodzaj studiów*: SSI/NSI/NSM</p>	<p>Przedmiot:</p>	<p>Grupa</p>
<p>2016/2017</p>	<p>SSI</p>	<p>BIAI</p>	<p>BDis3</p>
<p>Skład sekcji:</p>	<p>Kamil Ziętek Leszek Gzik</p>	<p>Prowadzący: OA/JP/KT/GD/BSz/GB</p>	<p>GB</p>
<p><i>Raport końcowy</i></p>			
<p>Temat projektu:</p> <p>Arena walk postaci</p> <p>Pliki projektu:</p> <p>https://github.com/LeszekGzik/ProjektBIAI</p>			
<p>Data oddania: dd/mm/rrrr</p>		<p>24/05/2017</p>	

Główne założenia projektu

Program generuje populację postaci o losowych wartościach dziewięciu parametrów decydujących o zdolnościach danej postaci. Druga populacja (przeciwników) nie jest tworzona losowo, lecz każdy kolejny jej członek ma parametry większe (lepsze) od poprzedniego. Następnie postacie toczą walki systemem turowym z kolejnymi, coraz silniejszymi przeciwnikami. Populacja "postaci" podlega ewolucji za pomocą algorytmu genetycznego, którego celem jest uzyskanie postaci o optymalnych parametrach tj. takich, które są w stanie wygrać jak najwięcej starć pod rząd bez przegranej (rolę współczynnika "fitness" pełni ilość pokonanych przez postać przeciwników).

Model postaci

Każda z postaci wchodzących w skład populacji jest opisana poprzez dziewięć wzajemnie oddziałujących ze sobą współczynników. Są to:

- *Max HP* - maksymalna ilość punktów zdrowia postaci. Jeśli punkty zdrowia spadną w trakcie walki do zera, postać ginie (i przegrywa walkę).
- *HP Regen* - ilość punktów zdrowia jakie postać odzyskuje na początku każdej rundy.
- *Base dmg* - obrażenia zadawane przeciwnikowi, gdy atak postaci trafi.
- *Crit rate* - szansa na to, że atak postaci będzie trafieniem krytycznym (w procentach)
- *Crit dmg* - mnożnik, który stosuje się do zadawanych przez postać obrażeń, jeśli trafi ona krytycznie (bazowo x1.5)
- *Hit rate* - szansa na to, że atak postaci trafi przeciwnika (w procentach, bazowo 50%)
- *Dodge rate* - szansa na to, że postać uniknie ataku przeciwnika; atrybut ten odejmuje się od wartości *Hit rate* wroga aby uzyskać realną szansę na trafienie
- *Block rate* - szansa na zablokowanie trafionego ataku przeciwnika
- *Block power* - mnożnik, który stosuje się do otrzymywanych przez postać obrażeń, jeśli uda jej się zablokować trafienie (bazowo x0.5, maleje wraz ze zwiększaniem atrybutu)

Na genotyp danej postaci składa się więc dziewięć wartości z zakresu od 0 do 255, gdzie każda wartość odpowiada jednemu współczynnikowi.

Każda postać posiada także pulę "punktów", które na etapie kreacji postaci zostają przydzielone do poszczególnych atrybutów zgodnie z proporcjami wynikającymi z genotypu postaci. Dla populacji "postaci" ilość punktów do rozdania jest stała (100), zaś dla przeciwników liniowo rosnąca (począwszy od 1, potencjalnie nieograniczona).

Tak więc realna wartość atrybutu postaci może być obliczona wzorem:

$$value(A_i) = pts * \frac{A_i}{\sum_{i=0}^8 A_i}$$

Gdzie:

A_i - gen opisujący i-ty atrybut

$value(A_i)$ - wartość i-tego atrybutu

pts - ilość punktów postaci

System walki

Walka pomiędzy dwoma postaciami rozgrywa się w serii tur. Każda tura przebiega w następujący sposób:

- 1) Obie postaci odzyskują punkty zdrowia równe ich *HP Regen*.
- 2) Postać 1 wykonuje atak.
- 3) Postać 2 wykonuje atak.
- 4) Jeśli któraś z postaci zginęła, walka kończy się.

W momencie wykonywania ataku, postać atakująca najpierw próbuje trafić obrońcę (szansa na trafienie to różnica między *Hit rate* napastnika, a *Dodge rate* obrońcy). Jeśli jej się powiodło, program sprawdza czy trafienie było krytyczne (jeśli tak, wówczas zadawane przez atak obrażenia ulegają zwiększeniu o stosowny mnożnik).

Na koniec zaś obrońca może spróbować zablokować atak. Blokowanie zmniejsza otrzymywane obrażenia, a przy odpowiednio wysokim współczynniku *Block power* może nawet zmniejszyć je do zera. W przypadku, gdy zablokowane zostanie trafienie krytyczne, oba mnożniki znoszą się wzajemnie (obrażenia liczone są jak dla zwykłego, nie-krytycznego trafienia).

Jeśli żadna z postaci nie zginie do tury 999, walka automatycznie kończy się remisem.

Warstwa ewolucyjna

Aby móc mówić o ewolucji, potrzebna jest populacja. W programie, poza wykonaniem walki między dwiema postaciami aby sprawdzić działanie opisanej wcześniej funkcjonalności, istnieje opcja utworzenia populacji - albo składającej się z losowych postaci, albo wczytanej z pliku. Jest to nasza populacja startowa, będąca podstawą do ewolucji. Podstawową miarą pozwalającą zbadać ewolucję postaci jest współczynnik fitness, będący wyznacznikiem przystosowania postaci do otoczenia. W naszym przypadku, rolę tego współczynnika pełni ilość przeciwników, z którymi dana postać jest w stanie wygrać. Każda postać toczy walkę z kolejnymi przeciwnikami (ich statystyki można dowolnie ustawić, domyślnie są to równomiernie rozłożone ustawienia), którzy liniowo zwiększają ilość punktów do rozdysponowania. Wartość fitness to przemnożona razy 10 ilość punktów ostatniego przeciwnika z którym udało się wygrać (jeśli postać poległa na przeciwniku o poziomie 120, dostaje 1200 pkt fitness), następnie toczy 10 walk z przeciwnikiem z którym przegrała - ilość wygranych z nim walk jest dodawana do współczynnika (jeśli wygrał 3 walki na 10 stoczonych, fitness to 1203). Takie badanie jest przeprowadzane dla każdego członka populacji. Następnie tworzone są kolejne pokolenia, według stałego schematu: selekcja, mutacja, krzyżowanie, ponowne badanie fitnessu nowopowstałego pokolenia. Każdy element warstwy ewolucyjnej można konfigurować w następującym zakresie:

- Selekcja - wybór najlepiej przystosowanych osobników, dokonywany na podstawie rankingu liniowego lub koła ruletki
- Mutacja - zawsze dotyczy jednej cechy można ustawić prawdopodobieństwo wystąpienia mutacji, jak również jej zakres (dodanie lub odjęcie: stałej wartości, losowej wartości z podanego zakresu lub ustalonego procentu cechy sprzed mutacji)
- Krzyżowanie - można wybrać prawdopodobieństwo wystąpienia krzyżowania dwóch losowo wybranych osobników, a także podzielić jaka ich część zostanie skrzyżowana w jednym punkcie, a jaka w dwóch.

Specyfikacja zewnętrzna (user manual)

Na GUI programu składają się dwie główne zakładki: *Simulator*, gdzie można przeprowadzić walkę dwóch dowolnych postaci i sprawdzić rezultaty, oraz *World*, gdzie mieści się właściwa (ewolucyjna) część programu.

Zakładka *World*, strona pierwsza:

The screenshot shows the 'World' tab of the 'Battle Arena BIAI' application. The interface includes the following elements:

- Population Settings:** A section with a 'Size of population' spinner (labeled 1) set to 100, a 'Create population' button, and a 'Population not created' status message.
- Fitness Calculation Settings:** A section with a 'Number of battles' spinner set to 10 and a 'Step (level of opponents)' spinner (labeled 2) set to 1.
- Character Settings:** A section with various character attributes (Max HP, HP Regen, Base dmg, Crit rate, Crit dmg, Hit rate, Dodge rate, Block rate, Block Power, Points) each with a spinner and a value. A 'Recalculate fitness' button is also present. A red number 3 is placed next to the 'Crit rate' attribute.
- Generation Control:** A 'Generation #' spinner (labeled 4) set to 0, and buttons for 'Import population' (labeled 5) and 'Export population' (labeled 6).
- Main Table:** A large table with columns: ID, Fitness, Previous fitness, Change, and Genotype.

1 - ustawienia rozmiaru populacji.

2 - ustawienia obliczania funkcji fitness; zalecane jest korzystanie z ustawień domyślnych, chyba że rozmiar populacji jest bardzo duży, lub bardzo mały

3 - "szablon" przeciwnika wykorzystywany przy obliczaniu funkcji fitness

4 - kontrolka do przełączania się między pokoleniami

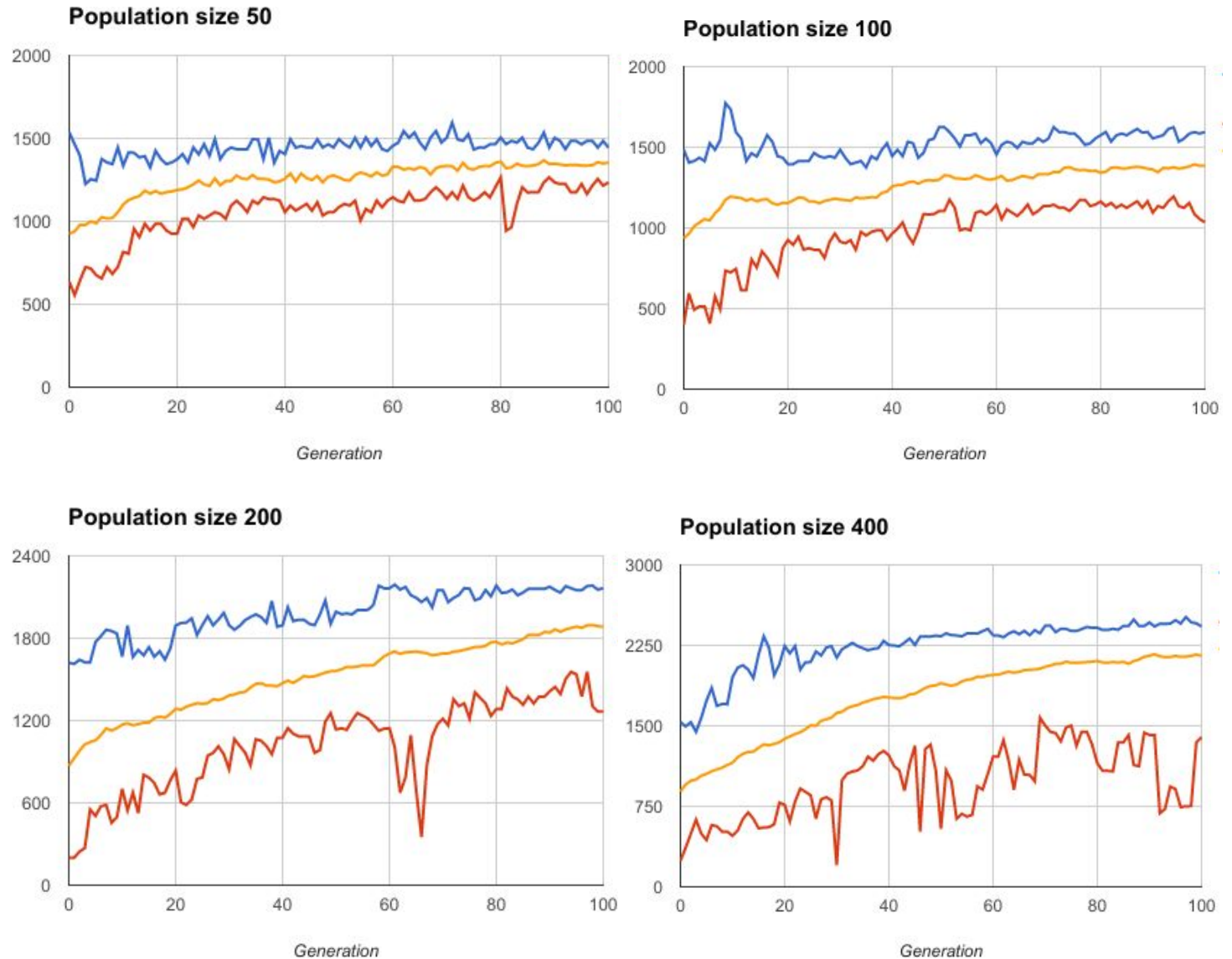
5 - import populacji z pliku tekstowego

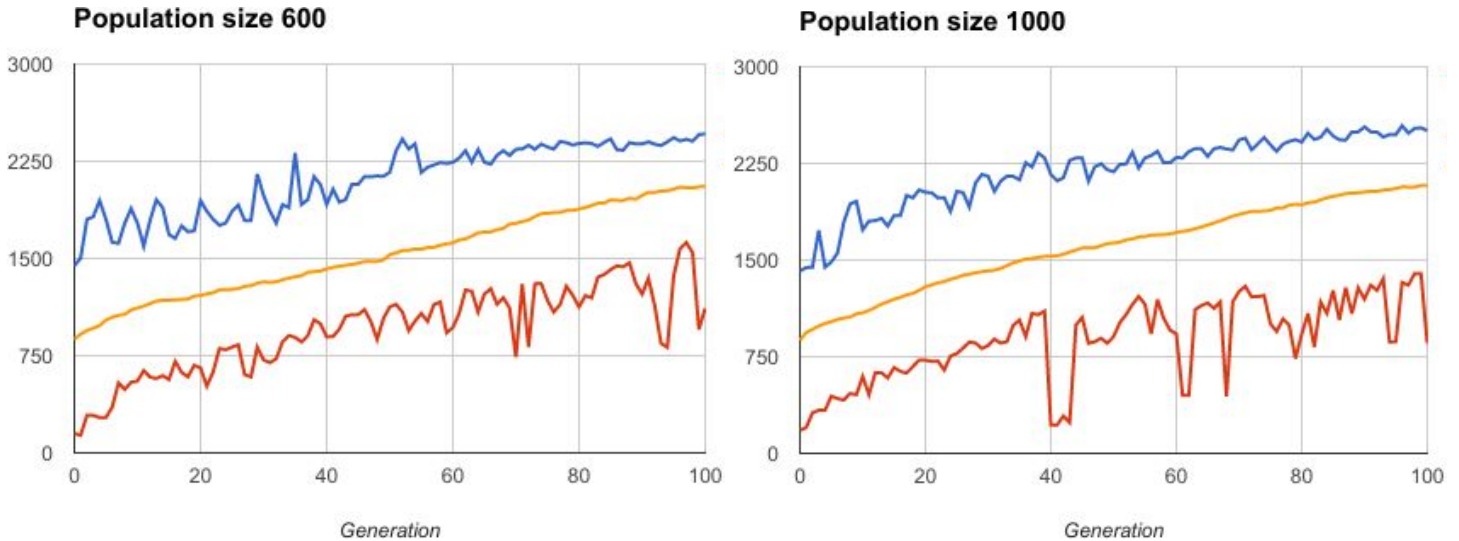
6 - eksport populacji do pliku

Testy ustawień ewolucji

Gdy nasz program miał już zaimplementowaną pełną funkcjonalność, mogliśmy przystąpić do testowania jego działania. Najpierw sprawdziliśmy zależność współczynnika fitness w skali całej populacji od konkretnych ustawień algorytmu genetycznego. Wszystkie wykresy zostały wykonane dla stu kolejnych pokoleń przy domyślnych ustawieniach wszystkich parametrów poza aktualnie badanym, i wskazują minimalną (czerwony), średnią (pomarańczowy) oraz maksymalną (niebieski) wartość współczynnika fitness w każdym pokoleniu.

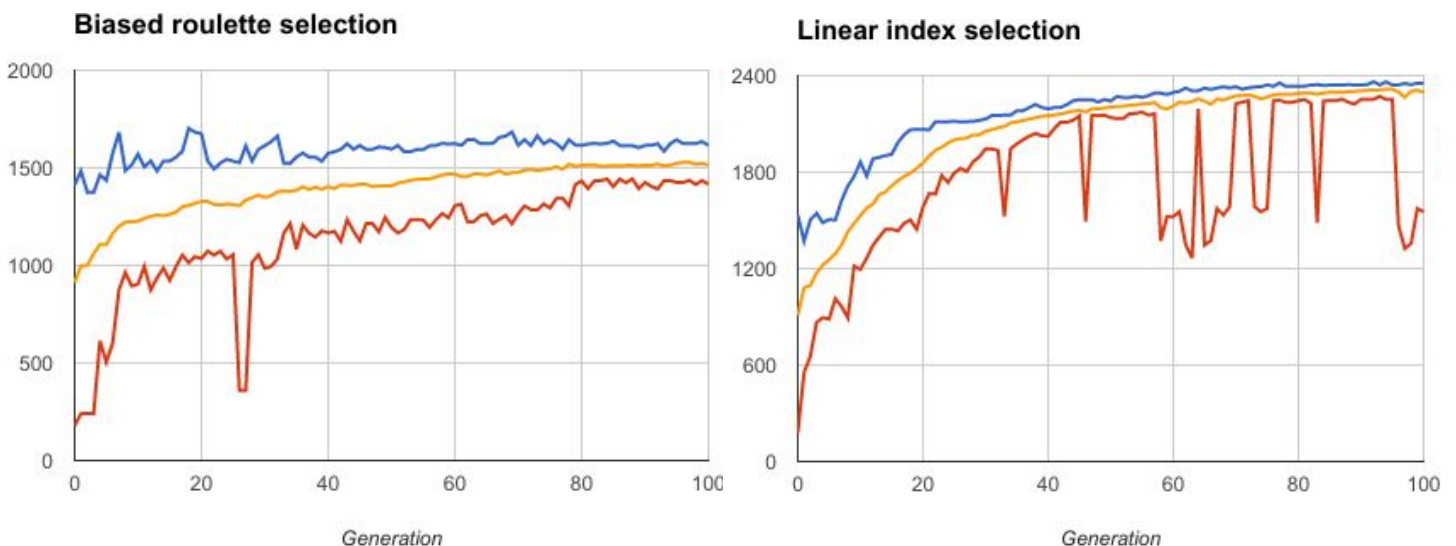
Testy rozpoczęliśmy od sprawdzenia wpływu rozmiaru populacji na wyniki działania algorytmu.





Jak widać, niezależnie od konkretnego rozmiaru populacji, występuje tendencja wzrostowa współczynnika fitness. Dla mniejszych rozmiarów (50 i 100) ewolucja polega raczej na eliminacji słabszych osobników, a wartość najlepszego osobnika pozostaje na stałym poziomie - zapewne dlatego, że nie ma tylu możliwości krzyżowania między poszczególnymi osobnikami. Zwiększanie rozmiaru populacji pozwala zauważyć wzrost maksymalnych wyników, ale jednocześnie gorszą niż przy mniejszych rozmiarach eliminację najslabszych osobników, co powoduje wzrost rozbieżności między wartością średnią a minimalną i maksymalną. Wynika to z faktu, że krzyżowanie przynosi lepsze efekty (przy małych populacjach dany osobnik najpewniej musi się kilka razy krzyżować z tym samym osobnikiem), ale spada jakość selekcji - przy większej populacji musi zostać wybranych więcej osobników do kolejnego pokolenia, więc "przechodzi" większa ilość słabych postaci. Najsensowniejszym ustawieniem wydaje się być opcja 200, jednak z powodu znaczącego wpływu na czas wykonywania obliczeń, domyślną wartością pozostało 100 jednostek.

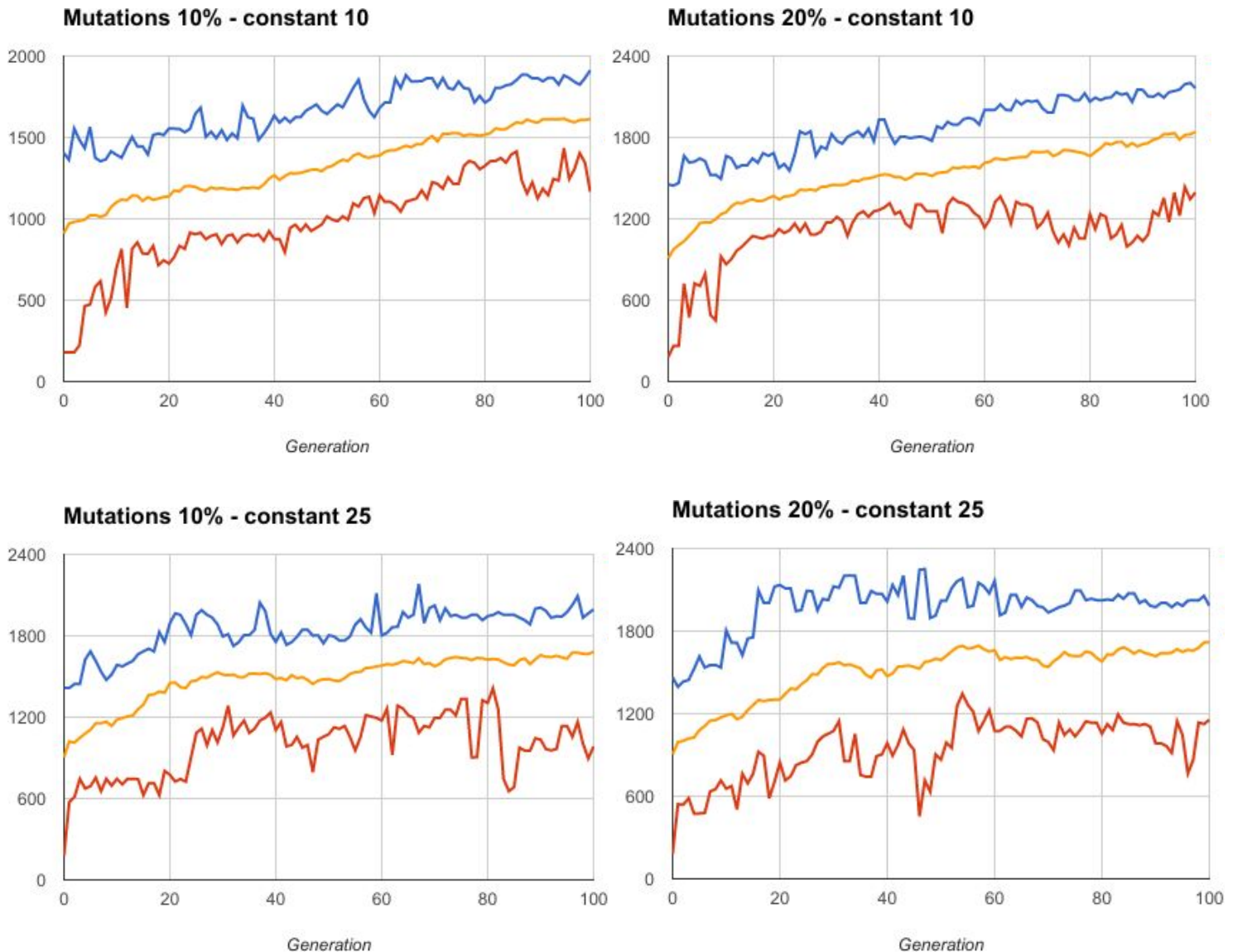
Kolejne badania zostały wykonane przy tej samej, liczącej sto osobników, populacji startowej. W celu umożliwienia odtworzenia przebiegu badań, populacja startowa została umieszczona w pliku `base_population.txt` i można ją zaimportować w programie. Wyniki w zależności od wybranej metody selekcji:



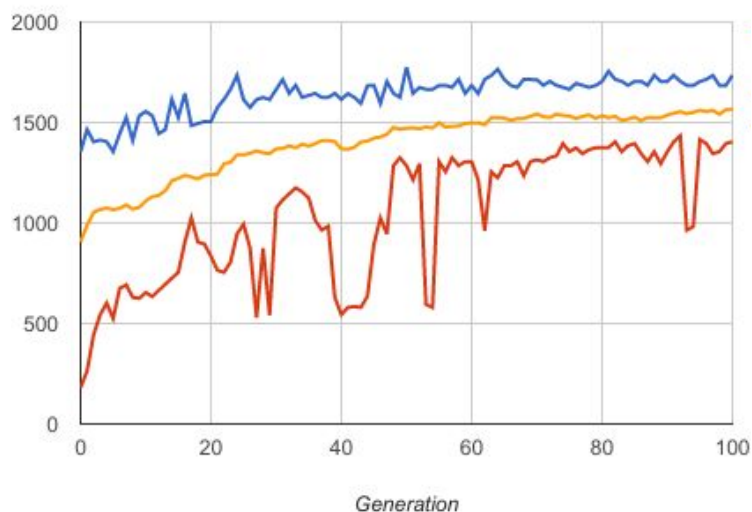
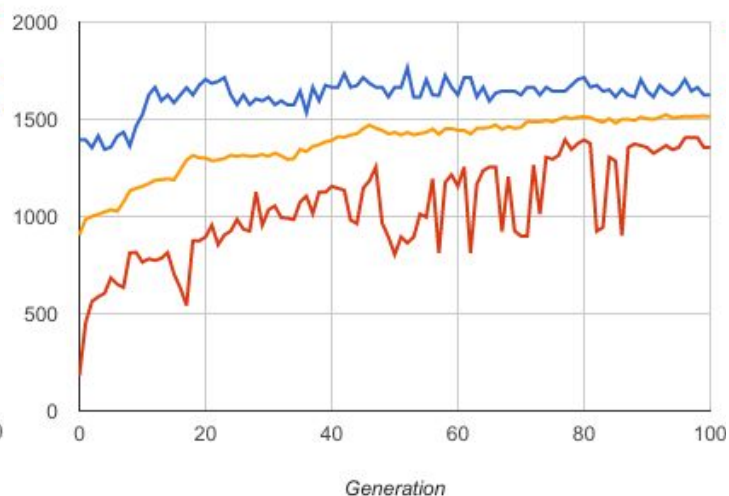
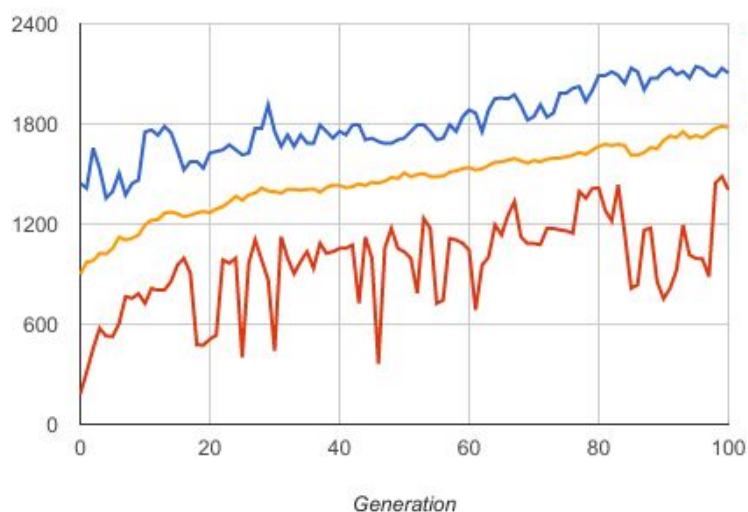
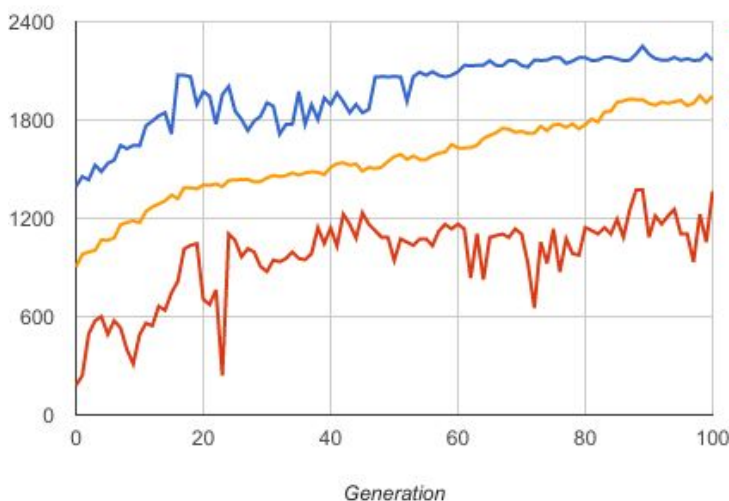
Ponieważ ranking liniowy mocniej faworyzuje sprawdzone rozwiązania (np. w populacji 100 osobników - osobnik najlepszy ma 100 razy większą szansę na przejście do kolejnego pokolenia niż osobnik najgorszy) jak widać na

wykresach powyżej fitness w rankingu liniowym rośnie znacznie szybciej, ale równie szybko się stabilizuje. Zdarzają się jednak osobniki dużo gorsi od pozostałych, ponieważ niezależnie od wartości jaka dzieli go od lepszego osobnika, prawdopodobieństwo jego wystąpienia jest zależne od pozycji w rankingu. W przypadku selekcji kołem ruletki, każdy zdecydowanie gorszy osobnik ma o wiele mniejsze prawdopodobieństwo ponownego wystąpienia, przez co takie osobniki są szybko eliminowane i zarówno najlepszy jak i najgorszy osobnik są zbliżone do średniej. Z drugiej strony, koło ruletki ogranicza mocno rozwój populacji - wiele podobnych osobników ma podobne prawdopodobieństwo wystąpienia, w przeciwieństwie do selekcji liniowej, gdzie nawet jeśli osobnik jest nieznacznie lepszy, to może mieć o wiele większe prawdopodobieństwo ponownego wystąpienia.

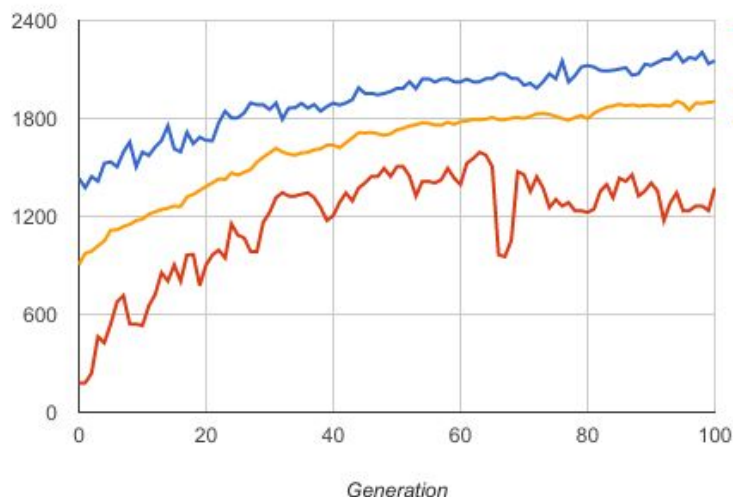
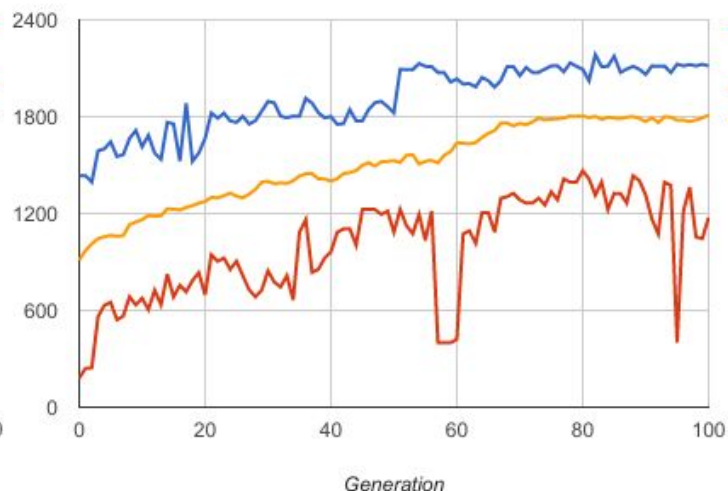
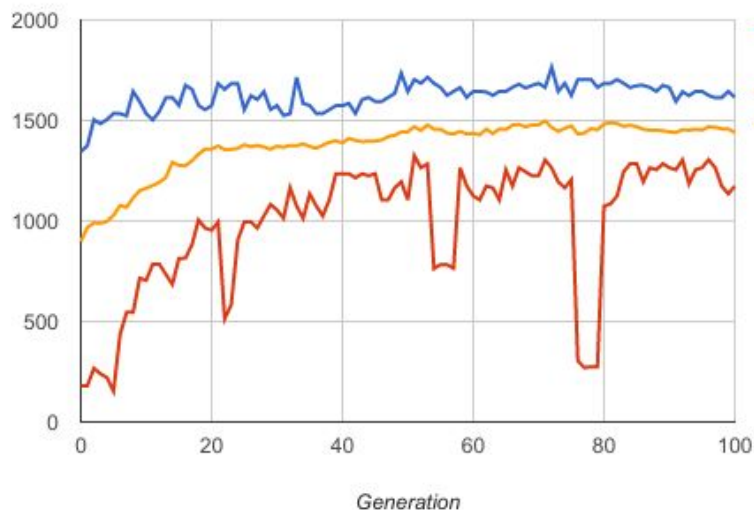
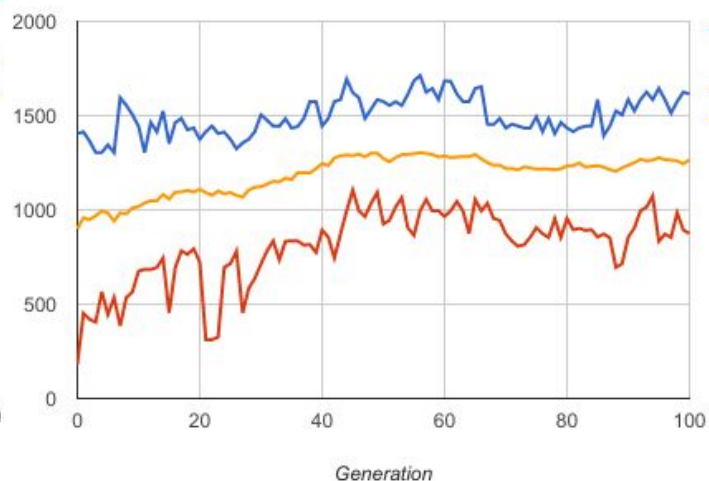
Zbadanie wpływu mutacji jest nieco bardziej skomplikowane - można zmieniać zarówno prawdopodobieństwo wystąpienia mutacji, jak i konkretny typ.



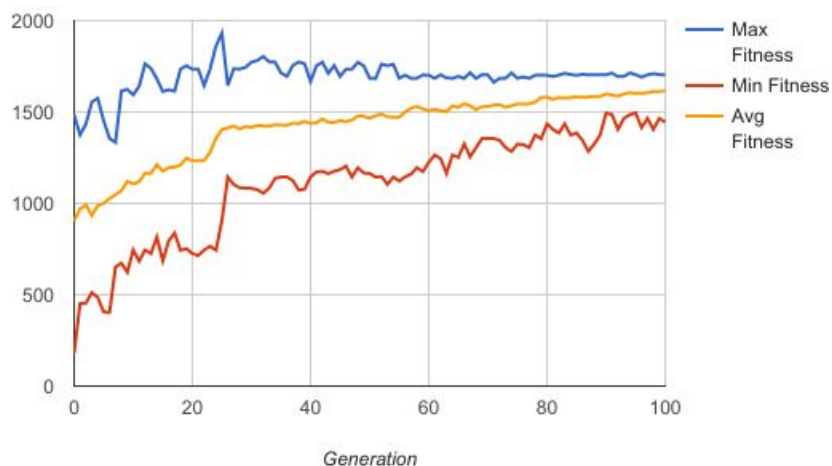
Ustawienie stałej zmiany wartości parametru nie powoduje znaczącego wpływu na wynik działania algorytmu. Jest to spowodowane skalą w jakiej operują te wartości (parametry od 0 do 255) - zmiana o stałą wartość jednego parametru nie powoduje zbyt drastycznych zmian w całkowitym rozkładzie punktów, szczególnie jeśli te parametry mają duże wartości.

Mutations 10% - percent 10**Mutations 20% - percent 10****Mutations 10% - percent 25****Mutations 20% - percent 25**

Dla zmiany o 10% sytuacja jest analogiczna do zmiany o stałą wartość - 10% z 255 to ok. 25, więc w najlepszym wypadku nawet jeśli mutacja zajdzie, to parametr zostanie zmodyfikowany nieznacznie, a dla mniejszych wartości zmiana ta będzie jeszcze mniej zauważalna. Zdecydowanie lepsze wyniki osiągnęło ustawienie zmiany o $\frac{1}{4}$ wartości - tutaj zajście mutacji powoduje znaczące zmiany w genotypie osobników, co przyczynia się do wzrostu maksymalnego fitnessu. Większe prawdopodobieństwo mutacji powoduje szybsze zwiększanie fitnessu, jednak ostateczna wartość jest mniej więcej taka sama.

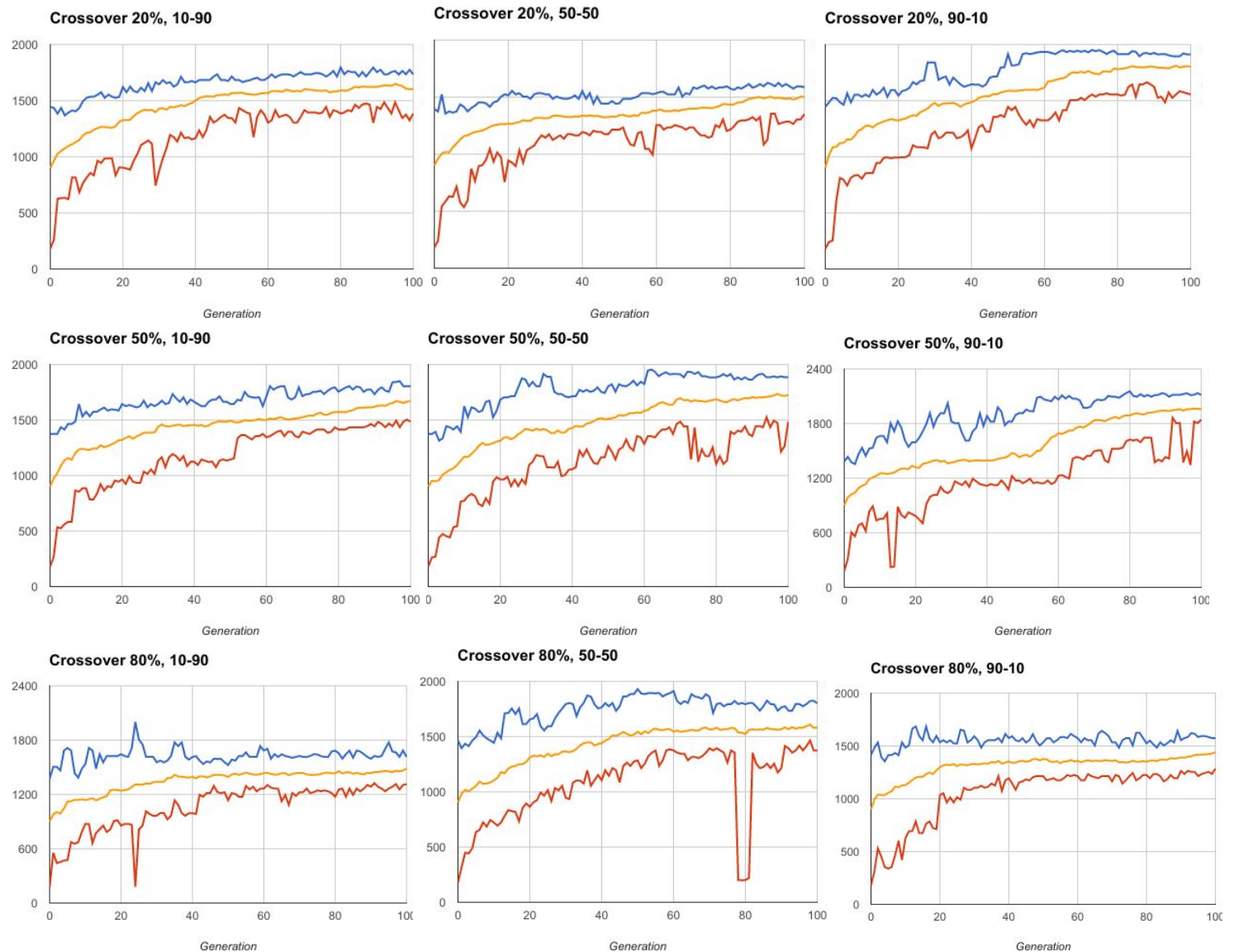
Mutations 10% - random 20**Mutations 20% - random 20****Mutations 10% - random 40****Mutations 20% - random 40**

Ustawienie zmiany o losową wartość między 0 a 20 dało podobny efekt jak stała 20, czyli nieznaczne zmiany w strukturze. Ustawienie 40 jako jedyne pozwoliło zauważyć cofanie się populacji w rozwoju - duże zmiany powodują spore różnice między kolejnymi pokoleniami, a ponadto fakt, że statystyki są opisane za pomocą wartości typu byte, rodzi w skutkach zagrożenie "przekręcenia" licznika, przez co zmiana byłaby jeszcze większa. Dla porównania - wyniki przy całkowicie wyłączonych mutacjach:

Mutations OFF

Jak widać, wyłączenie mutacji powoduje mocne ustabilizowanie współczynnika fitness w kolejnych pokoleniach. To zrozumiałe - w populacji ustaliło się kilka najlepszych konfiguracji genotypu, i jedyna zmiana polega na ich zamianie między kolejnymi osobnikami. Nie ma miejsca na żadne losowe zmiany - od początku operujemy tylko na tych zestawach cech, które zostały wygenerowane na początku, a wzrost średniego fitnessu zapewniony jest przez selekcję. Nawet jeśli więc konkretne ustawienie parametrów mutacji nie powoduje znaczących zmian w strukturze populacji, to muszą być one włączone aby umożliwić jej rozwój - w innym wypadku maksymalny osiągnięty fitness w kolejnych pokoleniach nie będzie mógł się zwiększać.

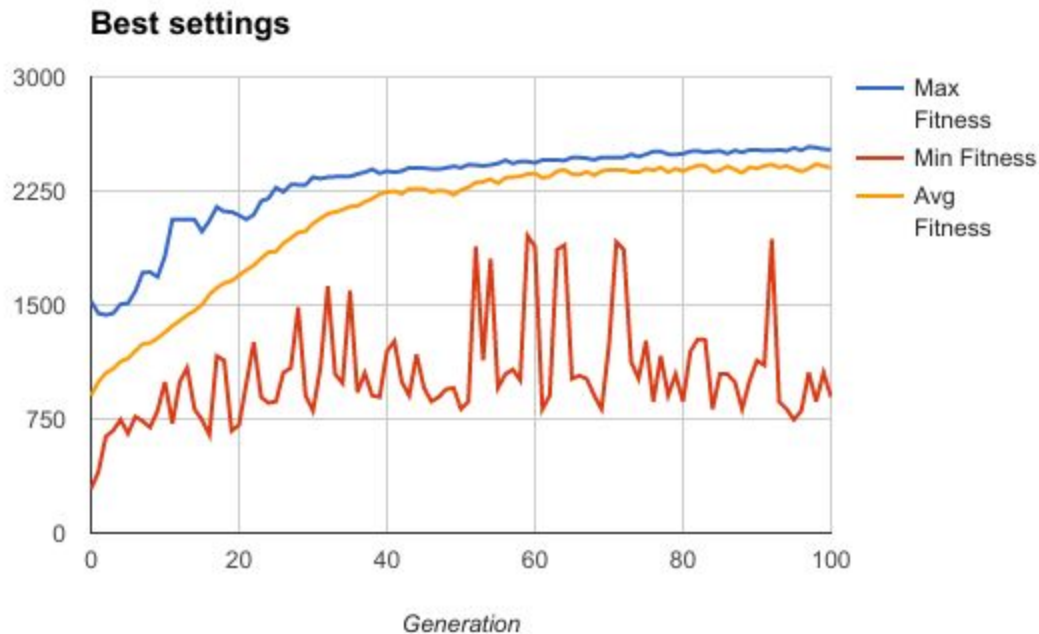
Ostatnią sprawdzoną zależnością jest prawdopodobieństwo i stosunek krzyżowania. Wiersze to różne ustawienia prawdopodobieństwa krzyżowania w ogóle, kolumny to kolejno: 10% szansy na krzyżowanie w jednym punkcie i 90% na krzyżowanie w dwóch, 50% szansy na każde krzyżowanie oraz 90% szansy na krzyżowanie w jednym punkcie i 10% na krzyżowanie w dwóch punktach.



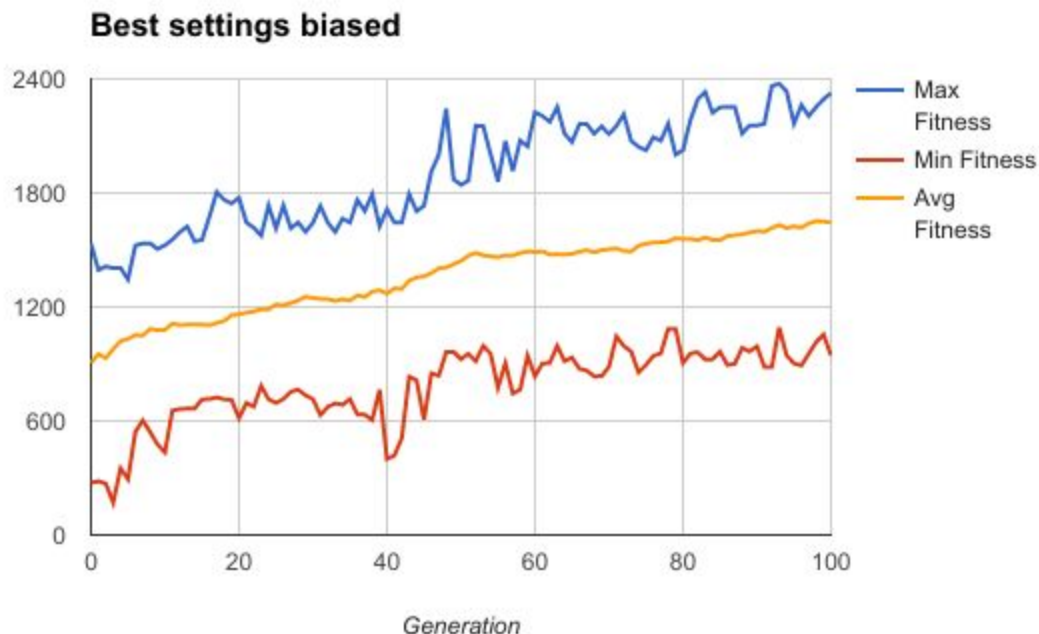
W każdym z przypadków ustawienie prawdopodobieństwa krzyżowania na 90% w jednym punkcie i 10% w dwóch punktach daje najlepsze efekty - krzyżowanie w dwóch punktach zbyt mocno "miesza" w genotypie, i kolejne postaci nie mają zbyt wiele wspólnego z poprzednimi, przez co ciężko mówić tu o ewolucji, a raczej o losowaniu statystyk. Z tego samego powodu ustawienie 80% prawdopodobieństwa krzyżowania w ogóle ogranicza rozwój populacji - w połączeniu z domyślnie aktywnymi mutacjami każde kolejne pokolenie ma zbyt różne statystyki od poprzedniego, aby mówić o rozwoju populacji.

Łącząc obserwacje z wszystkich powyższych wykresów, ustaliliśmy teoretycznie najlepszą kombinację ustawień ewolucji populacji:

- Rozmiar populacji: 250
- Metoda selekcji: liniowa
- Szansa mutacji: 20%
- Metoda mutacji: zmiana o 25% wartości
- Krzyżowanie: 50% szansy ogółem, 90% szansy na krzyżowanie w jednym punkcie

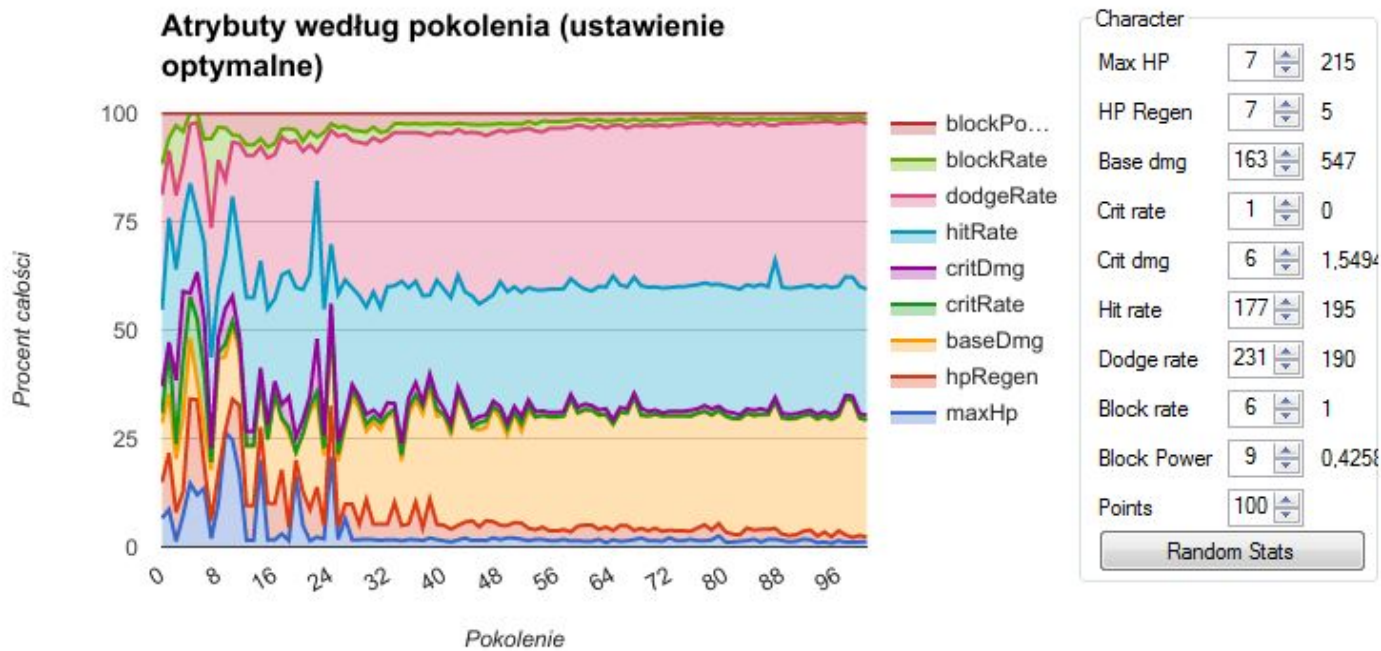


Wynik działania jest połączeniem wszystkich cech wybranych ustawień - szybki przyrost max fitnessu, intensywny wzrost (wartość maksymalna na poziomie 2500, średnia 2400 - są to najlepsze wyniki uzyskane we wszystkich testach), maksimum zbliżone do średniej. Bardzo jednak widoczna jest cecha selekcji liniowej, czyli ogromna przepaść między najslabszym a średnim osobnikiem. Aby zniwelować jej działanie, przeprowadziliśmy jeszcze raz test dla tej samej populacji początkowej i ustawień, zmieniając jednak typ selekcji na koło ruletki.



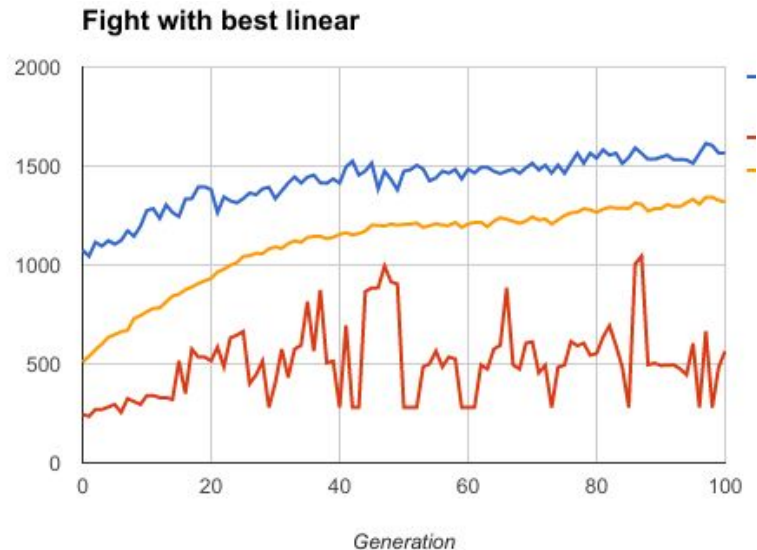
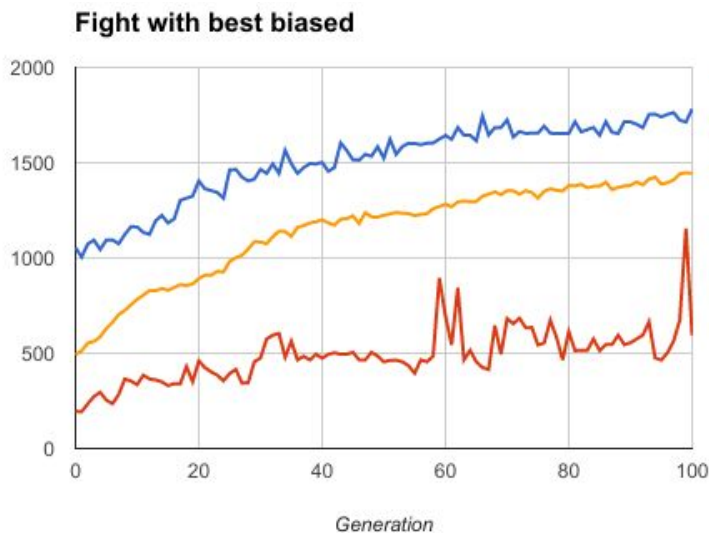
Najslabszy osobnik ma co prawda nieco lepsze statystyki (i mniej różniące się od średniej), jednak generalnie populacja osiągnęła gorsze wyniki niż przy selekcji liniowej. Ostatnie dwa badania pozwoliły nam wyznaczyć

cechy najlepszego osobnika przystosowanego do walki z przeciwnikiem mającym równomiernie rozłożone statystyki.

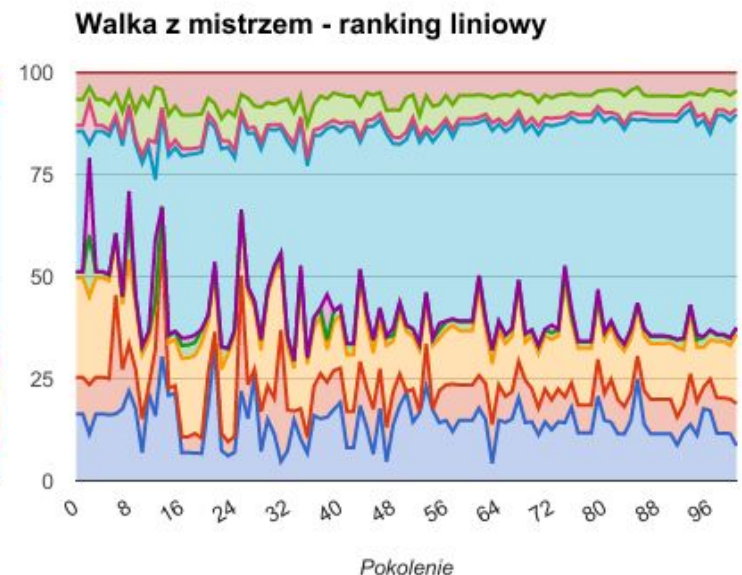
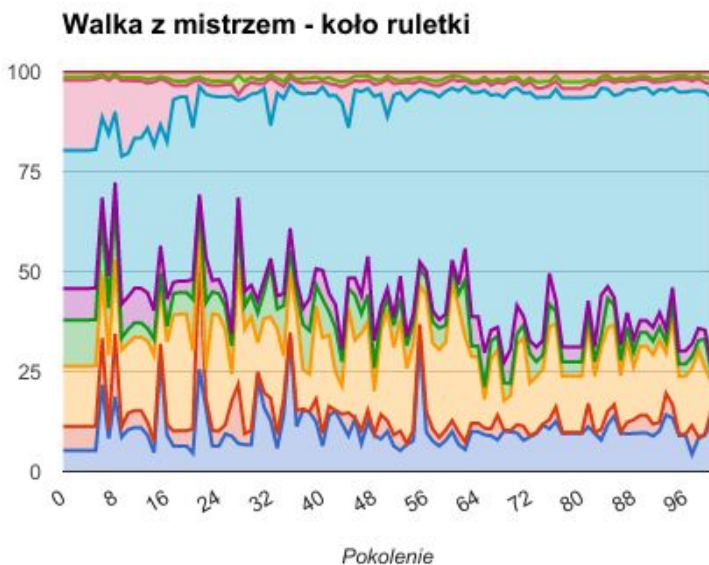


Testy przystosowania do środowiska

Wszystkie powyższe badania zostały przeprowadzone dla jednego typu przeciwnika, z równomiernie rozłożonymi statystykami. Algorytm genetyczny powinien jednak dostosowywać się do warunków środowiska, i działać równie dobrze przy każdym ustawieniu przeciwnika. Mając ustalonego lokalnie najlepszego przeciwnika, dla testu utworzyliśmy nową populację, ustawiając właśnie jego jako wzorzec przeciwnika do wyznaczania fitnessu.



Wyniki fitnessu można było przewidzieć - populacja co prawda została zmasakrowana, jednak nadal widoczny był jej stopniowy wzrost - a więc ewolucja działała, jednak z racji silnych ustawień przeciwnika jego pokonanie było zdecydowanie trudniejsze. Ciekawsze są wykresy przedstawiające atrybuty:

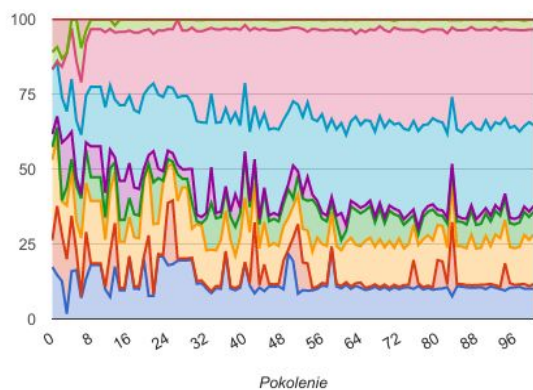


Zmiana jest wyraźnie zauważalna - pominięte poprzednio atrybuty maxHP i HPregen zyskały na wartości, podobnie z hitRate który przejął większość innych parametrów.

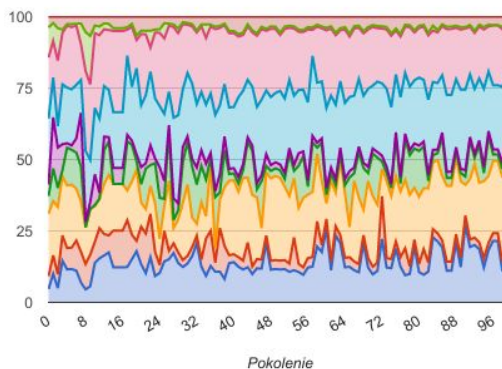
Aby lepiej uwidocznąć dostosowywanie się cech populacji do przeciwnika, przeprowadziliśmy jeszcze trzy testy, polegające na trzykrotnym uruchomieniu programu dla różnych populacji startowych, aby zobaczyć, czy faktycznie uzyskane na koniec wyniki będą takie same niezależnie od punktu wyjścia.

Test pierwszy - trzy uruchomienia dla domyślnego (równomierny rozkład statystyk) przeciwnika:

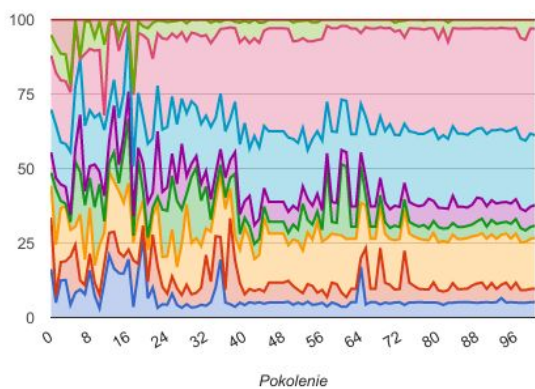
Atrybuty według pokolenia 1



Atrybuty według pokolenia 2



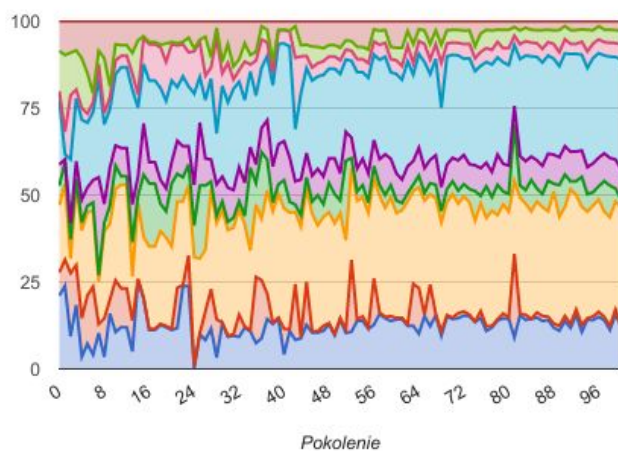
Atrybuty według pokolenia 3



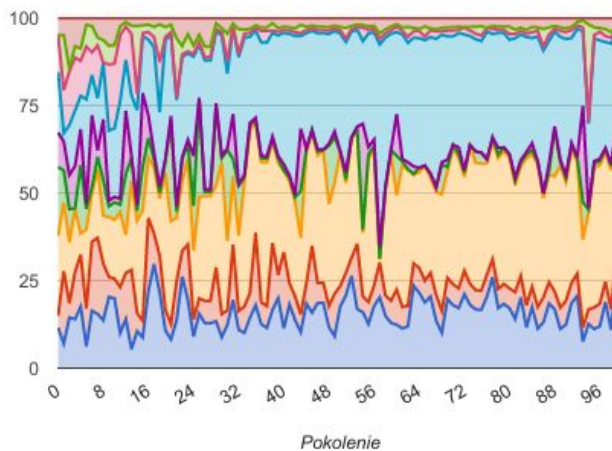
We wszystkich trzech próbach widać dość wyraźne tendencje w ostatnim pokoleniu - faworyzowane są głównie hitRate, dodgeRate, baseDmg i (w mniejszym stopniu) maxHp. Wyniki te są podobne do tych uzyskanych z optymalnymi ustawieniami, co nie powinno dziwić.

Test drugi - trzy uruchomienia przeciwko całkowicie losowemu przeciwnikowi:

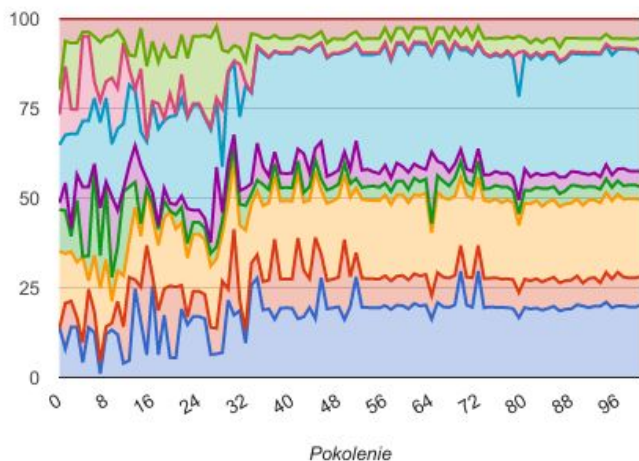
Atrybuty według pokolenia 4



Atrybuty według pokolenia 5



Atrybuty według pokolenia 6

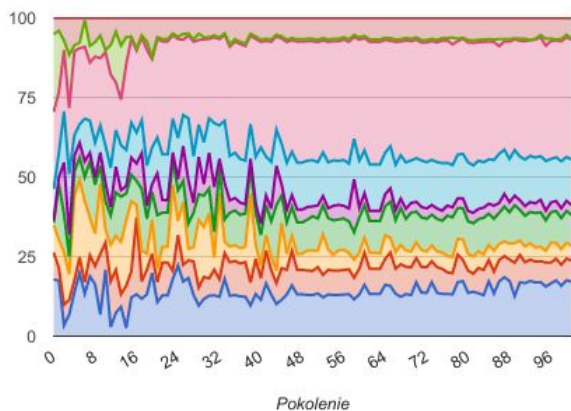


Character		
Max HP	62	616
HP Regen	209	87
Base dmg	73	131
Crit rate	89	14
Crit dmg	202	2,340
Hit rate	241	150
Dodge rate	138	57
Block rate	2	0
Block Power	185	-0,270
Points	100	
Random Stats		

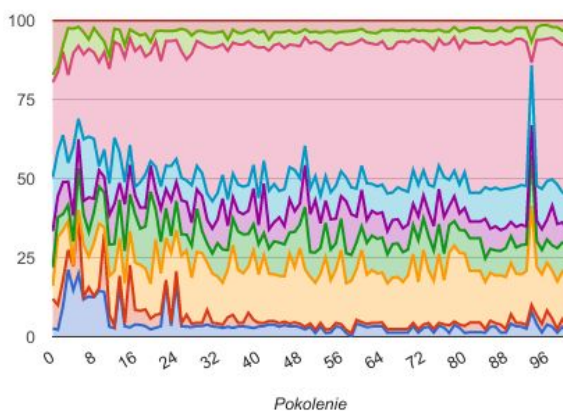
Drugi test dał inne wyniki - populacja zignorowała niemal całkowicie dodgeRate i skupiła się na rozwijaniu hitRate i baseDmg. Spowodował to zapewne fakt, że obrażenia zadawane przez przeciwnika są znacznie mniejsze niż poprzednio i potrzeba ich unikania zanikła.

Test trzeci i ostatni - trzy uruchomienia przeciwko ręcznie stworzonemu przeciwnikowi:

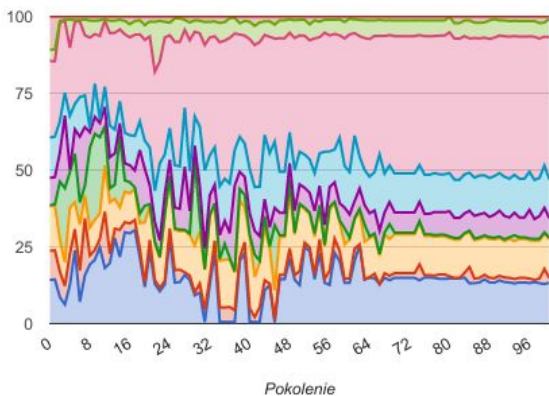
Atrybuty według pokolenia 7



Atrybuty według pokolenia 8



Atrybuty według pokolenia 9



Character		
Max HP	15	1600
HP Regen	4	20
Base dmg	21	430
Crit rate	20	40
Crit dmg	20	2,5
Hit rate	10	100
Dodge rate	4	20
Block rate	3	6
Block Power	3	0,35
Points	100	
Random Stats		

Jak widać, tym razem rozkład atrybutów jest dużo bardziej równomierny; populacja zaczęła rozwijać wcześniej ignorowane statystyki, takie jak critRate i blockRate, znacznie wzrosło też znaczenie dodgeRate. Przeciwnik ten ma dużą siłę ofensywną, więc postacie ewoluują w kierunku skuteczniejszych uników i bloków.

Wnioski

Projekt pozwolił nam zapoznać się z zasadą działania algorytmów genetycznych. Dzięki temu że zagadnienie wydaje się być zdecydowanie prostsze niż sieci neuronowe (które były alternatywnym tematem projektu), mogliśmy dobrze poznać sposób działania tych algorytmów i zaimplementować ich działanie od zera, bez posługiwania się gotowymi przykładami czy bibliotekami. Zaletą tego jest - naszym zdaniem - zdecydowanie lepsze i dogłębnierze poznanie tematu, niż gdybyśmy poznali tylko podstawy bardziej zaawansowanego algorytmu.

Mimo że zaimplementowana przez nas arena walk nie ma żadnego bezpośredniego zastosowania w rzeczywistości (jedynym potencjalnym zastosowaniem są walki w grach komputerowych, jednak mają one zazwyczaj dużo bardziej zaawansowane systemy walki), to warstwa genetyczna mogłaby być z powodzeniem zastosowana w dowolnym innym przypadku, który można sprowadzić do zestawu cech zapisywanych wartościami całkowitymi, i byłby możliwy do prostego porównania (wyznaczenia fitnessu). Operacje genetyczne bowiem wykonywane są jedynie na wartościach liczbowych, niezależnie od tego jakie one mają znaczenie.