WYDZIAŁ ARCHITEKTURY, BUDOWNICTWA
I SZTUK STOSOWANYCH

OPRACOWANIE GRY KOMPUTEROWEJ WYKORZYSTUJĄCEJ ALGORYTMY GENETYCZNE

PRACA INŻYNIERSKA

PROMOTOR:
dr inż. ŁUKASZ WALUSIAK

AUTOR:
ARTUR PILICHOWSKI

KIERUNEK: informatyka

TRYB: studia niestacjonarne

NUMER ALBUMU:10256

KATOWICE, 2022

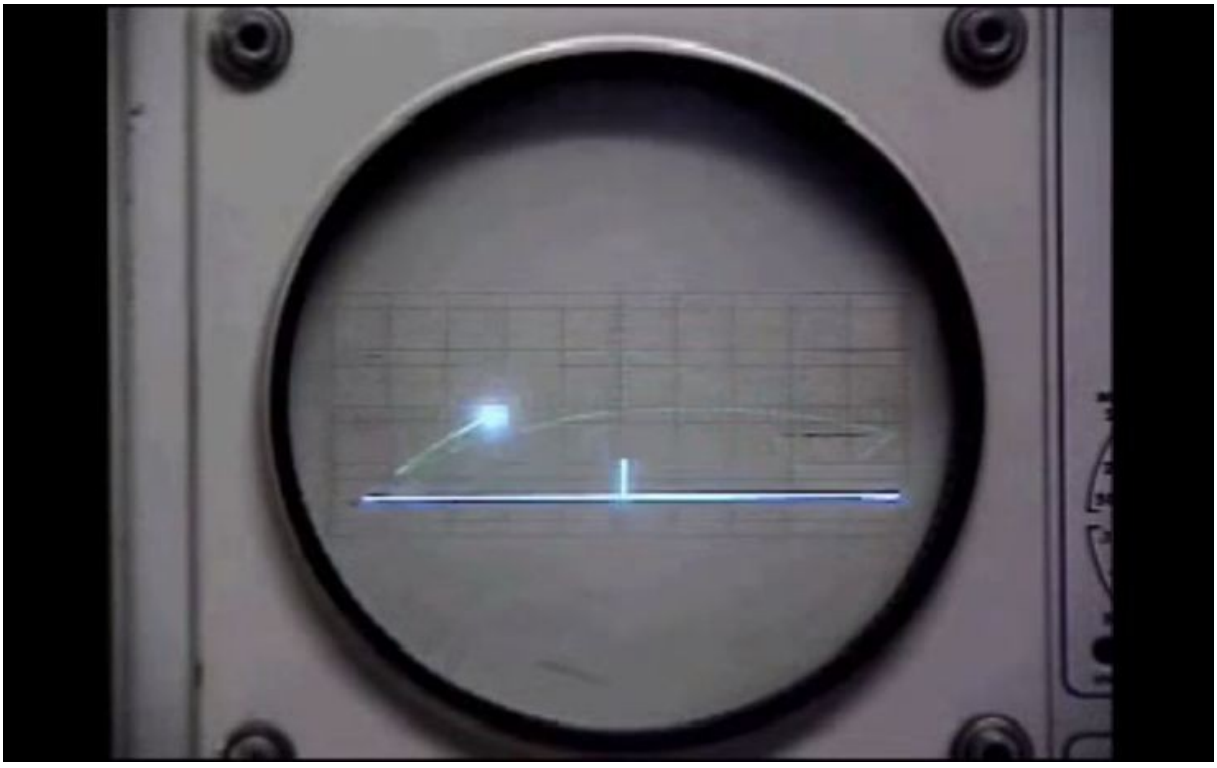
SPIS TREŚCI

1. WSTĘP

1.1 Gry komputerowe

Gry pełnią przede wszystkim funkcję ludyczną (zapewniając rozrywkę zarówno uczestnikom jak i obserwatorom) i prezentacyjną (ukazując sprawność fizyczną lub intelektualną uczestników). Choć stwierdzenie to wymagałoby dowodu (ang. proof), co mogłoby wiązać się z osobną pracą dyplomową, to nawet pobieżna obserwacja różnego rodzaju gier (od planszowych po różnorakie dyscypliny sportu) dostarczyłaby wielu dowodów (ang. evidence) zdających się je potwierdzać.

Nie inaczej jest w przypadku gier komputerowych, nawet u zarania ich istnienia, tj. w latach 40. i 50. XX wieku. Powstałe wówczas gry takie jak „Tennis for Two" z 1958 roku [1] czy późniejszy „Spacewar" zdecydowanie nie były projektami komercyjnymi, jednak nadawały się do prezentacji działania ówczesnych komputerów publiczności niezaznajomionej z budową i działaniem komputerów ani skomplikowanymi obliczeniami przez nie wykonywanymi.



Zdj. 1: Gra "Tennis for Two"

Współcześnie prezentacyjna funkcja gier komputerowych może być kojarzona z dążeniem do fotorealizmu grafiki 3D renderowanej w czasie rzeczywistym w wysokobudżetowych produkcjach, jednak nie wyczerpuje to dostępnych możliwości.

1.2 Algorytmy

Algorytm definiuje się jako ‘skończony ciąg jednoznacznych działań prowadzących do rozwiązania problemu należącego do danej klasy’ bądź ‘jednoznacznie zdefiniowaną procedurę obliczeniową, która dla otrzymanych danych wejściowych produkuje odpowiednie dane wyjściowe’ lub po prostu opis sposobu rozwiązania określonego problemu [2].

Algorytm zaimplementowany w programie komputerowym musi rozwiązywać problem w skończonym czasie i przy wykorzystaniu skończonych zasobów (głównie pamięci). Miarą kosztów jego działania jest złożoność obliczeniowa, odpowiednio: czasowa i pamięciowa, którą opisuje się jako funkcję rozmiaru danych wejściowych.

Dążenie do minimalizacji złożoności obliczeniowej przyczyniło się do opracowania różnorakich paradygmatów tworzenia algorytmów, takich jak metoda „dziel i zwyciężaj”, algorytmy zachłanne lub algorytmy genetyczne.

1.3 Algorytmy genetyczne

Algorytmy genetyczne są rodzajem heurystyki (lub metaheurystyką), inspirowanej biologicznymi procesami ewolucji i doboru naturalnego.

Ogólne działanie algorytmów genetycznych polega na wygenerowaniu zbioru potencjalnych rozwiązań (tzw. populacji) będącego podzbiorem wszystkich możliwych rozwiązań danego problemu, a następnie cyklicznemu przetwarzaniu go za pomocą operacji krzyżowania, mutacji i selekcji (operacje te są zwykle przynajmniej w pewnym stopniu losowe, stąd algorytmy genetyczne klasyfikuje się także jako algorytmy stochastyczne). Kolejne iteracje tych operacji (tzw. pokolenia) powinny powodować polepszenie średnich wyników w populacji.

Algorytmy genetyczne są stosowane w dziedzinach takich jak optymalizacja i przeszukiwanie, gdzie wykazują istotną przewagę nad tradycyjnymi algorytmami analitycznymi, dzięki wysokiej odporności, rozumianej jako równowaga między wydajnością a skutecznością, łatwości zastosowania do szerokiego spektrum problemów oraz niewrażliwości na pułapki stanowione przez rozwiązania pozorne (np. ekstrema lokalne w zadaniach optymalizacyjnych).

2. ZAŁOŻENIA I CEL PRACY

Celem niniejszej pracy jest implementacja algorytmu genetycznego wraz z funkcjonalnościami mającymi umożliwić śledzenie jego działania w trakcie wykonywania programu oraz gromadzić dane pozwalające na późniejszą analizę tegoż działania. Analiza ta powinna przynieść odpowiedź na pytanie: na ile zasadne jest użycie mechanizmu algorytmów genetycznych w taki sposób, jak w opisywanej tutaj aplikacji. Sama analiza jak i odpowiedź na wspomniane pytanie nie należą do zakresu niniejszej pracy.

2.1 Założenia

2.1.1 Mechanizm algorytmów genetycznych

Mechanizm algorytmów genetycznych, rozumianych jako rodzina algorytmów oparta o ideę doboru naturalnego i dziedziczności, posiada szereg elementów stałych [5], na czele z kluczowymi operacjami selekcji, krzyżowania i mutacji. Sposoby realizacji tych operacji (a także ich kolejność) mogą się jednak znacznie różnić [6], np. na skutek wymagań narzucanych przez rozwiązywany problem.

2.1.1.1 Tworzenie populacji

Pierwszym krokiem jest stworzenie populacji, czyli zbioru elementów zwanych osobnikami, z których każdy reprezentuje potencjalne rozwiązanie. Każdy osobnik posiada określony genom, czyli jeden lub więcej ciągów wartości odpowiadających (bezpośrednio lub w postaci zakodowanej [7]) danym, z których składa się rozwiązanie. Dla przykładu: w przypadku problemu komiwojażera populacja zawierałaby różne cykle Hamiltona przedstawione jako ciągi wierzchołków lub krawędzi składających się na dany cykl, natomiast w przypadku problemu, którego rozwiązaniem jest pojedyncza wartość liczbowa, populacja może być zbiorem liczb przedstawionych w postaci binarnej, jako ciągi zero-jedynkowe. Niezależnie od przyjętej formy i postaci, populacja powinna być losowym podzbiorem przestrzeni rozwiązań danego problemu. Rozmiar populacji jest ustalany arbitralnie.

2.1.1.2 Selekcja

Selekcja, zwana także reprodukcją, jest procesem mającym na celu wyłonienie z aktualnej populacji najbardziej wartościowych osobników (najlepszych rozwiązań) przy jednoczesnym zapewnieniu różnorodności genetycznej populacji w następnym pokoleniu.

W pierwszej kolejności należy określić jakość każdego z aktualnie rozpatrywanych rozwiązań. Dla każdego z nich obliczana jest jego wartość za pomocą tzw. funkcji celu (zwanej też, w nawiązaniu do teorii ewolucji, funkcją przystosowania). Obliczenia te

zazwyczaj uwzględniają wartości zapisane w genomie danego osobnika, ale postać funkcji celu jest zależna od rozwiązywanego problemu i formy, w jakiej zapisane są rozwiązania.

Następnie dokonywana jest selekcja na podstawie wartości osobników. Najprostszym sposobem jej realizacji jest metoda ruletki [8], czyli losowy wybór, w którym prawdopodobieństwo wybrania danego osobnika jest wprost proporcjonalne do jego wartości. Metoda ta jest wystarczająca do prawidłowego działania algorytmu, choć można ją wzbogacić o mechanizmy przeciwdziałające dominacji jednego z osobników lub zapewniające przetrwanie najlepszego z nich. Bez względu na wybraną metodę, wybrany osobnik zostaje dodany do nowego zbioru, który będzie stanowił następne pokolenie populacji lub podstawę dla jego utworzenia, jednocześnie pozostając w dotychczasowej populacji, co umożliwia jego ponowny wybór. Proces ten jest powtarzany do momentu osiągnięcia założonej liczebności nowego zbioru.

2.1.1.3 Krzyżowanie

W procesie krzyżowania dokonywana jest wymiana informacji genetycznej pomiędzy parami osobników z dotychczasowej populacji. Wymianie mogą podlegać losowe fragmenty genomu lub jego pojedyncze elementy (tzw. geny). Osobniki, które mają zostać skrzyżowanie, są również wybierane losowo, a po zakończeniu wymiany nie biorą udziału w dalszych iteracjach tego procesu. Operacja jest powtarzana dopóki w populacji występuje choć jedna para osobników nie poddanych krzyżowaniu.

2.1.1.4 Mutacja

Operacja mutacji polega na zmianie wartości losowych genów. Podobnie jak w naturze, prawdopodobieństwo zajścia takiej zmiany powinno być bardzo niskie.

2.1.2 Zastosowanie algorytmów genetycznych w grach komputerowych

Gry komputerowe nie są wymieniane wśród czołowych obszarów zastosowań algorytmów genetycznych, co może budzić zdziwienie, zważywszy na istnienie szeregu zagadnień w ramach tego medium, w których ukierunkowana losowość i adaptacyjność, charakterystyczne dla algorytmów genetycznych, wydają się być pożądane. Dotyczą one przede wszystkim dostosowania mechanik gry do działań graczy – czy to w celu zwiększenia realizmu interakcji, czy dla urozmaicenia rozgrywki – czego przykładem może być mechanika dynamicznego poziomu trudności.

Jednakże, nie można stwierdzić z całą pewnością, że algorytmy genetyczne mogą zawsze zapewnić odpowiednie rozwiązanie w we wszystkich kwestiach dotyczących mechaniki gier. Ma na to wpływ wiele czynników kształtujących tego typu algorytmy.

2.1.2.1 Cechy algorytmów genetycznych a rozwiązywanie problemów

Nawet pobieżna analiza problemów, do rozwiązywania których używane są algorytmy genetyczne, pozwala na wyróżnienie kilku cech wspólnych, które należałoby uznać za sprzyjające a może nawet konieczne.

Pierwszą z nich jest obecność funkcji matematycznej zawartej w opisie problemu, która może posłużyć za funkcję celu. Za przykład mogą posłużyć zadania optymalizacyjne polegające na poszukiwaniu ekstremum wspomnianej funkcji. Odpowiednio zdefiniowana funkcja celu pozwala na jednoznaczne przyporządkowanie wartości osobnika danej kombinacji jego genów [9]. Jest to istotne dla procesu znajdowania najlepszego rozwiązania – wprowadza element stałości i stabilizacji do tego silnie losowego procesu, ukierunkowując poszukiwania.

Wspomniana jednoznaczność wynika ze swoistej „statyczności" problemu, rozumianej jako niezmienność opisujących go danych i warunków determinujących rozwiązanie. Można to też określić jako poszukiwanie jak najlepszego rozwiązania dla jednego, ściśle określonego problemu. Przykładem dobrze ilustrującym sens tej idei jest wspomniany już problem komiwojażera, w przypadku którego w zadanym grafie ważonym należy odnaleźć najmniejszy cykl obejmujący wszystkie jego wierzchołki. Graf pozostaje zawsze taki sam – w trakcie rozwiązywania nie zmienia się ani liczba wierzchołków, ani połączenia między nimi, ani wagi krawędzi. Jakąkolwiek z wymienionych zmian należałoby zinterpretować jako wygenerowanie nowego grafu, a w konsekwencji – nowego problemu do rozwiązania. Analogicznie przykładem problemu nie spełniającego tego założenia mogłoby być rozpoznawanie pisma – zadanie wymagające analizy wielu próbek, z których każda stanowi właściwie osobny problem do rozwikłania. W tym przypadku zastosowanie znajdują sieci neuronowe, które na podstawie dotychczasowych problemów „uczą się" rozwiązywać kolejne. Co prawda algorytmy genetyczne również są stosowane do tworzenia systemów uczących się, jednak jest to „nauka" metodą prób i błędów przez wielokrotne podchodzenie do tego samego zadania. Możliwość zastosowania takiej metody jest istotną cechą łączącą omawiane tu problemy.

Kolejnym elementem, na który warto zwrócić uwagę, jest zakres dopuszczalnych rozwiązań. Stosowanie algorytmów genetycznych jest zasadne w przypadkach, w których możliwy kompromis pomiędzy jakością wyniku a czasem uzyskania go. Akceptowalność rozwiązań reprezentowanych przez osobniki jest określana poprzez porównanie przyporządkowanej im wartości funkcji celu z ustalonym wcześniej progiem. Wspomniany kompromis powoduje, że wartość tego progu jest istotnie niższa od maksymalnej możliwej

dla danego problemu. To z kolei oznacza, że osobnik będący końcowym rezultatem pracy algorytmu genetycznego jedno z rozwiązań „wystarczająco dobrych", nie koniecznie najlepsze. W przypadkach, w których taki kompromis nie jest możliwy, o wiele bardziej zasadne jest zastosowanie wyspecjalizowanego algorytmu analitycznego. [10]

2.1.2.2 Zastosowanie w grach – przeszkody

Podstawowym czynnikiem stojącym na przeszkodzie zastosowaniu mechanizmu algorytmów genetycznych w grach komputerowych zdaje się być ich interaktywność. Działania gracza powodują zmianę stanu gry, z punktu widzenia algorytmu tworząc nowy problem do rozwiązania.

Odbija się to przede wszystkim na ocenie populacji, gdyż wartości zwracane przez dotychczasową funkcję celu mogą okazać się błędne w nowych okolicznościach. Utrata jednoznaczności przyporządkowania wartości do danego osobnika wydaje się być nieuchronna. Nie jest przy tym pewne, czy w takich warunkach wartości genów mają bezpośrednie przełożenie na jakość reprezentowanego rozwiązania. Być może należy zidentyfikować zewnętrzne czynniki wpływające na „sprawność" osobników i oprzeć nową metodę ewaluacji właśnie na nich.

Niestety takie podejście wiąże się z ryzykiem wydłużenia czasu potrzebnego na wykonanie jednego cyklu pracy algorytmu genetycznego. Jeśli osobniki miałyby być powiązane z obiektami w grze, a ocena miała wynikać z jakości ich interakcji z innymi elementami gry (w tym z graczem), to selekcja potrwałaby zdecydowanie dłużej niż gdyby została przeprowadzona przy użyciu funkcji matematycznej przyjmującej geny jako argumenty.

Opisanych wyżej problemów można uniknąć, stosując algorytmy genetyczne do zadań niezależnych od działań gracza i innych źródeł zmienności, a nawet nie związanych samą rozgrywką, lecz z tworzeniem gry lub jej elementów. Zagadnienia te nie wchodzą jednak w zakres niniejszej pracy, a samo rozwiązanie jest jedynie unikiem.

Istnieje natomiast prawdopodobieństwo, że ukierunkowana losowość zapewniana przez algorytmy genetyczne wystarczy, by dać zadowalające wyniki nawet w tak niesprzyjających warunkach. Z punktu widzenia gracza dużo ważniejsza może okazać się obecność i ciągłość działania procesu dostosowywania elementów rozgrywki niż uzyskanie przez ten proces jakiegoś konkretnego rozwiązania.

2.2 Wykorzystane technologie

Najważniejszym elementem aplikacji wchodzącej w skład niniejszej pracy jest zaimplementowany algorytm genetyczny, natomiast gra stanowi jedynie platformę konieczną,

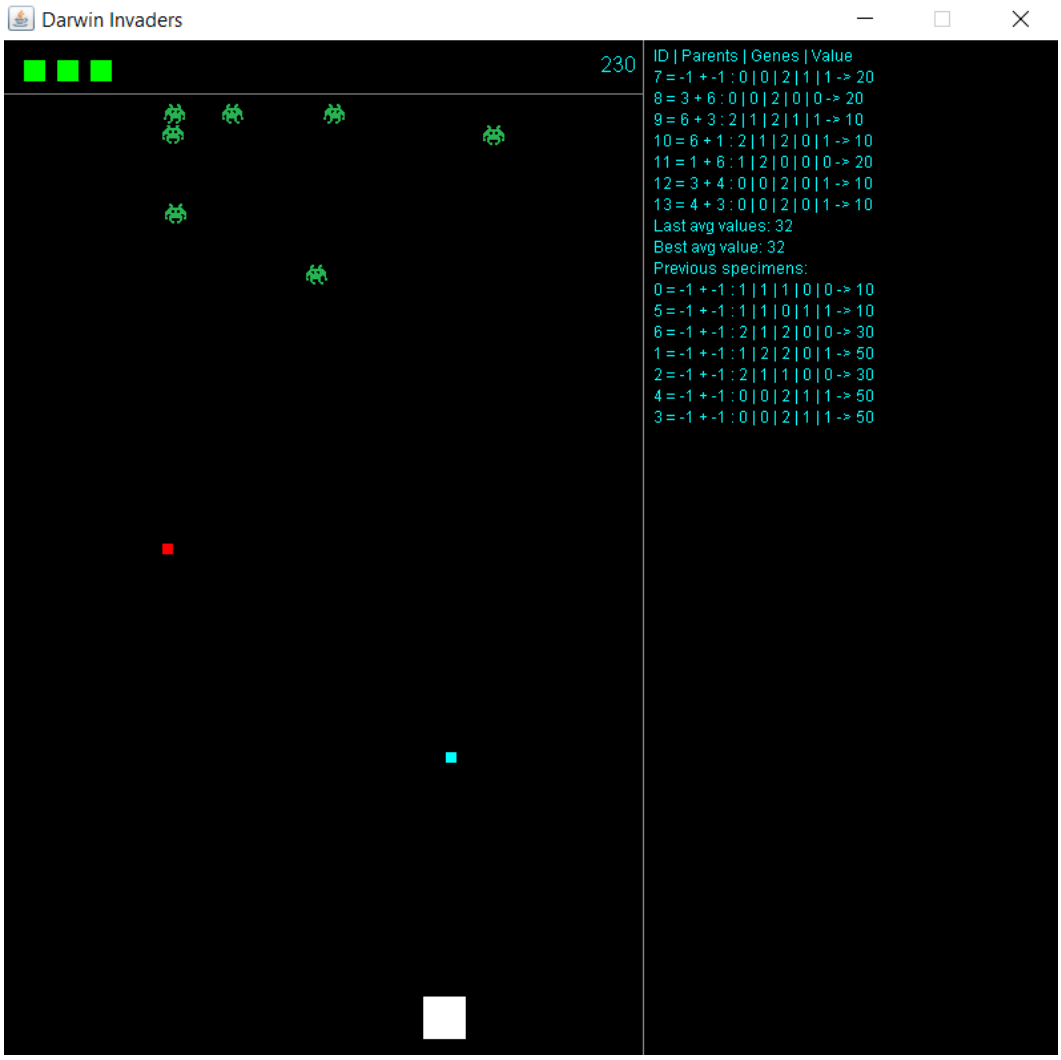
by implementacja mogła mieć miejsce. Tak określona hierarchia sprawia, że strona wizualna programu ma relatywnie niewielkie znaczenie, dlatego też wybór języka Java z wykorzystaniem biblioteki AWT okazał się wystarczający do realizacji wszystkich istotnych założeń dotyczących funkcjonalności programu.

3. OPIS APLIKACJI

3.1 Wygląd i działanie aplikacji

3.1.1 Okno aplikacji

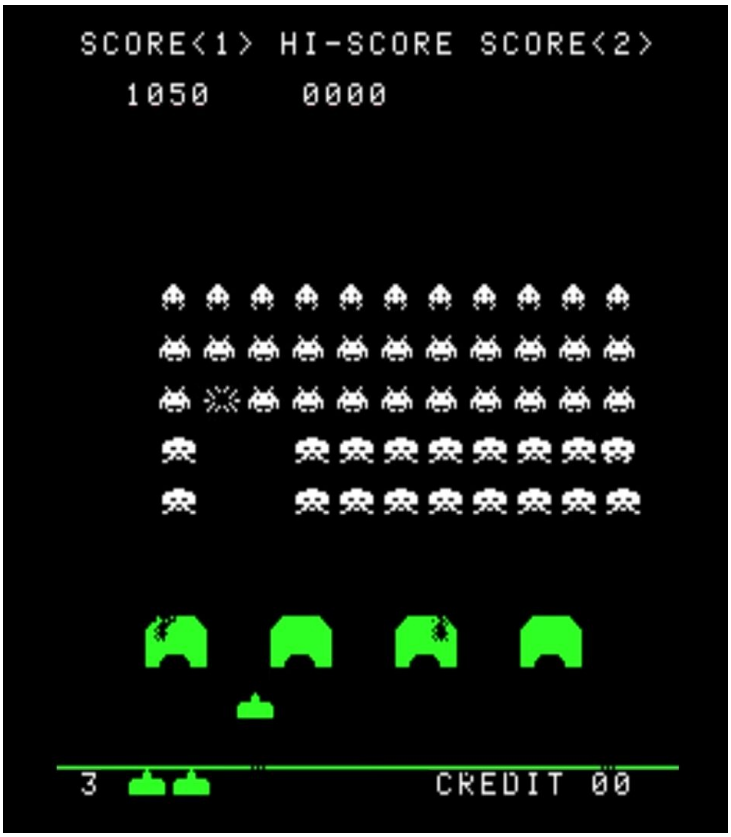
Okno aplikacji ma wymiary 800x800 pikseli i jest podzielone na dwa obszary: właściwy obszar gry o szerokości 480 pikseli oraz dodatkowy obszar, w którym wyświetlane są dane związane z pracą algorytmu genetycznego, o szerokości 320 pikseli. Wymiary te można zmienić jedynie poprzez modyfikację wartości odpowiadających im zmiennych w kodzie źródłowym aplikacji.



Rys. 1: Widok okna aplikacji podczas rozgrywki

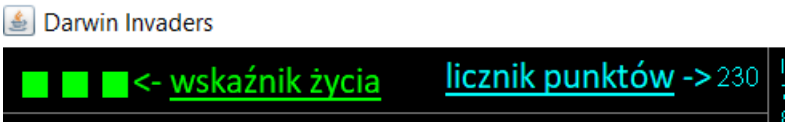
3.1.2 Zasady gry

Aplikacja jest inspirowana grą „Space Invaders”, co jest widoczne w wielu elementach rozgrywki, w tym w warstwie wizualnej, mimo iż implementacja mechanizmu algorytmów genetycznych wymusiła wprowadzenie szeregu istotnych zmian.



Rys. 2: Space Invaders (źr.: oldgamesdownload.com)

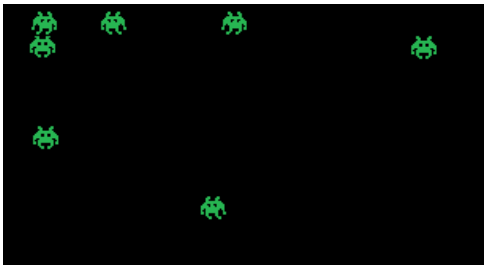
Na samej górze obszaru gry znajdują się dwa elementy: wskaźnik, będący graficzną reprezentacją licznika „życia” gracza, pod postacią maksymalnie trzech zielonych kwadratów oraz licznik punktów zdobytych przez gracza w trakcie aktualnej rozgrywki.



Rys. 3: Elementy GUI - wskaźnik życia i licznik punktów

Wrogowie są reprezentowani przez sprite’y wzorowane na kosmitach ze wspomnianej gry. Są oni tworzeni grupami (falami) i wprowadzani na górnym krańcu obszaru gry pojedynczo, w krótkich odstępach czasowych. Mogą poruszać się w lewo, w prawo lub w dół oraz strzelać. Wprowadzanie nowej fali rozpoczyna się dopiero w momencie wyczerpania

poprzedniej, tj. kiedy wszyscy dotychczasowi wrogowie zostaną wyeliminowani lub opuszczą obszar gry.



Rys. 4: Wygląd wrogów w grze

Gracz steruje obiektem reprezentowanym przez biały kwadrat u dołu okna. Może on poruszać się w lewo lub w prawo albo strzelać.

Pociski wystrzelone przez gracza i wrogów są reprezentowane odpowiednio przez jasnoniebieskie i czerwone kwadraty. Różnią się ona także kierunkiem (właściwie: zwrotem) ruchu – te pierwsze poruszają się pionowo w górę, natomiast te drugie poruszają się pionowo w dół.

Pociski gracza pojawiają się w grze po naciśnięciu przez niego przycisku odpowiadającego komendzie strzału (domyślnie jest to „w” lub „strzałka w górę”), jednak nie częściej niż pozwala na to odgórnie ustalony limit czasowy.

Wrogie pociski są wystrzeliwane przez losowego wroga w odgórnie ustalonych odstępach czasowych. Długość tego odstępu podlega modyfikacjom zależnym od liczebności wrogów – wraz z ich ubywaniem w ramach danej fali czas pomiędzy kolejnymi wystrzałami staje się krótszy.

Zadaniem wrogów jest dotarcie do dolnego krańca obszaru gry. Zadaniem gracza jest uzyskanie jak najwyższego wyniku punktowego poprzez niszczenie wrogów, jednocześnie jak najdłużej unikając kolizji z nimi i ich pociskami.

3.1.2.1 Kolizje

W trakcie gry może dojść do następujących kolizji pomiędzy obiektami:

- pomiędzy pociskiem gracza a wrogiem – pocisk zostaje usunięty z gry a dany wróg „zniszczony”, a jego wartość zostaje dodana do aktualnej puli punktów.
- pomiędzy pociskiem wroga a obiektem gracza – pocisk zostaje usunięty z gry a wartość licznika „życia” gracza zostaje zmniejszona o 1.

Jeśli wartość tego licznika spadnie to zera, gra zostaje zakończona.

- pomiędzy obiektem gracza a wrogiem – wróg zostaje „zniszczony” a jego wartość dodana do wyniku punktowego. Licznik „życia” ulega zmniejszeniu o 1.

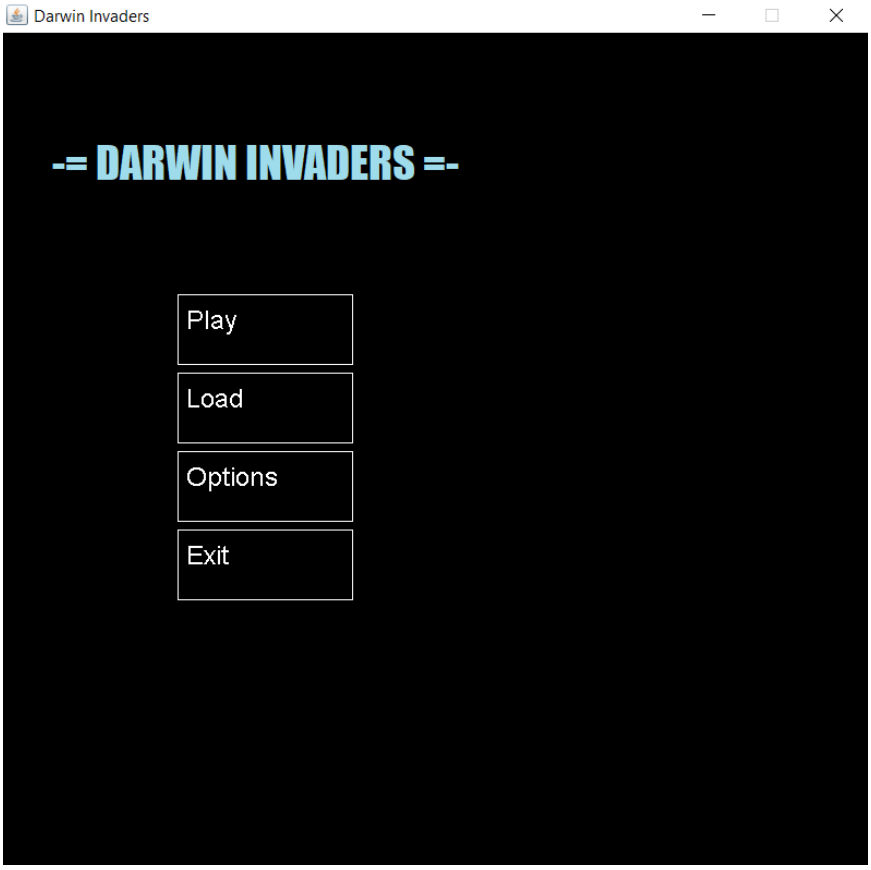
Jeśli dany wróg dotrze do dolnego krańca obszaru gry, zostanie on „zniszczony”, ale jego wartość będzie odjęta od wyniku punktowego.

3.1.2.2 Zakończenie gry

Gra kończy się w momencie, gdy wartość licznika „życia” osiągnie 0 lub gracz sam zakończy rozgrywkę.

3.1.3 Struktura aplikacji

3.1.3.1 Menu główne

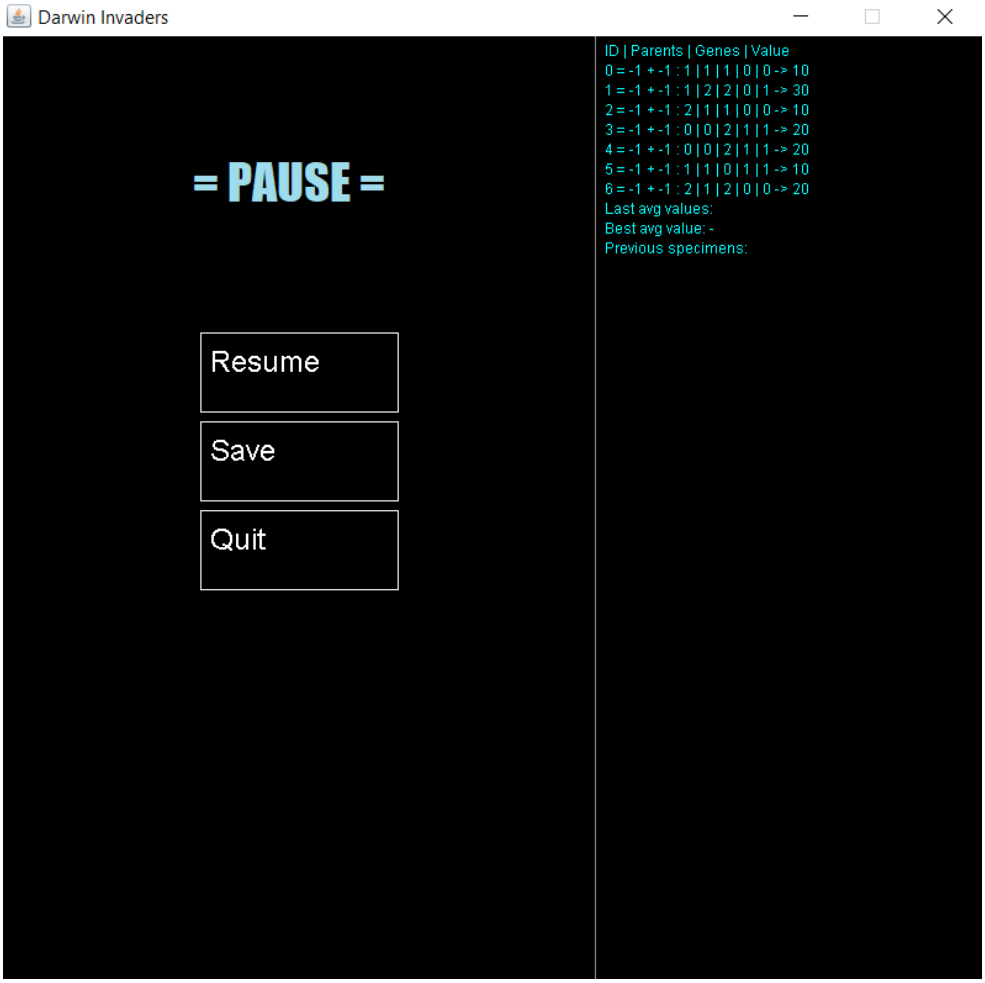


Rys. 5: Widok menu głównego

Menu główne zostaje wyświetlone zaraz po uruchomieniu aplikacji. Składa się ono z czterech przycisków:

- Play – kliknięcie spowoduje rozpoczęcie nowej rozgrywki;
- Load – kliknięcie spowoduje odczyt danych dotyczących stanu poprzednio zapisanej rozgrywki i wznowienie jej, o ile istnieją pliki z tymi danymi;
- Options – kliknięcie spowoduje przejście do ekranu opcji;
- Exit – kliknięcie spowoduje zakończenie pracy aplikacji.

3.1.3.2 Menu pauzy



Rys. 6: Widok menu pauzy

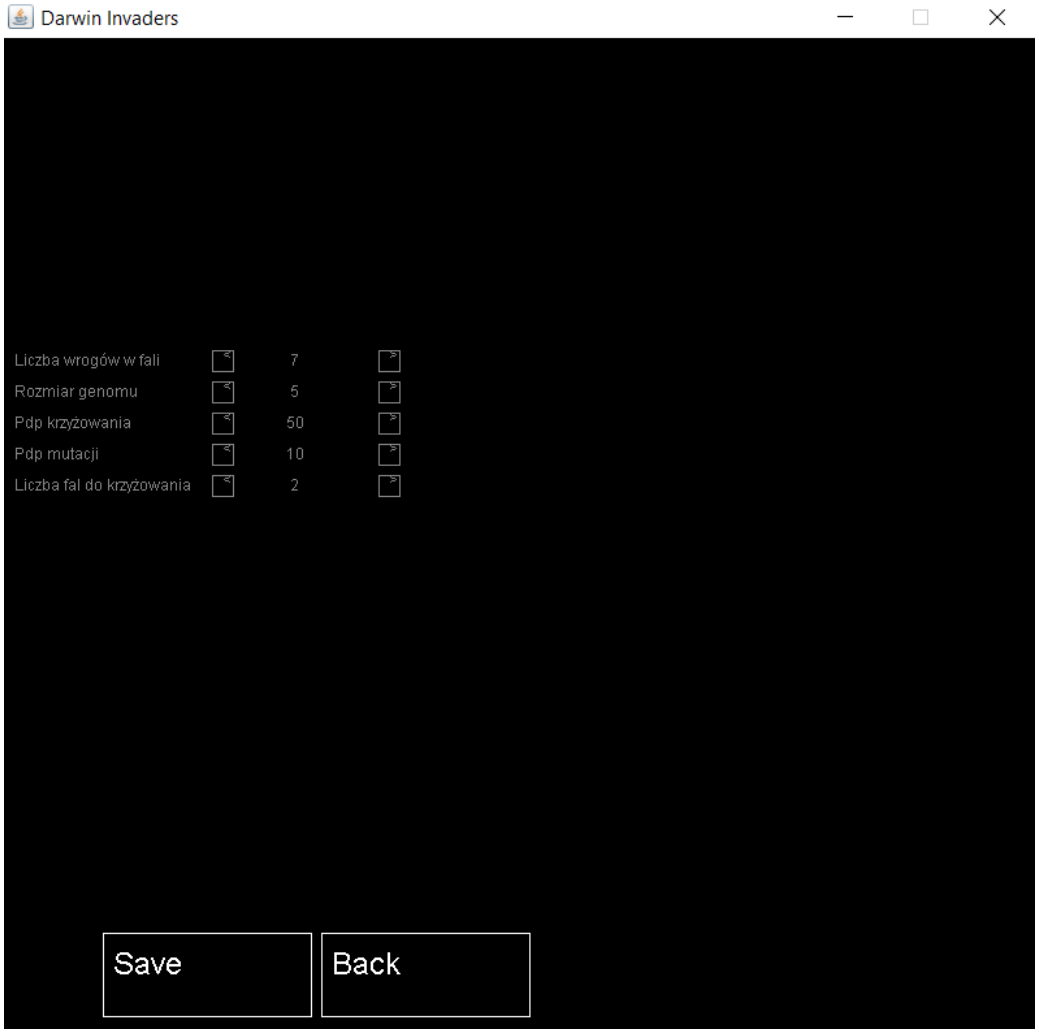
Jeśli w trakcie rozgrywki zostanie naciśnięty przycisk odpowiadający komendzie pauzy (domyślnie: klawisz „Escape”), to rozgrywka ta zostanie wstrzymana a w miejscu obszaru gry będzie wyświetlane menu pauzy. Obszar z danymi dotyczącymi pracy algorytmu genetycznego pozostanie nadal widoczny przez cały czas trwania pauzy.

Menu składa się z następujących przycisków:

- Resume – kliknięcie spowoduje zakończenie pauzy, powrót do aktualnej rozgrywki;
- Save – kliknięcie spowoduje zapis kluczowych danych dotyczących stanu aktualnej rozgrywki do odpowiednich plików;
- Quit – kliknięcie spowoduje zakończenie aktualnej rozgrywki i powrót do menu głównego.

Szczegółowe informacje o zapisie i odczycie gry znajdują się w sekcji 3.2.2.1.

3.1.3.3 Ekran opcji



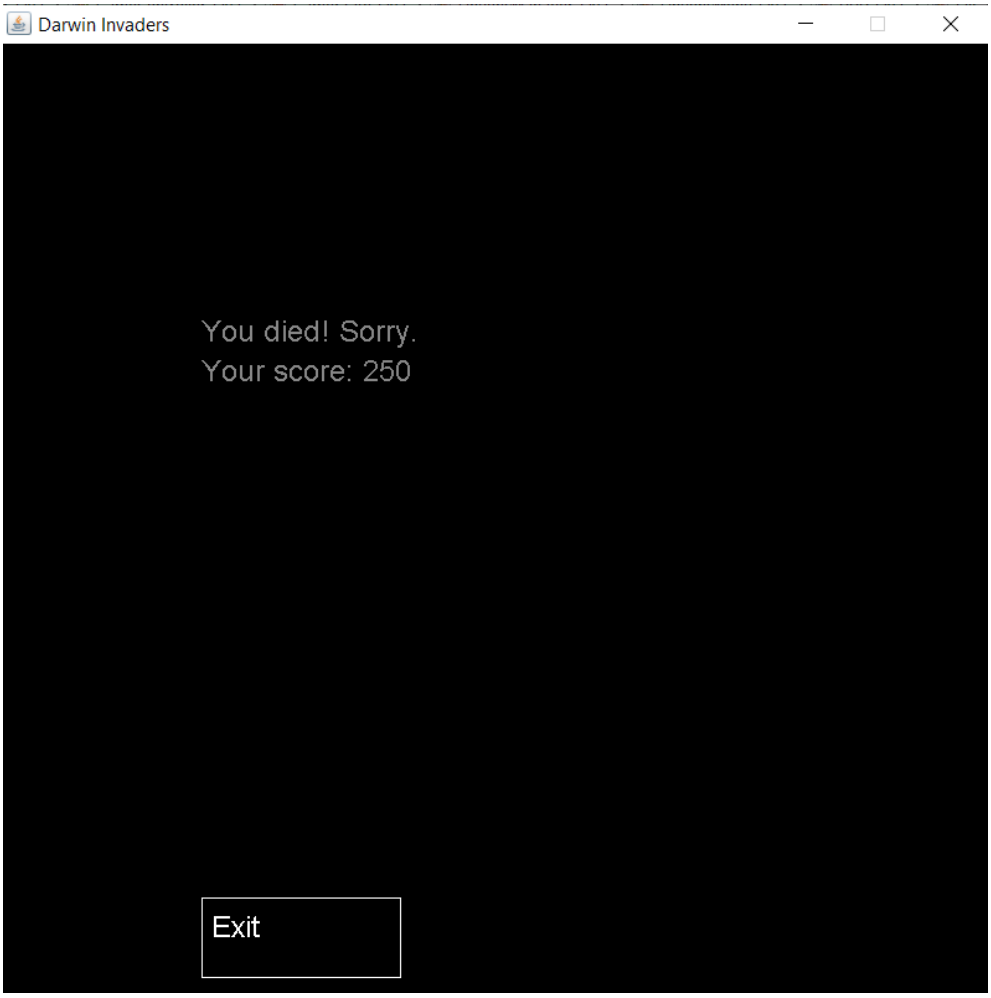
Rys. 7: Widok ekranu opcji

Ekran opcji pozwala na modyfikację parametrów istotnych dla pracy algorytmów genetycznych zaimplementowanych w aplikacji, czyli: rozmiar populacji (pod nazwą „Liczba wrogów w fali”), rozmiar genomu, prawdopodobieństwo krzyżowania i mutacji oraz rozmiar populacji dla implementacji drugorzędnej (pod nazwą „Liczba fal do krzyżowania”).

W skład ekranu opcji chodzą też dwa przyciski (poza przyciskami służącymi do modyfikacji wartości parametrów):

- Save – kliknięcie spowoduje przekazanie aktualnej wartości parametrów ze zmiennych tymczasowych do zmiennych globalnych, dzięki czemu będą one mogły zostać uwzględnione w najbliższej nowej rozgrywce i ewentualnie zapisane do pliku;
- Back – kliknięcie spowoduje powrót do menu głównego.

3.1.3.4 Ekran końca gry



Rys. 8: Widok ekranu końca gry

Jeśli gracz straci w trakcie rozgrywki wszystkie dostępne „życia”, to zostanie ona zakończona oraz zostanie wyświetlony komunikat pożegnalny wraz z informacją o liczbie punktów uzyskanych w trakcie tej rozgrywki. Kliknięcie przycisku „Exit” spowoduje powrót do menu głównego.

3.2 Silnik aplikacji

3.2.1 Implementacja algorytmów genetycznych

Najważniejszym elementem opisywanej tu aplikacji są dwie niezależne implementacje algorytmów genetycznych. Pierwsza z nich obsługuje zachowanie wrogów i ma znaczenie priorytetowe – to jej działanie jest przedmiotem obserwacji będącej podstawowym celem tej aplikacji. Druga implementacja obsługuje proces ustalania początkowych pozycji na których mają pojawić się wrogowie wprowadzani do gry. Jej znaczenie jest jednak marginalne, co okazało się na stosunkowo późnym etapie tworzenia aplikacji, co z kolei sprawiło, że jej wymiana na lepsze rozwiązanie stała się nieopłacalna.

3.2.1.1 Obsługa zachowań wrogów

3.2.1.1.1 Kluczowe pojęcia

Z punktu widzenia algorytmu kolejne fale wrogów stanowią populację i jej pokolenia, natomiast sami wrogowie są równoważni osobnikom. Genom każdego wroga opisuje sekwencję ruchów, jakie będzie on cyklicznie wykonywał, zapisaną jako lista wartości całkowitoliczbowych z przedziału [0,2], gdzie „0" oznacza ruch w prawo, „1" oznacza ruch w lewo, a „2" ruch w dół.

3.2.1.1.2 Selekcja

Każdy wróg usunięty z rozgrywki – na skutek zniszczenia lub dotarcia do dolnego krańca obszaru gry – zostaje przechwycony przez obiekt klasy CrossingManager, a jego genom i wartość zostają pobrane do dalszej obróbki. Kiedy CrossingManager zbierze dane wszystkich wrogów z danej fali, przystępuje do wykonania standardowych operacji algorytmu genetycznego, rozpoczynając od selekcji.

```
public void collectSpecimens(ObjHandler handler)
{
    int n = handler.getSize();
    for(int i=0; i<n; i++)
    {
        GameObject spec = handler.get(i);
        if(!spec.isAlive()) //pobieramy tylko te, które już odpadły z gry
        {
            behaviours.add(((BasicEnemy)spec).copyBehaviour());
            values.add(((BasicEnemy)spec).getValue());
            ids.add(((BasicEnemy)spec).getId());

            Integer[] tmpInt = new Integer[2];
            int[] tmpint = ((BasicEnemy)spec).getParents();
            tmpInt[0] = tmpint[0];
            tmpInt[1] = tmpint[1];
            parents.add(tmpInt);
        }
    }
}
```

Rys. 9: Metoda klasy CrossingManager służąca do pobierania danych z wyeliminowanych wrogów

Wartość wrogów nie jest obliczana przy użyciu funkcji celu na podstawie ich genów, lecz jest ustalana w trakcie rozgrywki, czyli pomiędzy kolejnymi uruchomieniami opisywanego tu procesu. Wartość ta wynika bezpośrednio z odległości przebytej w pionie przez danego osobnika, co w prawdzie odpowiada liczbie wykonanych przez niego ruchów w dół, lecz nie jest to jedyny ani najbardziej istotny czynnik. Wpływ działań gracza sprawia, że za najbardziej wartościowego osobnika należałoby uznać takiego, który nie tylko byłby w stanie szybko dotrzeć do dolnego krańca obszaru gry, ale także byłby trudny do trafienia przez gracza dzięki odpowiedniej sekwencji ruchów na boki. Wszystkie te czynniki sprawiają,

że próby sformułowania funkcji oceniającej wartość osobników na podstawie ich genów wydają się mniej opłacalne od oparcia tej ewaluacji o ich rzeczywiste dokonania w trakcie rozgrywki. Do tego wartość wrogów bezpośrednio przekłada się na punkty uzyskiwane przez gracza za ich eliminację, co może wpłynąć na decyzje podejmowane przez niego w trakcie rozgrywki, powodując np. opóźnienie w eliminacji celów, aby uzyskać wyższy wynik punktowy.

```
int move = behaviour.get(behaviourStep);
model = sprite.getImage(move);
switch (move) {
    case 0:
        this.speedX = GlobalVars.enemySpeed;
        this.speedY = 0;
        break;
    case 1:
        this.speedX = -GlobalVars.enemySpeed;
        this.speedY = 0;
        break;
    default:
        this.speedY = GlobalVars.enemySpeed;
        this.speedX = 0;
        value += 10;
        break;
}
```

Rys. 10: Ustalanie ruchu na podstawie genów

Sama procedura selekcji jest realizowana za pomocą standardowej metody ruletki, tj. losowo z prawdopodobieństwem wybrania danego osobnika proporcjonalnym do jego wartości. Wylosowane w ten sposób osobniki są następnie poddawane operacji krzyżowania.

```
//selekcja
double[] chances = new double[n];
double valSum = 0.0;
for(int i=0; i<n; i++)
{
    chances[i] = values.get(i);
    valSum += chances[i];
}
//oblicz prawdopodobieństwo wylosowania
for(int i=0; i<n; i++)
    chances[i] /= valSum;
//przekształć na dystrybuantę
for(int i=1; i<n; i++)
    chances[i] += chances[i-1];
//losowanie do krzyżowania
for(int i=0; i<n; i++)
{
    double attempt = rand.nextDouble(); //losuj 0..1
    int pick =-1;       //wybrany element
    int j = 0;          //aktuajnie rozpatrywany element
    //wybierz pierwszy element, którego wart. dystrybuanty > od wylosowanej wart.
    //przeglądaj kolejne elementy do momentu wyboru lub do końca tablicy
    while((pick<0)&&(j<n))
    {
        if(chances[j]>=attempt) pick = j;
        j++;
    }
    //jeśli nie dokonano wyboru - przypisz ostatni element
    //Może zajść, jeśli dystrybuanta nie zsumuje się do 1.0
    if(pick<0) pick = n-1;
    tmpPop.add(behaviours.get(pick));
    tmpId.add(ids.get(pick));
}
```

Rys. 11: Fragment kodu odpowiadający za selekcję

3.2.1.1.3 Krzyżowanie

Spośród osobników, które zakwalifikowały się do niniejszego etapu, cyklicznie losowane są dwa, dopóki to możliwe. Następnie dla wybranej pary wykonywane jest losowanie mające ustalić, czy będą się ze sobą krzyżować. Jeśli wylosowana wartość nie przekracza uprzednio ustalonej wartości prawdopodobieństwa (w ustawieniach opcji opisanych w rozdziale 3.1.3.3), to wybrane osobniki zostaną poddane krzyżowaniu jednopunktowemu. Polega ono na ustaleniu losowego punktu podziału genomów obu osobników a następnie przekazaniu genów do osobników potomnych w taki sposób, że pierwszy potomek otrzymuje geny poprzedzające punkt podziału od pierwszego rodzica, a geny następujące po punkcie podziału od drugiego rodzica, natomiast drugi potomek otrzymuje je w odwrotnym porządku.

```
LinkedList<Integer> a = behaviours.get(x1);
LinkedList<Integer> b = behaviours.get(x2);

//losuj czy krzyżują
if((rand.nextInt(100)+1)<=GlobalVars.probMix)
{
    //miejsce cięcia
    int cut = rand.nextInt(GlobalVars.behaviourSize-1)+1;
    //listy tymczasowe
    LinkedList<Integer> na = new LinkedList<Integer>();
    LinkedList<Integer> nb = new LinkedList<Integer>();
    //przeniesienie elementów...
    for(int i=0; i<GlobalVars.behaviourSize; i++)
    {
        if(i<cut)
        {
            //a -> na, b-> nb
            na.add(a.get(i));
            nb.add(b.get(i));
        }
        else
        {
            //a -> nb, b-> na
            na.add(b.get(i));
            nb.add(a.get(i));
        }
    }
    a.clear();
    a = na;
    b.clear();
    b = nb;
```

Rys. 12: Fragment kodu realizującego krzyżowanie

Następnie potomkowie zostaną umieszczeni w puli przeznaczonej do dalszej obróbki. Jeśli wylosowana liczba przekroczyła wartość prawdopodobieństwa krzyżowania, to wybrane osobniki przechodzą do wspomnianej puli bez zmian. Tak samo dzieje się z osobnikiem nie mającym pary.

3.2.1.1.4 Mutacja

Procedura mutacji polega na zmianie wartości losowych genów. Realizowana jest poprzez przejrzenie wszystkich genów osobników, które zostały zakwalifikowane do niniejszej operacji. Dla każdego z tych genów losowana jest liczba, podobnie jak to miało miejsce podczas krzyżowania: jeśli liczba ta jest mniejsza od wartości prawdopodobieństwa zajścia mutacji, to wskazany gen otrzymuje nową wartość z zakresu [0,2].

```
//mutacja
n = behaviours.size();
for(int i=0; i<n; i++) //dla każdego osobnika
{
    for(int j=0; j<GlobalVars.behaviourSize; j++)  //dla każdego genu
        if((rand.nextInt(100)+1)<=GlobalVars.probMutation)
        {
            behaviours.get(i).set(j, rand.nextInt(3));
        }
}
```

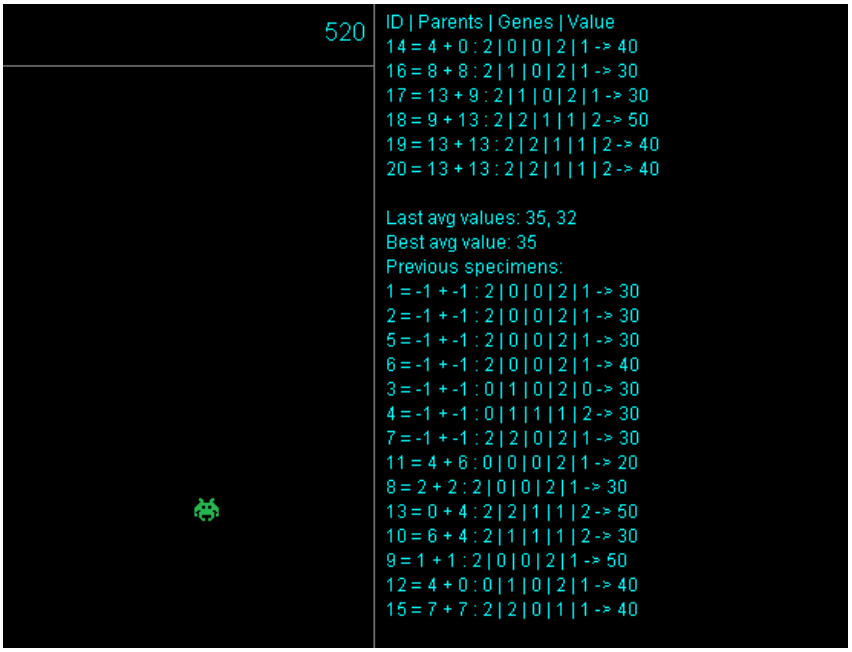
Rys. 13: Fragment kodu realizujący mutację

3.2.1.1.5 Operacje końcowe

Po zakończeniu procedury mutacji przetworzone genomy zostają wprowadzone do nowych wrogów (wraz z danymi identyfikacyjnymi ich rodziców), wygenerowanych na potrzeby kolejnej fali.

3.2.1.1.6 Prezentacja danych

W trakcie rozgrywki (włącznie z pauzą) w obszarze dodatkowym, znajdującym się po prawej stronie okna gry, wyświetlane są dane dotyczące pracy algorytmu obsługującego zachowania wrogów.



Rys. 14: Widok pracy algorytmu obsługującego zachowanie wrogów

W pierwszej kolejności są to dane dotyczące osobników z aktualnej fali, którzy zostali już wprowadzeni do rozgrywki i nadal są aktywni. Są one wyświetlane w postaci liczbowej i uporządkowane według następującej notacji:

$$\langle Identyfikator\ wroga \rangle = \langle Id\ rodzica\ A \rangle + \langle Id\ rodzica\ B \rangle : \langle Genom \rangle \rightarrow \langle Wartość\ wroga \rangle$$

Jeśli oba identyfikatory rodziców danego wroga mają taką samą wartość, to oznacza to, że wróg ten nie został poddany krzyżowaniu i w poprzedniej fali posiadał identyfikator o takiej właśnie wartości. Jeśli jednak identyfikatory rodziców są ujemne, to osobnik ten pochodzi z pierwszego pokolenia.

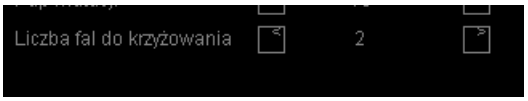
Następne dwa wiersze są zarezerwowane dla średnich wartości populacji: w pierwszym prezentowane są średnie dla odgórnie określonej liczby ostatnich pokoleń, natomiast w drugim znajduje się najwyższa średnia spośród wszystkich dotychczasowych fal. Dane te są aktualizowane w momencie wyczerpania aktualnej fali.

Kolejne wiersze przeznaczone są na listę z danymi wrogów, którzy nie uczestniczą już w rozgrywce. Liczba pozycji na tej liście jest ograniczona odgórnym limitem, po osiągnięciu którego najstarsze wpisy są sukcesywnie usuwane.

3.2.1.2 Obsługa punktów początkowych

Druga implementacja, która obsługuje proces ustalania punktów początkowych dla wrogów należących do danej fali, pracuje na takich samych zasadach jak ta, która została opisana powyżej (tj. w rozdziale 3.2.1.1), jednak z kilkoma istotnymi różnicami.

Przede wszystkim w tym przypadku osobnikami są kolejne fale, a ich genomami – listy wartości współrzędnych poziomych, wyznaczające punkty, w których mają pojawiać się wrogowie (współrzędna pionowa jest stała). Długość tych list (tożsama z wielkością genomu) jest równa liczebności wrogów przewidzianych na jedną falę, natomiast rozmiar populacji jest określony przez wartość parametru dostępnego z poziomu ekranu opcji pod nazwą „Liczba fal do krzyżowania”.



Rys. 15: Opcja do określania liczby fal do krzyżowania

Kolejną istotną różnicą jest sposób oceny osobników. Opisywany tu mechanizm miał w założeniu służyć z jednej strony eliminacji czystej losowości z procesu rozmieszczania wrogów, a z drugiej uniknięciu nadmiernej przewidywalności, która mogłaby doprowadzić do sytuacji, w której gracz zniszczyłby wszystkich wrogów w danej fali już w momencie ich

pojawienia się w grze, zanim zdążyliby się poruszyć. Z tego powodu za podstawę do oceny została przyjęta maksymalna liczba wrogów obecnych w rozgrywce w ramach danej fali. Najwyższa możliwa wartość jest równa rozmiarowi populacji wrogów. Obniżenie tego wyniku możliwe byłoby na skutek zniszczenia przez gracza części wrogów przed zakończeniem wprowadzania ich do gry, np. na skutek poprawnego przewidzenia miejsca ich pojawienia się. Niestety aktualnie przyjęte wymiary obszaru gry skutecznie utrudniają, a może nawet uniemożliwiają szybką eliminację celów (nie da się także wykluczyć, że to nie jedyny czynnik), przez co każda fala uzyskuje maksymalny wynik. W rezultacie mechanizm zaimplementowanego algorytmu genetycznego daje takie same efekty jak metoda czysto losowa. Nie sprawia to jednak, że staje się on zupełnie bezużyteczny, ponieważ nadal może służyć jako przykład negatywny.

Poprawę sytuacji mogłoby przynieść połączenie obu implementacji. Pozioma współrzędna punktu startowego zostałaby przypisana do każdego wroga jako kolejny chromosom jego genomu i podlegała obróbce równolegle z sekwencjami ruchów. W założeniu pozwoliłoby to wyeliminować sytuacje, w których potencjalnie wartościowe osobniki zostają szybko zniszczone, ponieważ zostały wprowadzone do gry zbyt blisko bocznych krańców obszaru gry, przez co – mając ograniczone pole manewru – stają się łatwym celem. Rozwiązanie to mogłoby także pozytywnie wpłynąć na różnorodność zachowań. Niestety zaproponowana tu zmiana wiązałaby się ze sporą ingerencją w kod źródłowy aplikacji – większą niż jest to możliwe w momencie pisania niniejszych słów.

3.2.2 Zapis i odczyt

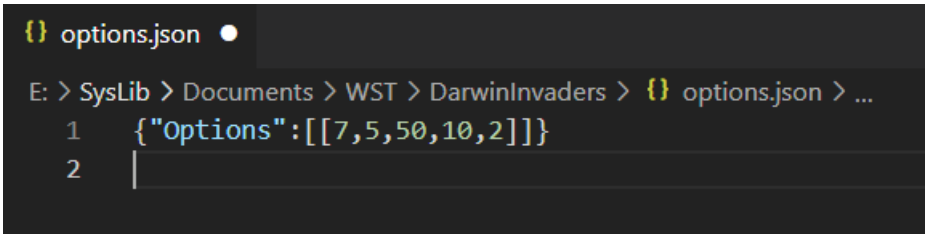
Dla ułatwienia użytkowania aplikacji, zwłaszcza w zakresie obserwacji i analizy pracy algorytmów genetycznych, została ona wzbogacona o szereg funkcjonalności związanych z obsługą plików.

3.2.2.1 Stan gry

Pierwszymi ze wspomnianych funkcjonalności są zapis i odczyt stanu gry, które powinny pozwolić na wydłużenie czasu obserwacji i zebranie danych z większej liczby pokoleń, co mogłoby być niemożliwe w ramach pojedynczej sesji. Operacje zapisu i odczytu przeprowadzane są na skutek działania użytkownika, tj. kliknięcie odpowiadających im przycisków znajdujących się odpowiednio w menu pauzy i menu głównym.

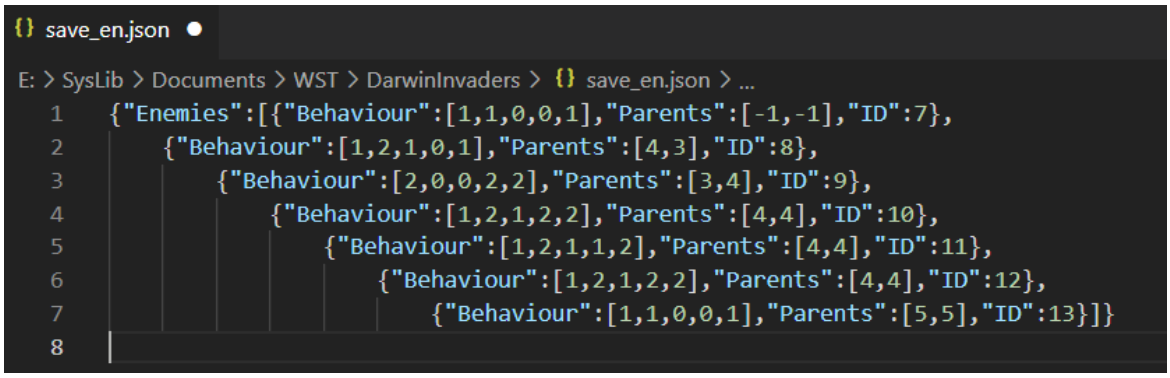
Zapisowi podlegają następujące dane:

- Wartości opcji



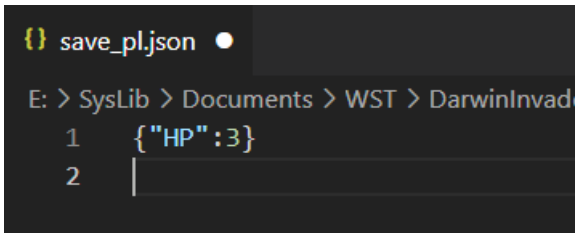
Rys. 16: Elementy danych podlegające zapisowi

- Dane wrogów z ostatniej w pełni wprowadzonej fali: identyfikatory własne i rodziców oraz sekwencje ruchów



Rys. 17: Elementy danych podlegające zapisowi

- Dane gracza: wartość licznika „życia”



Rys. 18: Elementy danych podlegające zapisowi

Dane te są przechowywane w plikach w formacie Json.

Wybór takiego właśnie zestawu danych podyktowany był względami praktycznymi – funkcjonalność związana z obsługą plików została dodana na relatywnie późnym etapie tworzenia aplikacji, przez co metody jej realizacji musiały być dostosowane do zastanych procedur i struktury programu. Odpowiednim rozwiązaniem okazało się pobieranie danych z każdego z wrogów w momencie wprowadzania ich do gry. W momencie zebrania kompletu danych z dwóch pokoleń, starsze z nich zostają odrzucone. Dzięki temu w każdym momencie gry, z drobnym wyjątkiem czasu przed zakończeniem wprowadzania pierwszego pokolenia, gracz posiada zasoby wystarczające do bezpiecznego dokonania zapisu oraz późniejszego odczytu, na skutek którego wrogowie zostaną odtworzeni w stanie odpowiadającym

momentowi rozpoczęcia fali, do której należeli. Podejście to pozwala także na ograniczenie ilości danych koniecznych do utrwalenia, np. nie ma potrzeby zachowywania wartości wrogów, ponieważ w chwili wprowadzania ich do gry dla każdego z nich wynosi ona tyle samo (czyli wartość minimalną, aktualnie 10).

Nieco inaczej jest w kwestii danych dotyczących gracza. Zapisywane są one w takim stanie, w jakim znajdują się w chwili wydania komendy zapisu, co może powodować pewnego rodzaju rozdźwięk pomiędzy odtworzonym stanem gracza a stanem wrogów, np. jeśli poziom „życia" gracza w momencie zapisu gry różnił się (tj. był niższy) od poziomu z początku fali. Na pierwszy rzut oka może się to wydawać błędem projektowym, jednak może być to jeden z tych błędów, które stają się kluczowym elementem gry, w tym przypadku: jako zabezpieczenie przed częstemu nadużyciu systemu zapisu, polegającym na odczytywaniu stanu sprzed momentu popełnienia przez gracza jakiegoś błędu, aby w ten sposób uniknąć jego konsekwencji. O ile za zwyczaj proceder ten jest niegroźny a nawet stanowi o swoistej przewadze gier nad rzeczywistością, to w przypadku opisywanej tu aplikacji niewątpliwie doprowadziłby do przekłamania danych, jakie są przez nią gromadzone na potrzeby analizy pracy algorytmu genetycznego. Dążenie do zapobiegnięcia nadużyciom (przynajmniej tym, które udało się zidentyfikować) stanowiło także postawę dla decyzji o usunięciu wartości licznika punktów z puli danych przeznaczonych do zapisu, aby gracz nie mógł ułatwiać sobie zdobywania punktów poprzez cykliczne odtwarzanie łatwej fali i zapisywanie gry (w szczególności: uzyskanego wyniku) przed jej wyczerpaniem.

3.2.2.2 Dziennik

Niezależnie od wspomnianych wyżej funkcjonalności dokonywany jest również zapis najważniejszych danych wszystkich wrogów wyeliminowanych z gry w trakcie ostatniej rozgrywki, pogrupowanych według pokoleń, z załączoną informacją o średniej wartości w ramach każdego pokolenia. Dane te są zapisywane automatycznie do pliku tekstowego podczas kończenia pracy programu na skutek kliknięcia przycisku „Exit" w menu głównym.

System zapisu takiego „dziennika” jest mechanizmem niezależnym od opisywanych wcześniej funkcjonalności związanych z utrwalaniem stanu gry. W przeciwieństwie bowiem do nich, nie jest on częścią klasy obsługującej operacje omawianych w rozdziale 3.2.2.1, lecz jest realizowany przez obiekt, którego głównym zadaniem jest przygotowanie i wyświetlanie danych dotyczących pracy głównej implementacji algorytmu genetycznego.

```
### Gen:0
2 = -1 + -1 : 2 | 0 | 0 | 0 | 0 -> 10
1 = -1 + -1 : 2 | 2 | 1 | 2 | 2 -> 40
0 = -1 + -1 : 2 | 2 | 0 | 1 | 1 -> 40
5 = -1 + -1 : 2 | 2 | 2 | 2 | 2 -> 50
6 = -1 + -1 : 0 | 0 | 0 | 0 | 2 -> 20
3 = -1 + -1 : 0 | 0 | 0 | 0 | 0 -> 10
4 = -1 + -1 : 1 | 0 | 0 | 0 | 2 -> 20
Avg value: 27
### Gen:1
7 = -1 + -1 : 2 | 2 | 1 | 2 | 2 -> 20
8 = 1 + 0 : 2 | 2 | 1 | 2 | 2 -> 20
10 = 0 + 5 : 2 | 2 | 2 | 2 | 2 -> 30
13 = 0 + 0 : 2 | 2 | 0 | 1 | 1 -> 20
11 = 5 + 0 : 2 | 2 | 2 | 1 | 1 -> 40
9 = 0 + 1 : 2 | 2 | 0 | 1 | 2 -> 50
12 = 6 + 6 : 0 | 0 | 0 | 0 | 2 -> 20
Avg value: 28
### Gen:2
15 = 11 + 11 : 2 | 2 | 2 | 1 | 1 -> 20
17 = 11 + 11 : 2 | 2 | 2 | 2 | 1 -> 20
16 = 11 + 11 : 2 | 2 | 0 | 1 | 1 -> 40
18 = 13 + 13 : 2 | 2 | 0 | 1 | 1 -> 40
19 = 11 + 11 : 2 | 2 | 2 | 1 | 1 -> 60
20 = -1 + -1 : 1 | 2 | 1 | 1 | 2 -> 40
```

Rys. 19: Fragment pliku "dziennika"

System ten wykorzystuje także specyfikę działania programu w zakresie obsługi obiektów (wrogów i pocisków) wyeliminowanych z rozgrywki. Bez względu na to, czy uległy one kolizji, czy opuściły obszar gry, nie są one natychmiast owo usuwane z pamięci, lecz jedynie oznaczane, jako „martwe”. Dopiero później wszystkie listy przechowujące te obiekty zostają przejrzane i oczyszczone ze zbędnych elementów.

```
//kolizja wróg-pocisk
n = playerProjectileHdlr.getSize();
for(int j=0; j<n; j++)
{
    GameObject tmpBullet = playerProjectileHdlr.get(j);
    colDetected = false;
    int m = enemyHdlr.getSize();
    i = 0;
    while((i<m) && !colDetected)
    {
        GameObject tmpObj = enemyHdlr.get(i);
        if(tmpBullet.getHitbox().intersects(tmpObj.getHitbox()))
        {
            tmpBullet.kill();
            tmpObj.kill();
            hud.incScore(((BasicEnemy)tmpObj).getValue());
            colDetected = true;
        }
        i++;
    }
}
```

Rys. 20: Fragment kodu realizującego kolizje.
Metoda kill() ustawia jedynie odpowiednia flagę.

```
//czyszczenie list z "martwych" obiektów
if(enemyHdlr.getSize() >0)
{
    crMan.collectSpecimens(enemyHdlr);
    displayer.SaveTheDead();
}
enemyHdlr.clearDead();
enemyProjectileHdlr.clearDead();
playerProjectileHdlr.clearDead();
```

Rys. 21: Fragment kodu obsługującego czyszczenie list obiektów

Dzięki takiemu podejściu, niejako wiążącemu ów proces ze strukturą kodu, o wiele łatwiejsze staje się dodawanie kolejnych funkcjonalności, które opierają swoje działanie na danych pozyskanych z obiektów usuniętych z rozgrywki. Jako przykład należałoby wymienić samą implementację algorytmu genetycznego oraz omawiany tu system zapisu.

```
/**
 * Czyści listę obiektów, ale tylko z elementów oznaczonych jako "martwe"
 */
public void clearDead()
{
    int i = objects.size()-1 ;

    while(i >= 0)
    {
        if(objects.get(i).isAlive() == false) objects.remove(i);
        i--;
    }
}
```

Rys. 22: Fragment kodu - metoda czyszcząca listę

4. WNIOSKI

- Implementacja algorytmów genetycznych w grze komputerowej jest bez wątpienia możliwa.
- Pobieżne obserwacje działania aplikacji zdają się wskazywać na brak negatywnego wpływu wspomnianej implementacji na rozgrywkę, przynajmniej z punktu widzenia gracza.
- Negatywny wpływ elementów gry – zwłaszcza interaktywności – na pracę algorytmu jest niestety wyraźnie widoczny, na co wskazuje między innymi nieregularność zmian średniej wartości populacji.
- Jednoznaczne stwierdzenie zasadności użycia algorytmów genetycznych jako jednego z mechanizmów w grach komputerowych będzie wymagało przeprowadzenia znacznie rozleglejszych badań niż te, jakie mogły mieć miejsce w ramach niniejszej pracy.

5. STRESZCZENIE

Celem niniejszej pracy jest implementacja algorytmu genetycznego wraz z funkcjonalnościami mającymi umożliwić śledzenie jego działania w trakcie wykonywania programu oraz gromadzić dane pozwalające na późniejszą analizę tegoż działania. Implementacja ta została dokonana w ramach aplikacji napisanej w języku Java w formie gry komputerowej. Istnieje szereg przesłanek przemawiających za i przeciw stosowaniu algorytmów genetycznych w grach komputerowych. Zakres niniejszej pracy nie pozwolił uzyskać jednoznacznej odpowiedzi co do zasadności takiego połączenia, jednak wspomniana aplikacja powinna stanowić dobre narzędzie dla dalszych badań w tym temacie.

The purpose of this paper is to implement a genetic algorithm along with functionalities allowing to track its operations during program execution and to collect data for the subsequent analysis of this operation. This implementation was made within an application written in Java in the form of a computer game. There are a number of reasons for and against the use of genetic algorithms in computer games. The scope of this work did not allow for a clear answer as to the legitimacy of such a combination, however, the mentioned application should be a good tool for further research on this topic.

6. BIBLIOGRAFIA

1. Ivory J.: A Brief History of Video Games [w:] The Video Game Debate: Unravelling the Physical, Social, and Psychological Effects of Digital Games / edited by Rachel Kowert and Thorsten Quandt. New York: Routledge, 2016, s.10-30.
2. Christodoulou M., Szczygieł E., Kłapa Ł., Kolarz W.: Algorytmika i Programowanie. Materiały szkoleniowe dla nauczycieli. Krosno: P.T.E.A. Wszechnica Sp. z o.o., 2018. ISBN 978-83-951529-1-7.
3. Cormen T. [et al.]: Introduction to Algorithms. Third Edition. Cambridge: The MIT Press, 2009. ISBN 978-0-262-03384-8.
4. Goldberg D.: Algorytmy genetyczne i ich zastosowania. Warszawa: Wydawnictwo WNT, 1995. Wydanie I. ISBN: 83-204-1852-6.
5. Whitley D.: A Genetic Algorithm Tutorial, [wyd:] Computer Science Department, Colorado State University, 1993.
6. Gwiazda T.: Algorytmy Genetyczne: Kompendium. Warszawa: Wydawnictwo Naukowe PWN, 2007. Wydanie I. ISBN 978-83-01-15168-3.
7. Deb K.: An Introduction to Genetic Algorithms *Sadhana Journal*, vol. 24 (4-5), [wyd:] Kanpur Genetic Algorithms Laboratory, 1999, str. 293-315
8. Holland J.H.: Adaptation in natural and artificial systems [wyd:] MIT Press, 1992. Wydanie I. ISBN 0-262-58111-6.
9. Beasley D., Bull D.R., Martin R.R.: An Overview of Genetic Algorithms: Part 1, Fundamentals *University Computing*, vol. 15(2), 1993, str. 58-69
10. Whitley D.: An overview of evolutionary algorithms: practical issues and common pitfalls *Information and Software Technology*, vol. 43(14), 2001, str. 817-831

7. WYKAZ ILUSTRACJI

8. WYKAZ FOTOGRAFII