Dokumentacja projektu PSZT

Autorzy: Łopusiński Paweł, Kaczor Bartosz, Domagała Bartosz

# Decyzje projektowe

## Treść zadania

Należy stworzyć prostą grę typu "orbit" (http://www.2dplay.com/orbit/orbit.swf) ,   
a następnie zastosować algorytm ewolucyjny znajdujące wartości wejściowe, które maksymalizują długość "życia" obiektu.

## Środowisko programistyczne

Projekt został napisany w języku C++ z użyciem bibliotek Qt w wersji 5.0.

## Algorytm rozwiązania zadania

Zdecydowaliśmy się skorzystać z algorytmu ewolucyjnego µ + λ. Daje on potencjalnie szansę na dość szybkie znalezienie rozwiązania, dzięki licznym krzyżowaniom i dodatkowym mutacjom. Ponadto najsilniejsi osobnicy danej populacji, przechodzą do kolejnych iteracji algorytmu, mogąc dalej się rozmnażać, zwiększając szansę na otrzymanie silniejszych potomków.

Algorytm (µ, λ) odrzuciliśmy, ponieważ nie bierze on pod uwagę populacji rodziców w aktualnej iteracji, co może skutkować odrzuceniem silniejszych osobników, a zależało nam aby jak najdłużej „utrzymać przy życiu” najsilniejszych osobników, celem zwiększenia szansy na znalezienie osobnika, który jak najbardziej zbliży się do zadanej wartości długości „życia”.

Algorytm 1+1 jak i jego zrównoleglona wersja, wydały nam się zbyt mało dynamicznymi – istniało duże ryzyko skupienia się generowanych potomków wokół pewnego maksimum lokalnego (lub maksimów lokalnych) i pozostanie w nim poprzez kilkanaście kolejnych iteracji.

Ostatecznie doszliśmy do wniosku, iż zastosowany algorytm µ + λ pozwala osiągnąć odpowiedni stosunek między eksploracją a eksploatacja przestrzeni wyników.

# Instrukcja dla użytkownika

## Kompilacja

Do kompilacji wymagane są biblioteki Qt w wersji 5.0.

## Obsługa

Do obsługi programu wystarczy mysz komputerowa. Po otwarciu okna programu możemy wygenerować poziom ustawiając odpowiednie parametry: docelowy czas „życia” komety, ilość planet, minimalną i maksymalną wagę planet, a następnie klikając przycisk „Generuj poziom”. Wagę każdej wygenerowanej planety możemy zobaczyć najeżdżając na nią kursorem. Następnie możemy wybrać tryb gry „Graj samemu” lub symulować rozwiązania klikając przycisk „Symuluj” (mamy tu dwie możliwości do wyboru, które różnią się metodą powstawania zarodków). W drugim przypadku, na wstępie zostanie wygenerowana i zasymulowana pierwsza, losowa generacja. Następnie ustawiając ilość generacji na kliknięcie i wybierając opcję „Następna populacja” możemy obserwować działanie algorytmu. Na ekranie zobaczymy punkty początkowe osobników oraz drogę jaką przebyli. Po każdym kroku możemy zobaczyć szczegółowe dane na temat każdego osobnika klikając „Wyświetl zestawienie”. Jeśli wybierzemy opcję „Odtwórz” pod jednym z osobników wyświetlonych w zestawieniu, program odtworzy jego trasę lotu w czasie rzeczywistym.

# Opis struktury programu

## 1. Struktura Vector (vector.h, vector.cpp)

Jest to definicja dwuwymiarowego wektora i związanych z nim operacji, które wykorzystywane są w obliczeniach w trakcie symulacji.

## 2. Klasy Planeta (planeta.h, planeta.cpp) oraz Kometa (kometa.h, kometa.cpp)

Klasa Planeta jest reprezentacją planety zarówno na ekranie jak i w algorytmie symulacji. Ponadto jest klasą bazową dla klasy Kometa, określającej obiekt reprezentujący „próbkę” badaną w programie.

## 3. Klasy Populacja (populacja.h, populacja.cpp) oraz Populacja2 (populacja2.h, populacja2.cpp)

Klasy te określają populację oraz cały jej proces rozmnażania i mutacji aż do powstania nowych zarodków. Za pomocą publicznej metody *tworzNowaPopulacje();* tworzymy nową populację badanych obiektów, przechodząc przez cały proces algorytmu. Klasa **Populacja2** jest pochodną klasy **Populacja** – został przeciążony system tworzenia nowych zarodków. W klasie bazowej są one uśredniane, natomiast w pochodnej interpolowane.

## 4. Klasa Symulation (symulation.h, symulation.cpp)

Klasa ta jest odpowiedzialna za symulowanie fizyki w programie. Do poprawnego działania, przed uruchomieniem obliczeń należy podać listę planet (przeszkód), osobnika którego zachowanie chcemy zasymulować oraz maksymalną wartość czasu „życia”.

## 5. Klasa Zestawienie (zestawienie.h, zestawienie.cpp, zestawienie.ui)

Klasa odpowiedzialna za wyświetlenie okna dialogowego, zawierającego zestawienie parametrów początkowych oraz czasów „życia” populacji aktualnie wyświetlanej na ekranie. Z poziomu tego okna można odtworzyć lot wybranego osobnika.

## 6. Klasa Replay (replay.h, replay.cpp)

Klasa odpowiedzialna za poprawne odtworzenie lotu osobnika wybranego z okna dialogowego zestawienia.

## 7. Klasa KometaScene (kometascene.h, kometascene.cpp)

Jest to przeciążona wersja klasy QGraphicsView z pakietu Qt, w której zaimplementowaliśmy własne metody obsługi myszy, aby umożliwić samodzielną rozgrywkę.

## 8. Struktury Wiadomosc i ProstaWiadomosc (wiadomosc.h)

Stworzenie tych struktur było wymuszone poprzez mechanizmy pakietu Qt – ich wykorzystanie znacznie ułatwiło przekazywanie danych z wątków roboczych do wątku GUI.

## 9. Klasa MainWindow (mainwindow.h, mainwindow.cpp, mainwindow.ui)

Klasa tworząca główne okno programu, przygotowująca program do pracy. W tej klasie zawiera się obsługa wszystkich kontrolek widocznych w oknie.

# Wnioski dotyczące rezultatów

Działanie algorytmu jest zgodne z oczekiwaniami. Najsilniejsi osobnicy oraz najsilniejsi potomkowie zostają rodzicami w kolejnych populacjach. Zdarza się też, że niektórzy osobnicy są w stanie przetrwać kilkadziesiąt populacji, jeśli ich czas życia jest dostatecznie długi, tym samym nadal tworząc silnych potomków. Prowadzi to do otrzymywania zadowalających wyników, dążących do zadanego czasu „życia”. Dużo zależy od początkowej populacji, która generowana w całości pseudolosowo może znaleźć się blisko jak i daleko od planet. W przypadku rozwijania programu warto byłoby się zastanowić nad algorytmem, który wybierał by optymalne współrzędne oraz wektory prędkości dla początkowych osobników. Można by także pomyśleć o wyborze początkowej populacji przez użytkownika. Człowiek bowiem wybierałby najprawdopodobniej punkty przewidując początkowy ruch komety, co daje przewagę nad wyborem pseudolosowym. Powyższe zabiegi mogłyby jeszcze bardziej usprawnić działanie algorytmu i umożliwić szybsze znajdowanie osobników o dłuższych czasach życia.

Wiele symulacji pozwoliło nam ustalić, że wyniki są porównywalne bez względu na to czy zastosujemy tworzenie zarodków poprzez interpolację czy poprzez uśrednianie. Przy wielu populacjach różnicy praktycznie nie ma, obie metody nadają się do rozwiązania tego problemu.

Użytkownik w grze kieruje się przede wszystkim intuicją i stara się przewidzieć początkowy ruch komety ze względu na odległość planet oraz ich przyciąganie. Daje to dużą przewagę nad programem, który wybiera stricte losowe miejsca i prędkości, nie wykluczając również pojawienia się komety na planecie, co skutkuje zerowym czasem „życia”.