Uniwersytet Warmińsko-Mazurski w Olsztynie
Wydział Matematyki i Informatyki

Kierunek: Informatyka

Krzysztof Pietraszko

Symulacja ruchu drogowego na skrzyżowaniu z sygnalizacją świetlną sterowaną algorytmem ewolucyjnym

Praca inżynierska wykonana
w Katedrze Metod Matematycznych Informatyki
pod kierunkiem
Dr Inż. Bartosza Nowaka

2019, Olsztyn

University of Warmia and Mazury in Olsztyn
Faculty of Mathematics and Computer Science

Computer Science

Krzysztof Pietraszko

Traffic simulation on junction with traffic lights controlled by an evolutionary algorithm

Engineer's Thesis is performed
in the Department of Mathematical
Methods in Computer Science
under supervision of
Dr Inż. Bartosz Nowak

2019, Olsztyn

Streszczenie

Celem pracy była implementacja algorytmu ewolucyjnego optymalizującego sygnalizację świetlną na skrzyżowaniu.
Algorytmy ewolucyjne to jedna z kategorii technik sztucznej inteligencji. Znajdują szerokie zastosowanie w rozwiązywaniu problemów optymalizacyjnych. Ich podstawą są mechanizmy czerpiące inspirację z ewolucji gatunków.
Usprawnienie sygnalizacji świetlnej przedstawiono jako problem optymalizacyjny z funkcją celu zależną od symulacji ruchu drogowego. Do implementacji aplikacji wykorzystano silnik Unity i język C#. Przeprowadza ona proces optymalizacji z ustawieniami określonymi przez użytkownika. W pracy opisano przebieg algorytmu i pracy samej aplikacji, a wyniki uczenia zaprezentowano w formie wykresów. Zaproponowano również możliwe kierunki rozwoju aplikacji.

Abstract

The goal of the thesis was to implement an evolutionary algorithm that optimises traffic lights on a junction.
Evolutionary algorithms are a category of artificial intelligence techniques. They are widely used to solve optimisation problems. Their main mechanisms are inspired by the evolution of species.
Traffic lights improvement was presented as an optimisation problem with a goal function dependant on a simulation of the traffic. Unity engine and C# language were used to implement the application. It performs the optimisation process with settings provided by the user. This thesis describes the algorithm and the functionality of the application itself. The results of optimisation were presented in the form of charts. Possible directions for further extension of the application were also proposed.

Spis treści

Wstęp

Komputery są obecnie niezwykle popularnymi urządzeniami znajdującymi zastosowanie w wielu dziedzinach życia. Są pomocne także w zadaniach inżynieryjnych takich jak problemy optymalizacyjne.

Problemy optymalizacyjne

Problemy w wielu obszarach matematyki, inżynierii, ekonomii, medycyny i statystyki mogą być przedstawione w kategoriach optymalizacji.
Określenie problemu optymalizacyjnego rozpoczyna się od ustalenia zbioru zmiennych niezależnych lub parametrów. Często formułuje się również ograniczenia, które wyznaczają dozwolone wartości zmiennych. Inną istotną składową problemu optymalizacyjnego jest funkcja celu, której wartość zależy od zmiennych. Rozwiązaniem takiego problemu jest zbiór dozwolonych wartości zmiennych, dla których funkcja przyjmuje optymalną wartość (minimalną lub maksymalną, zależnie od badanego zagadnienia)[1].

Do rozwiązywania problemów optymalizacyjnych często stosowane są metody sztucznej inteligencji. Dostarczają lepszych, szybszych i bardziej precyzyjnych rozwiązań niż konwencjonalne techniki, szczególnie w złożonych problemach inżynieryjnych. Spośród ich charakterystycznych cech warto wymienić następujące:
— Metody te „pamiętają" swoje poprzednie odkrycia,
— Dostosowują swoją wydajność (np. decydują o eksploracji lub eksploatacji), co przekłada się na unikanie utknięcia w minimum lokalnym,
— Mogą planować swoje działania długofalowo [2].

Algorytmy ewolucyjne

Przykładowym typem metod sztucznej inteligencji są algorytmy ewolucyjne. Poszukują one optymalnego rozwiązania problemu w sposób, który czerpie inspirację z mechanizmu ewolucji gatunków. Jedną z głównych cech wyróżniających ten zbiór technik jest to, że jednocześnie zmieniana jest pewna populacja rozwiązań, a jej przekształcenia przypominają procesy zachodzące w przyrodzie. Zrozumienie algorytmów ewolucyjnych wymaga przyswojenia pewnych kluczowych pojęć, takich jak dostosowanie, stanowiące ocenę danego rozwiązania, oraz krzyżowanie i mutacja, będące mechanizmami przekształcania populacji.

Krzyżowanie tworzy nowe rozwiązanie łącząc cechy dwóch innych, podczas gdy mutacja w losowy sposób modyfikuje istniejące rozwiązanie. Przebieg algorytmu ewolucyjnego rozpoczyna się od utworzenia początkowej populacji rozwiązań, które zazwyczaj są generowane losowo. W związku z tym zakłada się, że jest to różnorodny zbiór, zawierający dobre i złe cechy. Algorytm następnie generuje szereg kolejnych populacji (pokoleń), z pomocą wyżej wspomnianych mechanizmów przekształcających [3]. Kluczowe jest tu zachowanie równowagi między wykorzystywaniem znanych pozytywnych cech (a więc krzyżowaniem) a odkrywaniem nowych możliwości w przestrzeni dopuszczalnych rozwiązań (a więc mutacją). Istnieją modele mające na celu zmniejszenie tego problemu. Przykładowo, działanie algorytmu można podzielić na fazy: pozyskiwania, akumulacji i wykorzystywania wiedzy [4]. Innym rozwiązaniem jest przyjęcie zmiennego współczynnika mutacji danego pewną funkcją. Taką technikę przyjęto w tej pracy.

Przykładowe zastosowania algorytmów ewolucyjnych

Do ciekawych zastosowań algorytmów ewolucyjnych należą m.in. antena statku kosmicznego [5], czy też wentylator silnika [6], których kształt został zaprojektowany przez taki algorytm. Obszary przejawiające duży potencjał to chociażby planowanie produkcji [7], przewidywanie zmian temperatury na Ziemi[8] i wiele innych [9].

W tej pracy omówiono wykorzystanie algorytmu ewolucyjnego do usprawnienia płynności ruchu drogowego na skrzyżowaniu poprzez optymalizację sygnalizacji świetlnej. Stworzono aplikację przeprowadzającą proces uczenia z możliwością konfiguracji jego ustawień. Efektem końcowym pracy aplikacji jest wyświetlenie optymalnych parametrów sygnalizacji.

Usprawnienie sygnalizacji świetlnej jako problem optymalizacyjny

Każda zmienna niezależna to czas, przez jaki wybrane sygnalizatory są zielone. Ewaluacja funkcji celu każdorazowo wymaga przeprowadzenia symulacji ruchu pojazdów, którą w tym celu zaimplementowano. Symulacja trwa, dopóki określona liczba samochodów nie opuści skrzyżowania, a czas jej trwania decyduje o ocenie danego osobnika.

Zawartość pracy

W rozdziale *Użyte technologie* opisano wykorzystane narzędzia i język programowania, a także czynniki decydujące o wyborze silnika.
W rozdziale *Projekt aplikacji* opisano wymagania funkcjonalne i pozafunkcjonalne projektu oraz główny przypadek użycia.
Rozdział *Opis teoretyczny algorytmu ewolucyjnego* szczegółowo przedstawia algorytm

ewolucyjny, który zaimplementowano w aplikacji.
W rozdziale *Opis aplikacji* zaprezentowano aplikację z punktu widzenia użytkownika i wyjaśniono jej funkcjonowanie. Opisano też ustawienia, które może zmieniać użytkownik, a także dodatkowe narzędzia powstałe w trakcie implementacji.
W rozdziale *Wyniki uczenia sygnalizacji* opisano rezultaty kilku przykładowych przebiegów procesu uczenia.
W podsumowaniu opisano, czego udało się dokonać w ramach pracy. Przedstawiono też największe wyzwania i możliwe kierunki rozwoju aplikacji.

Użyte technologie

Głównymi czynnikami decydującymi o doborze silnika były:
— Prostota interakcji z kartą graficzną,
— Dobre narzędzia do debugowania kodu,
— Jakość i kompletność dokumentacji,
— Wydajność,
— Możliwość prostej implementacji interfejsu graficznego,
— Wcześniejsze doświadczenie,
— Szybkość kompilacji – istotna przy częstych zmianach w kodzie.

Unity

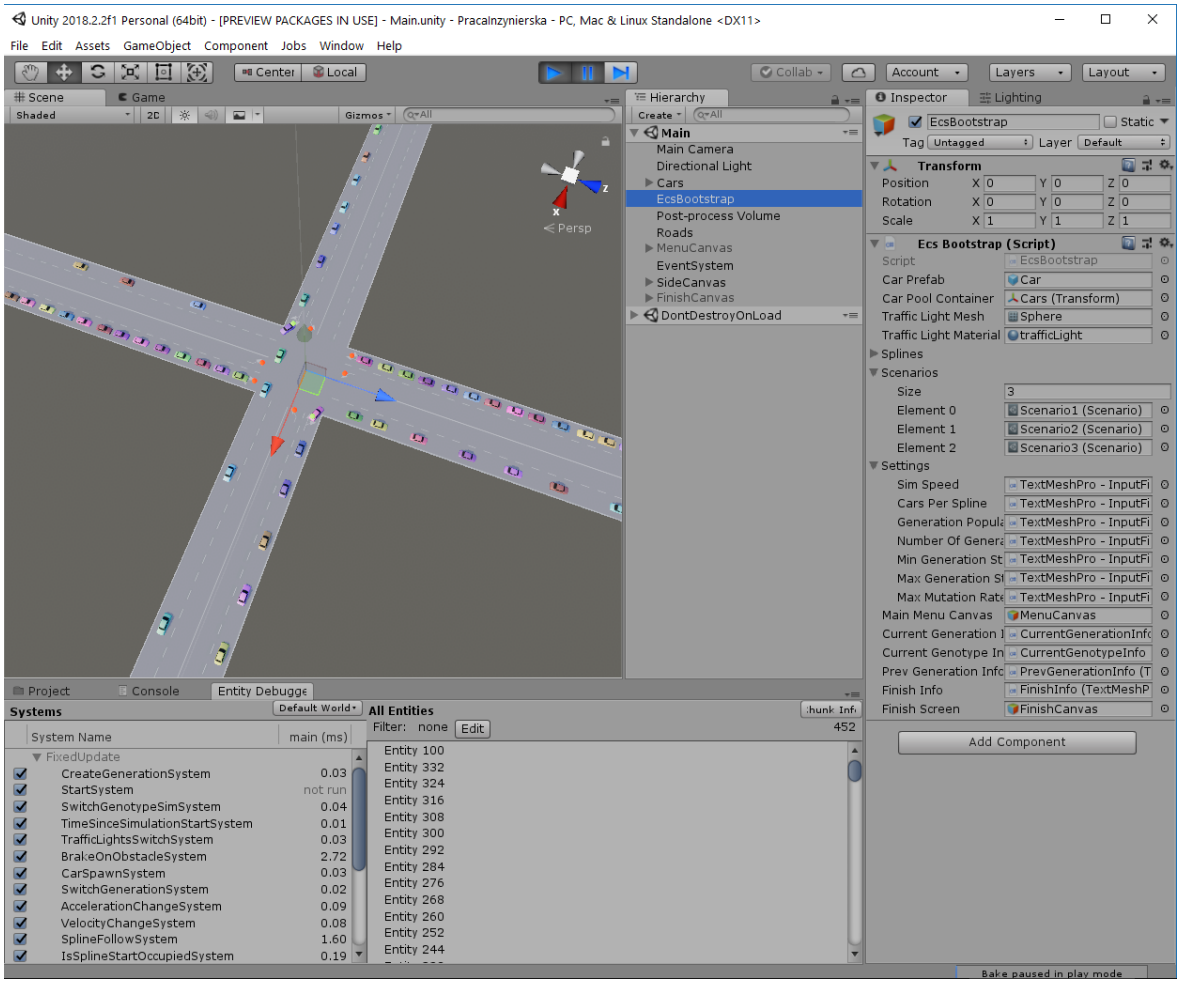
Unity to wieloplatformowy silnik autorstwa firmy Unity Technologies, którego głównym zastosowaniem jest tworzenie gier dwuwymiarowych i trójwymiarowych oraz symulacji. W porównaniu do podobnych narzędzi wyróżnia się prostotą obsługi, dużym naciskiem na edytory wizualne i zintegrowanym środowiskiem, które nie wymaga skomplikowanej konfiguracji. Wspiera wiele API graficznych (DirectX, OpenGL, Metal i Vulkan)[10] pod wspólnym interfejsem. Dzięki temu użytkownicy silnika mogą w łatwy sposób tworzyć wersje swoich aplikacji na wiele platform, bez konieczności uciążliwego dostosowywania swojego kodu. Podobne uproszczenia dostępne są również w zakresie obsługi urządzeń wejścia (jak mysz, klawiatura, kontrolery, ekrany dotykowe), a także dźwięku. Na wspomnienie zasługuje również wbudowany edytor WYSIWYG[1] interfejsu graficznego aplikacji. Silnik można też rozszerzać o własne narzędzia, na przykład graficzny edytor krzywych. Programować swoją aplikację można jedynie w języku C# [11]. Interfejs edytora Unity przedstawiono na rysunku 1.

Język C#

C# to silnie i statycznie typowany, obiektowy język programowania zaprojektowany przez firmę Microsoft. Pozwala pisać kod z zachowaniem wszystkich założeń obiektowego paradygmatu programowania, czyli: abstrakcji, hermetyzacji, polimorfizmu i dziedziczenia. Mimo że C# to język obiektowy, posiada wiele funkcji inspirowanych językami funkcyjnymi, które pozwalają m.in. bardzo łatwo dokonywać filtrowania, przekształcania czy sortowania kolekcji obiektów. Ułatwia to pisanie czytelnego, a przy

[1] WYSIWYG (What You See Is What You Get) - „otrzymujesz to co widzisz"

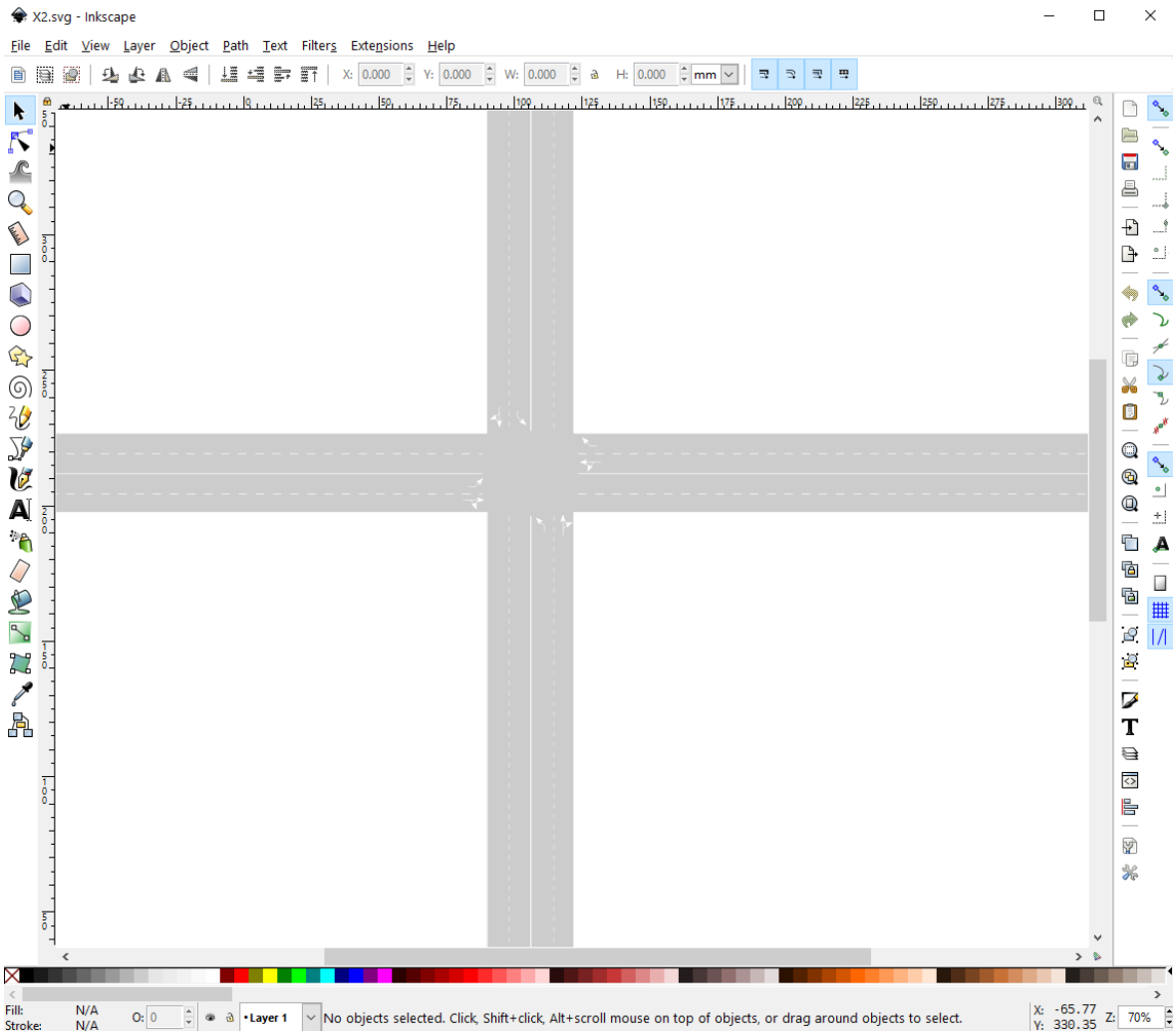
tym zwięzłego, kodu. Pierwotnie kod C# mógł być uruchamiany jedynie na systemie Windows. Zmieniło się to jednak w ostatnich latach dzięki środowiskom takim jak Mono oraz .NET Core i obecnie wspierane są inne systemy, takie jak Linux, macOS, iOS czy Android [12].



Rysunek 1. Interfejs edytora Unity

Inkscape

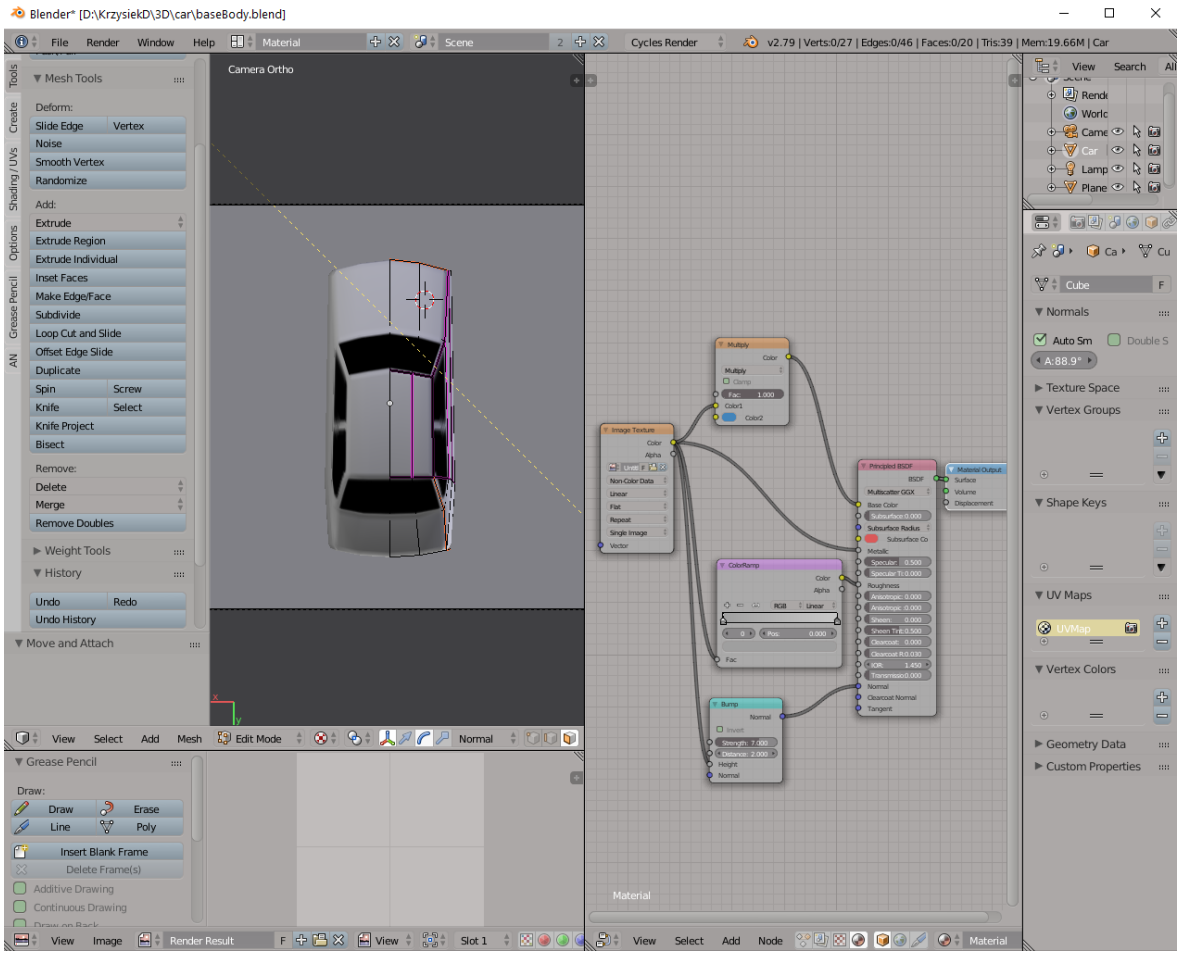
Inkscape to darmowy program do tworzenia grafiki wektorowej. Tworząc w nim obrazy korzysta się z podstawowych kształtów takich jak elipsy, wielokąty, krzywe czy strzałki. Inkscape używa formatu Scalable Vector Graphics (SVG). Mimo że program opiera się na formacie wektorowym, plikiem wynikowym rysunku w tej pracy był obraz rastrowy, który można eksportować w dowolnej rozdzielczości. Decyzja ta wynika z ograniczeń w importowaniu grafik wektorowych w silniku Unity. Za użyciem Inkscape przemawiała przede wszystkim mnogość narzędzi i ustawień do tworzenia kształtów i wyrównywania ich położenia. Wygląd interfejsu programu przedstawiono na rysunku 2. W tej pracy został użyty do stworzenia grafiki skrzyżowania.



Rysunek 2. Interfejs programu Inkscape

Blender

Blender to darmowy program do tworzenia grafiki trójwymiarowej stworzony przez organizację Blender Foundation. Posiada narzędzia do modelowania, teksturowania, renderowania i animacji. Oprócz tego można z jego użyciem wykonywać proste symulacje fizyczne, a nawet zrealizować postprodukcję filmów. Niewątpliwie zaletą programu jest połączenie ogromnej liczby narzędzi w jeden pakiet. Dodatkowe atuty to dobra integracja z silnikiem Unity, wygoda użytkowania i wydajność. W tej pracy posłużył do wykonania trójwymiarowego modelu samochodu oraz ikony aplikacji. Interfejs Blendera przedstawia rysunek 3.



Rysunek 3. Interfejs programu Blender

Projekt aplikacji

Aplikacja jest wysoce wyspecjalizowana – przeprowadza wyłącznie proces symulacji i uczenia. Użytkownik może konfigurować ustawienia tego procesu.

Wymagania funkcjonalne

— Przeprowadzanie procesu uczenia optymalnych czasów świecenia sygnalizatorów na skrzyżowaniu za pośrednictwem algorytmu ewolucyjnego, wykorzystującego symulację ruchu pojazdów do oceny rozwiązań,
— Konfiguracja ustawień symulacji, będącej bazą procesu uczenia,
— Konfiguracja ustawień algorytmu ewolucyjnego,
— Wyświetlanie na bieżąco informacji o przebiegu uczenia,
— Wyświetlenie wyników uczenia po jego zakończeniu.

Wymagania pozafunkcjonalne

— Stabilność – aplikacja musi pracować bezawaryjnie przez wiele godzin,
— Wydajność – aplikacja musi działać na tyle wydajnie, aby symulacja mogła być przeprowadzana kilkadziesiąt razy szybciej niż czas rzeczywisty.

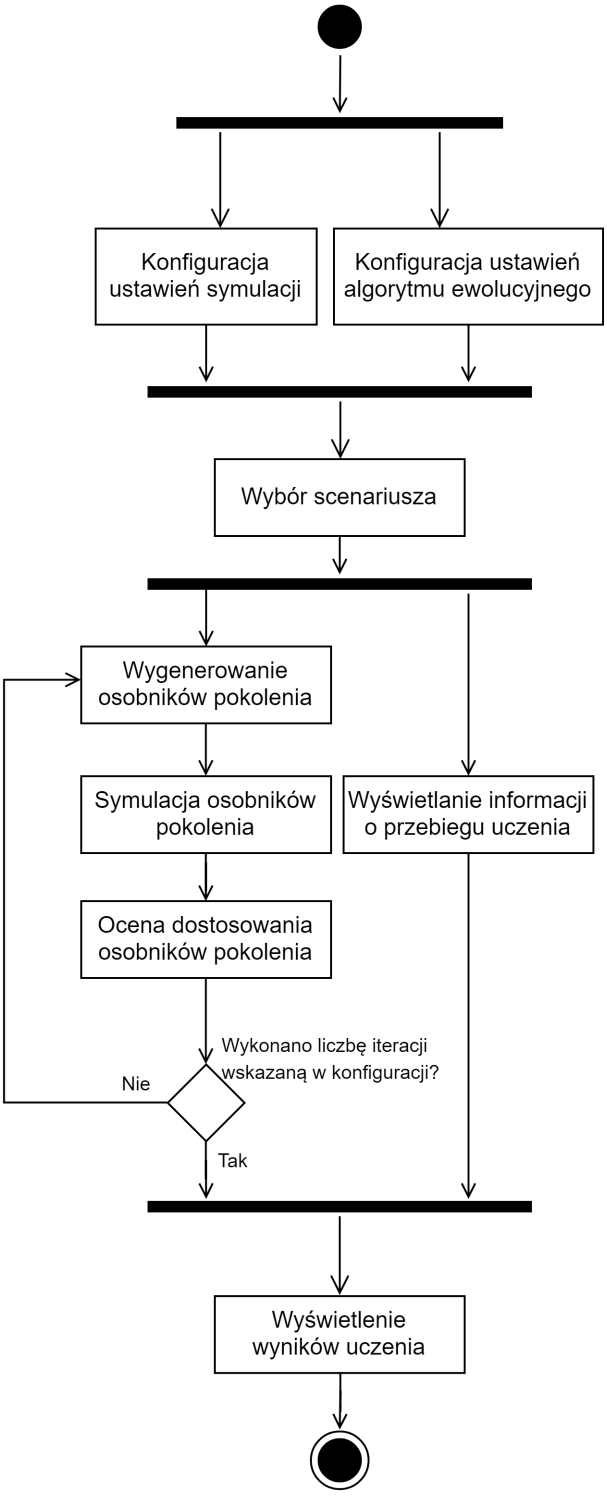
Przypadki użycia

W projekcie występuje tylko jeden przypadek użycia: „Przeprowadź proces optymalizacji sygnalizacji świetlnej". Przedstawia go diagram aktywności na rysunku 4.

Przebieg przypadku użycia „Przeprowadź proces optymalizacji sygnalizacji świetlnej"

Użytkownik najpierw może ustawić konfigurację symulacji oraz algorytmu ewolucyjnego. Następne wybiera scenariusz (czyli kolejność zapalania sygnalizatorów), co powoduje rozpoczęcie procesu uczenia. Aplikacja kolejno generuje, symuluje, a potem ocenia osobników pokolenia, jednocześnie wyświetlając na ekranie informacje o przebiegu uczenia. Pokolenia są kolejno generowane tak długo, aż zostanie osiągnięta ich

liczba określona w konfiguracji. Następnie aplikacja wyświetla ekran końcowy prezentujący wyniki uczenia.



Rysunek 4. Diagram aktywności przypadku użycia „Przeprowadź proces uczenia sygnalizacji”

Opis teoretyczny algorytmu ewolucyjnego

Proces uczenia sygnalizacji świetlnej w tej pracy został zrealizowany z pomocą algorytmu ewolucyjnego. Ten rozdział opisuje kolejne kroki, z których składa się proces uczenia. Szczegóły dotyczące zastosowania poniższego algorytmu zostały opisane w następnym rozdziale.

Pierwsze pokolenie

Przebieg uczenia rozpoczyna się od wygenerowania pierwszego pokolenia osobników. Osobnik to właściwie zbiór wartości określonych parametrów. W pierwszym pokoleniu wartości te generowane są losowo z zakresu określonego w konfiguracji algorytmu.

Funkcja dostosowania

Osobniki wygenerowanego pokolenia poddawane są ocenie, która określa ich dostosowanie (ang. *fitness*). Funkcja dostosowania określona jest wzorem:

$$f(x) = max(t(1), \ldots, t(n)) - t(x) \tag{1}$$

gdzie:

- x – oceniany osobnik
- n – liczba osobników w pokoleniu
- $t(a)$ – czas, w jakim określona liczba samochodów przejechała przez skrzyżowanie, przy sygnalizacji ustawionej wg parametrów osobnika a

Generowanie nowego pokolenia

Najlepszy osobnik poprzedniego pokolenia jest kopiowany wprost do nowego. Wybór reszty osobników odbywa się drogą selekcji turniejowej. Z poprzedniego pokolenia losowane są 3 osobniki. Spośród tych trzech wybierany jest ten o najwyższej wartości funkcji dostosowania. Taki turniej powtarzany jest $2n$ razy, gdzie n to liczba osobników w pokoleniu (określona w konfiguracji algorytmu). Następnie ze zbioru $2n$ osobników wybieranych jest n o najlepszej wartości funkcji dostosowania. Ta pula poddana jest następnie krzyżowaniu lub mutacji.

Mutacja i krzyżowanie

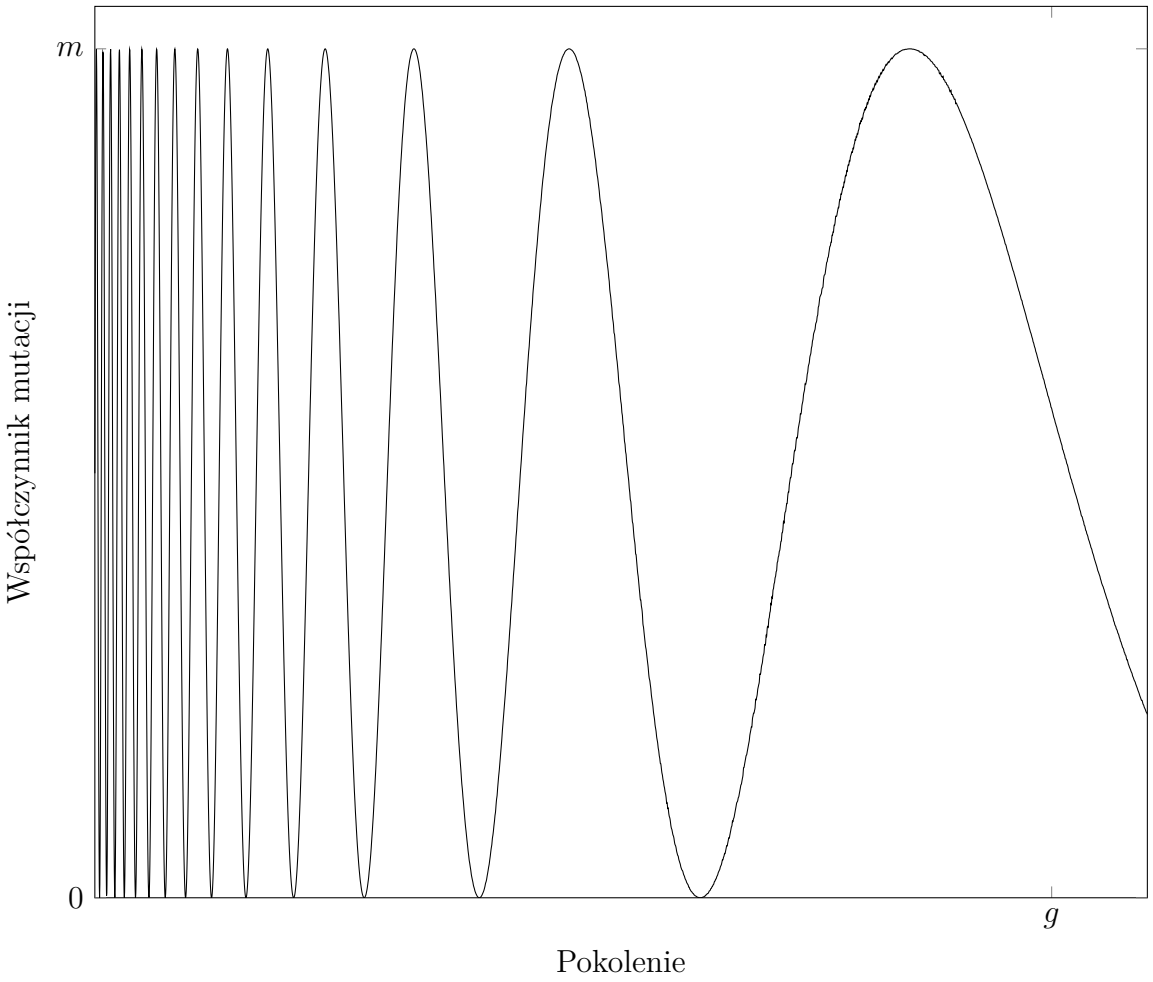
Każdy osobnik z wygenerowanej puli, z wyjątkiem najlepszego zachowanego z poprzedniego pokolenia, zostaje zmutowany albo skrzyżowany z innym osobnikiem. Decyduje o tym ważone losowanie. Prawdopodobieństwo, że osobnik zostanie poddany mutacji jest zmienne i wyznacza je funkcja:

$$f(x) = (sin(x * \frac{1}{\frac{g}{1000} + \frac{x}{100}}) + 1) : 2 \cdot m \tag{2}$$

gdzie:

- x – numer obecnego pokolenia
- g – liczba wszystkich pokoleń, określona w konfiguracji algorytmu
- m – maksymalny współczynnik mutacji, określony w konfiguracji algorytmu

Wykres tej funkcji przedstawiono na rysunku 5.
Korzyści wynikające ze stosowania dynamicznego współczynnika mutacji opisał Thierens [13]. Również przy powstawaniu tej pracy zmienne prawdopodobieństwo mutacji okazało się lepszym rozwiązaniem w porównaniu do stałego współczynnika.



Rysunek 5. Funkcja określająca współczynnik mutacji

Mutacja polega na losowym dodaniu lub odjęciu od wartości każdego parametru osobnika losowej liczby z zakresu 1–20.
Jeśli w wyniku wyżej opisanego losowania zostanie podjęta decyzja, że osobnik ma zostać poddany krzyżowaniu, to z puli losowo wybierany jest inny osobnik, od którego ten pierwszy przejmie wartość co drugiego parametru. Pozostałe parametry pozostają bez zmian.

Po zastosowaniu krzyżowania lub mutacji nowe pokolenie jest gotowe do oceny funkcją dostosowania opisaną wcześniej, co zamyka cykl. Kolejne pokolenia są generowane na postawie poprzednich i oceniane tak długo, aż osiągnięta zostanie liczba pokoleń określona w konfiguracji algorytmu.

Opis aplikacji

Po uruchomieniu aplikacji, użytkownikowi najpierw ukazuje się ekran ładowania (przedstawiony na rysunku 6), a następnie menu główne (widoczne na rysunku 7).

Rysunek 6. Ekran ładowania aplikacji

Menu główne

W górnej części menu znajdują się pola liczbowe, do których użytkownik może wpisać ustawienia symulacji oraz algorytmu ewolucyjnego. Może też skorzystać z ustawień domyślnych.
Dostępne ustawienia symulacji to:

— Prędkość symulacji – jest to mnożnik prędkości symulowania ruchu drogowego. Przy ustawieniu 1x przeliczane jest 30 kroków symulacji na sekundę. Jedynym górnym ograniczeniem tego ustawienia jest moc obliczeniowa procesora. Domyślna wartość to 100x.

— Liczba samochodów na pasie – jest to liczba samochodów, które wjadą na każdy pas ruchu (pasy są widoczne w lewym dolnym rogu rysunku 7). Domyślna wartość to 20.

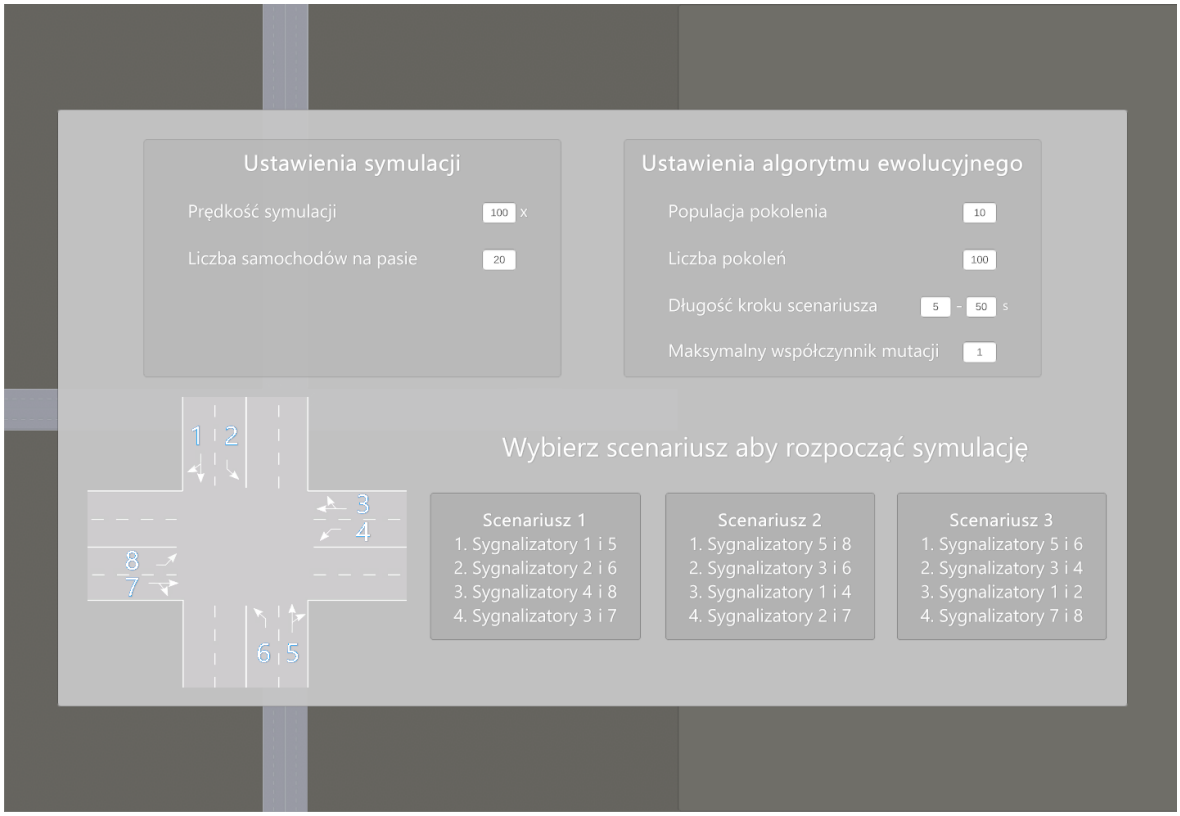
Ustawienia algorytmu ewolucyjnego przedstawiają się następująco:

— Populacja pokolenia – liczba osobników w każdym pokoleniu (stała n w funkcji 1 w poprzednim rozdziale).

— Liczba pokoleń – liczba pokoleń, po których proces uczenia ma zostać przerwany. Oprócz tego wpływa na współczynnik mutacji (stała g w funkcji 2 w poprzednim rozdziale).

— Długość kroku scenariusza – zakres, w jakim muszą mieścić się wartości parametrów osobników, a więc długości kroków scenariusza.

— Maksymalny współczynnik mutacji – stała m w funkcji 2 w poprzednim rozdziale.



Rysunek 7. Menu główne aplikacji

Po wybraniu ustawień użytkownik musi wybrać scenariusz. Jest to sekwencja sygnalizatorów, które będą zielone w określonym kroku scenariusza. W lewym dolnym rogu menu (na rysunku 7) widoczny jest schemat z ponumerowanymi sygnalizatorami, mający na celu ułatwienie użytkownikowi zrozumienia różnicy między scenariuszami.

Scenariusz

Na przykładzie Scenariusza 1 widocznego w menu: podczas pierwszego kroku scenariusza zielone będą sygnalizatory nr 1 i 5 (oznaczone na schemacie po lewej), pozostałe zaś będą czerwone. Po przejściu do drugiego kroku zielone będą sygnalizatory nr 2 i 6, podczas gdy pozostałe będą czerwone. Analogicznie w kolejnych krokach scenariusza. Długość każdego z tych kroków opisuje obecnie symulowany osobnik.

Przebieg uczenia i symulacji ruchu

Po wybraniu scenariusza rozpoczyna się proces uczenia. Aplikację podczas tego procesu przedstawia rysunek 8. Lewa część ekranu prezentuje postęp symulacji ruchu drogowego z zastosowaniem parametrów obecnie symulowanego osobnika. Z początkiem symulacji osobnika samochody wjeżdżają na skrzyżowanie i poruszają się zgodnie z sygnalizacją. Sygnalizatory przełączają się w odpowiednim czasie, a kiedy skończy się ostatni krok scenariusza, cykl sygnalizacji się zapętla – wraca do pierwszego kroku. Symulacja trwa tak długo, aż wszystkie samochody opuszczą skrzyżowanie. Wtedy czas trwania symulacji jest zapisywany, zostanie później użyty do oceny dostosowania. Aplikacja przechodzi do symulacji kolejnego osobnika pokolenia – na skrzyżowanie wjeżdża ta sama liczba samochodów, sygnalizacja zmienia się tym razem w odstępach określonych kolejnym osobnikiem. Cykl powtarza się, aż symulacji poddane zostaną wszystkie osobniki pokolenia. Następnie aplikacja dokonuje oceny i generuje nowe pokolenie, co opisano w rozdziale *Opis teoretyczny algorytmu ewolucyjnego*. Potem odbywa się symulacja osobników nowego pokolenia. Cykl ten powtarza się, aż osiągnięta zostanie liczba pokoleń określona w konfiguracji.
Prawa część ekranu przedstawia stan uczenia – numer obecnego pokolenia i osobnika. Po pierwszym pokoleniu dodatkowo w tej części wyświetlana jest informacja o tym, jaki był czas trwania symulacji najlepszego osobnika poprzedniego pokolenia. Prezentuje to rysunek 9. Z upływem pokoleń czas symulacji najlepszego osobnika spada, co jest widoczne, jeśli porówna się rysunek 10 do rysunku 9.

Ekran końcowy

Po ukończeniu procesu uczenia użytkownikowi ukazuje się ekran końcowy (widoczny na rysunku 11), który przedstawia wyniki uczenia. Informacje prezentowane na tym ekranie to czas trwania symulacji najlepszego osobnika ostatniego pokolenia i jego parametry. Na tym ekranie znajduje się również przycisk *Wyjdź*, który pozwala na zamknięcie aplikacji.

Rysunek 8. Proces uczenia i symulacji ruchu podczas pierwszego pokolenia

Ruch pojazdów

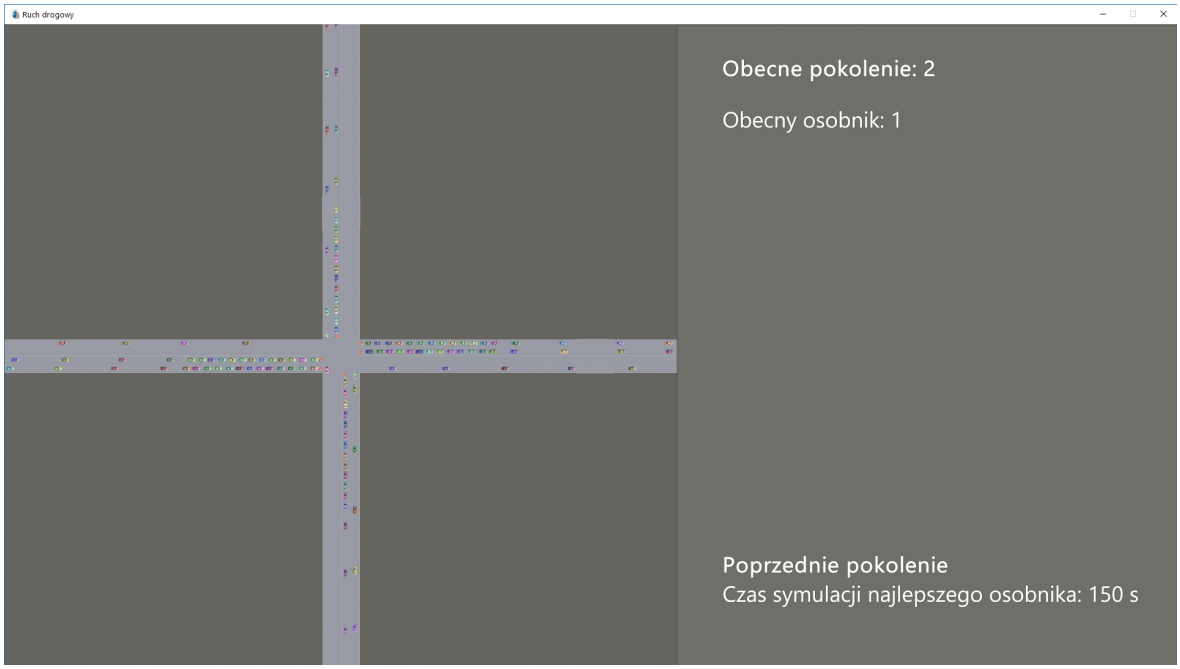
Każdy pojazd porusza się wzdłuż jednej z 12 krzywych sklejanych – przykładową przedstawiono na rysunku 13 kolorem zielonym. Każda z nich składa się z krzywych Béziera drugiego stopnia.

Dodatkowe proste narzędzia

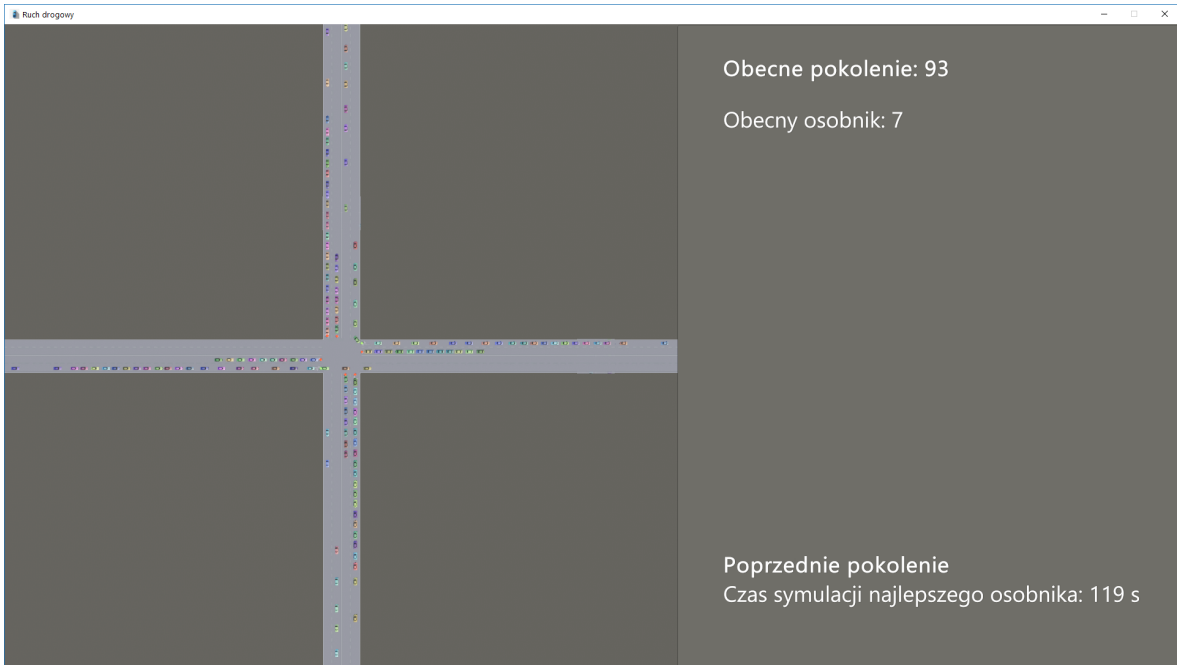
Przy okazji powstawania tej aplikacji stworzono proste narzędzie do modyfikacji krzywych sklejanych. Pozwala ono w łatwy sposób przesuwać ich punkty kontrolne w trybie graficznym (co przedstawiono na rysunku 12) lub wprowadzając ich współrzędne (co ukazuje rysunek 13).

Stworzono również narzędzie do szybkiego tworzenia scenariuszy, widoczne na rysunku 14. Pozwala ono określić liczbę kroków i listę sygnalizatorów, które będą w danym kroku zielone.

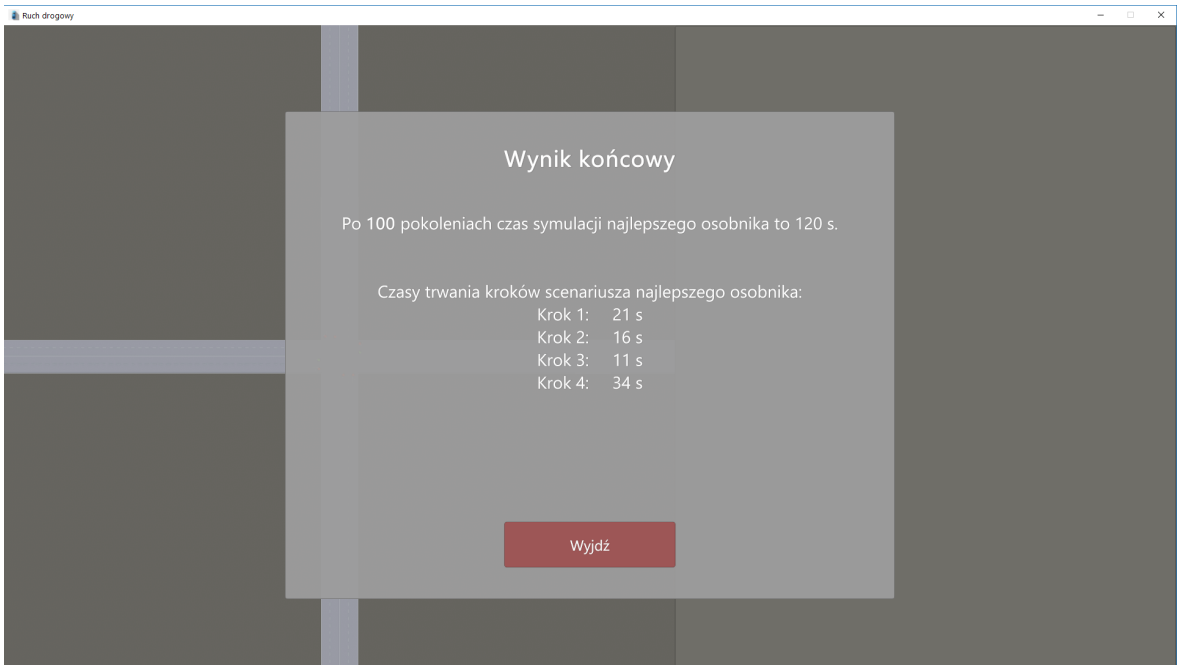
Narzędzia te skróciły czas potrzebny na przygotowanie aplikacji i ułatwiły testowanie różnych rozwiązań.



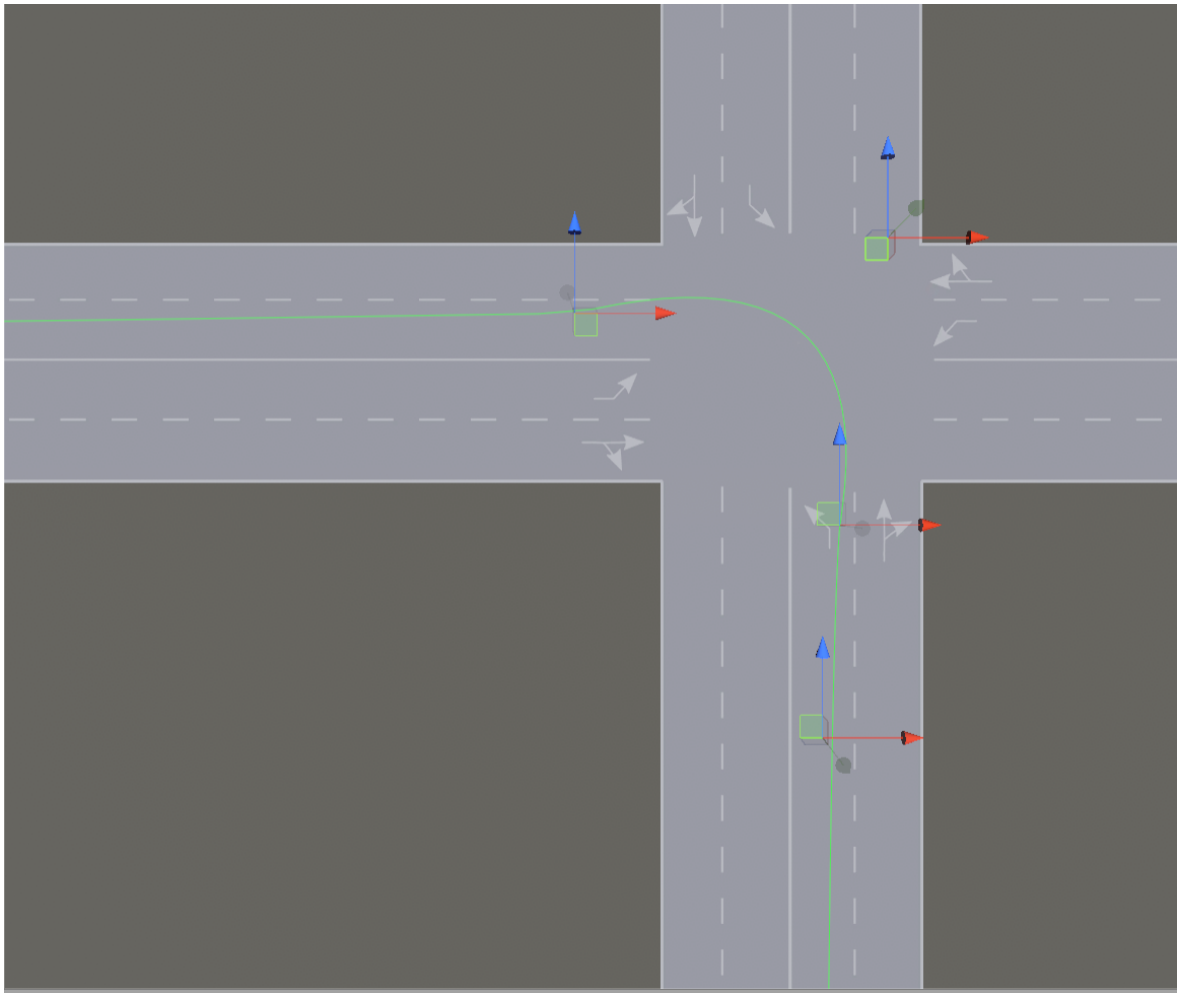
Rysunek 9. Proces uczenia i symulacji ruchu podczas drugiego pokolenia



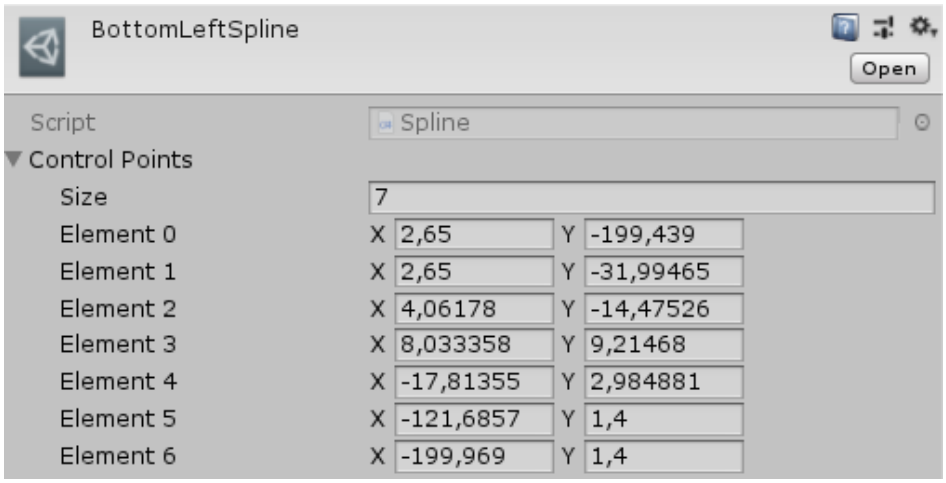
Rysunek 10. Proces uczenia i symulacji ruchu podczas 93. pokolenia



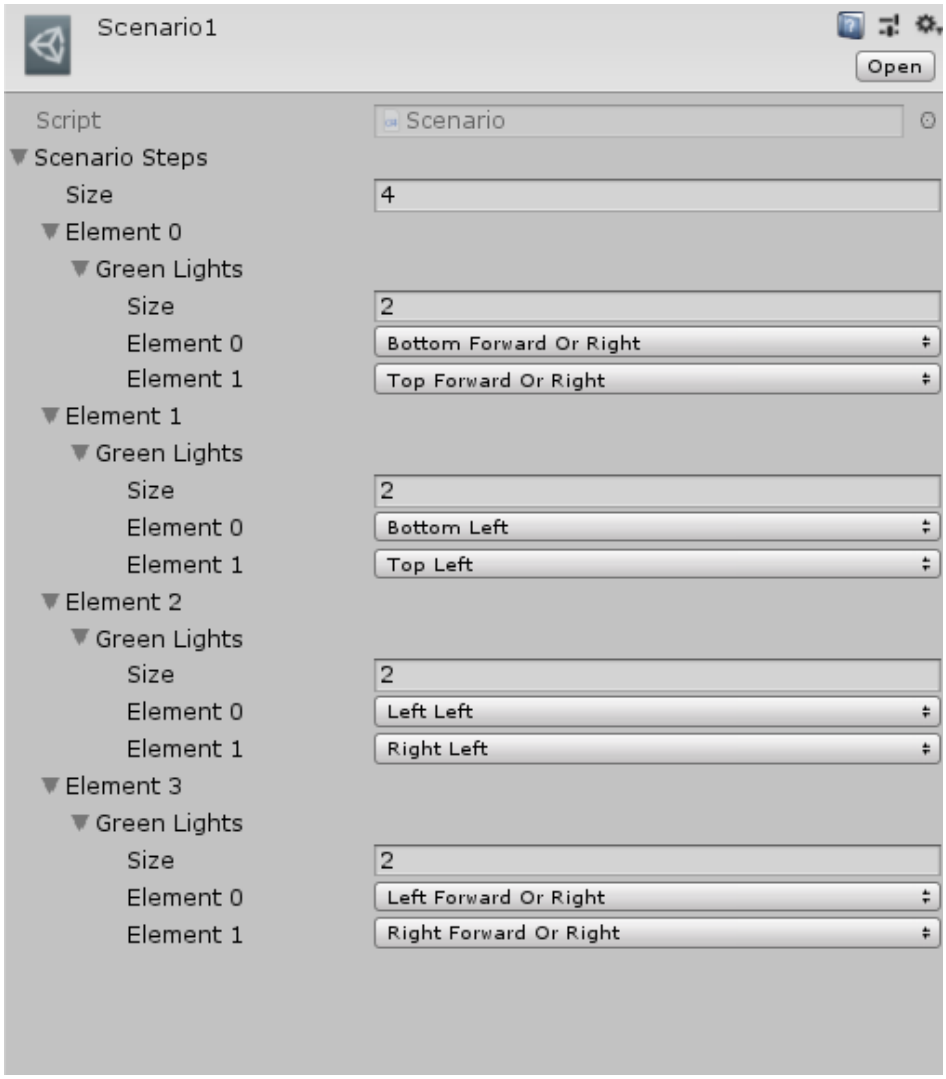
Rysunek 11. Ekran końcowy



Rysunek 12. Narzędzie do modyfikacji krzywych sklejanych w trybie graficznym



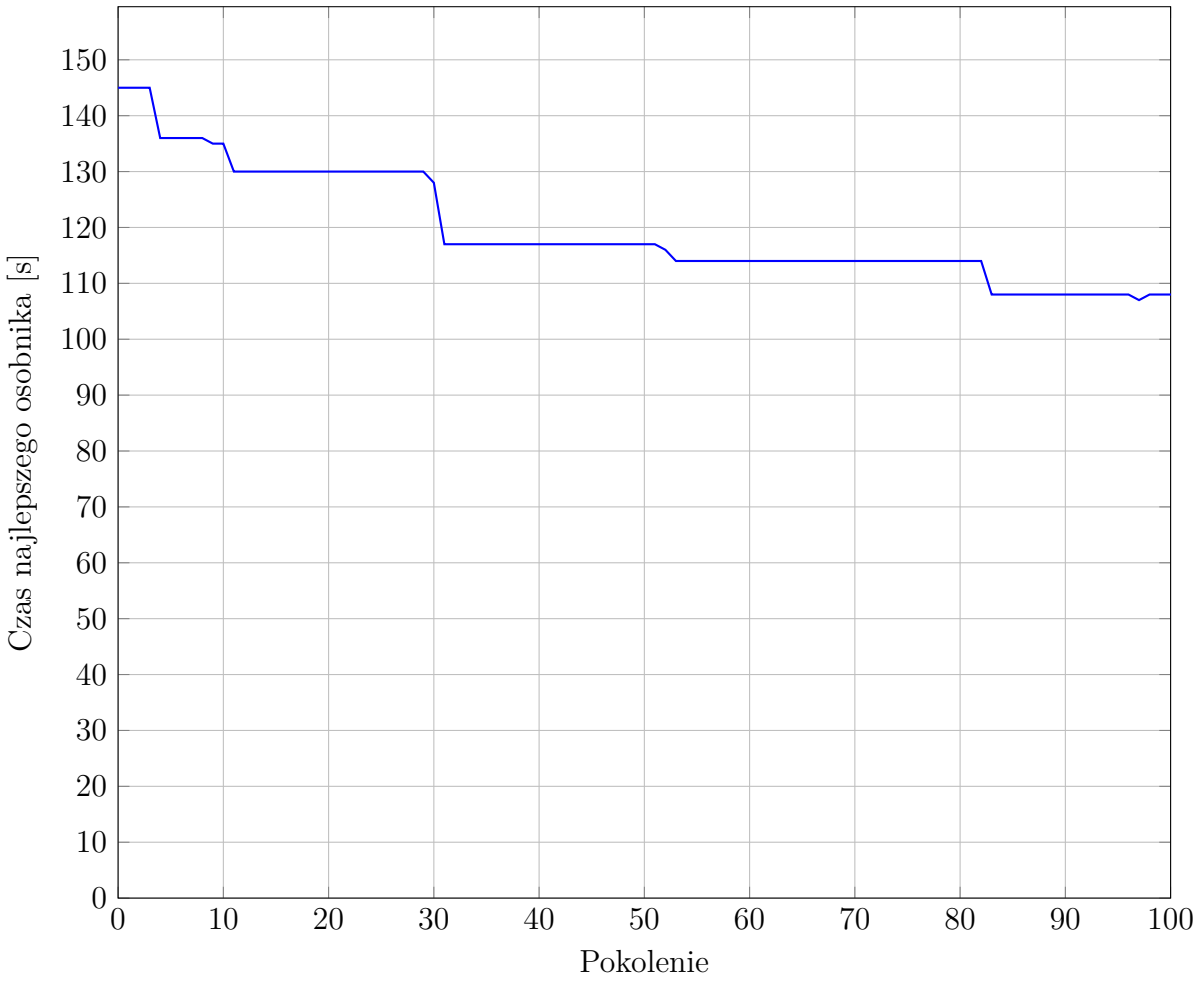
Rysunek 13. Narzędzie do modyfikacji krzywych sklejanych w trybie tekstowym



Rysunek 14. Narzędzie do tworzenia scenariuszy

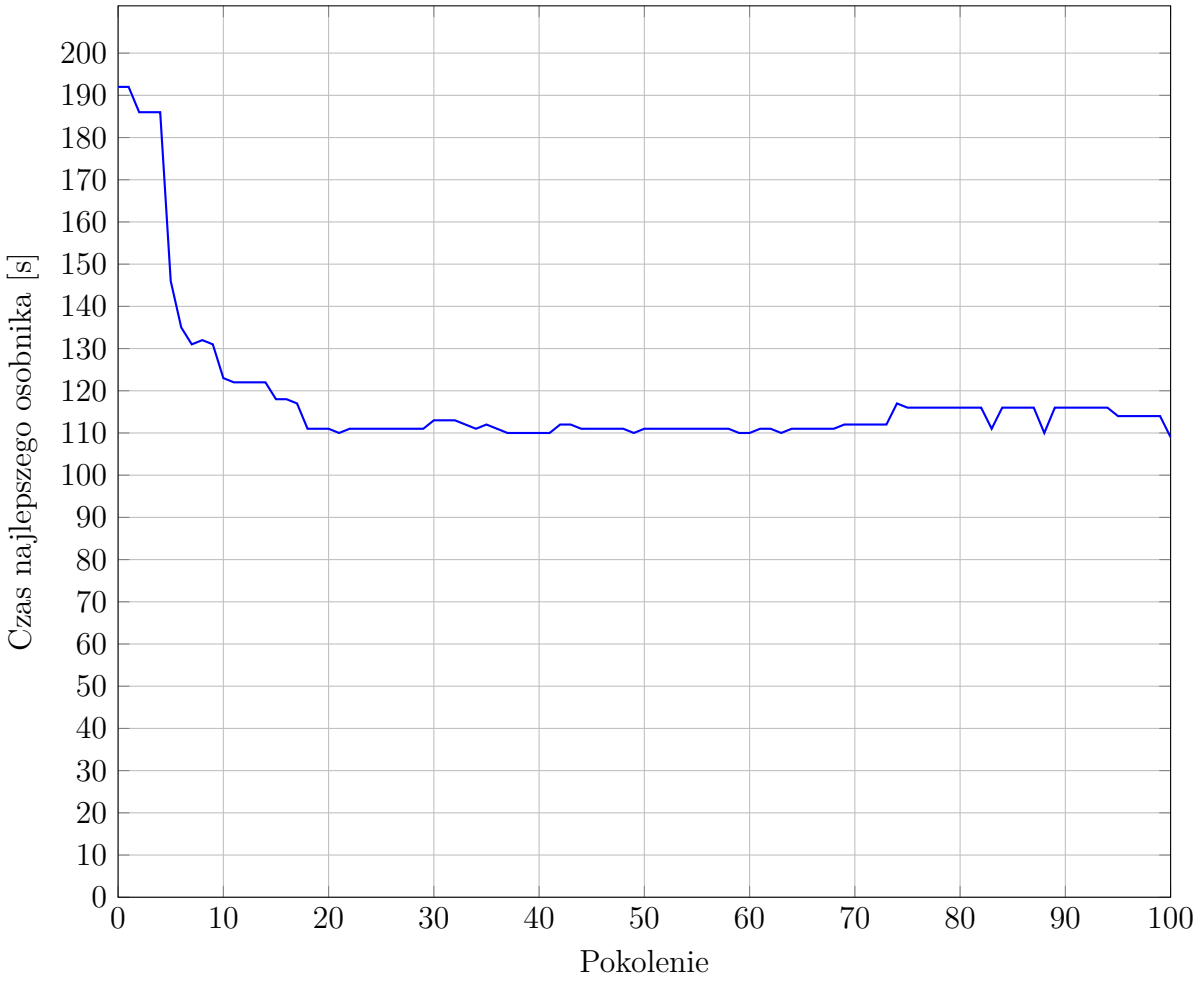
Wyniki uczenia sygnalizacji

W tym rozdziale przedstawione są przykładowe wyniki uczenia sygnalizacji. Rysunek 15 przedstawia rezultaty z użyciem algorytmu opisanego w rozdziale *Opis teoretyczny algorytmu ewolucyjnego*. W tym przykładzie rozmiar populacji pokolenia wynosił 10 osobników, a pozostałe ustawienia były domyślne. Jeśli porównać najlepszego osobnika ostatniego i pierwszego pokolenia, to nastąpiła 26-procentowa poprawa czasu.



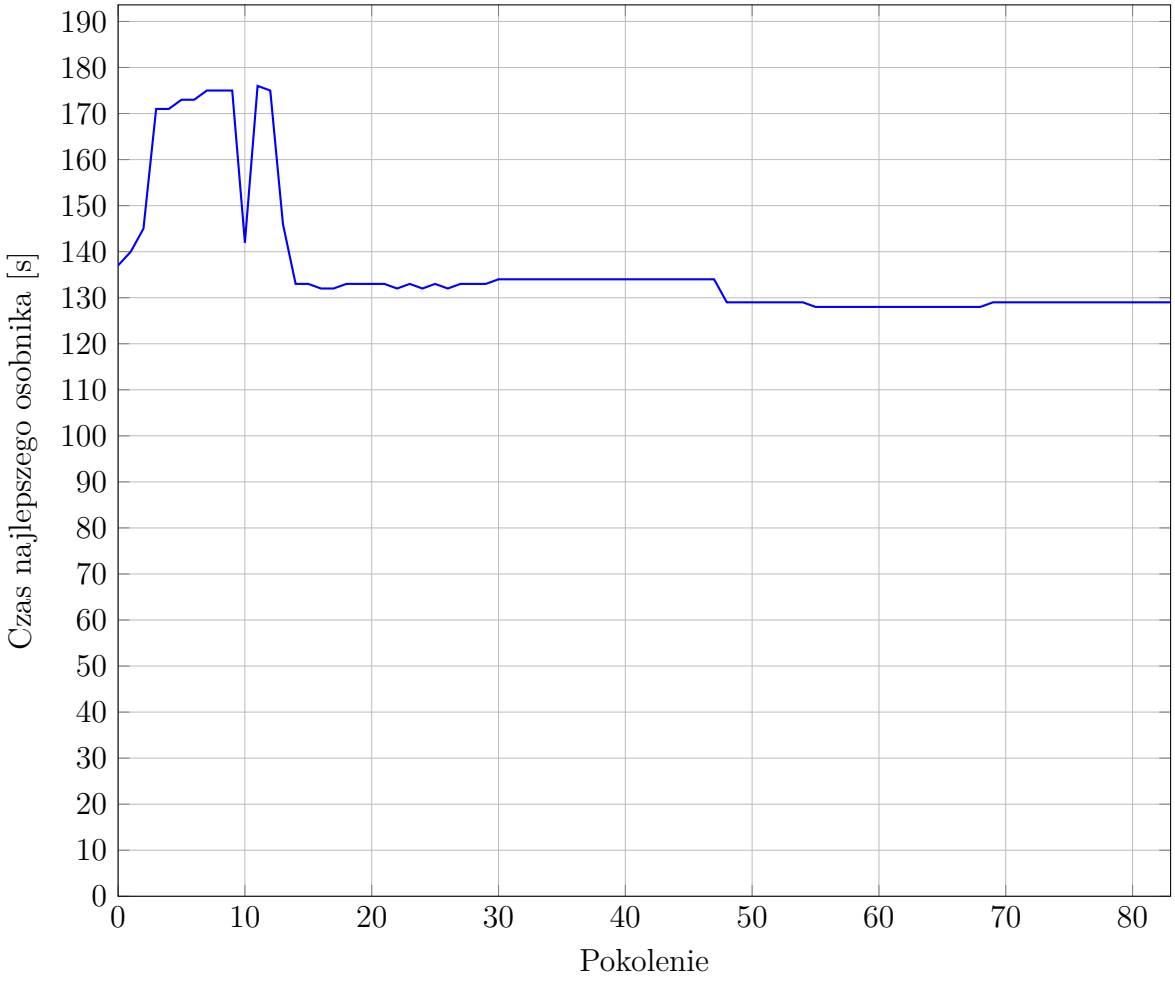
Rysunek 15. Wynik uczenia z 10 osobnikami w pokoleniu

Rysunek 16 przedstawia wyniki z wykorzystaniem tego samego algorytmu co wyżej, z tą różnicą, że rozmiar turnieju to 2 osobniki, a najlepszy osobnik nie przechodził bezwarunkowo do kolejnego pokolenia. Rozmiar populacji pokolenia to ponownie 10 osobników, maksymalny współczynnik mutacji to 0,33, a do pozostałych ustawień użyto domyślnych wartości. Ostatecznie uzyskano tutaj 43-procentowy spadek czasu, jednak to w dużej mierze zasługa wysokiego czasu w pierwszym pokoleniu.



Rysunek 16. Wynik uczenia inną wersją algorytmu

Z kolei na rysunku 17 przedstawiono wyniki z wykorzystaniem stałego współczynnika mutacji równego 0,1 oraz populacji o rozmiarze 20 osobników. Uzyskano tu jedynie 6-procentowy spadek czasu między pierwszym a ostatnim pokoleniem.



Rysunek 17. Wynik uczenia z 20 osobnikami w pokoleniu i stałym współczynnikiem mutacji

Podsumowanie

Celem pracy było stworzenie aplikacji usprawniającej płynność ruchu drogowego poprzez optymalizację sygnalizacji świetlnej.

Symulacja ruchu drogowego

Zaimplementowano symulację ruchu pojazdów reagujących na sygnalizatory i inne pojazdy, odpowiednio hamujących lub przyspieszających w odpowiednich momentach. Ta symulacja stała się podstawą procesu optymalizacji decydując o ocenie każdego rozwiązania. Kod jest na tyle zoptymalizowany, że na współczesnych procesorach symulacja może bezproblemowo być uruchomiona ze 100-krotnym przyspieszeniem. Przy takim ustawieniu 100-pokoleniowy proces uczenia z 10 osobnikami na pokolenie trwa około godziny. Dalszą drogą do zwiększenia wydajności mogłoby być wprowadzenie wielowątkowości – silnik Unity umożliwia wykorzystanie wszystkich wątków procesora, wymaga to jednak nieco innej struktury kodu i rozważenia przypadków charakterystycznych dla obliczeń równoległych.
Głównym wyzwaniem w implementacji symulacji było czasochłonne szukanie błędów w kodzie oraz jego optymalizacja.

Uczenie sygnalizacji świetlnej

Zaimplementowano algorytm ewolucyjny, który optymalizuje parametry sygnalizatorów świetlnych. Udostępniono możliwość konfiguracji ustawień algorytmu. Na podstawie rozdziału *Wyniki uczenia sygnalizacji* można stwierdzić, że algorytm jest dość skuteczny w znajdowaniu coraz lepszych rozwiązań. Jego implementacja była największym wyzwaniem w tej pracy, w szczególności dobór odpowiedniej techniki mutacji i krzyżowania osobników. Zastosowany algorytm można rozwinąć, wprowadzając dynamiczną regulację jego parametrów (jak choćby współczynnika mutacji) zależną od tempa postępów uczenia w poprzednich pokoleniach.

Stabilność aplikacji jest na zadowalającym poziomie – nawet przy wielogodzinnej pracy nie występują błędy.

Postawiony cel pracy udało się osiągnąć, a aplikacja spełnia postawione wymagania funkcjonalne i pozafunkcjonalne. Można ją rozwinąć wprowadzając inne typy skrzyżowań lub dodając możliwość tworzenia własnych scenariuszy przez użytkownika.

Bibliografia

[1] Murray, W., Wright, M. H., Gill, P. E. *Practical Optimization.* Emerald Publishing Limited, 1982.

[2] Badar, A. Q. H., Umre, D. B. S., Junghare, D. A. S. Study of artificial intelligence optimization techniques applied to active power loss minimization.

[3] Cohoon, J., Karro, J., Lienig, J. Advances in evolutionary computing. Springer-Verlag, Berlin, Heidelberg, 2003.

[4] Berger-Tal, O., Nathan, J., Meron, E., Saltz, D. The exploration-exploitation dilemma: A multidisciplinary framework. *PLOS ONE 9*, 4 (2014).

[5] Globus, A., Hornby, G., Linden, D., Lohn, J. Automated antenna design with evolutionary algorithms.

[6] Leon, N., Uresti, E., Arcos, W. Fan-shape optimisation using cfd and genetic algorithms for increasing the efficiency of electric motors. *IJCAT 30* (11 2007).

[7] Wall, M. B. *A Genetic Algorithm for Resource-constrained Scheduling.* PhD thesis, A Genetic Algorithm for Resource-constrained Scheduling, Cambridge, MA, USA, 1996.

[8] Stanisławska, K., Krawiec, K., Kundzewicz, Z. W. Modeling global temperature changes with genetic programming. *Comput. Math. Appl. 64*, 12 (2012).

[9] Steinbuch, R. Successful application of evolutionary algorithms in engineering design. *Journal of Bionic Engineering 7*, 4 (2010).

[10] Unity Manual - Graphics API support. https://docs.unity3d.com/Manual/GraphicsAPIs.html (dostęp 17.01.2019).

[11] Programming in Unity. https://unity3d.com/programming-in-unity (dostęp 19.01.2019).

[12] Albahari, J. *C# 7.0 in a Nutshell.* O'Reilly UK Ltd., 2017.

[13] Thierens, D. *Adaptive mutation rate control schemes in genetic algorithms.* IEEE, 2002.

Spis rysunków