

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

lebelebe

Abstract

webewebe

Spis treści

Streszczenie	1
Abstract	2
Wstęp	4
Problemy optymalizacyjne	4
Algorytmy ewolucyjne	4
Przykładowe zastosowania algorytmów ewolucyjnych	5
Użyte technologie	6
Unity	6
Język C#	6
Inkscape	7
Blender	9
Projekt aplikacji	10
Wymagania funkcjonalne	10
Wymagania pozafunkcjonalne	10
Przypadki użycia	10
Przebieg przypadku użycia „Przeprowadź proces optymalizacji sygnalizacji świetlnej”	10
Opis teoretyczny algorytmu ewolucyjnego	12
Pierwsze pokolenie	12
Funkcja dostosowania	12
Generowanie nowego pokolenia	12
Mutacja i krzyżowanie	13
Opis aplikacji	15
Menu główne	15
Scenariusz	17
Przebieg uczenia i symulacji ruchu	17
0.1. Ekran końcowy	17
Bibliografia	20
Spis rysunków	21

Wstęp

Komputery są obecnie niezwykle popularnymi urządzeniami znajdującymi zastosowanie w wielu dziedzinach życia. Są pomocne także w zadaniach inżynierskich 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 inżynierskich. 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. Jednym z głównych czynników 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 (ang. *fitness*) 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 genetycznego 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 odpowiedniej 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 optymalizacji sygnalizacji świetlnej na skrzyżowaniu. 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.

TODO: OPIS ROZDZIAŁÓW

Użyte technologie

Głównymi czynnikami decydującymi o doborze technologii 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

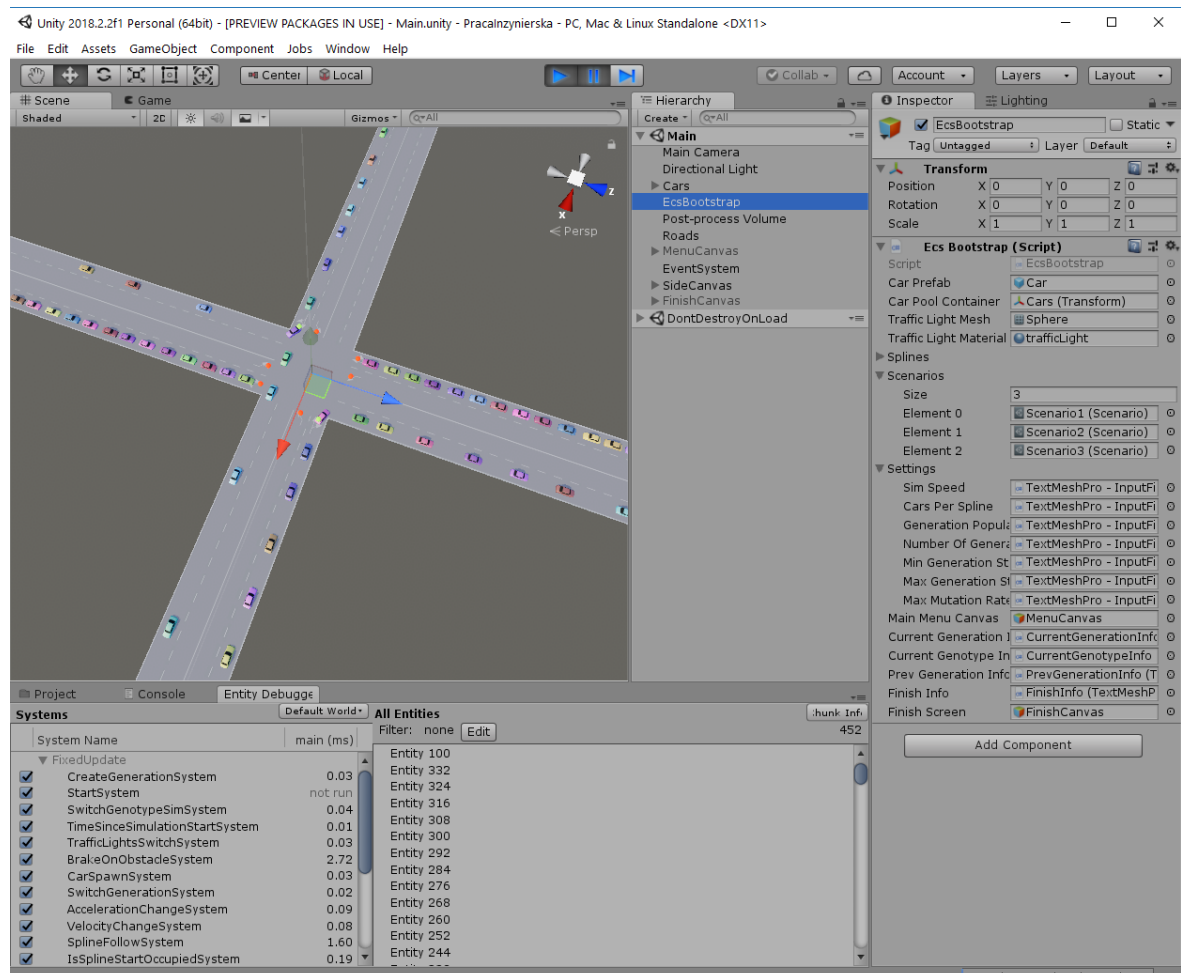
Unity to wieloplatformowy silnik autorstwa firmy Unity Technologies, którego głównym zastosowaniem jest tworzenie gier dwuwymiarowych i trójwymiarowych oraz symulacji. W porównaniu do podobnych narzędzi wyróżnia się prostotą obsługi, dużym naciskiem na edytory wizualne i zintegrowanym środowiskiem, które nie wymaga skomplikowanej konfiguracji. Wspiera wiele API graficznych (DirectX, OpenGL, Metal i Vulkan)[10] pod wspólnym interfejsem. Dzięki temu twórcy mogą w łatwy sposób tworzyć wersje swoich aplikacji na wiele platform bez 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 WYSIWYG¹ 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 m.in. pozwalają 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”

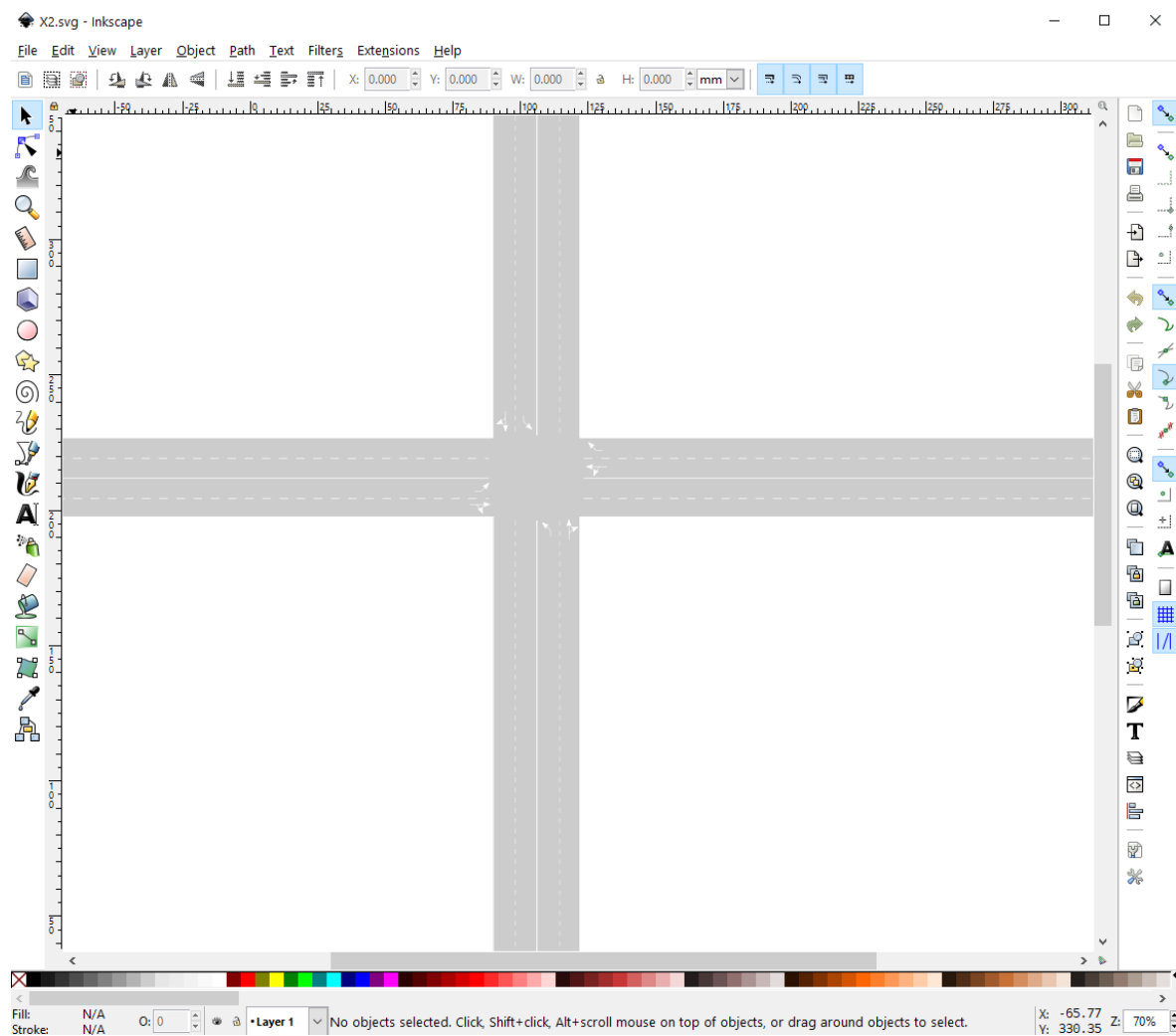
tym zwięzłego, kodu. Pierwotnie kod C# mógł być uruchamiany jedynie na systemie Windows. Zmieniło się to jednak w ostatnich latach dzięki środowiskom takim jak Mono oraz .NET Core i obecnie wspierane są inne platformy, takie jak Linux, macOS, iOS czy Android [12].



Rysunek 1. Interfejs edytora Unity

Inkscape

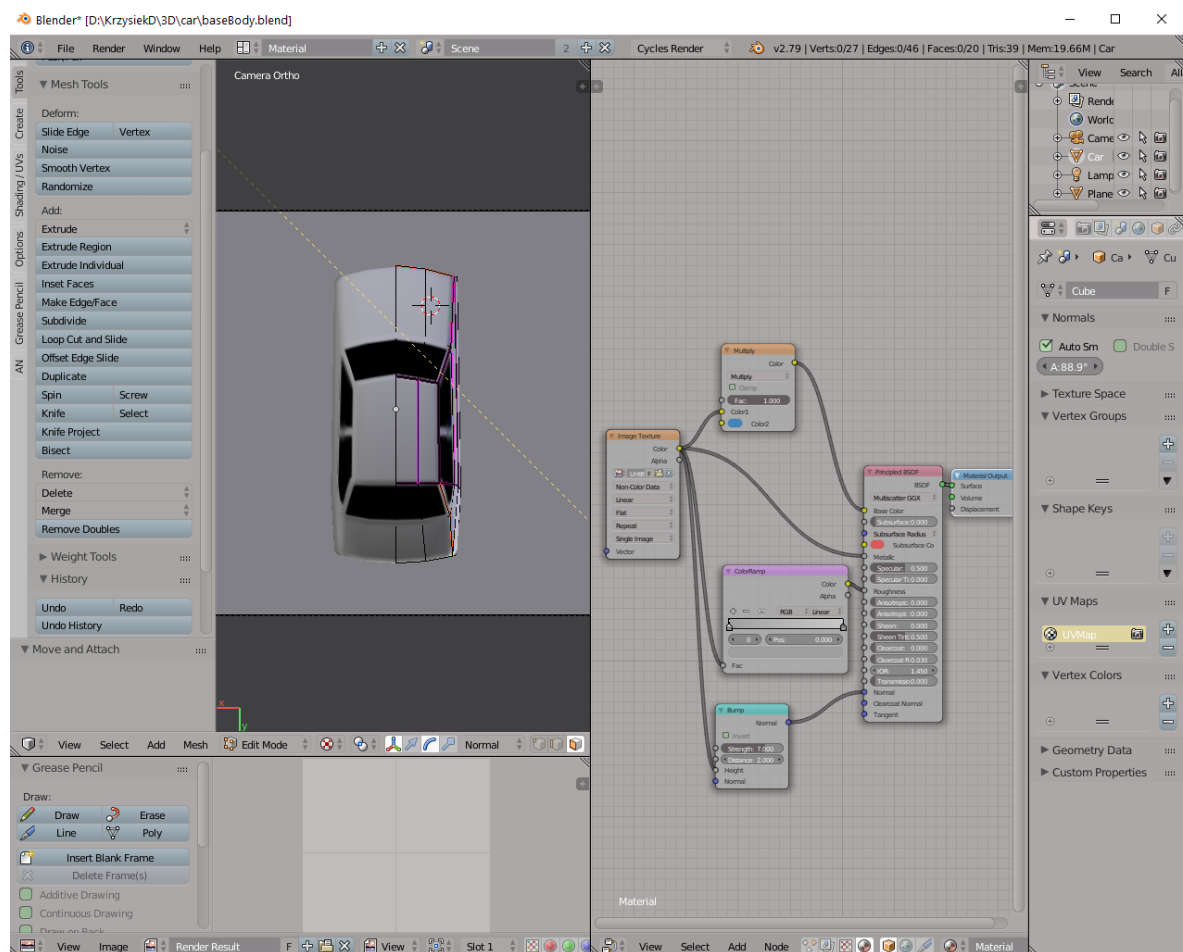
Inkscape to darmowy program do tworzenia grafiki wektorowej. Tworząc w nim obrazy korzysta się z podstawowych kształtów takich jak elipsy, wielokąty, krzywe czy strzałki. Inkscape używa formatu Scalable Vector Graphics (SVG). Mimo że program opiera się na formacie wektorowym, plikiem wynikowym rysunku 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

- Przeprowadzenie procesu uczenia optymalnych czasów świecenia sygnalizatorów na skrzyżowaniu za pośrednictwem algorytmu ewolucyjnego,
- 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

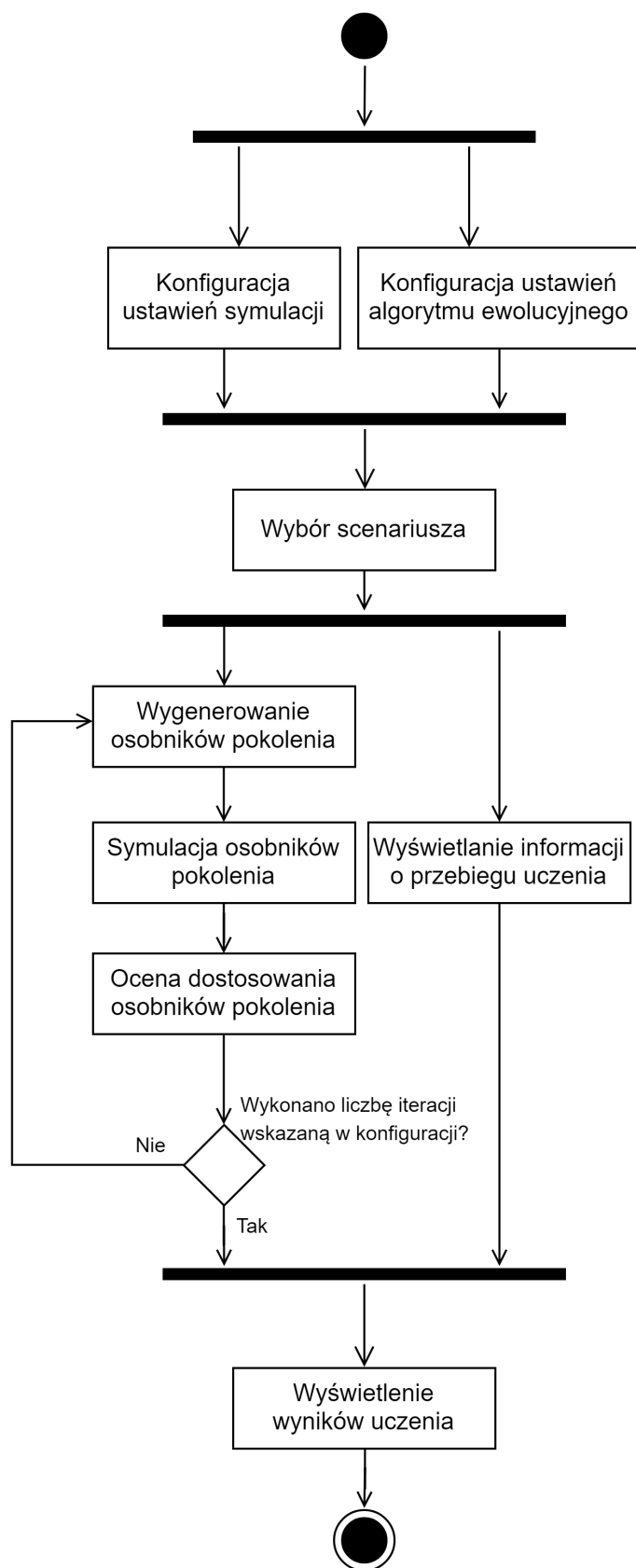
- Stabilność – aplikacja musi pracować bezawaryjnie przez wiele godzin,
- Wydajność – aplikacja musi działać na tyle wydajnie, aby symulacja mogła być przeprowadzana kilkadziesiąt razy szybciej niż czas rzeczywisty.

Przypadki użycia

W projekt 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. Następnie wybiera scenariusz (czyli kolejność zapalania się sygnalizatorów) co powoduje rozpoczęcie procesu uczenia. Aplikacja kolejno generuje, symuluje, a potem ocenia osobniki 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:

$$f(x) = \max(t(1), \dots, t(n)) - t(x) \quad (1)$$

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 ważne losowanie. Prawdopodobieństwo, że osobnik zostanie poddany mutacji jest zmienne i wyznacza je funkcja:

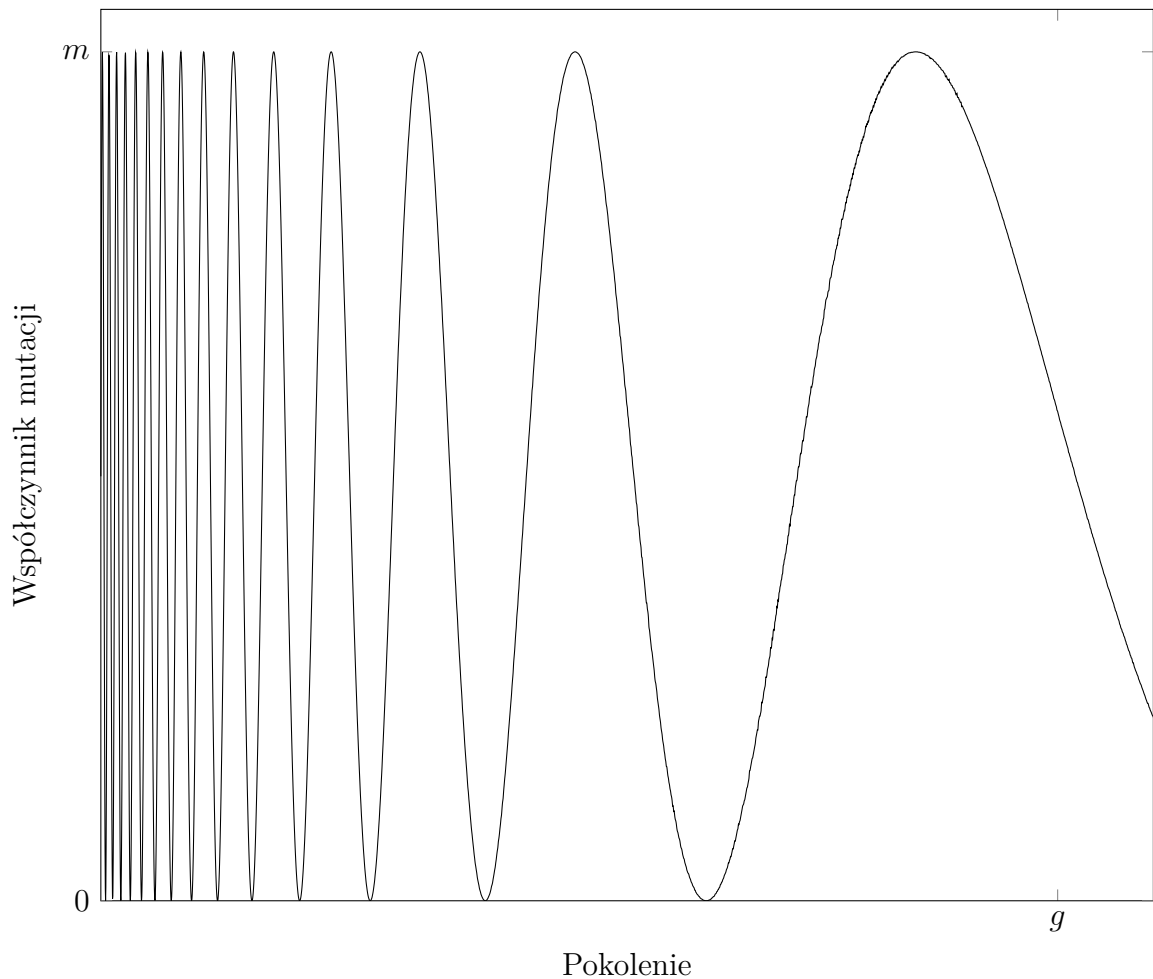
$$f(x) = \left(\sin\left(x * \frac{1}{\frac{g}{1000} + \frac{x}{100}}\right) + 1 \right) : 2 \cdot m \quad (2)$$

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 podstawie 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.

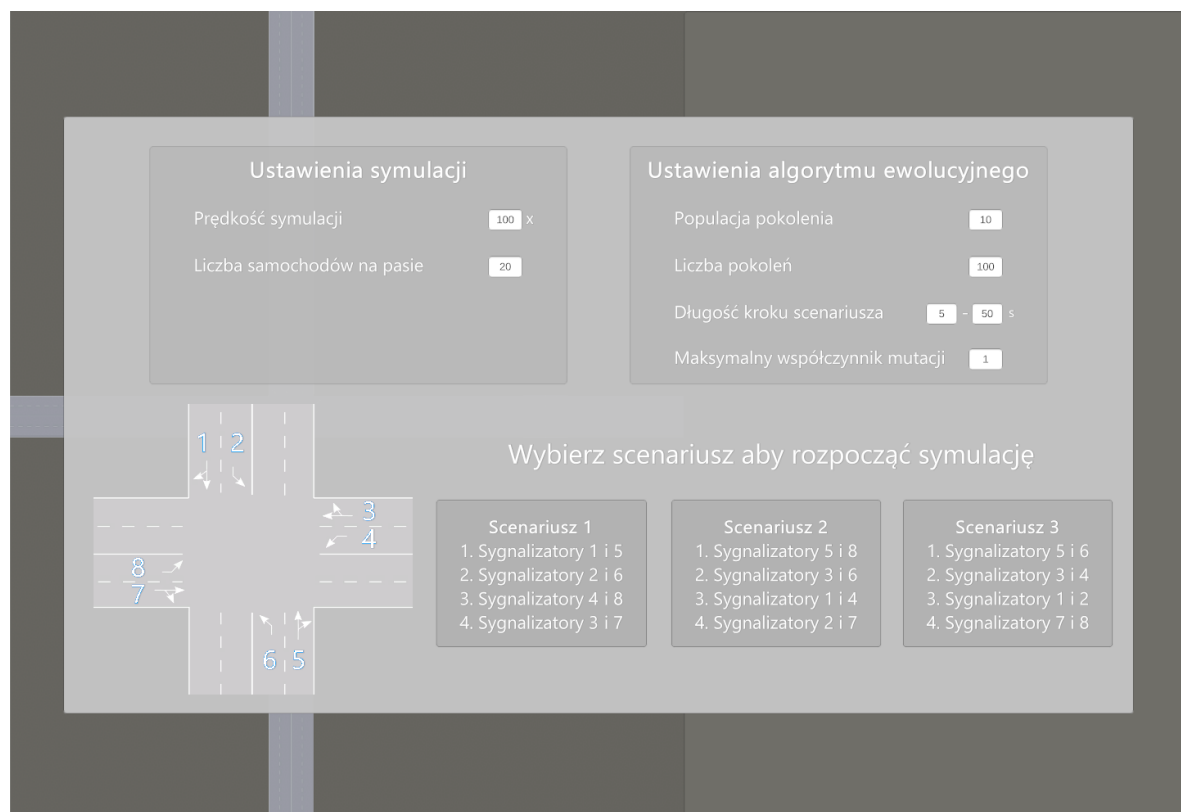
Dostępne ustawienia symulacji to:

- 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 się mieścić 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

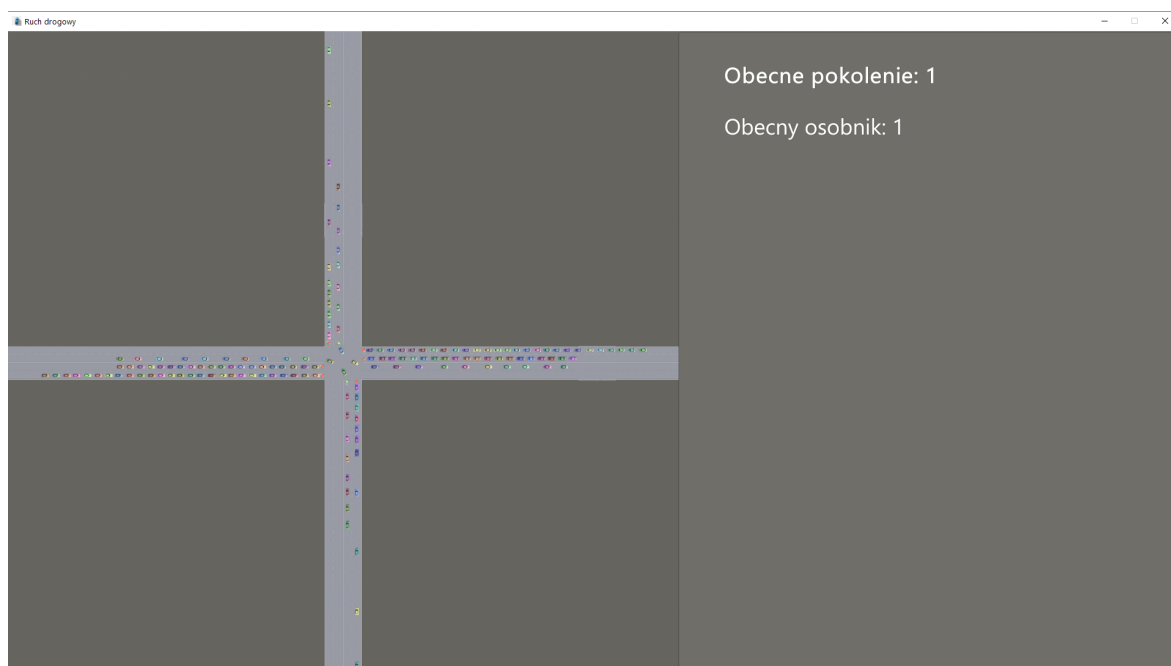
Na przykładzie Scenariusza 1 widocznego w menu: podczas pierwszego kroku scenariusza zielone będą sygnalizatory nr 1 i 5 (oznaczone na schemacie po lewej), zaś pozostałe będą czerwone. Po przejściu do drugiego kroku zielone będą sygnalizatory nr 2 i 6, podczas gdy pozostałe będą czerwone. Analogicznie 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 widocznie, jeśli porówna się rysunek 10 do rysunku 9.

Ekran końcowy



Rysunek 8. Proces uczenia i symulacji ruchu podczas pierwszego pokolenia



Rysunek 9. Proces uczenia i symulacji ruchu podczas drugiego pokolenia



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

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

1.	Interfejs edytora Unity	7
2.	Interfejs programu Inkscape	8
3.	Interfejs programu Blender	9
4.	Diagram aktywności przypadku użycia „Przeprowadź proces uczenia sygnalizacji”	11
5.	Funkcja określająca współczynnik mutacji	13
6.	Ekran ładowania aplikacji	15
7.	Menu główne aplikacji	16
8.	Proces uczenia i symulacji ruchu	18
9.	Proces uczenia i symulacji ruchu	18
10.	Proces uczenia i symulacji ruchu	19