

Kierunek: Informatyka

**Bartłomiej Lipiński**

Numer Albumu: 402694

**Symulacja zachowań pojazdów w  
mieście stworzona w oparciu o  
silnik Unity**

Simulation of vehicle behavior in city created in  
Unity engine

Praca napisana  
pod kierunkiem:  
dr. Sebastiana Lindnera  
w Katedrze Funkcji Rzeczywistych  
Wydziału Matematyki i Informatyki  
Uniwersytetu Łódzkiego

## Spis treści

Rozdział 1 Wstęp.....	4
Rozdział 2 O projekcie .....	5
Główne założenia .....	5
Inspiracje .....	5
Rozdział 3 Mechanika symulacji .....	7
Opis głównych elementów .....	7
Światła .....	7
Miasto .....	8
Pojazdy .....	11
Ulice .....	12
Aspekt graficzny .....	13
Światła .....	13
Miasto .....	14
Pojazdy .....	15
Ulice .....	16
Rozdział 4 Implementacja kodu .....	18
Środowisko Unity .....	18
Własne skrypty .....	18
Punkty docelowe .....	18
Światła .....	18
Miasto .....	18
Pojazdy .....	18
Ulice .....	18
Ułatwienia edycji .....	19
Rozdział 5 Dalszy rozwój .....	19
Optymalizacja .....	19

Poprawa algorytmu ruchu .....	20
Edytor ustawień miasta i pojazdów .....	22
Edytor dróg .....	23
Rozdział 6 Zakończenie .....	25
Rozdział 7 Bibliografia .....	26

## Streszczenie

TODO

## Summary

TODO

# Rozdział 1 Wstęp

Celem symulacji ruchu pojazdów w miejskim środowisku jest stworzenie w miarę realistycznego modelu, który odzwierciedla rzeczywiste warunki drogowe i zachowania kierowców. Wykorzystanie silnika Unity pozwala na implementację zaawansowanych algorytmów oraz dodatkowych narzędzi, które wspierają realizację tego celu.

Symulację można zastosować jako:

### 1. Narzędzie planowania urbanistycznego:

- Używane do testowania różnych scenariuszy planowania infrastruktury drogowej i oceny ich wpływu na ruch miejski za pomocą zebranych danych.

### 2. Grę/aplikację rozrywkową:

- Użycie w grach komputerowych i symulatorach, gdzie realistyczny ruch uliczny jest kluczowym elementem.

# Rozdział 2 O projekcie

## Główne założenia

Główne założenia symulacji ruchu pojazdów w mieście w silniku Unity obejmują szeroki zakres technicznych aspektów, w tym:

### 1. Algorytmy ruchu:

- Implementacja algorytmów ruchu pojazdów, które określają ich zachowanie na drodze, takie jak utrzymywanie odpowiedniego odstępu od innych pojazdów oraz prawidłowe pokonywanie skrzyżowań i zakrętów.
- Zastosowanie różnych technik sterowania, takich jak algorytmy przyspieszania, hamowania i skręcania, aby odzwierciedlić zachowanie prawdziwych kierowców.

### 2. Zachowania kierowców:

- Symulacja reakcji kierowców na różne sytuacje, takie jak zmiana prędkości ruchu, żeby uniknąć zderzenia.

### 3. Symulacja świateł drogowych:

- Implementacja logiki świateł drogowych, aby regulować ruch pojazdów na skrzyżowaniach i w innych miejscach, gdzie wymagane jest kontrolowanie przepływu ruchu.

Te założenia są kluczowe dla stworzenia wiarygodnej i realistycznej symulacji ruchu drogowego, która może być wykorzystana do testowania różnych scenariuszy drogowych, badania wpływu infrastruktury drogowej na ruch uliczny, oraz analizy efektywności strategii zarządzania ruchem. Ich skuteczna implementacja pozwala na dokładne odwzorowanie rzeczywistych warunków drogowych i poprawne modelowanie zachowań kierowców.

## Inspiracje

Symulacja ruchu pojazdów w miejskim środowisku stanowi jedno z kluczowych wyzwań w tworzeniu realistycznych gier symulacyjnych oraz narzędzi urbanistycznych. W takim projekcie istotne jest czerpanie inspiracji z istniejących, dobrze ocenianych symulacji miejskich, takich jak "*City Skylines*" 1 i 2,. W tym rozdziale będzie omówione, jak te gry inspirowały projekt oraz które elementy zostały zaadaptowane w naszej symulacji.

"City Skylines" jest grą symulacyjną, która stała się wzorem w dziedzinie zarządzania miastem. Gra oferuje zaawansowany system symulacji ruchu drogowego, który charakteryzuje się realistycznym odwzorowaniem ruchu pojazdów oraz dynamiką przepływu ruchu miejskiego. Kluczowe elementy, które zainspirowały nas w naszym projekcie, obejmują:

### 1. Algorytmy ścieżek (Pathfinding):

- o "City Skylines" wykorzystuje zaawansowane algorytmy, takie jak A\* (A-star), aby efektywnie wyznaczać trasy dla pojazdów. Jest to algorytm znajdowania najkrótszej ścieżki w grafie ważonym między dwoma dowolnymi wierzchołkami. W projekcie został zaimplementowany prostszy algorytm, który nadal pozwala na efektywne przeprowadzanie symulacji.

### 2. Symulacja świateł drogowych:

- o W "City Skylines" zarządzanie światłami drogowymi jest kluczowym elementem, który wpływa na płynność ruchu. Zaadaptowane mechanizmy zarządzania sygnalizacją świetlną, poprawiają realizm ruchu w symulacji.

Czerpanie inspiracji z "City Skylines 1" oraz "City Skylines 2" pozwoliło na stworzenie realistycznej i efektywnej symulacji ruchu pojazdów w miejskim środowisku w silniku Unity. Implementacja działających ścieżek jak i zarządzania ruchem znacząco zwiększyła realizm i funkcjonalność naszej symulacji. Wykorzystanie tych inspiracji pozwala na tworzenie bardziej realistycznych i angażujących symulacji miejskich, które mogą znaleźć zastosowanie zarówno w grach, jak i narzędziach do planowania urbanistycznego.

## Rozdział 3 Mechanika symulacji

Aby ułatwić edycję i zarządzanie informacjami, projekt został zaprojektowany w sposób modułowy od samego początku. Takie podejście umożliwia łatwe dodawanie i modyfikowanie poszczególnych elementów systemu bez konieczności wprowadzania zmian w całym projekcie.


Modułowość w projekcie przynosi kilka kluczowych korzyści:

### 1. Łatwość tworzenia i edytowania ustawień miasta:

- Dzięki modułowej budowie, nowe ustawienia miasta mogą być tworzone i dostosowywane bez ingerencji w resztę projektu. Można łatwo zmieniać parametry symulacji związane z ruchem ulicznym.

### 2. Łatwość tworzenia i edytowania ustawień pojazdów:

- Podobnie jak w przypadku ustawień miasta, konfiguracje pojazdów mogą być łatwo zmieniane lub dodawane. Modułowa struktura umożliwia szybkie dodawanie nowych typów pojazdów, modyfikację ich parametrów technicznych.

Modułowe podejście do projektowania i implementacji systemu znacząco zwiększa jego elastyczność i skalowalność. Umożliwia łatwe dodawanie i modyfikowanie różnych elementów, automatyzację procesów oraz intuicyjną obsługę, co jest kluczowe dla efektywnego zarządzania złożonymi projektami symulacyjnymi. Dzięki temu, zespół może szybko reagować na zmieniające się  wymagania i wprowadzać niezbędne poprawki bez potrzeby przebudowy całego systemu.

## Opis głównych elementów


### Światła

Symulacja świateł drogowych (rys. 3.1) w projekcie ma kilka kluczowych cech:

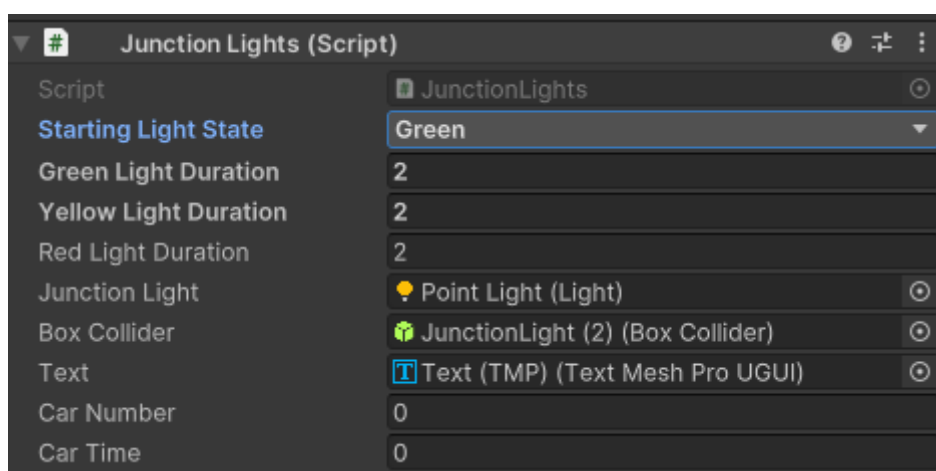
#### 1. Aktualny stan świateł:

- Światła mogą znajdować się w jednym z trzech stanów: zielonym, żółtym lub czerwonym.

#### 2. Dostosowywanie czasu trwania świateł:

- Użytkownik ma możliwość dostosowania czasu trwania każdego pojedynczego światła. Może zmieniać długość trwania stanów zielonego, żółtego i czerwonego, aby symulować różne warunki drogowe i sytuacje ruchu 
3. **Zatrzymywanie ruchu w przypadku czerwonego światła:**
- Gdy światło jest w stanie czerwonym, ruch pojazdów jest zatrzymywany. To kluczowa cecha symulacji, która odzwierciedla rzeczywiste funkcjonowanie sygnalizacji świetlnej na skrzyżowaniach.
4. **Średni czas przejazdu przez dane światło:**
- Symulacja zbiera informacje o średnim czasie przejazdu każdego pojazdu przez konkretne światło i wyświetla je w odpowiednim miejscu. To istotne dla analizy przepustowości skrzyżowań i optymalizacji czasu oczekiwania pojazdów na sygnalizacji świetlnej.

Te funkcje umożliwiają realistyczną symulację zachowania się ruchu na skrzyżowaniach, pozwalając na eksperymentowanie z różnymi ustawieniami sygnalizacji świetlnej i ocenę ich wpływu na płynność ruchu drogowego.



Rysunek 3.1 Zrzut ekranu przedstawiający komponent światel..

## Miasto

W projekcie symulacji ruchu drogowego istnieje jedno miasto (rys. 3.2), które ma kluczowe zadania:

### 1. Zarządzanie infrastrukturą drogową:



- jest odpowiedzialne za przechowywanie listy punktów docelowych. To dzięki niemu symulacja ma pełny obraz infrastruktury drogowej w mieście.

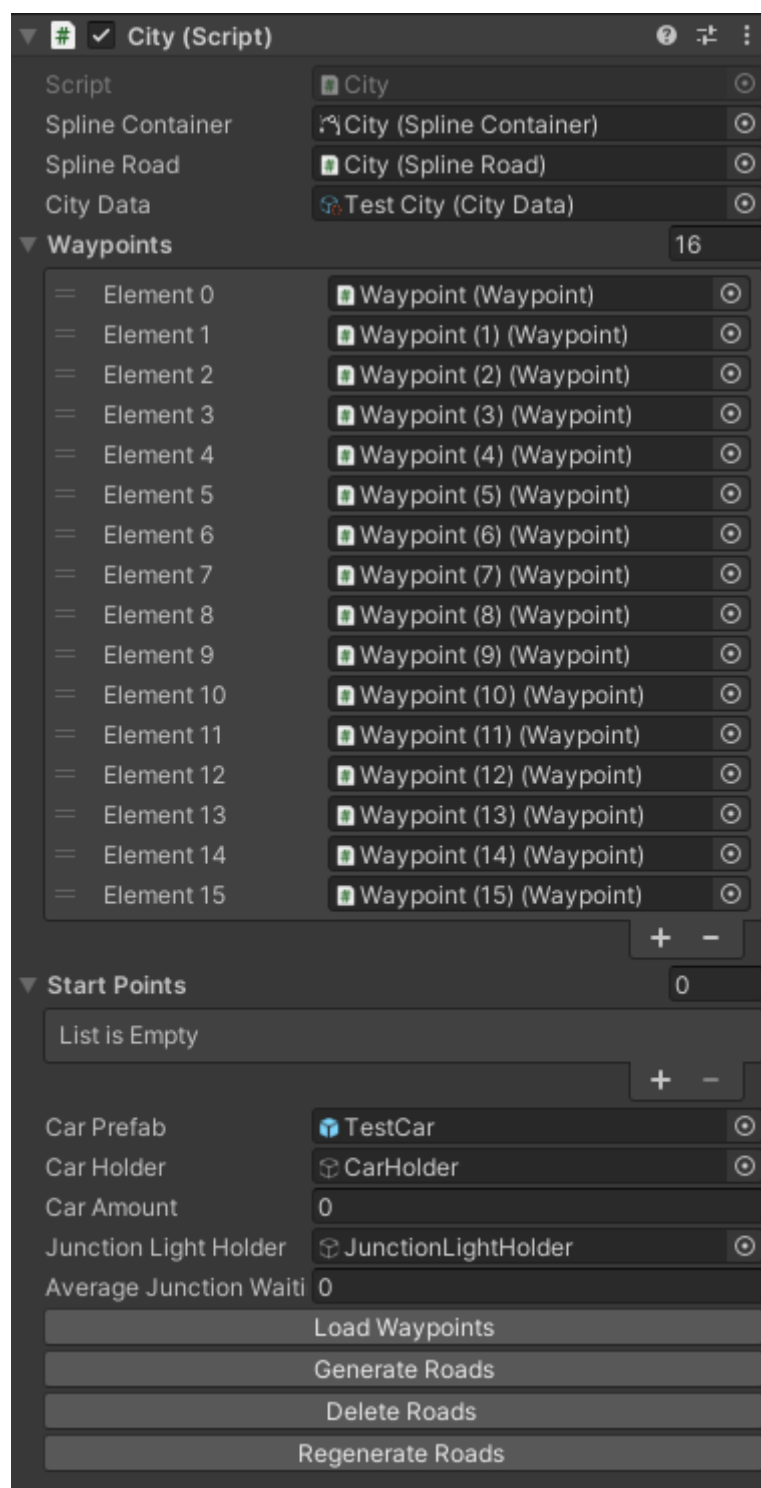
## 2. Monitorowanie stanu świateł drogowych:

- Komponent zbiera dane z wszystkich świateł drogowych w symulacji. Te informacje są wykorzystywane do wyświetlania na interfejsie użytkownika średniego czasu przejazdu przez wszystkie światła. Dzięki temu użytkownik może śledzić, jak zmienia się przepustowość na skrzyżowaniach w różnych warunkach ruchu.

## 3. Analiza zatoru drogowego:

- Średni czas przejazdu przez światła drogowe dostarcza istotnych danych na temat przepustowości dróg i skrzyżowań. Na podstawie tych informacji możliwe jest ocenienie, jak bardzo miasto jest narażone na korki oraz identyfikacja obszarów o największym prawdopodobieństwie wystąpienia zatorów drogowych.

Działanie tego komponentu pozwala nam na klarowne spojrzenie na ruch w mieście, umożliwiając zarówno monitorowanie aktualnego stanu sygnalizacji świetlnej, jak i analizę jego wpływu na przepustowość dróg. To kluczowe dla identyfikacji potencjalnych obszarów do optymalizacji oraz podejmowania decyzji mających na celu poprawę płynności ruchu drogowego.



Rysunek 3.2 Zrzut ekranu przedstawiający komponent miasta.

## Pojazdy

Pojazdy (rys. 3.3) w symulacji poruszają się za pomocą silnika fizycznego, co umożliwia realistyczne odwzorowanie ich zachowania na drodze. Istnieją trzy kluczowe elementy, które charakteryzują sposób, w jaki poruszają się te pojazdy:

### 1. Dostosowywanie prędkości:

- Silnik fizyczny umożliwia dynamiczne modyfikowanie prędkości pojazdów w zależności od warunków otoczenia. Maksymalna prędkość każdego pojazdu jest dostosowywana w taki sposób, aby uniknąć kolizji z innymi pojazdami oraz przeszkodami na drodze.

### 2. Opóźnienie reakcji:

- Aby zwiększyć realizm symulacji, modyfikacje prędkości są wprowadzane z lekkim opóźnieniem. Oznacza to, że reakcja pojazdu na zmiany warunków drogowych nie jest natychmiastowa, co odzwierciedla zachowanie prawdziwych kierowców na drodze.

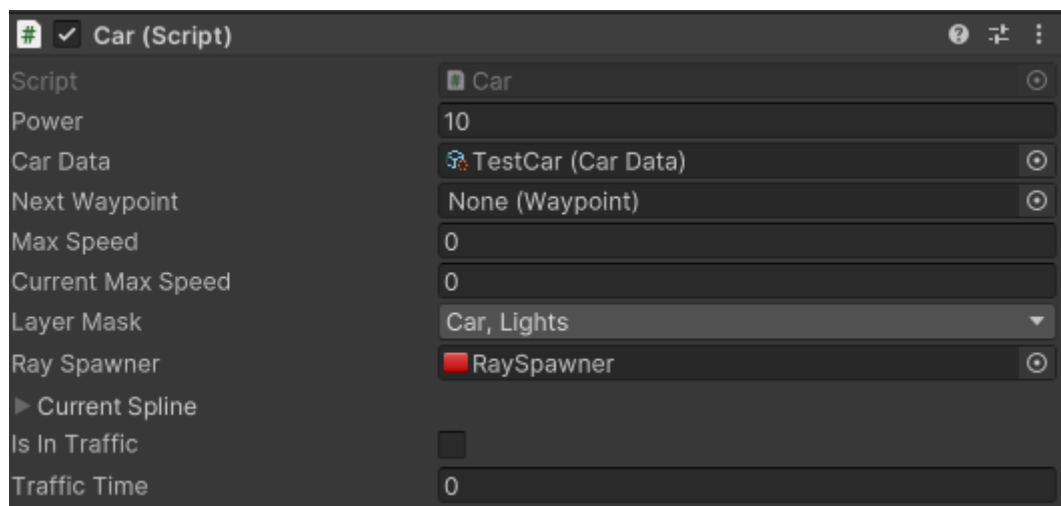
### 3. Losowanie kolejnego punktu docelowego:

- Po dotarciu do aktualnego punktu docelowego, samochód losuje kolejny punkt z listy dostępnych punktów. Dzięki temu symulacja odzwierciedla naturalne przemieszczanie się pojazdów w mieście, gdzie kierowcy wybierają kolejne cele podróży na podstawie aktualnych warunków drogowych i ruchu.

### 4. Zakończenie trasy:

- Gdy samochód dotrze na koniec trasy, czyli do punktu, który nie prowadzi nigdzie, pojazd zostaje usunięty z symulacji. Ten mechanizm pozwala na cykliczne wykorzystywanie pojazdów i zapobiega gromadzeniu się niepotrzebnych obiektów w symulacji.

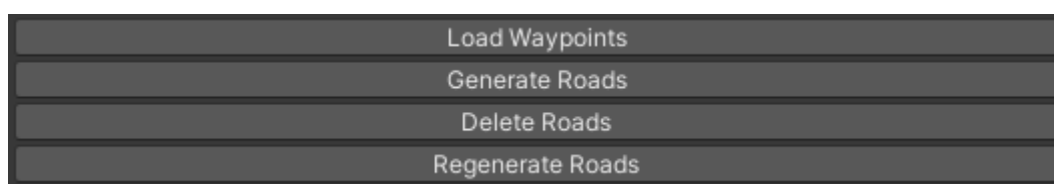
Dzięki tym funkcjom samochody w symulacji są w stanie dynamicznie reagować na zmieniające się warunki drogowe, co przyczynia się do bardziej realistycznego odwzorowania ruchu ulicznego w mieście.



Rysunek 3.3 Zrzut ekranu przedstawiający komponent auta.

## Ulice

System tworzenia ulic był jednym z najtrudniejszych problemów do rozwiązania. Po wielu różnych próbach zdecydowałem na znaczne ułatwienie w modyfikowaniu infrastruktury poprzez dodatkowe menu(rys. 3.4).



Rysunek 3.4 Zrzut ekranu przedstawiający dodatkowe ułatwienia edycji infrastruktury drogowej w mieście.

Po naciśnięciu przycisku „Generate Roads” w komponencie miasta, program przystępuje do dynamicznego tworzenia ulic w mieście. Proces ten przebiega w następujący sposób:

### 1. Przegląd punktów docelowych:

- Program przechodzi przez wszystkie punkty docelowe miasta, uwzględniając listę punktów. Każda droga jest tworzona z uwzględnieniem tych punktów, co pozwala na uwzględnienie wszystkich kluczowych lokalizacji w sieci drogowej.

### 2. Dzielanie drogi na segmenty:

- Droga od punktu do innego punktu docelowego jest dzielona na odpowiednio wiele segmentów, co pozwala na tworzenie realistycznych i dokładnych graficznych reprezentacji dróg w mieście.

### 3. Tworzenie obiektu drogi:

- o Na podstawie tych segmentów tworzony jest kształt drogi. Jest to realizowane poprzez tworzenie krzywej Beziera, która jest elastyczna i pozwala na wygładzanie zakrętów oraz dostosowanie kształtu drogi do topografii terenu.

### 4. Dostosowywanie kształtu drogi:

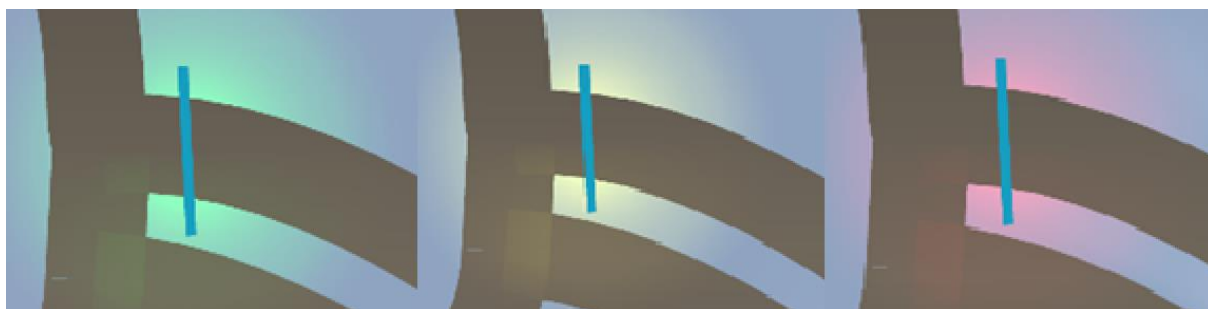
- o Jako że droga jest tworzona jako krzywa Beziera, istnieje możliwość manipulowania jej kształtem, włączając w to wyginanie i dostosowywanie trajektorii drogi. Skrypt automatycznie poprawia grafikę, aby zapewnić płynne i realistyczne wygląd drogi.

Ten proces dynamicznego tworzenia ulic pozwala na elastyczne dostosowywanie infrastruktury drogowej w mieście, uwzględniając zmieniające się warunki i potrzeby. Dzięki temu symulacja może reprezentować różnorodne układy drogowe.

## Aspekt graficzny

Dzięki modułowej konstrukcji projektu, zmiana grafiki jest wyjątkowo prosta i nie wpływa na funkcjonalność mechanizmów symulacji. Taka architektura pozwala na łatwe aktualizacje wizualne, bez ryzyka zakłócenia działania systemu.

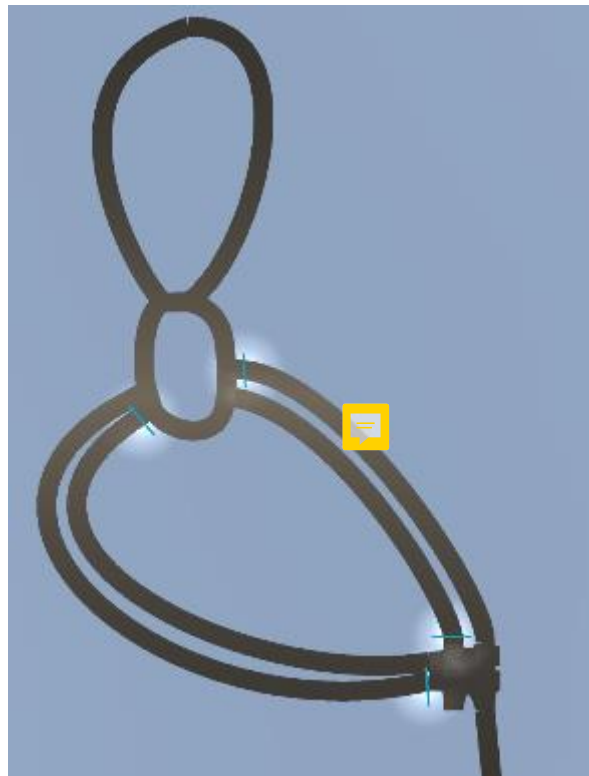
## Światła



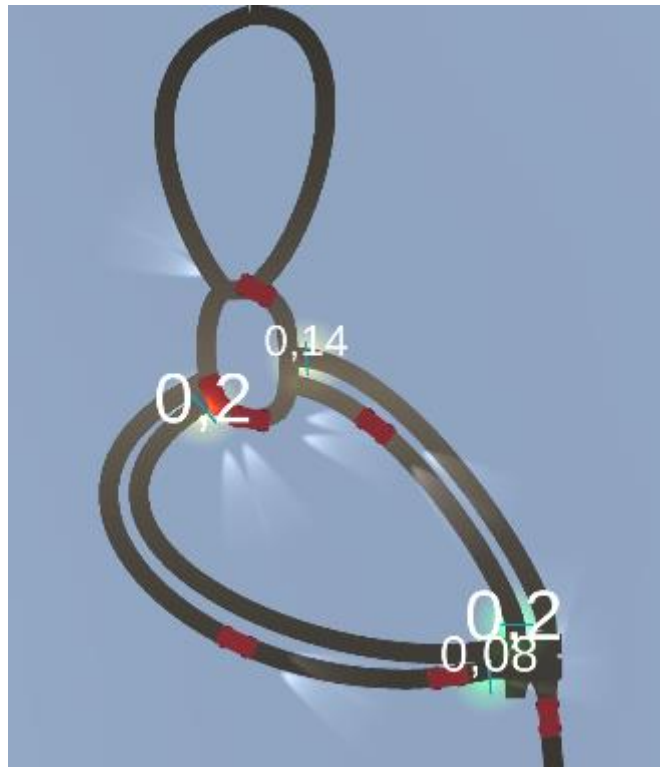
*Rysunek 3.5 Zrzuty ekranu pokazujące graficzne działanie światel sygnalizacyjnych podczas działania symulacji.*

## Miasto

Miasto (rys. 3.6) można rozbudowywać za pomocą różnych komponentów. Punkty docelowe są połączone drogami, po których poruszają się pojazdy. Sygnalizatory dodają dynamiki, zmieniając kolory świateł w określonych odstępach czasu.



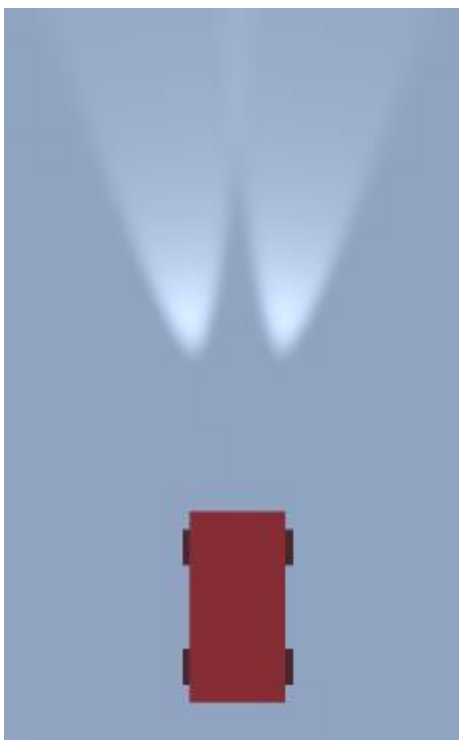
*Rysunek 3.6 Zrzut ekranu pokazujące przykładowy kawałek miasta..*



Rysunek 3.7 Zrzut ekranu pokazujące przykładowy kawałek miasta podczas symulacji..

## Pojazdy

Ponieważ widok jest ustawiony od góry, jednym z najważniejszych aspektów grafiki pojazdów (rys. 3.8) jest kolor, który można łatwo zmienić w zależności od potrzeb. Dzięki temu można łatwo odróżnić różne pojazdy od siebie, co jest istotne dla przejrzystości wizualnej. Dodatkowo, światła pojazdów odgrywają kluczową rolę w poprawie widoczności. Światła nie tylko zwiększają realizm sceny, ale także pomagają użytkownikom lepiej śledzić ruch pojazdów i zrozumieć dynamikę ruchu miejskiego w aplikacji.



*Rysunek 3.8 Zrzut ekranu pokazujący graficzną interpretację pojazdu.*

## Ulice

System graficzny dróg stanowi wyjątek od tej reguły. Możliwe są jedynie zmiany parametrów materiału, takich jak czysty kolor i odbijanie światła. Nie zdążyłem jednak dodać działającej tekstury do dróg, dlatego, ani pasy, ani strzałki nie są wyświetlane.





*Rysunek 3.9 Zrzut ekranu pokazujący graficzną interpretację drogi przed zastosowaniem wykręcania..*



*Rysunek 3.10 Zrzut ekranu pokazujący graficzną interpretację drogi po zastosowaniu wykręcania..*

## Rozdział 4 Implementacja kodu

### Środowisko Unity

Mechaniki zostały zrealizowane w Unity 2022.3.3f1, **wykorzystując** wbudowany silnik fizyczny dla ruchu pojazdów. Modele zostały stworzone w Unity, a skrypty napisane w C# za pomocą Visual Studio 2022.

### Własne skrypty

#### Punkty docelowe

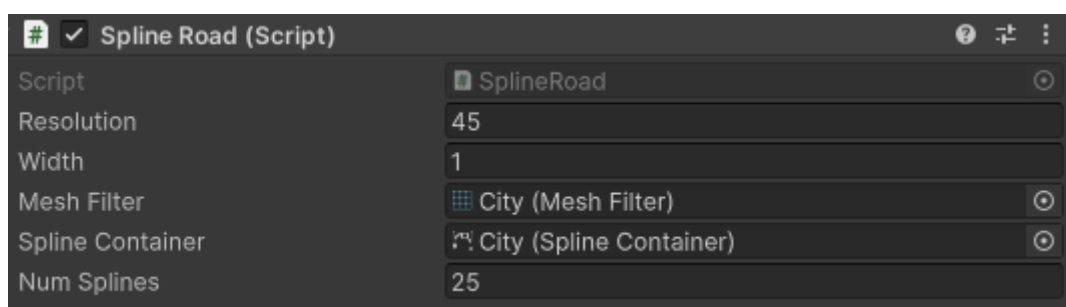
Każdy punkt trasy jest opatrzony informacjami na temat docelowego miejsca, do którego prowadzi. Dodatkowo, użytkownik ma możliwość decydowania o tym, czy w danym punkcie mogą pojawiać się pojazdy, oraz określenia interwału czasowego, z jakim pojazdy będą pojawiać się w danym miejscu. Te informacje są kluczowe dla kontrolowania dynamicznego zachowania pojazdów w symulacji oraz dla tworzenia różnorodnych scenariuszy ruchu.

#### Światła

#### Miasto

#### Pojazdy

#### Ulice



Rysunek 4.4 Zrzut ekranu przedstawiający komponent pozwalający na dynamiczne generowanie dróg.

## Rozdział 5 Dalszy rozwój

### Optymalizacja

Dobra optymalizacja jest kluczowa w każdym projekcie, ponieważ pozwala na zwiększenie skali bez spadku wydajności. Polega ona na dostosowaniu i usprawnieniu działania systemu, aby mógł obsługiwać większe obciążenia przy zachowaniu płynności i efektywności.

W kontekście obecnego rozwiązania, problemem jest ograniczona zdolność obsługi dużej liczby pojazdów. Przyczyną tego jest sposób, w jaki silnik fizyczny przetwarza dane. Dla każdego pojazdu silnik fizyczny wykonuje zbyt wiele obliczeń, co prowadzi do znacznego obciążenia urządzenia. Każde dodatkowe obliczenie zwiększa zapotrzebowanie na zasoby, co w rezultacie spowalnia działanie całego systemu. W skrajnym przypadku może to prowadzić do spadku wydajności, co jest szczególnie problematyczne w aplikacjach wymagających płynnego działania, takich jak symulacje.

Aby rozwiązać ten problem, należy zidentyfikować i zoptymalizować te aspekty silnika fizycznego, które generują największe obciążenie. Możliwe podejścia obejmują:

1. **Redukcję złożoności obliczeń:** Uproszczenie modelu fizycznego dla pojazdów, na przykład poprzez zmniejszenie liczby zmiennych lub parametrów, które są monitorowane i przetwarzane.
2. **Korzystanie z poziomów szczegółowości (LOD):** Implementacja technik, które zmniejszają szczegółowość obliczeń dla pojazdów, które są dalej od kamery lub mniej istotne w danym momencie.
3. **Optymalizacja algorytmów:** Poprawa efektywności algorytmów fizycznych, aby mogły wykonywać te same zadania przy mniejszym zużyciu zasobów.
4. **Wykorzystanie równoległego przetwarzania:** Wykorzystanie technologii wielordzeniowych procesorów do rozdzielenia obciążeń obliczeniowych między różne rdzenie, co może znacząco poprawić wydajność.

5. **Inteligentne zarządzanie zasobami:** Dynamiczne alokowanie zasobów do najbardziej potrzebujących elementów systemu, co może pomóc w utrzymaniu płynnej pracy nawet przy dużym obciążeniu.

Wdrożenie tych i innych technik optymalizacyjnych pozwoli na znaczące zwiększenie liczby obsługiwanych pojazdów bez spadku wydajności, co przełoży się na lepsze doświadczenie użytkowników i bardziej efektywne wykorzystanie zasobów systemowych.

## Poprawa algorytmu ruchu

Obecny algorytm ruchu pojazdów w naszej symulacji nie jest idealny. Auta nie biorą pod uwagę przeszkód innych niż te, które znajdują się bezpośrednio przed nimi. Powoduje to problemy z poruszaniem się na zakrętach, rondach i innych złożonych elementach infrastruktury drogowej.

- **Ograniczone wykrywanie przeszkód:** Obecny system używa pojedynczego promienia (raycast) wysyłanego do przodu od pojazdu, aby wykrywać przeszkody. Taki sposób wykrywania nie sprawdza obszarów po bokach lub za pojazdem, co prowadzi do problemów na bardziej skomplikowanych trasach.

### Możliwe rozwiązania



1. **Zwiększenie liczby promieni (raycastów):**
  - **Opis:** Zamiast jednego promienia, użycie kilku promieni rozchodzących się w różnych kierunkach (np. do przodu, na boki, pod kątem).
  - **Zalety:** Lepsze wykrywanie przeszkód wokół pojazdu, co pozwoli na bardziej płynne poruszanie się, szczególnie na zakrętach i rondach.
  - **Wady:** Zwiększenie liczby promieni znacząco zwiększy obciążenie obliczeniowe silnika fizycznego. Może to prowadzić do spadku płynności symulacji, szczególnie przy dużej liczbie pojazdów.
2. **Zastąpienie promieni overlapem:**
  - **Opis:** Zamiast promieni, użycie metody overlap, wbudowanej w silnik fizyczny Unity. Metoda ta tworzy niewidzialne pudełko wokół pojazdu, które sprawdza kolizje w określonym obszarze.

- **Zalety:** Overlap może sprawdzać większy obszar nie ograniczając się tylko do jednego kierunku. Pozwala to na lepsze wykrywanie przeszkód, co może poprawić nawigację pojazdów w złożonych sytuacjach.
- **Wady:** Chociaż overlap zmniejsza liczbę pojedynczych promieni, nadal może wymagać znacznych zasobów obliczeniowych, zwłaszcza jeśli wiele pojazdów jest w ruchu jednocześnie. Dodatkowo, implementacja overlapu może wymagać dostosowania algorytmów ruchu, aby poprawnie interpretować dane z wykrytych kolizji.

Dodatkowo, należy zauważyć, że obecny algorytm ruchu pojazdów w naszej symulacji nie uwzględnia nachylenia terenu. Brak tego elementu może prowadzić do niewłaściwego zachowania się pojazdów, szczególnie na stromych drogach lub podjazdach, gdzie prędkość i manewrowanie mogą być ograniczone ze względu na ukształtowanie terenu.

Możliwe rozwiązania:

#### 1. Uwzględnienie stromości terenu:

- Zmodyfikowanie algorytmu ruchu pojazdów w taki sposób, aby uwzględniał stromość terenu przy określaniu prędkości i manewrowaniu pojazdów. W ten sposób można zapobiec sytuacjom, w których pojazdy próbują poruszać się po zbyt stromych drogach.

#### 2. Dopasowanie prędkości do nachylenia terenu:

- Możliwe jest dostosowanie prędkości pojazdów w zależności od nachylenia terenu. Na stromych drogach prędkość może być ograniczona, aby zapewnić bezpieczne poruszanie się pojazdów i uniknięcie utraty kontroli.

Obecny algorytm ruchu pojazdów można znacznie ulepszyć, zmieniając metodę wykrywania przeszkód. Zamiast pojedynczego promienia, można użyć kilku promieni lub zastosować metodę overlap, aby dokładniej i efektywniej wykrywać przeszkody. Chociaż każda z tych metod ma swoje wady i zalety, implementacja overlapu wydaje się być bardziej obiecująca pod względem dokładności i elastyczności. Wymaga to jednak odpowiednich optymalizacji, aby nie obciążać nadmiernie silnika fizycznego i zapewnić płynność symulacji.

Dodatkowo uwzględnienie stromości terenu w algorytmie ruchu pojazdów może znacząco poprawić realizm symulacji i umożliwić bardziej dokładne odwzorowanie

rzeczywistych warunków drogowych. Jednakże, konieczne jest przeprowadzenie odpowiednich testów i optymalizacji, aby zapewnić, że dodanie tego elementu nie wpłynie negatywnie na wydajność i płynność symulacji.

## Edytor ustawień miasta i pojazdów

Obecna opcja zmieniania wartości ustawień miasta i pojazdów jest wbudowana bezpośrednio w edytor Unity, co oznacza, że aby wprowadzić jakiekolwiek zmiany, konieczny jest dostęp do całego projektu. Taki sposób pracy ma kilka istotnych wad:

1. **Problem z powtarzalnością testów:** Każda zmiana ustawień wymaga ręcznego dostępu do edytora Unity, co utrudnia przeprowadzanie spójnych i powtarzalnych testów. Zmiany dokonywane ręcznie mogą być niedokładne lub różnić się między testami, co wpływa na wyniki i ich wiarygodność.
2. **Trudność obsługi:** Wymóg korzystania z edytora Unity do zmiany ustawień sprawia, że proces jest bardziej skomplikowany i mniej dostępny dla użytkowników, którzy nie są zaznajomieni z edytorem Unity. Ponadto, każda zmiana wymaga ponownego uruchomienia edytora, co jest czasochłonne i nieefektywne.
3. **Brak dynamicznego zarządzania:** Zmiany dokonywane w edytorze nie mogą być wprowadzane w czasie rzeczywistym podczas działania symulacji, co ogranicza możliwość dynamicznego testowania różnych scenariuszy i dostosowywania parametrów na bieżąco.

Aby rozwiązać te problemy, można wprowadzić dodatkowy interfejs użytkownika (UI) w projekcie, który umożliwiłby:

1. **Resetowanie całej sceny:** Interfejs powinien pozwalać na szybkie resetowanie sceny do stanu początkowego bez konieczności ponownego uruchamiania edytora Unity. To usprawni proces testowania i pozwoli na łatwe przeprowadzanie wielokrotnych testów w kontrolowanych warunkach.
2. **Zmiana ustawień między symulacjami:** Interfejs powinien umożliwiać użytkownikom łatwe wprowadzanie zmian w ustawieniach miasta i pojazdów bezpośrednio z poziomu aplikacji. Możliwość dynamicznego dostosowywania

parametrów pozwoli na bardziej efektywne testowanie różnych scenariuszy i szybkie dostosowywanie symulacji do zmieniających się potrzeb.

3. **Intuicyjność i dostępność:** Nowy interfejs użytkownika powinien być intuicyjny i łatwy w obsłudze, nawet dla osób nieznających się na Unity. Przyjazny dla użytkownika design sprawi, że zmiany ustawień będą mogły być dokonywane szybko i bez błędów.

Wprowadzenie takiego interfejsu użytkownika znacząco poprawi efektywność procesu testowania, ułatwi obsługę i umożliwi dynamiczne zarządzanie ustawieniami symulacji, co jest kluczowe dla rozwoju i optymalizacji projektu.

## Edytor dróg

Głównym problemem tworzenia siatki infrastruktury w projekcie jest jej czytelność. Za każdym razem jak jest dodany punkt trzeba wczytać wszystkie punkty docelowe od nowa. Dodatkowo resetuje to nadane przez użytkownika kształty dróg (krzywe Beziera), co znacznie wydłuża i utrudnia rozbudowę istniejącej części.

Przykładowa symulacja

TODO



## Rozdział 6 Zakończenie

TODO

## Rozdział 7 Bibliografia

[https://www.youtube.com/watch?v=ZiHH\\_BvjoGk&t=850s](https://www.youtube.com/watch?v=ZiHH_BvjoGk&t=850s)

<https://docs.unity3d.com/ScriptReference/Physics.OverlapBox.html>

TODO