

Artificial City

Symulacja ruchu miejskiego na skrzyżowaniu Alei Trzech Wieszczów z ul. Karmelicką i Królewską w Krakowie

Jakub Popielarz, Katarzyna Pyrczak, Kinga Florek

Projekt stworzony w ramach przedmiotu
Symulacja Dyskretna Systemów Złożonych

Spis treści

1	Opis problemu	2
2	Przegląd literatury i istniejących projektów	2
3	Propozycja rozwiązania problemu	4
4	Proponowany przez nas model symulacji	5
5	Implementacja	5
5.1	Symulacja samochodów	6
5.1.1	Pas jezdni	6
5.2	Symulacja tramwajów	7
5.3	Symulacja pieszych	7
5.3.1	Przejście dla pieszych	8
5.4	Interakcja z użytkownikiem	8
5.5	Wizualizacja	9
6	Wyniki symulacji testowych	11
6.1	Parametry domyśle - sytuacja najbardziej zbliżona do rzeczywistej na drodze	12
6.2	Zmniejszenie prawdopodobieństwa zwalniania samochodów	12
6.3	Zwiększenie prawdopodobieństwa zwalniania samochodów	12
6.4	Wydłużenie zielonego światła na Alejach	13
6.5	Wydłużenie zielonego światła na ul. Karmelickiej i Królewskiej	13
7	Statystyki i wnioski	13
8	Podsumowanie	15

1 Opis problemu

W naszym projekcie przedstawimy model skrzyżowania Alei Trzech Wieszców z ul. Kar-melicką i Królewską w Krakowie. Przybliżona wielkość symulowanego przez nas obszaru jest widoczna na poniższym zdjęciu skrzyżowania (100m x 100m).



Celem naszego projektu jest symulacja ruchu miejskiego z uwzględnieniem ruchu samo-chodów osobowych, tramwajów i pieszych wraz z symulacją sygnalizacji świetlnej.

2 Przegląd literatury i istniejących projektów

W tym rozdziale przedstawimy kilka najważniejszych fragmentów z prac, projektów czy artykułów, które podejmują podobny do naszego problem. Opracowując nasze własne al-gorytmy oraz przygotowując się do tworzenia modelu czerpaliśmy wiedzę oraz inspirację z poniższych rozwiązań.

Traffic Simulation Framework — a Cellular Automaton-Based Tool for Simulating and Investigating Real City Traffic [1]

“In the paper we present the Traffic Simulation Framework (TSF) which is an information system for simulating a vehicular traffic in the real city road networks. The system is equipped with a comprehensive visualization which enables users to set and modify simulation parameters and road network preferences. It also supports gathering detailed data related with the simulated traffic. “

“The TSF model is a probabilistic cellular automaton which inherits some foundations from well-known Nagel-Schreckenberg model (...) it is much more elaborated than the standard NaSch model, since its intention is to emulate the real road traffic in big agglomerations.”

Algorithms for the Traffic Light Setting Problem on the Graph Model [2]

“In this paper, we use a graph model to represent the traffic network.”

Section 2 - The Traffic Graph Model and Its Properties

“(...) we can transform the traffic network of a city into a traffic graph model. Here we use a graph $G = (V, E)$ to represent the traffic network of a city, where each vertex $v \in V$ denotes the intersection of some roads and each edge $(u, v) \in E$ denotes the road segment between two vertices u and v . Since each intersection $v \in V$ is controlled by a traffic light, we use $t(v)$ to denote the starting time of green light on v . That is, the interval of green light on v can be written as $[t(v), (t(v) + T/21) \bmod T]$.”

Two Lane Traffic Simulations using Cellular Automata [3]

“We examine a simple two lane cellular automaton based upon the single lane CA introduced by Nagel and Schreckenberg. We point out important parameters defining the shape of the fundamental diagram. Moreover we investigate the importance of stochastic elements with respect to real life traffic.”

“We have presented straightforward extensions of the cellular automata approach to traffic flow so that it includes two-lane traffic. The basic scheme introduced here is fairly general, essentially consisting of two rules: Look ahead in your own lane for obstructions, and look in the other lane if there is enough space. The flow-density relations of several realizations of this scheme have been investigated in detail; possible artifacts for certain parameter choices have been pointed out.”

Nagel-Schreckenberg simulation (lane change model) [4]

Traffic simulation based on Nagel-Schreckenberg model with:

- n lanes,
- traffic lights,
- speed limits,
- obstacles.

“In classic NS model cars move around in circle, so there is the same number of cars all the time. In this model they are generated in each iteration and once they reach the end of the road they are destroyed.”

Nagel-Schreckenberg Model of Traffic – Study of Diversity of Car Rules [5]

“ The probabilistic cellular automata model of traffic represents a lane as one-dimensional lattice of L cells. Each cell is either occupied by one of N cars or is empty. A car can move with the velocity determined by integer values bounded by a speed limit V_{max} . At time t a car is identified by a cell number. At time t a car is identified by a cell number $x_i^{(t)}$ occupied by the car and its velocity $v_i^{(t)}$.”

At each discrete time step $t \rightarrow t + 1$ the positions and velocities of all cars are synchronously updated."

Definition of a Cellular Automaton [?]

“A cellular automaton is a model of a system of “cell” objects with characteristics such as **a grid, a state and the neighborhood** of each cell”

“We’re not really talking about real-world time here, but about the CA living over a period of time, which could also be called a generation and, in our case, will likely refer to the frame count of an animation.”

“The formula for calculating CELL’s state at any given time t is as follows:
CELL state at time $t = f(\text{CELL neighborhood at time } t - 1)$ ”

3 Propozycja rozwiązania problemu

Problem ten może zostać rozwiązany poprzez zastosowanie automatów komórkowych, a dokładniej modelu Nagela-Schreckenbergera (NaSch). Zazwyczaj ten model wykorzystywany jest do symulacji ruchu na autostradach, jednak nie stoi na przeszkodzie aby po odpowiednim dostosowaniu użyć go do symulacji ruchu miejskiego.

W metodzie modelowania, która bazuje na automatach komórkowych, obszar jest dzielony na komórki o takiej samej wielkości, a obiekty posiadają określony zestaw parametrów. Kolejnym fundamentem tej metody jest dyskretyzacja czasu. W każdym kroku czasowym pojazd przemieszcza się zgodnie z pewnymi regułami. Każdy obiekt(pojazd, tramwaj, pieszy) posiada prędkość v_i , która musi być większa od 0 i jednocześnie nie może przekraczać ustalonej dla danego obiektu prędkości maksymalnej.

Prędkość w takim modelu określa ilość komórek przebytą w trakcie jednego kroku czasowego, a zachowanie obiektu zależy od parametrów deterministycznych, losowych oraz od otoczenia komórki, w której w danej chwili znajduje się pojazd.

Reguła automatu komórkowego Nagela-Schreckenberga przewiduje cztery etapy ruchu obiektu w jednym kroku czasowym:

1. Przyspieszenie pojazdu do prędkości nie większej niż V_{max} ,

2. sprawdzenie czy nie dojdzie do kolizji z pojazdem jadącym przed nim i gdy jest taka konieczność, zmniejszenie prędkości, tak aby cały czas utrzymywać bezpieczny dystans,
3. Zmniejszenie prędkości pojazdu zgodnie z określonym prawdopodobieństwem.
4. Przesunięcie pojazdu o określoną liczbę komórek wynikająca z jego prędkości.

Pierwszy etap jest deterministyczny, gdyż zależy tylko od parametrów początkowych. Drugi etap, który wprowadza randomizację, warunkuje w symulacji element realistyczny. Pozwala odwzorować losowe zachowania kierujących oraz interakcje między pojazdami.

W naszej symulacji został zmodyfikowany etap 1, gdyż to, czy pojazd przyspieszy, czy zahamuje, zależy również od tego, jakie światło jest ustawione na jego pasie. Modyfikacji podległa również prędkość pojazdów, która w naszej symulacji, w celu jej płynniejszego działania, niekoniecznie jest liczbą całkowitą.

4 Proponowany przez nas model symulacji

Na potrzeby symulacji i wizualizacji postanowiliśmy wprowadzić następujące uproszczenia i założenia:

- nie uwzględniamy ruchu autobusów miejskich i autobusów prywatnych przewoźników,
- na obszarze przez nas symulowanym kierowcy nie mają już możliwości zmiany pasów (zbyt blisko skrzyżowania),
- ulica Karmelicka i Królewska przecinają Aleje pod kątem prostym
- obszar symulacji nie obejmuje miejsc, w których znajdują się przystanki tramwajowe (pragnęliśmy zachować rzeczywiste odległości, mimo nie zachowania kąta skrzyżowania)

5 Implementacja

Do implementacji algorytmów modelujących ruch samochodów, pieszych i tramwajów oraz sygnalizacji świetlnej postanowiliśmy wykorzystać język Python. Do wizualizacji symulacji użyliśmy biblioteki Pygame [6]. Postanowiliśmy tak dlatego, że Python pozwala w stosunkowo łatwy sposób wizualizować dane, tworzyć GUI oraz - co najważniejsze - wykonywać obliczenia liczbowo lub symbolicznie.

Implementacja opiera się o model Nagela-Schreckenberga (dalej NaSch). W miarę rozwijania projektu odeszliśmy jednak od ścisłej implementacji automatów komórkowych. Miało to na celu poprawę jakości strony wizualnej oraz dokładniejsze odwzorowanie rzeczywistości. W związku z tym, mimo że pierwotnie komórki miały umowny rozmiar $1 \times 1m$, odległości, prędkości oraz zmiany prędkości mogą mieć wartości nie będące liczbami całkowitymi.

5.1 Symulacja samochodów

Utrzymane zostały fazy poszczególnych kroków modelu NaSch symulacji dla każdego samochodu:

1. Zwiększenie prędkości samochodu
2. Sprawdzenie, czy taka prędkość będzie bezpieczna - czy nie dojdzie do zderzenia
3. Jeśli nie - zmniejszenie prędkości do bezpiecznej wartości
4. Losowe rozstrzygnięcie, czy samochód zmniejszy prędkość
5. Poruszenie z tak ustaloną prędkością

By bardziej zwiększyć płynność oraz dokładność symulacji zaimplementowane zostało dynamiczne określanie bezpiecznego dystansu. Osiągane jest to przez pomnożenie obecnej wartości prędkości samochodu przez określony eksperymentalnie współczynnik. Jednak żeby samochody nie zbliżały się do siebie zbyt bardzo bezpieczna odległość jest równa co najmniej jeden metr.

Każdy obiekt samochodu posiada m.in. następujące pola:

```
self.size # słownik o kluczach: "length", "width"- rozmiar pojazdu
self.max_velocity # liczba - maksymalna predkosc pojazdu
self.velocity # liczba - obecna predkosc
self.velocity_change # liczba - zmiana predkosci przy przyspieszaniu i
                    # zwalnianiu
self.slowdown_probability # prawdopodobienstwo losowego spowolnienia
self.travelled # odleglosc przebyta od poczatku pasa
self.safe_distance # odleglosc od obiektu przed uznawana za bezpieczna
self.safe_dist_coeff # wspomniany wyzej wspolczynnik
```

Jednak prawdopodobnie najistotniejszym dla realizmu symulacji elementem klasy `Vehicle` (implementującej samochód) jest metoda która odpowiednio zmniejsza prędkość samochodu. Jej argumentem jest obiekt samochodu bezpośrednio przed danym:

```
def keep_safe(self, vehicle_ahead):
    '''Slow down to not crash into other vehicle'''
    ahead_travelled = vehicle_ahead.travelled - vehicle_ahead.size["length"]

    ahead_travelled -= self.safe_distance
    if self.travelled > ahead_travelled:
        diff = self.travelled - ahead_travelled
        new_velocity = self.velocity - diff
        self.velocity = max(0, new_velocity)
```

5.1.1 Pas jezdni

Za odpowiednie zaktualizowanie świateł samochodów odpowiada klasa `Lane`. To właśnie w niej w odpowiedniej kolejności (zgodnej z modelem NaSch) wywoływane są metody z klasy `Vehicle`.

Każdy pas na symulacji jest oddzielnym obiektem `Lane`, zawierającym tablicę samochodów znajdujących się na nim. Nowe samochody są tworzone poza ekranem - pod warunkiem że najbliższy samochód na pasie jest odpowiednio daleko losowana jest liczba z przedziału $<0, 1>$. Jeśli jest ona mniejsza niż określone prawdopodobieństwo stworzenia

samochodu - inicjalizowany jest nowy obiekt `Vehicle`, który następnie zostaje dodany do tablicy pojazdów.

Klasa `Lane` koordynuje również zmiany świateł drogowych w odpowiednich interwałach czasowych. Ze względu na zaimplementowane wcześniej mechanizmy unikania kolizji między samochodami 'zaświecenie' światła czerwonego realizowane jest przez odpowiednie stworzenie samochodu o ujemnej prędkości maksymalnej oraz ujemnej zmianie prędkości. W połączeniu z tym, że samochód nie może mieć prędkości mniejszej niż 0 skutkuje to powstaniem samochodu, który się nie przemieszcza. W takim przypadku wszystkie kolejne pojazdy - zbliżające się do niego zatrzymują się, by uniknąć kolizji. Zmiana światła na zielone oznacza po prostu usunięcie tego pojazdu z pasa. Oczywiście takie pojazdy byłyby widoczne na ekranie, jednak nie są one wyświetlane.

5.2 Symulacja tramwajów

Tramwaje zostały zaimplementowane z użyciem opisanej wyżej klasy `Vehicle`. Tramwaje są inicjalizowane z innymi parametrami, bardziej odpowiadającymi tym pojazdom - rzadziej zwalniają, są dłuższe, wolniejsze oraz wolniej przyspieszają. Jediną różnicą względem mechanizmów kontrolujących samochody stanowi tak na prawdę mechanizm tworzenia nowych tramwajów - w przeciwieństwie do samochodów (i krakowskiego MPK) nasze tramwaje poruszają się zgodnie z rozkładem. Ze względu na małą ilość różnic przemieszczają się one po pasach zaimplementowanych klasą `TramLane`, dziedziczącą po klasie `Lane`. Klasa `TramLane` przesłania odpowiednie funkcje odpowiedzialne za tworzenie nowych pojazdów. Zamiast polegać na losowości odliczana jest po prostu odpowiednia, zdefiniowana przez nas, ilość czasu.

5.3 Symulacja pieszych

Ruch pieszych również opiera się na wszystkich etapach modelu Nagela-Schreckenberga.

Obiekt klasy `Pedestrian` posiada takie pola jak:

```
self.length # dlugosc pieszego
self.width  # szerokosc pieszego
self.vMax   # predkosc maksymalna
self.velocity # obecna predkosc
self.vChange # zmiana predkosci
self.travelled # odleglosc przebyta od poczatku pasa
self.safeDistance # odleglosc od poprzedniego pieszego
```

Dla pieszych również ważne jest, aby utrzymywali odpowiedni dystans. To zapewnia funkcja `keepSafeVelocity`.

```
def keepSafeVelocity(self, prevPedestrian):
    dist = self.distance(prevPedestrian)
    if dist < self.safeDistance + self.velocity:
        self.velocity = dist

def distance(self, prevPedestrian):
    """ zwraca dystans miedzy obecnym a poprzednim pieszym """
    return prevPedestrian.travelled - self.travelled
```

5.3.1 Przejście dla pieszych

Przejście dla pieszych to w naszej implementacji dwa pasy, po których piesi poruszają się w przeciwnych kierunkach. Każdy pas to obiekt klasy `Path`. Posiada on tablicę pieszych, na których są wywoływane metody klasy `Pedestrian`. Nowi piesi powstają przed przejściem dla pieszych i są dodawani do tablicy pieszych zgodnie z ustawionym parametrem prawdopodobieństwa. Usuwani są z niej wtedy, gdy wartość zmiennej *travelled* pieszego przekroczy wartość *length* pasa, po którym się porusza.

Nie udało się zrealizować synchronizacji między pieszymi a resztą pojazdów, dlatego wnioski symulacji opierają się tylko na samochodach oraz tramwajach.

5.4 Interakcja z użytkownikiem

Do symulacji postanowiliśmy dodać kilka przycisków pozwalających użytkownikowi:

- zatrzymać lub uruchomić symulację
- zwiększyć lub zmniejszyć prędkość symulacji
- zwiększyć lub zmniejszyć długość trwania światła zielonego (na pasach pionowych lub poziomych) - wydłużenie go na pasach poziomych powoduje wydłużenie czasu trwania światła czerwonego na pasach pionowych i na odwrót
- zwiększyć lub zmniejszyć prawdopodobieństwo zwalniania pojazdów

Oprócz nich wprowadziliśmy podsumowanie końcowe symulacji zawierające informacje o:

- czasie trwania symulacji
- ilości samochodów które przejechały przez skrzyżowanie
- średniej ilości samochodów przejeżdżających skrzyżowanie na sekundę
- analogicznych wartościach dla tramwajów

Dzięki temu możemy sprawdzić jaką przepustowość ma skrzyżowanie, przy różnych parametrach początkowych.

5.5 Wizualizacja

Tak jak zostało wspomniane wcześniej, do wizualizacji użyliśmy biblioteki pygame. Każdy element wizualizacji tzn. jezdnie, światła, linie drogowe, przyciski oraz piesi stworzone są za pomocą prostokątów. Jedynie aby poprawić jakość wizualizacji do przedstawienia samochodów oraz tramwajów użyliśmy gotowych ikon. Wizualizacja zachowuje proporcje skrzyżowania i pojazdów.

Główne pola i metody klasy do tworzenia wizualizacji naszej symulacji:

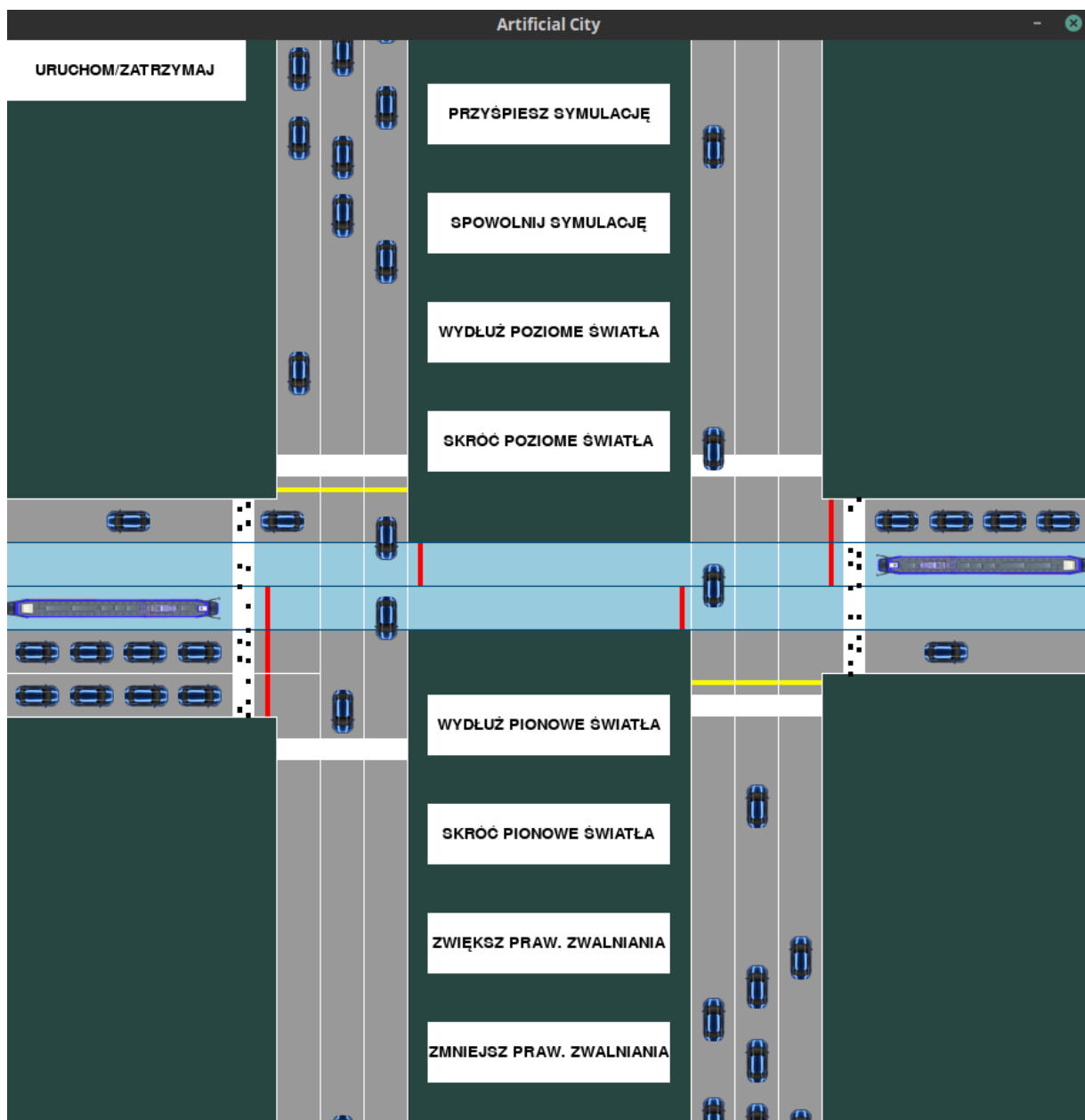
```
class Visualisation():
    def __init__(self, win, lane, cell_size, clc, zlc, tlc):
        self.win = win # okno programu
        self.width, self.height = win.get_width(), win.get_height() #
                                                                    wielkosc okna programu
        self.lane = lane # szerokosc pasa ruchu
        self.px = cell_size # ilosc px ktore odzwierciedlaja 1m w
                                                                    rzeczywistosci
        self.carlane_c = clc # wspolrzedne pasow samochodowych
        self.zebralane_c = zlc # wspolrzedne przejsc dla pieszych
        self.tramlane_c = tlc # wspolrzedne torow tramwajowych

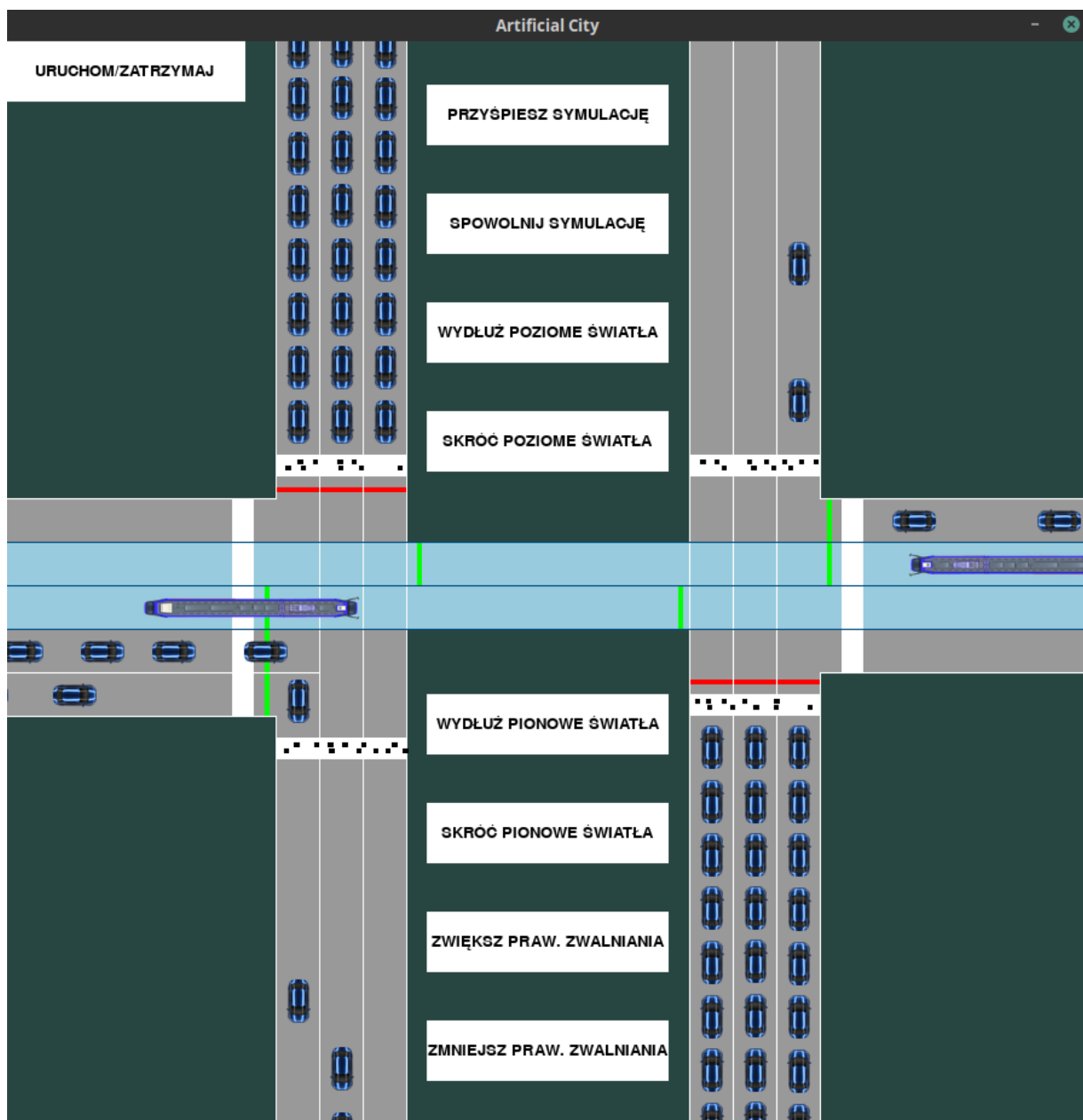
    def draw(self, CAR_LANES, PEDESTRIAN_LANES, TRAM_LANES):
        self.win.fill(darkgreen)
        self.__drawIntersection()
        self.drawLights(CAR_LANES, TRAM_LANES)
        self.__drawLines()
        self.drawVehicles(CAR_LANES, PEDESTRIAN_LANES, TRAM_LANES)

    def drawVehicles(self, CAR_LANES, PEDESTRIAN_LANES, TRAM_LANES):
        for key, lane in CAR_LANES.items():
            self.drawCarOnLane(key, lane.vehicles)

        for key, lane in TRAM_LANES.items():
            self.drawTramOnLane(key, lane.vehicles)

        for key, lane in PEDESTRIAN_LANES.items():
            self.drawPedestrianOnLane(key, lane.pedestrians)
```





6 Wyniki symulacji testowych

Postanowiliśmy przeprowadzić kilka testowych symulacji z różnymi parametrami początkowymi, aby sprawdzić jak zmieni się wtedy przepustowość skrzyżowania. Będziemy testować wpływ parametru określającego prawdopodobieństwo zwalniania samochodów oraz wpływ długości zielonego światła na Alejach lub ul. Karmelickiej i Królewskiej na tworzenie się korków i ogólną przepustowość.

6.1 Parametry domyśle - sytuacja najbardziej zbliżona do rzeczywistej na drodze

```
Run: artificial_city x
/home/kinga/PycharmProjects/BITAI/1/venv/bin/python /home/kinga/Uni/semestr4/sdsz/ArtificialCity/artificial_city.py
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
Simulation running: True
Simulation running: False
-----
Simulation duration: 60.000
Number of cars that went through the intersection: 111
Average number of cars exiting the intersection per second: 1.850
Number of trams that went through the intersection: 10
Average number of trams exiting the intersection per second: 0.167
Number of pedestrians that went through the intersection: 649
Average number of pedestrians exiting the intersection per second: 10.817

Process finished with exit code 0
|
```

6.2 Zmniejszenie prawdopodobieństwa zwalniania samochodów

```
Run: artificial_city x
Current car slowdown probability: 0.39
Current car slowdown probability: 0.38
Current car slowdown probability: 0.37
Current car slowdown probability: 0.36
Current car slowdown probability: 0.35
Simulation running: True
Simulation running: False
-----
Simulation duration: 60.000
Number of cars that went through the intersection: 228
Average number of cars exiting the intersection per second: 3.800
Number of trams that went through the intersection: 10
Average number of trams exiting the intersection per second: 0.167
Number of pedestrians that went through the intersection: 612
Average number of pedestrians exiting the intersection per second: 10.200

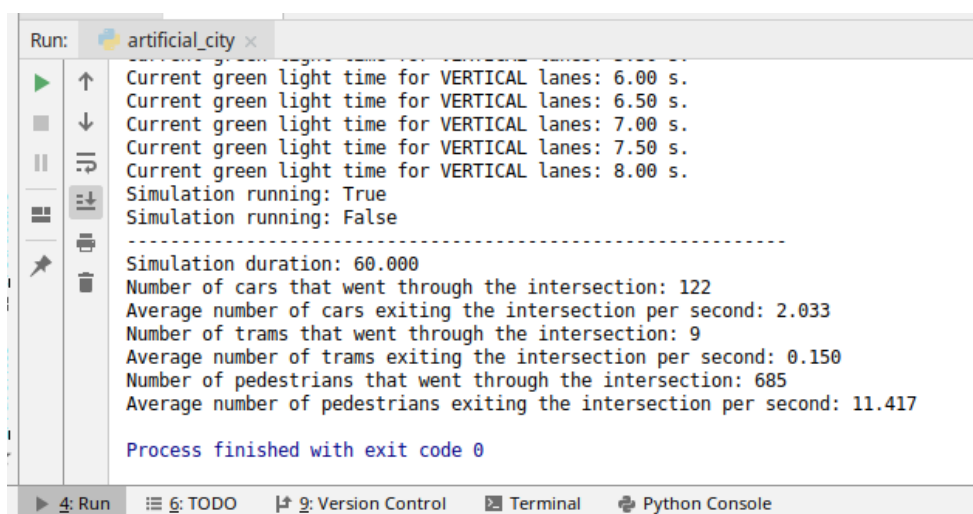
Process finished with exit code 0
```

6.3 Zwiększenie prawdopodobieństwa zwalniania samochodów

```
Run: artificial_city x
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
Current car slowdown probability: 0.46
Current car slowdown probability: 0.47
Current car slowdown probability: 0.48
Simulation running: True
Simulation running: False
-----
Simulation duration: 60.000
Number of cars that went through the intersection: 78
Average number of cars exiting the intersection per second: 1.300
Number of trams that went through the intersection: 9
Average number of trams exiting the intersection per second: 0.150
Number of pedestrians that went through the intersection: 582
Average number of pedestrians exiting the intersection per second: 9.700

Process finished with exit code 0
```

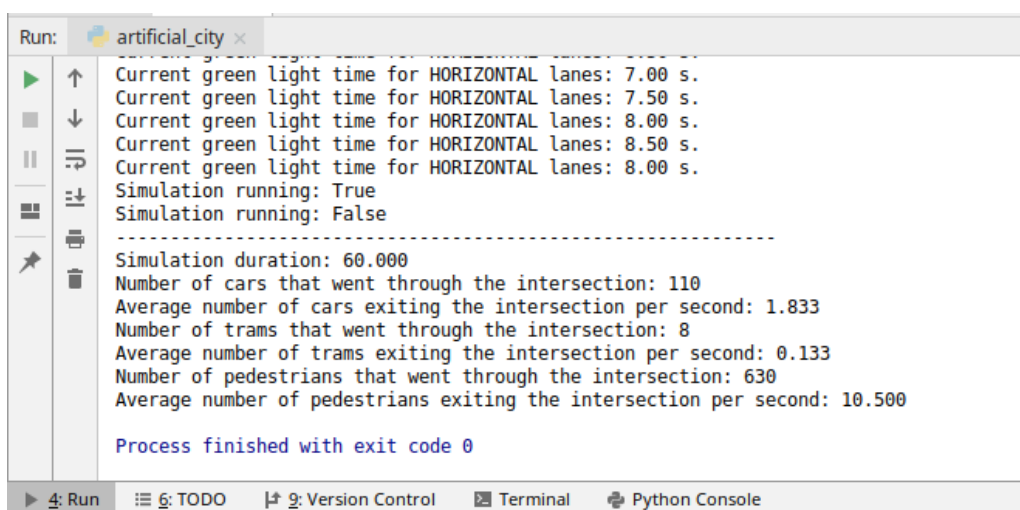
6.4 Wydłużenie zielonego światła na Alejach



```
Run: artificial_city x
Current green light time for VERTICAL lanes: 6.00 s.
Current green light time for VERTICAL lanes: 6.50 s.
Current green light time for VERTICAL lanes: 7.00 s.
Current green light time for VERTICAL lanes: 7.50 s.
Current green light time for VERTICAL lanes: 8.00 s.
Simulation running: True
Simulation running: False
-----
Simulation duration: 60.000
Number of cars that went through the intersection: 122
Average number of cars exiting the intersection per second: 2.033
Number of trams that went through the intersection: 9
Average number of trams exiting the intersection per second: 0.150
Number of pedestrians that went through the intersection: 685
Average number of pedestrians exiting the intersection per second: 11.417

Process finished with exit code 0
```

6.5 Wydłużenie zielonego światła na ul. Karmelickiej i Królewskiej

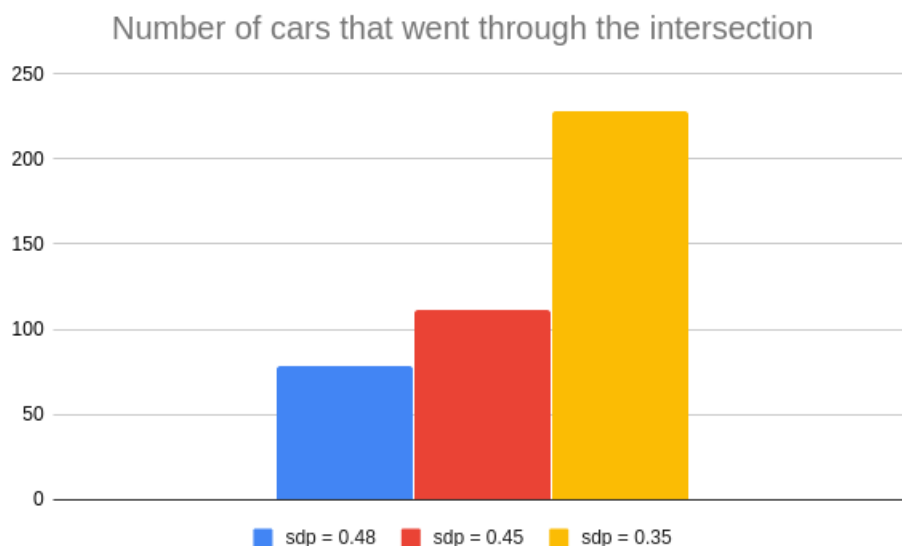


```
Run: artificial_city x
Current green light time for HORIZONTAL lanes: 7.00 s.
Current green light time for HORIZONTAL lanes: 7.50 s.
Current green light time for HORIZONTAL lanes: 8.00 s.
Current green light time for HORIZONTAL lanes: 8.50 s.
Current green light time for HORIZONTAL lanes: 8.00 s.
Simulation running: True
Simulation running: False
-----
Simulation duration: 60.000
Number of cars that went through the intersection: 110
Average number of cars exiting the intersection per second: 1.833
Number of trams that went through the intersection: 8
Average number of trams exiting the intersection per second: 0.133
Number of pedestrians that went through the intersection: 630
Average number of pedestrians exiting the intersection per second: 10.500

Process finished with exit code 0
```

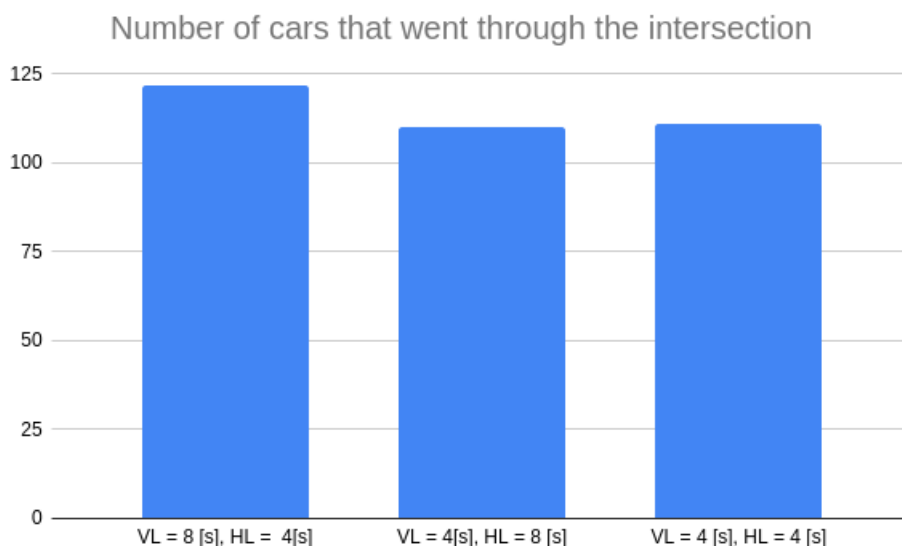
7 Statystyki i wnioski

Jak możemy łatwo zauważyć zmniejszenie prawdopodobieństwa zwalniania pojazdów nawet o małą wartość znacząco wpływa na przepustowość skrzyżowania. Tworzy się wtedy zdecydowanie mniej korków, samochody ruszają płynniej. Jest to dobre odzwierciedlenie rzeczywistej sytuacji, gdzie korki nie tworzą się tylko z oczywistych wzdługów jak np. zwężenia dróg, ale również z powodu powolnej reakcji kierowców lub nieumyślnego zwalniania podczas chociażby zastanawiania się, w którą stronę jechać.



	sdp = 0.48	sdp = 0.45	sdp = 0.35
Simulation duration	60	60	60
Number of cars that went through the intersection	78	111	228
Average number of cars exiting the intersection per second	1.3	1.85	3.8
Number of trams that went through the intersection	9	10	10
Average number of trams exiting the intersection per second	0.15	0.167	0.167
Number of pedestrians that went through the intersection	582	649	612
Average number of pedestrians exiting the intersection per second	9.7	10.817	10.2

Z drugiej strony tylko zwiększając dwukrotnie długość świateł na Alejach udało się nam uzyskać większą przepustowość skrzyżowania. Potwierdza to jedynie nasze przypuszczenia, że panuje tam większy ruch i aby zapobiegać tworzeniu się korków na skrzyżowaniu światła na Alejach powinny być dłuższe.



	VL = 8 [s], HL = 4[s]	VL = 4[s], HL = 8 [s]	VL = 4 [s], HL = 4 [s]
Simulation duration	60	60	60
Number of cars that went through the intersection	122	110	111
Average number of cars exiting the intersection per second	2.033	1.833	1.85
Number of trams that went through the intersection	9	8	10
Average number of trams exiting the intersection per second	0.15	0.133	0.167
Number of pedestrians that went through the intersection	685	630	649
Average number of pedestrians exiting the intersection per second	11.417	10.5	10.817

8 Podsumowanie

W projekcie zaproponowaliśmy zastosowanie automatu komórkowego do symulacji ruchu miejskiego z wprowadzeniem zmiennych parametrów wpływających na sposób poruszania się pojazdów.

Otrzymane wyniki dają odpowiedź, czemu w centrach miast tak często tworzą się korki. Powodem tego są losowe zwalnianie kierowców oraz opóźniony czas reakcji podczas ruszania, co w przypadku wielu samochodów, kumuluje się i przyczynia do powstawania zatorów drogowych.

Literatura

- [1] *Traffic Simulation Framework — a Cellular Automaton-Based Tool for Simulating and Investigating RealCity Traffic*. <http://iis.ipipan.waw.pl/2009/proceedings/iis09-64.pdf>.
- [2] *Algorithms for the Traffic Light Setting Problem on the Graph Model*. <http://par.cse.nsysu.edu.tw/~cbyang/person/publish/mc07traffic.pdf>.
- [3] *Two Lane Traffic Simulations using Cellular Automata*. <https://arxiv.org/pdf/cond-mat/9512119.pdf>.
- [4] *Nagel-Schreckenberg simulation (lane change model)*. <https://github.com/morethanoneanimal/Nagel-Schreckenberg-simulation>.
- [5] *Nagel-Schreckenberg Model of Traffic – Study of Diversity of Car Rules*. <https://link.springer.com/content/pdf/10.1007>
- [6] *Pygame Documentation*. <https://www.pygame.org/docs/>.