Programowanie Równoległe i Rozproszone
Projekt

Radosław Churski
Paweł Kurbiel

4 czerwca 2018

Spis treści

1 Temat projektu

Jako temat projektu wybraliśmy model symulacyjny prostego ruchu drogowego. Można w nim rozróżnić następujące podmioty:

samochód porusza się po drogach i skrzyżowaniach. Jeśli napotka przed sobą skrzyżowanie może na nie wjechać tylko wtedy, kiedy odpowiedni sygnalizator świetlny ma zapalone zielone światło. Samochód zna tylko najbliższe otoczenie.

skrzyżowanie jest wyposażone w dwa sygnalizatory świetlne, po jednym dla każdej z dwóch osi. Z założenia, samochody jadące na przeciw siebie "mają"to samo światło. Skrzyżowanie odpowiada za sterowanie sygnalizacją świetlną.

sygnalizacja świetlna może mieć jeden z dwóch kolorów, zielony lub czerwony.

mapa reprezentuje świat w którym poruszają się samochody i znajdują się drogi oraz skrzyżowania. Z mapy samochody dowiadują się o swoim najbliższym otoczeniu.

2 Realizacja

2.1 Uruchomienie; dane wejściowe

Program przyjmuje trzy argumenty:

- ścieżka do pliku z mapą
- liczba samochodów
- czas pracy programu

2.1.1 Plik z mapą

Można stwierdzić, że plik z mapą jest podzielony na dwie części. Jedną z nich jest pierwszy wiersza zawierający dwie liczby naturalne. Stanowią one informację o rozmiarach mapy.

Drugą częścią jest właściwa mapa. Rozróżnialne są trzy rodzaje znaków składające się na mapę:

- znak drogi (`R`)
- znak skrzyżowania (`C`)
- znak pustego pola (`.`)

Przykładowy plik z mapą:

```
5 10
RRRR......
...R......
RRRCRCRCRR
.....R.R.R
RRRRRR.RRR
```

Poniżej zamieszczony został fragment klasy `mapReader`:

```
private final static char roadChar = 'R';
private final static char crossingChar = 'C';

public static Map readMap(String path) {
    [...]
}
```

Atrybuty `roadChar` oraz `crossingChar` przechowują znaki, które podczas czytania pliku są interpretowane jako odpowiednio drogi i skrzyżowania. Każdy inny napotkany znak (oczywiście, oprócz końca linii) jest interpretowany jako puste pole. Takie rozwiązanie pozostawia możliwość na łatwe rozbudowanie obsługiwanych rodzajów pól mapy.

2.1.2 Liczba samochodów

Argument ten determinuje ile samochodów zostanie "wypuszczonych„ na drogi. Są one generowane w następujący sposób:

```
int numCars = Integer.parseInt(args[1]);
ArrayList<Road> roads = map.getallRoads();
Random random = new Random();

for (int i = 0; i < numCars; i++) {
    cars.add(new Car(
        i,
        roads.get(random.nextInt(roads.size())).getPosition(),
        new Velocity(random.nextInt(maxVelocity - minVelocity) + minVelocity),
        map)
    );
}
```

W pętli `for` dodajemy do listy `cars` kolejne samochody, przekazując do konstruktora kolejne liczby naturalne stanowiące identyfikatory samochodów, pozycję losowanych pól dróg, losowaną prędkość jako obiekt klasy `Velocity` oraz mapę. Klasa `Main` posiada statyczne atrybuty determinujące minimalną i maksymalną prędkość z jaką mają się poruszać samochody.

2.1.3 Czas pracy programu

Ostatni parametrem przyjmowanym przez program jest informacja przez ile sekund ma pracować. Po tym, jak wszystkie potrzebne obiekty zostaną utworzone główny wątek programu jest usypiany na odpowiedni czas, po czym wywołuje metody terminujące na wszystkich obiektach pracujących z własnymi wątkami.

2.2 Implementacja samochodów

2.3 Implementacja skrzyżowań ze światłami

2.4 Wyjście

3 Podsumowanie