Miłosz Ostrowski M.P.Ostrowski@stud.elka.pw.edu.pl
Mikołaj Markiewicz M.Markiewicz@stud.elka.pw.edu.pl

Prowadzący:
Rafał Biedrzycki
rbiedrzy@elka.pw.edu.pl

(Średnio)zaawansowane programowanie w C++ (ZPR2012L)

Dokumentacja końcowa

1) **Temat:**

   Implementacja prostego symulatora ruchu miejskiego.

2) **Opis modelu:**

   Podczas symulacji poruszają się obiekty różnego typu: samochody małe, duże oraz piesi. W różnych miejscach ulic rozmieszczone są "inteligentne" kamery – każda posiada swoje parametry: dokładność, kąt widzenia, kierunek obserwacji. Kamery są próbkowane co 1 sek. Jeżeli coś zauważy – generuje obserwacje i podaje współrzędne zaobserwowanych obiektów zmierzone zgodnie ze swoimi parametrami oraz w celu weryfikacji działania symulatora, rzeczywiste współrzędne wraz z czasem logowania.

3) **Problemy napotkane podczas tworzenia aplikacji:**

   Podczas tworzenia aplikacji, głównym problemem były ograniczenia czasowe. Z tego powodu nie udało nam się zrealizować funkcjonalności dodatkowej projektu. Jednym z problemów który ukazał się pod koniec projektu, gdy symulacja była skończona, był wybór biblioteki do stworzenia prostego GUI w postaci biblioteki Allegro – brak prostej obsługi zdarzeń spowodował konieczność poświęcenia sporej ilości pracy. Biblioteka ta jednakże została wybrana tylko do celów wizualizacyjnych.

4) **Struktura aplikacji:**

   Aplikacja została podzielona na trzy główne części: Model, Widok i Kontroler (MVC).
   Model odpowiada za wszelkie obliczenia związane z przebiegiem symulacji. Kontroler zajmuje się czytaniem danych z plików konfiguracyjnych oraz zarządzaniem odświeżania modelu oraz widoku poprzez kontrolujący to Timer.
   Widok odrysowuje przebieg symulacji w postaci prostych obiektów geometrycznych:
   - Samochody - prostokąty różniące się kolorem w zależności od rozmiaru,
   - Piesi - trójkąty,
   - Kamery - koła i wycinki kół pokazujące ich zasięg.

5) **Zrealizowana funkcjonalność:**

   a) Dane pobierane są z plików XML, których szablony dołączone są do projektu w katalogu *template_xml/* Dane domyślnie szukane są w katalogu *xml_data/*
      - Mapa ulic w postaci węzłów grafu i krawędzi pomiędzy nimi (graf nieskierowany). *[streets.xml]*

- Obiekty w poruszające się w symulacji, ich parametry i trasy. *[objects.xml]*
- Kamery i ich parametry. *[dispatcher.xml]*

b) Samochody poruszają się po gładkich trasach, nie skręcają 'w miejscu'. Przyśpieszają i zwalniają przed zakrętami.
c) Piesi poruszają się po prostych trasach, skręcają 'w miejscu'.
d) Zakręty na trasach samochodów wyznaczane są automatycznie przez wyliczenie odpowiednich krzywych Beziera zależnych wielkością 'promienia' od stałej zdefiniowanej w projekcie.
e) Kamery logują obserwacje co 1 sekundę do pliku o nazwie złożonej z numeru (timestamp) z rozszerzeniem *.txt*, w chwili pierwszego zauważonego obiektu.
f) Wizualizacja graficzna w postaci prostych obiektów geometrycznych oraz proste menu pozwalające wystartować, pauzować, restartować, zapętlić lub zakończyć symulację. Zapętlenie polega na tym, że gdy obiekt dojedzie do końca swojej trasy - zaczyna ją przemierzać od początku.
g) Pliki z danymi domyślnie szukane są w katalogu *xml_data/* o nazwach podanych w (pkt. a). W przypadku braku któregokolwiek z plików, należy wskazać prawidłową, względną do uruchamianej aplikacji, ścieżkę do pliku lub zakończyć działanie programu.
h) Ze względu na ograniczenia czasowe nie zostały zrealizowane funkcje dodatkowe z dokumentacji wstępnej.

6) Konwencja jednostek:

a) Początek układu współrzędnych (0, 0) w lewym górnym rogu
b) Jednostki: 1.0 = 1 metr
c) Widoczny obszar symulacji to 200 x 200 [m]
d) Kąty podawane w stopniach, w aplikacji zamieniane na radiany

7) Parametry obiektów:

a) <u>Kamery:</u>

i) ID - liczba całkowita
ii) Położenie - współrzędne x, y
iii) Kierunek patrzenia - kąt w stopniach, [0=E, 90=S, 180=W, 270=N]
iv) Kąt widzenia - całkowity kąt widzenia w stopniach
v) Zasięg widzenia - liczba metrów
vi) Precyzja - podawana w %

b) <u>Obiekty:</u>

Wspólne parametry:

i) Typ - samochód / pieszy
ii) ID - rejestracja samochodu / nazwa pieszego
iii) Prędkość maksymalna - prędkość maksymalna dla samochodu / prędkość pieszego - obie podawane w [km / h]

Dodatkowe parametry samochodów:

iv) Przyśpieszenie - podawane w [m/s]
v) Masa - podawana w [kg]

vi) Rozmiar - mały / duży

Trasy obiektów:

vii) Lista kolejnych punktów trasy - współrzędne x, y

c) Mapy ulic:

Lista wierzchołków:

i) ID - numer wierzchołka

ii) Położenie - współrzędne x, y

Lista krawędzi:

iii) Łączenie - numery wierzchołków pomiędzy którymi występuje krawędź

8) Wykorzystanie dodatkowych bibliotek:

a) Allegro 5 - wizualizacja symulacji

b) Ważniejsze biblioteki z boost:

i) Boost.Thread - użyta to wielowątkowości aplikacji

ii) Boost.Chrono - użyta do liczenia czasu

iii) Boost.Bind - w celu ograniczenia ilości kodu

iv) Boost.shared_ptr - sprytne wskaźniki

v) Boost.lexical_cast i Boost.format – formatowanie tekstu

vi) Boost.Filesystem - zarządzanie ścieżkami

vii) Boost.PropertyTree – parsowanie plików konfiguracyjnych

c) Kontenery i algorytmy z biblioteki standardowej STL

9) Użyte wzorce projektowe:

Architektura projektowanego symulatora okazała się na tyle prosta, że wystarczyło użycie prostych rozwiązań. Użyte zostały tylko przestawione poniżej wzorce w mniejszym lub większym udziale:

- MVC - rozdzielenie logiki aplikacji na model, widok i kontroler
- Obserwator - timer – zarejestrowani obserwatorzy sprawdzają czas timera w oczekiwaniu na upłynięcie timeoutu.
- Singleton - logger - logowanie do pliku danych pochodzących z symulacji
- Wizytator -model propagacji zdarzeń użytkownika do kontrolera – kontroler jest wizytowany przez wyzwolone zdarzenia.
- Komenda - boost.bind - anonimowe funktory

10) Kompilacja i uruchamianie programu:

a) Windows, Visual Studio 2010: (instrukcja w pkt. 15)
   i) Wymaga linkowania bibliotek boost do projektu
   ii) Wymaga dołączenia biblioteki allegro
   iii) Wskazanie katalogów z plikami nagłówkowymi
b) Unix, g++: (instrukcja instalacji bibliotek w pkt. 14)
   i) Polecenie make - kompilacja programu
   ii) make clean - usunięcie plików pośrednich
   iii) make test - proste testy aplikacji

11) Testy:

Nasz projekt został oparty głównie na wzorcu MVC, w związku z czym większość sterowania przepływa przez kontroler. Jednocześnie znaczna część operacji odbywa się wewnątrz modelu, który jest odizolowany od reszty programu. To powoduje, że pisanie testów jednostkowych czy też modułowych jest bardzo utrudnione. Jednocześnie, program realizuje funkcję symulatora, przez co samo uruchomienie jest poniekąd prowadzeniem testu. Zwłaszcza semantyka testów biblioteki boost::test wydaje się wysoce nieodpowiednia do naszego projektu. Aby jednak umieścić jakiekolwiek testy, pokazać znajomość i chociaż rąbek możliwości bibliotek automatyzujących ten proces, stworzyliśmy drugą ścieżkę przebiegu programu tworzoną poleceniem *make test*(automatyzacja procesu testowania).Polega to na bardzo prostym sprawdzeniu kilku podstawowych funkcji, których używa model – jest to bardziej prezentacja znajomości zagadnienia niż ważna funkcjonalność symulatora.

12) Podsumowanie:

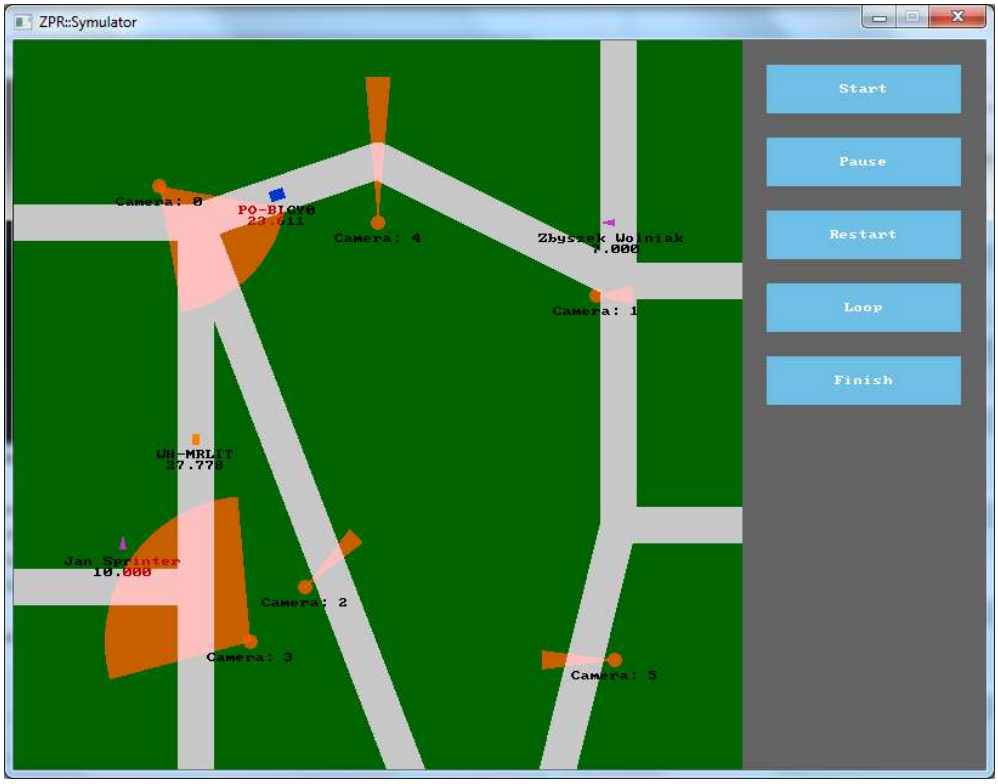
Podczas pracy nad projektem natknęliśmy się na niejeden problem, jednakże dzięki dostępnym rozbudowanym bibliotekom oraz przykładom z wykładów, które były bardzo pomocne, rozwiązania niekiedy stawały się trywialne i praca szybko posuwała się do przodu. Znajomość niektórych wzorców projektowych pomogła na etapie projektowania, aczkolwiek w naszej aplikacji znaleźliśmy zastosowanie tylko dla kilku z wielu. Nie chcieliśmy też na siłę używać wzorców nie pasujących do projektu, gdyż wiele rzeczy rozwiązaliśmy prostymi technikami programowania, nie utrudniając sobietym samym.

Dzięki aktywnej pracy nad projektem w połączeniu z uczestnictwem w wykładach poznaliśmy wiele zaawansowanych technik nowoczesnego C++ (boost::bind), które znacząco ułatwiają prace, czynią kod czytelniejszym i przenoszą logikę aplikacji na wyższy poziom abstrakcji. Mamy nadzieję wykorzystać nabytą wiedzę jak najszybciej w pracy zawodowej.

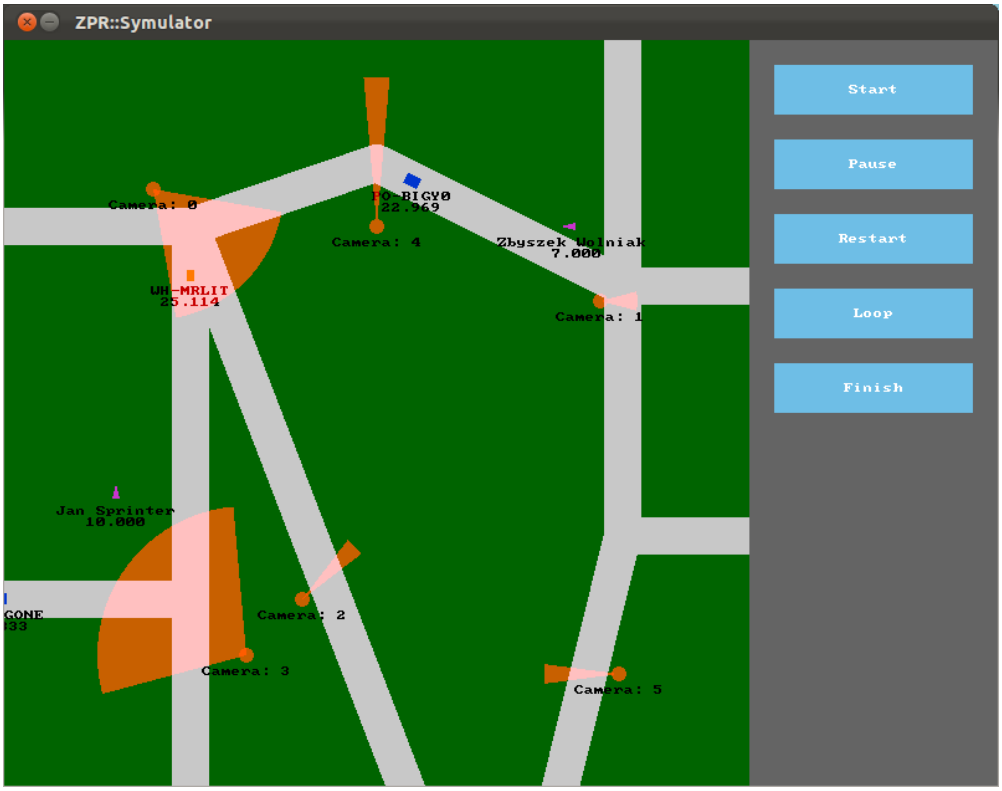
Przygotowana przez nas aplikacja, pełni główną rolę jako symulator. Wizualizacja była dla nas celem drugoplanowym, skupiliśmy się na wierności modelu i spójności wewnętrznej projektu. I choć program mógłby być bardziej przyjazny użytkownikowi, z lepszym interfejsem graficznym, jesteśmy zadowoleni z projektu, gdyż priorytetem dla nas było wnętrze symulacji, a strona graficzna – tylko miłym dodatkiem.

13) Przykładowy screen aplikacji:

Windows:



Unix:



14) Instrukcja instalacji biblioteki allegro i boost pod Ubuntu:

ALLEGRO:

```
sudo apt-get install subversion
sudo apt-get install cmake
sudo apt-get install make

mkdir allegro-svn
cd allegro-svn

svn co https://alleg.svn.sourceforge.net/svnroot/alleg/allegro/branches/5.1

cd 5.1

mkdir Build
cd Build

cmake .. -DGRADE_DEBUG=on -DSHARED=off

make
sudo make install
```

BOOST:

```
Download boost_1_49_0.tar.bz2.
   http://sourceforge.net/projects/boost/files/boost/1.49.0/boost_1_49_0.tar.bz2/download

W katalogu gdzie chcemy wstawić bibliotekę, wykonać: (zazwyaczaj/usr/local/boost_1_49_0)

sudo tar --bzip2 -xf /sciezka/wybrana/wyzej/do/boost_1_49_0.tar.bz2

cd /usr/local/boost_1_49_0
sudo ./bootstrap.sh --prefix=/usr/local/

sudo ./b2 install
```

15) Instrukcja instalacji w środowisku Windows (Visual Studio 2010):

Biblioteka: http://static.allegro.cc/file/library/allegro/5.0.6/allegro-5.0.6-msvc-10.0.zip

Instrukcja instalacji:

http://wiki.allegro.cc/index.php?title=Windows,_Visual_Studio_2010_and_Allegro_5

+ lokalizacja plików nagłówkowych: