Symulacja ruchu pociagów

Prezentację tworzył Michał Pluta

we współpracy z Adam Mościcki, Jan Wiśniewski, Edwin Jarosiński, Paweł Kowalczyk , Margarita Kirillova

24 stycznia 2014

Podsumowanie

1 O projekcie
- Treść

2 User Stories
- Obserwator
- Projektant torów
- Logistyk

3 Dokonania
- Edwin
- Rita
- Paweł
- Adam
- Michał
- Janek
- Razem

4 Koniec

Treść

Należy stworzyć aplikację wizualizującą ruch pociągów. Aplikację należy skonstruować w taki sposób, aby z pliku wczytywany był układ torów z dowolną liczbą skrzyżowań. Układ torów i początkowe rozmieszczenie pociągów powinno być definiowane i wczytywane z pliku. Na skrzyżowaniu domyślnie jest sygnalizacja, ponadto każdy pociąg posiada radar, który pozwala mu określić odległość od innego pociągu jadącego po tym samym torze. Po torowisku może jeździć dowolna liczba pociągów z różnymi prędkościami. Pociągi muszą jeździć tak, aby nie doszło do kolizji. Program musi posiadać moduł sterowania prędkościami pociągów, tak aby nie dochodziło do kolizji! Historia sterowania pociągami(ich prędkości w poszczególnych chwilach czasu i ich zmiany winny być zapamiętywane w pliku). Program musi posiadać możliwość odtwarzania ruchu pociągów zapisanego w pliku z historią.

User Stories - Obserwator

1. Jako obserwator chciałbym :
   - a) aby program informował o zablokowaniu się ruchu we wszystkich kierunkach na skrzyżowaniu (możliwe jest wczytanie mapy doprowadzającej do takiej sytuacji) (may have)
   - b) widzieć jak poruszają się pociągi (must have)
   - c) aby można było sprawdzić prędkości poszczególnych pociągów (should have)
   - d) mieć możliwość zatrzymania symulacji w dowolnej chwili (must have)
   - e) mieć możliwość przyspieszania i zwalniania symulacji (should have)
   - f) sprawdzać w danej chwili właściwości pociągów (cel podróży i punkty przez) i zadanej trasy (should have)
   - g) aby stacje były reprezentowane na mapie w postaci graficznej (must have)

User Stories - Obserwator

- h) aby pociąg zwalniał podczas dojeżdżania do stacji i do skrzyżowań (should have)
- i) aby pociąg stopniowo zwiększał swoją prędkość ruszając z miejsca (should have)
- j) aby przy każdym skrzyżowaniu można było sprawdzić stan wszystkich sygnalizacji (need to have)
- k) aby program miał możliwość zapisania do pliku przebiegu symulacji (must have)
- l) pociągi powinny być sterowane w sposób zapobiegający kolizjom
- m) aby symulacja była przedstawiona w czytelnej, ładnej oprawie graficznej
- n) aby była możliwość odtworzenia symulacji na podstawie zapisanego pliku

User Stories - Projektant torów

2 Jako projektant torów chciałbym :

- a) mieć możliwość ustawiania stacji w dowolnym punkcie mapy (must have)
- b) mieć możliwość łączenia stacji torami (jednokierunkowymi lub dwukierunkowymi) (must have)
- c) mieć możliwość kreowania otoczenia przez bardzo uproszczone elementy, takie jak przeszkody na mapie (may have)
- d) mieć możliwość ustalenia dowolnej odległości między stacjami (niezależnie od reprezentacji graficznej trasy) (good to have)
- e) projektować skrzyżowania torów (must have)
- f) ustawiać priotytety przejazdów pociągów na poszczególych skrzyżowaniach (should have)

User Stories - Projektant torów

- g) mieć możliwość łaczenia wielu odcinków torów i wiele skrzyżowań w segment. Na jednym segmencie nie może znajdować się więcej niż jeden pociąg (nice to have)
- h) aby plik opisujący mapę miał składnię umożliwiającą względnie nieskomplikowaną edycję mapy (must have)
- i) aby z programem dostarczony był graficzny edytor mapy (may have)
- j) tworzyć dowolnie dużą mapę (nice to have)

User Stories - Logistyk

3 Jako logistyk chciałbym:

- a) znać odległości pomiędzy poszczególnymi stacjami (must have)
- b) edytując plik wejściowy ustawiać ilość wagonów przed symulacją (should have)
- c) ustawiać maksymalne prędkości poszczególnych pociągów przed symulacją (przez plik wejściowy)
- d) aby po dojechaniu pociągu do stacji docelowej generowany był raport zawierający między innymi pokonaną odległość i czas przejazdu (nice to have)

User Stories - Logistyk

- e) aby pociąg dostosowywał prędkość do wolniejszego pociągu jadącego przed nim (must have)
- f) mieć możliwość zadania czasu minimalnego, przez jaki pociąg powinien stać na stacji (nice to have)
- g) móc przypisać pociągowi trasę specyfikując punkty (stacje) przez które ma on przejechać (must have)
- h) aby pociąg miał możliwość wyznaczenia właściwej dla siebie trasy na podstawie danych stacji: początkowej i końcowej, a także dowolnej ilości punktów (stacji) "przez"(must have)

Indywidualne dokonania: Edwin

Brałem udział w tworzeniu obsługi zapisu i odczytu pliku. Tworząc klasę DecodeXml zapewniłem proste wczytywanie danych symulacji z pliku o rozszeszeniu .xml z prostą, zaproponowaną przez zespół składnią. W klasie DecodeXml korzystam przede wszystkim z przygotowanych bibliotek Qt. Klasa stworzona przez programistów tworzących QtDomDocument świetnie nadaje się do naszego parsera.

Indywidualne dokonania: Edwin

Stworzyłem edytor map, który jest w pełni kompatybilny z symulatorem tworzonym przez pozostałą część zespołu. Edytor przede wszystkim składa się z klasy MainWindow i MapPainter. Klasa MainWindow odpowiada za wyświetlanie menu i wszystkich dostępnych toolbarów. Z kolei klasa MapPainter z użyciem biblioteki QtPainter jest odpowiedzialna za wyświeltlanie grafiki. Animacje rysowane są jednynie przy eventach myszy, więc są one rysowane dosyć szybko. Edytor choć miał być jedynie dodatkiem, stanowi wygodny sposób edytowania naszych plików .xml.

Indywidualne dokonania: Rita

- Starałam się zrobić otwieranie okna na początku, ale przez problem z kompem Adam mnie wyprzedził
- Rysowałam w Illustratorze i Photoshopie kafelki, wyliczałam żeby pasowało wszystko do siebie
- pogrzebałam w mapach
- testowałam na żądanie, aby sprawdzić czy działa

Indywidualne dokonania: Paweł

Moja praca zawiera się w ładowaniu postaci kaflowej mapy do programu. Chodziło m.in. o część przetwarzania kafli na wewnątrzprogramową postać grafu. Główna praca zawiera się w drugim sposobie ładowania mapy - pseudograficznym. W tym trybie użytkownik w obrębie odpowiednich znaczników żysuje"w pliku tekstowym ścieżkę kafli według określonych, ale dosyć luźnych reguł pozwalających np. na dodawanie komentarzy. Każdy kafel jest reprezentowany przez odpowiednie litery symbolizujące kierunek toru, uporządkowane w taki sposób, by możliwe było dowolne dozwolone łączenie torów. Przewagą tej metody nad pisaniem mapy w czystym xml-u była duża szybkość tworzenia mapy oraz możliwość łatwego wyobrażenia sobie jej wyglądu.

Indywidualne dokonania: Paweł

Ten sposób jednak przegrywa jako narzędzie do tworzenia map z graficznym edytorem, który pojawił się w 3. sprincie. Nadal jednak jest to metoda pozwalająca szybko generować np. mapy powtarzalne.
Reszta mej pracy polegała na pozbawianiu projektu kilku pomniejszych walorów rozrywkowych, jak choćby naprawa błędu przenikania zawracających pociągów. Miałem również pewien wkład w interpretowaniu kafelków na poziomie silnika.

Indywidualne dokonania: Adam

Moim wkładem w projekt jest zaprogramowanie praktycznie większości części graficznej (tekstury tworzone przez Ritę, parser przez Edwina i Pawła).
Na grafikę składa się główne okno, widżet informujący o stanach pociągów oraz widżet wyświetlający mapę wraz z obiektami. Biblioteka wykorzystana do stworzenia okna programu to Qt z dodatkiem OpenGL.
Klasy przeze mnie stworzone to MainWindow, TileGLWidget oraz InformationWidget.

Indywidualne dokonania: Adam - Klasy stworzone

MainWindow z pliku mainwindow.h:

- Klasa odpowiada za wyświetlanie menu z opcjami zapisu i odczytu symulacji, włączania i zatrzymywania symulacji, zmianę szybkości symulacji oraz przybliżania i oddalania mapy.
- Jej głównym elementem jest widżet wyświetlający mapę oraz widżet wyświetlający informacje.
- Realizuje pętle aktualizującą stan wewnętrznego silnika symulacji i odświeża widok.
- Wywołuje klasę DecodeXml do odczytywania i zapisywania stanu symulacji.
- Klasa wykorzystuje system slotów i sygnałów do łapania sygnałów emitowanych przez klasę InformationWidget (zmiana aktualnego pociągu) oraz sygnałów emitowanych przez klasę TileGLWidget (informacja w dolnym pasku o zagnieżdżeniu się pociągów).

Indywidualne dokonania: Adam - Klasy stworzone

TileGLWidget z pliku tileglwidget.h:

- Inicjuje bibliotekę graficzną OpenGL, która jest odpowiedzialna za wyświetlanie
- Rysuje kafelki/semafory/pociągi/czaszki
- Reaguje na przyciśniecie przycisków przybliżania i oddalania mapy
- Pozwala na przesuwanie mapy przy użyciu lewego przycisku myszy
- Pozwala wyznaczyć (poprzez wywołanie odpowiedniej funkcji i wyświetlenie jej rezultatu) odległość między danymi stacjami
- Sprawdza, który pociąg jest aktualnie wybrany i zmienia jego kolor, tak aby odróżniał się od pozostałych oraz podświetla stacje na jego trasie

Indywidualne dokonania: Adam - Klasy stworzone

InformationWidget z pliku informationwidget.h:

- Wyświetla czas trwania symulacji
- Pozwala zmieniać aktualnie badany pociąg
- Podczas poruszania pociągu wyświetla prędkość/nazwę/rozkład/czas stania pociągów na stacjach
- Gdy dany pociąg dojedzie wyświetla ponadto czas przejazdu, średnią prędkość, czas postojów

Indywidualne dokonania: Michał

Moim głównym udziałem w projekcie jest część algorytmiczna silnika. Mając do dyspozycji interfejs zaprojektowany przez Janka miałem za zadanie:

- Napisać metodę do wyznaczania trasy dla pociągu (za pomocą algorytmu Dijkstry)
- Wymyśleć sposób interpretacji kafelków tworząc na ich podstawie skrzyżowania, stawiając między nimi tory i łącząc je logicznie w segmenty (segment jest podstawowym narzędziem zapobiegającym kolizjom - na jednym segmencie może przebywać najwyżej jeden pociąg)
- Znaleźć sposób na wykrywanie zakleszczeń wzajemnych i samozakleszczeń oraz informować o takim zdarzeniu moduł graficzny

Indywidualne dokonania: Michał

Zrobiłem jeszcze kilka innych, mniej skomplikowanych rzeczy:

- Napisanie i zastosowanie metody perfectAngle(), która pozwala na wyznaczenie właściwego kąta, pod jakim pociąg powinien być rysowany podczas skręcania
- Resetowanie i czyszczenie silnika
- Inne nieskomplikowane rzeczy

Indywidualne dokonania: Michał - Metody

Kilka ważnych metod:

- Engine::addTile() - metoda ta interpretuje kafelek we wspomniany sposób
- Engine::findShortest() - metoda wyznaczająca najkrótszą trasę między dwoma punktami
- Engine::Train::findRoute() - metoda wyznaczająca trasę przez nieograniczoną liczbę punktów (wywołuje metodę findShortest())
- Engine::getSegmentFor() - metoda zwracająca właściwy segment dla toru, który ma być ustawiony między danymi skrzyżowaniami, w miarę potrzeby dokonuje również łączenia segmentów

Indywidualne dokonania: Janek

W tym projekcie zajmowałem się głownie silnikiem. Zaprojektowałem interfejs silnika i napisałem klasy pozwalające na animację pociągu. Stworzyłem również mechanizm sterowania sygnalizacjami świetlnymi. Miałem również swój niewielki wkład w wyświetlaniu obrazu w opengl.

Indywidualne dokonania: Janek - Klasy

Klasa Train reprezentuje pociąg. Posiada ona parametry takie jak maksymalna prędkość, aktualna prędkość, ilość wagonów, stacje docelowe i aktualna pozycja. Metoda update powoduje zapisanie do klasy TrainState nowego stanu pociągu. Wykorzystuje ona funkcję move() która przyjmuje za parametr odległość i zwraca informację czy udało się przesunąć pociąg. Modelowy pociąg składa się z dwóch części niewidocznej strefy hamowania (zależnej od prędkości) i z rzeczywistej części pociągu (wyświetlanej w postaci wagonów). Taka reprezentacja pociągu gwarantuje możliwość bezpiecznego hamowania i zapobiega kolizjom.

Wspólny wysiłek

Wszyscy również byli zaangażowani w testowanie aplikacji, zgłaszanie błędów, planowanie projektu, pisanie User Stories itp.

Zakończenie

Dziękujemy.