Projekt Zespołowy

Edytor pojazdów i map

Wojciech Kowalik, Konrad Miśkiewicz, Mateusz Pielat

17 listopada 2015

1 XML - do dodania gdzieś

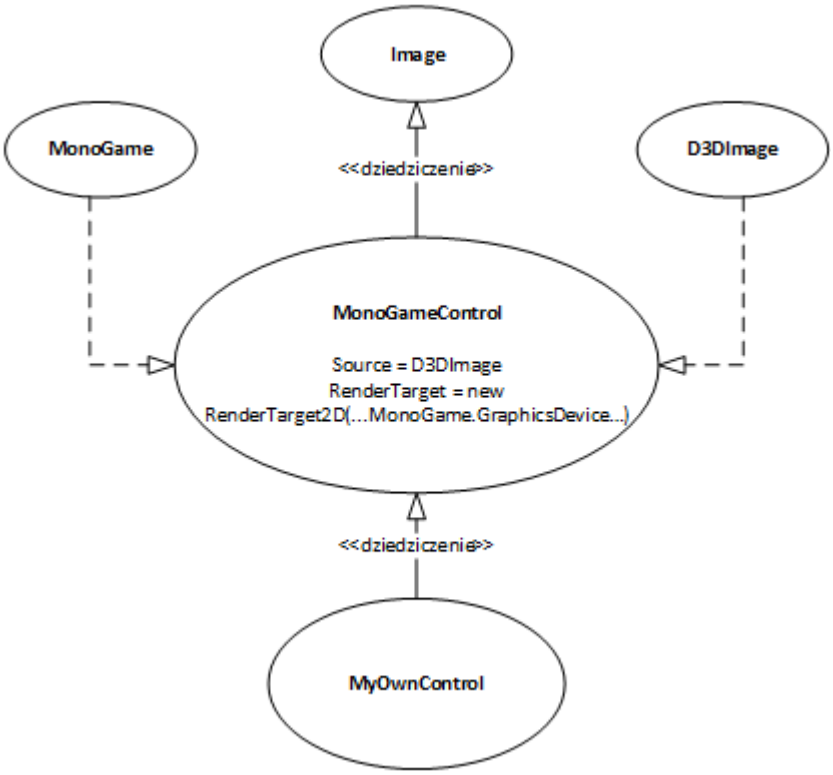
```
<vmd xmlns="pl.pw.mini.KowMisPie.SRL" width="512" height="512">
  <polygon>
        <point x = "160" y="119" />
        <point x = "58" y="284" />
        <point x = "357" y="275" />
    </polygon>
    <polygon>
        <point x = "393" y="32" />
        <point x = "390" y="398" />
        <point x = "457" y="400" />
        <point x = "458" y="33" />
    </polygon>
    <polygon>
        <point x="233" y="327" />
        <point x="30" y="316" />
        <point x="27" y="60" />
        <point x="357" y="60" />
        <point x="362" y="23" />
        <point x="466" y="21" />
        <point x="467" y="7" />
        <point x="16" y="42" />
        <point x="15" y="334" />
        <point x="258" y="406" />
        <point x="315" y="358" />
    </polygon>
</vmd>

<vvd xmlns="pl.pw.mini.KowMisPie.SRL">
    <orientation>
        <point x="0" y="0" />
        <angle>1.9091524329963763</angle>
    </orientation>
    <polygon>
        <point x="112" y="93" />
        <point x="359" y="95" />
        <point x="433" y="165" />
        <point x="371" y="222" />
        <point x="111" y="222" />
    </polygon>
</vvd>
```

2 Połączenie technologii WPF i MonoGame

Aby interfejs użytkownika był możliwie prosty, a tworzenie map i pojazdów maksymalnie wydajne połączone zostały technologie Windows Presentation Foundation (WPF) i MonoGame. Technologia WPF domyślnie nie umożliwia osadzenia zewnętrznego silnika graficznego, więc aby móc skorzystać z MonoGame w tym środowisku konieczne było napisanie odpowiedniej kontrolki.

W związku z tym stworzona została kontrolka *MonoGameControl*. W rzeczywistości jest ona klasą abstrakcyjną, która zawiera trzy metody abstrakcyjne. Aby z niej skorzystać konieczne jest utworzenie własnej klasy dziedziczącej po *MonoGameControl* i implementującej metody: **Initialize()**, **Uninitialize()** oraz **Render(TimeSpan time)**. Budowę takiej kontrolki oraz jej implementację przedstawiono poniżej.

2.1 Schemat kontrolki hostującej MonoGame



Schemat 1. MonoGameControl

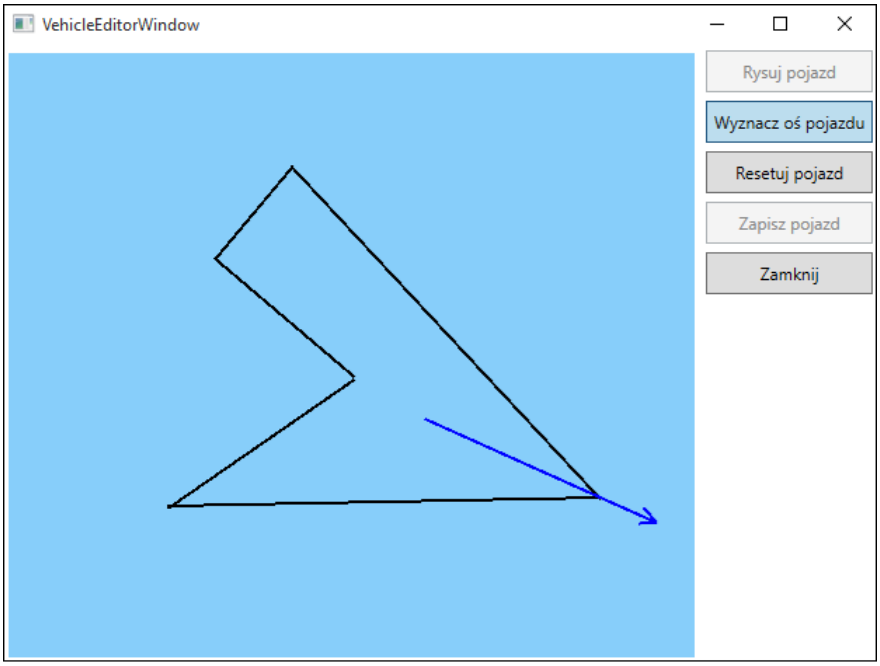
2.2 Metody abstrakcyjne wymagające nadpisania

- **abstract void Initialize()** - wywoływana przy tworzeniu kontrolki
- **abstract void Unitialize()** - wywoływana przy destrukcji kontrolki
- **abstract void Render(TimeSpan time)** - wywoływana cyklicznie, odpowiada za ponowne renderowanie obrazu wyświetlanego na kontrolce

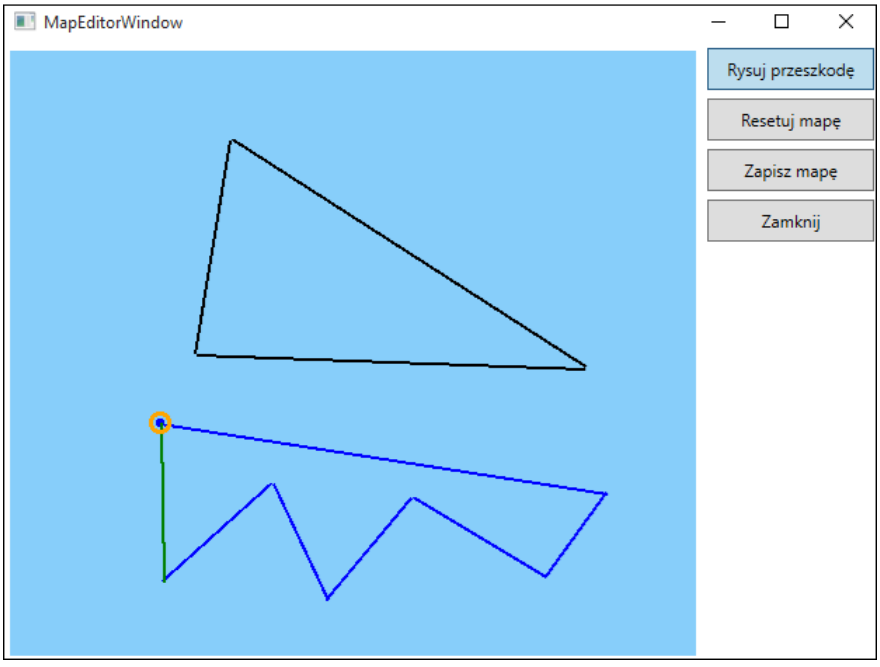
2.3 Przykład Implementacji własnej kontrolki hostującej MonoGame

```
public class MyOwnControl : MonoGameControl
{
    private SpriteBatch spriteBatch;

    protected override void Initialize()
    {
        spriteBatch = new SpriteBatch(GraphicsDevice);
    }

    protected override void Unitialize()
    {
        spriteBatch.Dispose();
    }

    protected override void Render(TimeSpan time)
    {
        GraphicsDevice.Clear(Color.LightSkyBlue);
        GraphicsDevice.RasterizerState = RasterizerState.CullNone;

        spriteBatch.BeginDraw();

        // DRAW

        spriteBatch.End();
    }
}
```

3 Osadzenie obrazka



Rysunek 1. Edytor pojazdu



Rysunek 2. Edytor map

3.0.1 Moduł wizualizacji

- System powinien umożliwiać wczytywanie uprzednio zapisanych w odpowiednim formacie map i pojazdów.
- System powinien umożliwiać ustawienie początku i końca trasy pojazdu.
- System powinien umożliwiać ustawienie prędkości animacji ruchu pojazdu.
- System powinien umożliwiać pauzowanie trwającej wizualizacji.
- System powinien umożliwiać przejście suwakiem do dowolnego momentu symulacji.
- System powinien umożliwiać zapisywanie wygenerowanej symulacji.

3.0.2 Moduł rysowania pojazdów i map

- System powinien umożliwiać rysowanie wielokątów reprezentujących pojazdy lub przeszkody omijane przez pojazd.
- System powinien zamykać wielokąt, jeżeli użytkownik nie zamknie go sam.
- System powinien umożliwiać zapisywanie narysowanych pojazdów/map do postaci wektorowej.

3.0.3 Moduł generowania pojazdów i map z plików graficznych

- System powinien umożliwiać wczytanie pliku graficznego w jednym z dozwolonych formatów.
- System powinien umożliwiać dobór parametrów trasowania.
- System powinien wyświetlać użytkownikowi wynik trasowania dla aktualnie dobranych parametrów.
- System powinien umożliwiać zapisywanie wygenerowanych pojazdów/map do postaci wektorowej.

3.0.4 Moduł algorytmu

- System powinien znajdować ścieżkę między punktem startowym a punktem końcowym dla danego pojazdu z uwzględnieniem przeszkód.
- System powinien eksportować wynik obliczeń do listy rozkazów.

4 Wymagania niefunkcjonalne

- Aplikacja powinna poprawnie działać na systemach operacyjnych z rodziny *Windows*.
- System powinien poinformować użytkownika, gdy przeprowadzenie symulacji jest niemożliwe.
- Dozwolone formaty dla plików graficznych: **bmp**, **jpeg**, **png**, **gif**.
- Rysowanie pojazdów i map nie powinno odbywać się poprzez rysowanie ciągłe, a raczej poprzez dodawanie kolejnych punktów do wielokątów.
- Podczas rysowania wielokątów ich krawędzie nie mogą się przecinać.
- System powinien dać się łatwo tłumaczyć na różne języki:
  - Powinien mieć wbudowany język polski i angielski.
  - Dodanie nowego języka powinno ograniczać się do zmian w pliku konfiguracyjnym.

5 Metodologia pracy

Aplikacja będzie tworzona zgodnie z modelem przyrostowym (ang. *incremental development*). Zakłada on realizację jedynie pewnej części systemu w każdym kolejnym 'cyklu pracy. Poszczególne porcje funkcjonalności powinny być spójne, aby możliwe było przetestowanie i dostarczenie ich w wersji finalnej klientowi.

5.1 Uzasadnienie wyboru

Preferencja modelu przyrostowego nad modelem kaskadowym wynika ze struktury naszej aplikacji. Można ją podzielić na wiele odrębnych modułów działających (do pewnego stopnia) oddzielnie i niezależnie od siebie. Ponadto, ta metodologia zakłada brak konieczności definiowania z góry szczegółowych wymagań poszczególnych części systemu, dzięki czemu proces tworzenia jest bardziej elastyczny niż w przypadku modelu kaskadowego.

6 Harmonogram pracy

Architektura systemu		
WK	repozytorium i struktura katalogów	17.11.15
KM	przygotowanie diagramu klas	
ALL	zdefiniowanie reprezentacji pojazdu i mapy	
ALL	zdefiniowanie reprezentacji rozkazów ruchu pojazdu	
Interfejs graficzny		
MP	przygotowanie mock-up poszczególnych okien interfejsu	
KM	implementacja interakcji pomiędzy oknami	
Tworzenie pojazdów i map		
KM	serializacja i deserializacja do XML	
WK	tworzenie pojazdów i przeszkód przy pomocy kursora	
MP	selekcja trasowanych grafik rastrowych pod względem rozszerzeń i treści	01.12.15
MP	trasowanie z użyciem wybranej biblioteki; wyświetlanie efektu użytkownikowi dla odpowiednio dobranych parametrów trasowania	
Mock-up algorytmu		15.12.15
KM	losowe generowanie tras	
KM	eksport trasy do listy rozkazów	
Wizualizacja		
WK	tworzenie przeszkód i pojazdu na mapie	
WK	ustawianie początku i końca trasy	
MP	animacja ruchu na podstawie rozkazów	
KM	selekcja trasowanych grafik rastrowych pod względem rozszerzeń i treści	
MP	implementacja suwaka osi czasu wizualizacji	
Algorytm		12.01.16
ALL	opracowanie koncepcji algorytmu wyszukiwania drogi	
ALL	implementacja algorytmu wyszukiwania drogi	
ALL	eksport i mapowanie wyników obliczeń do listy rozkazów	

ALL - zadania przydzielone dla wszystkich
WK - zadania przydzielone dla Wojciecha Kowalika
KM - zadania przydzielone dla Konrada Miśkiewicza
MP - zadania przydzielone dla Mateusza Pielata

7 Kamienie milowe

Każdy kamień milowy stanowi oddzielny, funkcjonalny moduł systemu. Wszystkie z nich kończą się testami i dokumentacją.

1. **Moduł tworzenia map i pojazdów przez użytkownika**

   Zostanie stworzony interfejs użytkownika oraz możliwe będzie ręczne tworzenie map i pojazdów z wykorzystaniem wbudowanego edytora i zapisywanie ich do formatu wektorowego

2. **Moduł wczytywania map i pojazdów z plików graficznych**

   Możliwe będzie wczytywanie pojazdów i map z plików graficznych oraz eksportowanie ich do formatu wektorowego

3. **Moduł wizualizacji ruchu**

   Użytkownik może wybierać pojazd, mapę oraz zaznaczać początek i koniec trasy. Mock-up algorytmu wygeneruje losową trasę dla pojazdu, która zostanie przedstawiona użytkownikowi. Użytkownik będzie miał możliwość ustawienia prędkości symulacji, pauzy oraz przejścia w czasie suwakiem.

4. **Moduł algorytmu wyszukiwania drogi**

   Zaimplementowany zostanie algorytm wyznaczania drogi, który zastąpi wcześniejszy mock-up. Wszystkie moduły aplikacji zostaną ze sobą zintegrowane.