Specyfikacja implementacyjna
Projekt indywidualny

Mateusz Smoliński

27 kwietnia 2019

Temat:	**Wersja:**
Projekt aplikacji	1.0
Autor:	**Data:**
Mateusz Smoliński e-mail: 01131251@pw.edu.pl	27.04.2019

Spis treści

1 Opis środowiska **2**

2 Diagramy klas **2**

3 Opis klas/metod **4**
3.1 Klasa MainWindow 4
3.2 Klasa Parser 4
3.3 Przestrzeń nazw MapObjects 4
3.3.1 Klasa Point 5
3.3.2 Klasa Line 5
3.3.3 Klasa River 5
3.3.4 Klasa City 5
3.3.5 Klasa CityData 5
3.4 Przestrzeń nazw Simulator 5
3.4.1 Klasa Map 6
3.4.2 Klasa MapConfiguration 6
3.4.3 Klasa MainControler 6
3.5 Przestrzeń nazw Events 6
3.5.1 Interfejs IGameEvent 7
3.5.2 Klasy zdarzeniowe 7

4 Plan testów **7**
4.1 Testy akceptacyjne programu 7
4.2 Testy jednostkowe 7

1 Opis środowiska

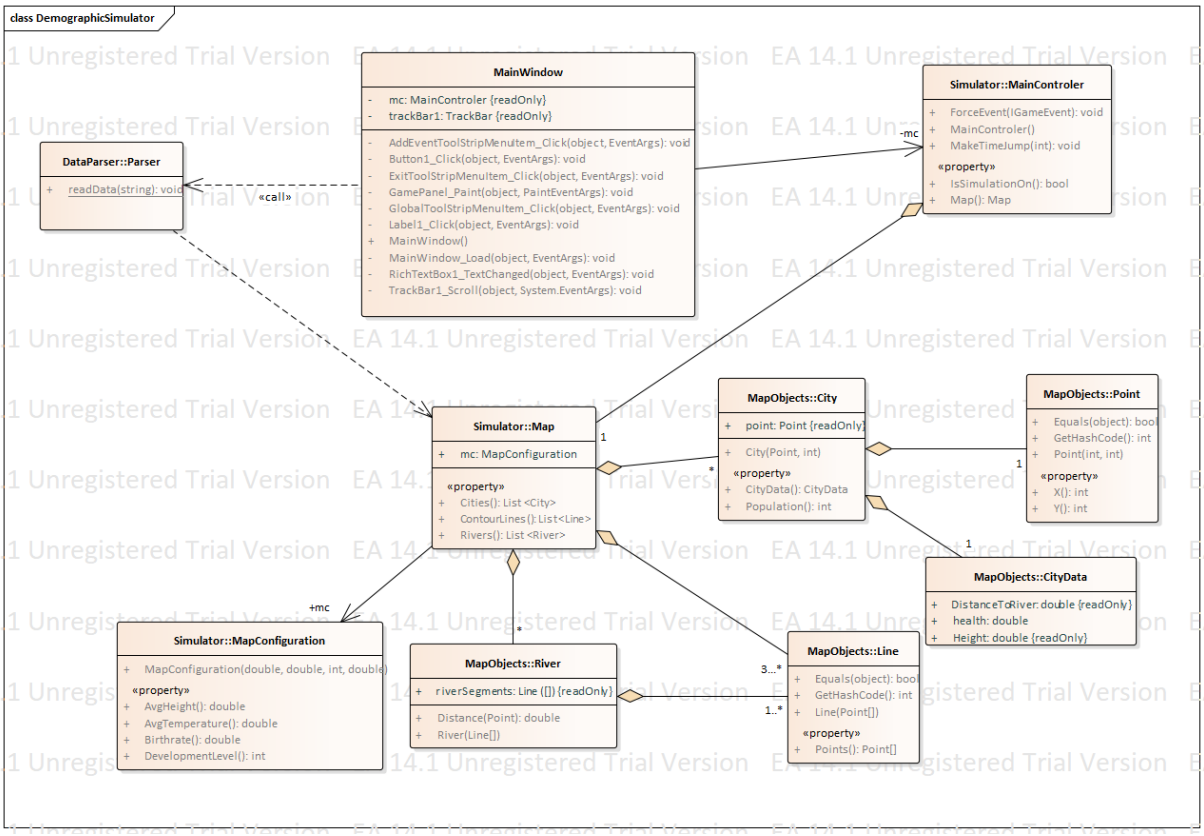
Aplikacja Demographic Simulator jest pisana w ramach realizacji projektu indywidualnego, którego tematem jest zapoznanie ze środowiskiem Visual Studio i napisanie aplikacji w języku C#. Powstaje w języku C# wersji 7.3 (.NET Framework 4.7.2), w środowisku Microsoft Visual Studio 2019 w wersji 16.0.2. Implementacja oraz testowanie programu odbywać się będą na komputerach o następujących parametrach:

- 64-bitowy system operacyjny Windows 10 Home ver. 1803,
- procesor Intel Core i5-7200U,
- pamięć RAM 8,00 GB,
- karty graficzne Intel HD Graphics 620 + NVIDIA GeForce 940MX.

2 Diagramy klas

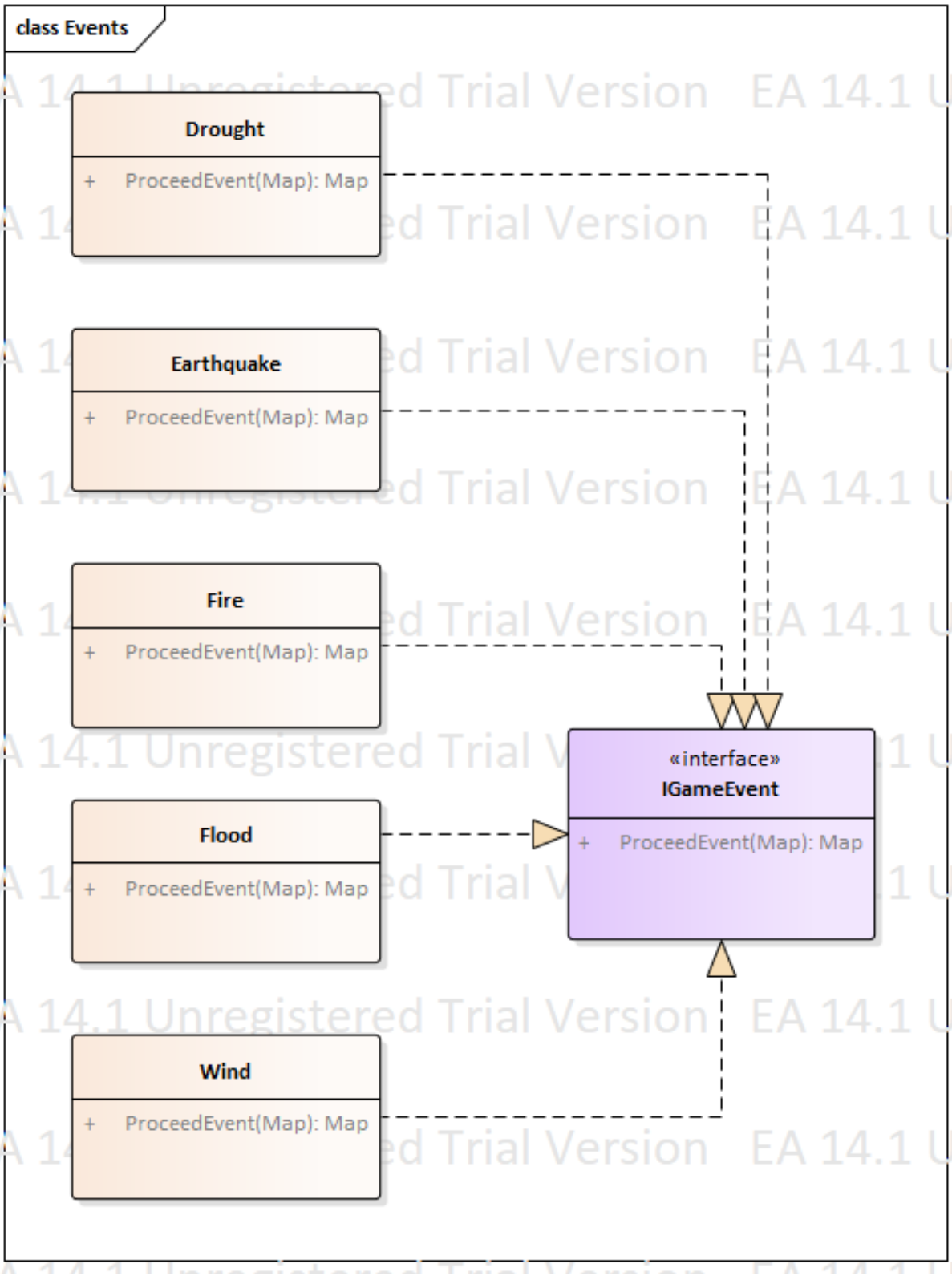
Program składa się z kilku modułów – odpowiedzialnych za wyświetlanie na ekranie, odczyt z pliku, przechowywanie danych aplikacji, operowanie tymi danymi oraz obsługę wydarzeń.

Główną strukturę programu przedstawia następujący diagram:



Rysunek 1: Diagram klas głównej części programu

Dodatkowo, klasy MainWindow i MainControler obsługują zdarzenia programu, które przedstawia poniższy diagram:



Rysunek 2: Diagram zdarzeń wywoływanych w trakcie programu

Wszystkie diagramy są wykonane za pomocą wersji próbnej programu Enterprise Architect 14.1.

3 Opis klas/metod

3.1 Klasa MainWindow

Jest to bazowa klasa tego programu, odpowiadająca przede wszystkim za interfejs graficzny. Składa się ona z wielu odpowiadających za reagowanie zdarzeniowe na działania użytkownika. Wśród najważniejszych logicznych implementacji, które muszą być zawarte w implementacjach tych metod, warto wspomnieć o:

- metodzie, która wyświetli dane o mieście wskazywanym przez kursor w odpowiednim panelu po prawej stronie ekranu,
- metodzie, która będzie odpowiadać za uruchamianie i zatrzymywanie symulacji, wykorzystując przy tym oddzielny wątek,
- metodzie wybierającej i generującej wydarzenia na mapie, na podstawie informacji otrzymanej od użytkownika.

Do zadań tej klasy należeć będzie także realizacja podstawowych zadań interfejsu – wywołanie wczytywania danych z pliku, wyświetlenie pomocy, informacji o autorze programu czy też wyłączenie programu.

3.2 Klasa Parser

Ta klasa odpowiada za wczytanie danych z pliku tekstowego, które zostaną użyte w trakcie działania programu. Wszystkie metody w tej klasie są statyczne, przez co obiekt klasy Parser nie będzie powstawać w żadnym momencie trwania programu. Metody te w przypadku niepowodzenia korzystają z wyjątków, które przekazują do metody nadrzędnej – metoda *ReadData* przekazuje te wyjątki do klasy *MainWindow*, która odpowiada za przekazanie komunikatu o błędzie użytkownikowi.

- `Map ReadData(string inputFilePath)`

Metoda *ParseFile* jest główną metodą tej klasy. Otrzymuje ona ścieżkę do pliku w postaci napisu, po czym ma za zadanie otworzyć ten plik, przeanalizować każdą jego linię oraz zapisać dane w postaci zrozumiałej dla programu. Na podstawie składni pliku oraz linii oddzielających etapy wczytywania, oznaczonych # na początku, odczytuje linię tekstu i dokonuje jej interpretacji.

Klasa w późniejszej wersji programu będzie podzielona na mniejsze metody, odpowiadające za odczyt konkretnego rodzaju linii. Wszystkie dane będą zapamiętywane w pamięci programu do opisanych w dalszej części dokumentu obiektach klas z przestrzeni nazw *MapObjects*.

W przypadku niepowodzenia spowodowanego błędami składniowymi metoda *ReadData* zwróci wyjątek, który zostanie później przekazany klasie *MainWindow* i przedstawiony użytkownikowi. W przypadku powodzenia metoda zwraca przygotowany obiekt klasy *Map*.

3.3 Przestrzeń nazw MapObjects

Klasy z tej przestrzeni nazw są odpowiedzialne za przechowywanie informacji o elementach poddawanych symulacji. Będą one zawierać zarówno dane geometryczne potrzebne do rysowania ich w programie, jak i dane demograficzne wykorzystywanie do symulowania życia na mapie.

3.3.1 Klasa Point

Ta klasa jest najprostszą klasą w programie. Jej zadaniem jest przechowywanie współrzędnych na mapie w postaci pary liczb. Jej metody ograniczą się do tworzenia punktów i ich porównywania na podstawie współrzędnych.

3.3.2 Klasa Line

Klasa *Line* jest kolejną prostą klasą przechowującą dane geometryczne. Jej zadaniem jest przechowywanie dwóch punktów (obiektów klasy *Point*, które oznaczają geometryczny początek i koniec linii. Podobnie jak w przypadku klasy *Point* jej metody ograniczają się do konstruktora obiektu oraz metod porównujących ze sobą obiekty klasy *Line* na podstawie współrzędnych punktów.

3.3.3 Klasa River

Ta klasa przechowuje kolekcję obiektów klasy *Line*, interpretując je jako zestaw powiązanych ze sobą odcinków rzeki. W przeciwieństwie do dwóch poprzednich klas, klasa *River* ma więcej zadań niż tylko przechowywanie danych – wykonuje ona także obliczenia potrzebne do wyznaczenia dystansu.

- `public double Distance(Point point)`

Te metoda odpowiada za wykonanie wspomnianych wcześniej obliczeń matematycznych potrzebnych do wyznaczenia dystansu od rzeki. Jako argument otrzymuje punkt, którego dystans od rzeki ma za zadanie wyznaczyć, po czym zwraca najkrótszą odległość spośród odległości od poszczególnych odcinków rzeki.

3.3.4 Klasa City

Ta klasa będzie przechowywać podstawowe informacje o mieście. Należą do nich współrzędne miasta, traktowane w programie jako punkt o danych współrzędnych, a także populacja i inne dane o mieście. Pozostałe dane są przechowywane w zewnętrznym obiekcie, który jest opisany poniżej.

Miasto będzie umożliwiało wykonywanie podobnych operacji, co punkt i linia – porównywanie miast ze sobą i sprawdzanie ich jednoznaczności w celu ułatwienia przyszłych iteracji.

3.3.5 Klasa CityData

Ta klasa składać się będzie jedynie z właściwości opisujących miasto. Będą to między innymi:

- dystans do rzeki,
- współczynnik zdrowotności miasta,
- wysokość bezwzględna miasta.

3.4 Przestrzeń nazw Simulator

Klasy znajdujące się w tej przestrzeni pełnią funkcję serca samej symulacji życia. Odpowiadają one za przechowywanie globalnych informacji o mapie oraz przeprowadzanie obliczeń przesuwających symulację do przodu.

3.4.1 Klasa Map

Klasa przechowuje wszystkie struktury zawarte w przestrzeni *MapObjects*. Składa się z następujących elementów:

- listy miast występujących na mapie,
- listy rzek występujących na mapie,
- listy linii pełniących funkcję konturu dla mapy,
- obiekt klasy *MapConfiguration*, przechowujący globalne informacje o danym terytorium.

3.4.2 Klasa MapConfiguration

Ta klasa pełni analogiczną rolę do klasy *MapObject*. Przechowuje konkretne właściwości, dotyczące tym razem globalnych informacji potrzebnych do symulacji. Zawarte tu będą między innymi:

- średnia wysokość obszaru,
- średnia temperatura,
- współczynnik przyrostu naturalnego,
- poziom rozwoju cywilizacyjnego.

3.4.3 Klasa MainControler

W tej klasie znajdują się wszystkie najważniejsze operacje kalkulujące zmiany na mapie. Pełni ona także funkcję pośrednika pomiędzy główną klasą interfejsu (*MainWindow*), a klasami przechowującymi dane o programie.

- `public void MakeTimeJump(int timePeriod)`

Najważniejsza metoda tej klasy odpowiada za przesunięcie mapy o dany okres, liczony w miesiącach. Wykorzystuje do tego dane zawarte w strukturach programu. Do jej zadań należy modyfikacja poszczególnych współczynników miast, sprawdzenie prawdopodobieństwa potencjalnych zdarzeń i ewentualne wywołanie tych zdarzeń.

- `public void ForceEvent(IGameEvent gameEvent)`

Ta metoda otrzymuje jako argument zdarzenie wywołane przez interfejs użytkownika. Jej zadaniem jest wywołanie zdarzenia i wykonanie zmian na mapie gry bez wywoływania przesunięcia czasowego.

3.5 Przestrzeń nazw Events

Ta przestrzeń składa się z interfejsu *IGameEvent* oraz kilku klas implementujących ten interfejs. Zadaniem tych klas jest przeprowadzenie zdarzenia w danym punkcie mapy.

3.5.1 Interfejs IGameEvent

- `Map ProceedEvent(Map map, Point center)`

Ten interfejs składa się z dokładnie jednej metody, która jako argumenty otrzymuje mapę gry oraz punkt będący epicentrum wywoływanego zdarzenia. Każda klasa implementująca ten interfejs może zmodyfikować mapę w inny sposób, zależny od danego zdarzenia.

3.5.2 Klasy zdarzeniowe

Do klas implementujących zdarzenia należą wymienione wcześniej na diagramie klasy:

- *Drought*,
- *Earthquake*,
- *Fire*,
- *Flood*,
- *Wind*.

Każda z tych klas ma za zadanie wywołać docelową zmianę na mapie rozgrywki, korzystając z informacji zawartych w klasie kontenerowej *Map*.

4 Plan testów

W ramach projektu zostaną przeprowadzone testy oprogramowania, które składać się będą z testów ogólnych oraz jednostkowych.

4.1 Testy akceptacyjne programu

Testy ogólne będą polegać na przetestowaniu programu z perspektywy użytkownika. Polegać będą na przeprowadzeniu testowych symulacji przez interfejs graficzny, sprawdzenie kilku nieprzewidywalnych wyborów i próby wprowadzenia niewłaściwych plików do programu. Wszystkie sytuacje, które nie zostały obsłużone przez program zostaną poprawione w ostatecznej wersji aplikacji.

4.2 Testy jednostkowe

Do testów jednostkowych, przeprowadzanych z poziomu kodu muszą należeć testy następujących zagadnień:

1. Testy klas zdarzeniowych – metody zdarzeniowe będą otrzymywać przygotowane obiekty *Map*, po czym testy będą sprawdzać, czy nastąpiły zmiany zgodne z przewidywaniami wywołującego zdarzenia testera z uwzględnieniem ewentualnych elementów losowych.
2. Testy klas przechowujących dane – testowane będzie poprawne działanie metod obliczających (takich jak odległość od rzeki) oraz metod *equals* sprawdzających tożsamość punktów i linii.
3. Testy klasy Parser, polegające na podaniu plików tekstowych z celowo przygotowanymi różnymi rodzajami błędów.

4. Testy klas z przestrzeni *Simulator*, polegające przede wszystkim na wywoływaniu ręcznym zdarzeń i różnych czasowych skoków z łatwymi do przewidzenia skutkami.

Data:	**Autor:**	**Zakres:**	**Zatwierdził:**	**Wersja:**
11.04.2019	MS	Utworzenie dokumentu, wstępny opis klas	MS	0.1
23.04.2019	MS	Wygenerowanie diagramów, poprawki	MS	0.2
27.04.2019	MS	Uzupełnienie opisów klas i testów	MS	1.0