Języki i metody programowania 2
Sprawozdanie z projektu nr 2

Mateusz Bocheński
Eryk Banaś

07.06.2017

1 Opis ogólny

1.1 Nazwa programu

WireWorld

1.2 Poruszany problem

Celem projektu jest stworzenie aplikacji w języku Java implementującej automat komórkowy WireWorld Briana Silvermana. Komórka może znajdować się w jednym z czterech stanów:

1. Pusta (kolor biały)
2. Głowa elektronu (kolor czerwony)
3. Ogon elektrony (kolor żółty)
4. Przewodnik (kolor czarny)

Kolejne generacje budowane są z wykorzystaniem zestawu pięciu zasad:

- Komórka pozostaje Pusta, jeśli była Pusta.
- Komórka staje się Ogonem elektronu, jeśli była Głową elektronu.
- Komórka staje się Głową elektronu tylko wtedy, gdy dokładnie 1 lub 2 sąsiadujące komórki są Głowami Elektronu, była Przewodnikiem i nie była Ogonem.
- Komórka staje się Przewodnikiem jeśli była Ogonem elektronu oraz w każdym innym wypadku.

Do sprawdzenia stanu komórek w danej iteracji stosowane jest sąsiedztwo Moore'a.

2 Opis funkcjonalności

2.1 Jak korzystać z programu

Obsługa programu odbywa się głównie za pomocą interfejsu graficznego. Menu zostało opracowane intuicyjnie, w razie wątpliwości polecamy przeczytanie instrukcji znajdującej się pod pozycją menu "Pomoc", a następnie ponownie "Pomoc".

Uwaga: projekt należy zaimportować jako projekt Maven, żeby można było korzystać z bibliotek.

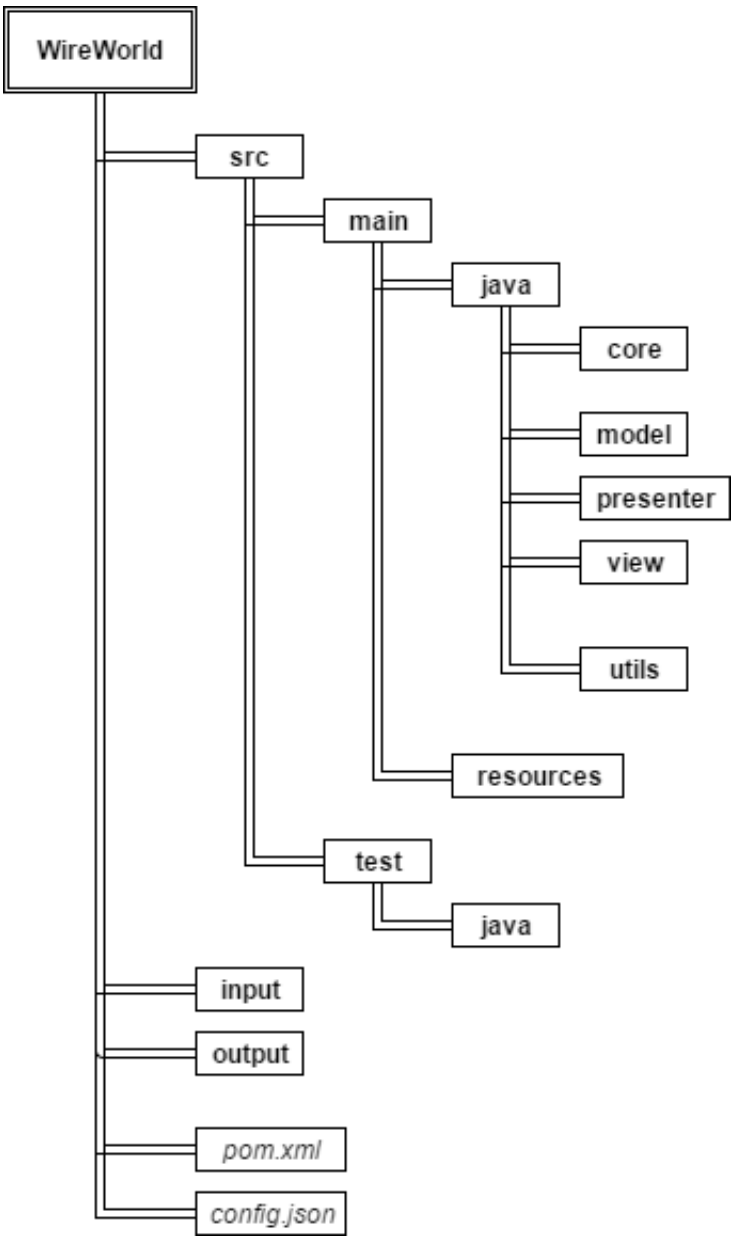
2.2 Uruchomienie programu

W celu skorzystania z programu należy uruchomić klasę App.java znajdującą się w folderze aplikacji w ścieżce: src/main/java/.

3 Format danych i struktura plików

3.1 Struktura

Szablon struktury katalogowej zgodnej z Mavenem:



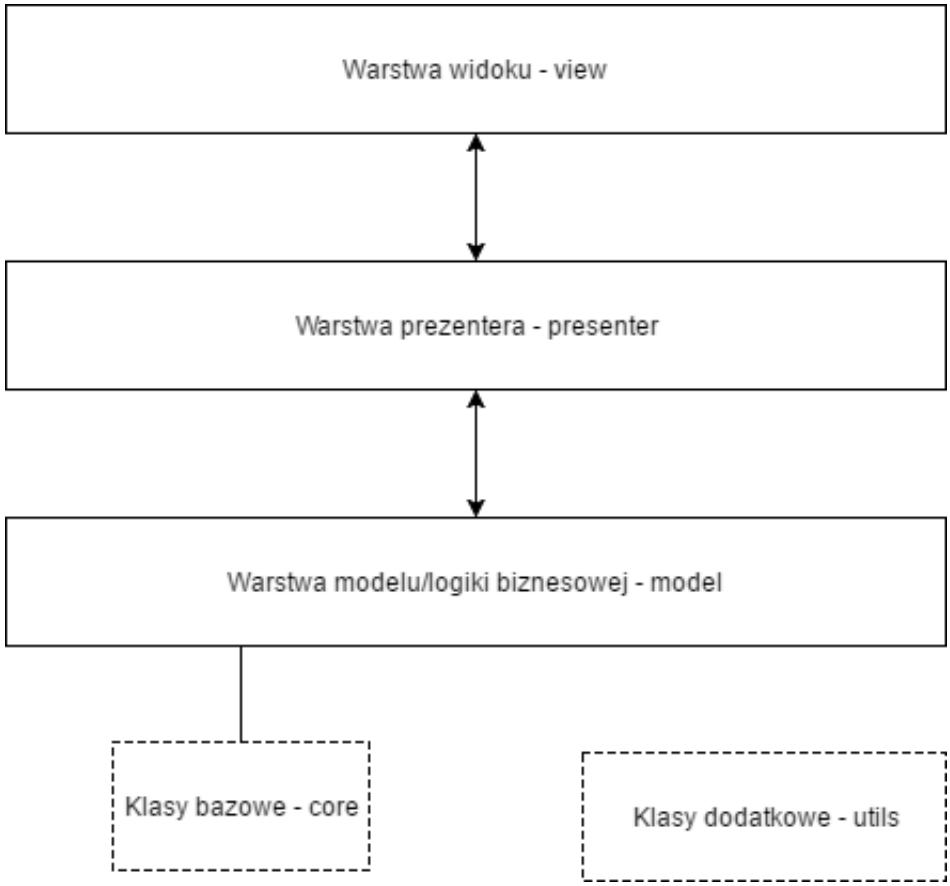
Aplikacja jest podzielona następująco:

- input - folder zawierający przykładowe dane generacji wczytywane do aplikacji. Użytkownik ma możliwość wczytania do programu innych danych początkowych (z innym umiejscowieniem komórek na planszy), musi tylko przestrzegać formatu danych zgodnego z plikiem przykładowym:
  - exampleInputData.json
    - WireCell - lista współrzędnych x, y pojedynczych komórek, które będą umieszczone na planszy (w gridzie)
    - WireLine - definicja struktury (obiektu) składającej się z kilku komórek położonych pionowo lub poziomo (w kształcie linii), zawiera współrzędne komórki początkowej oraz komórki końcowej

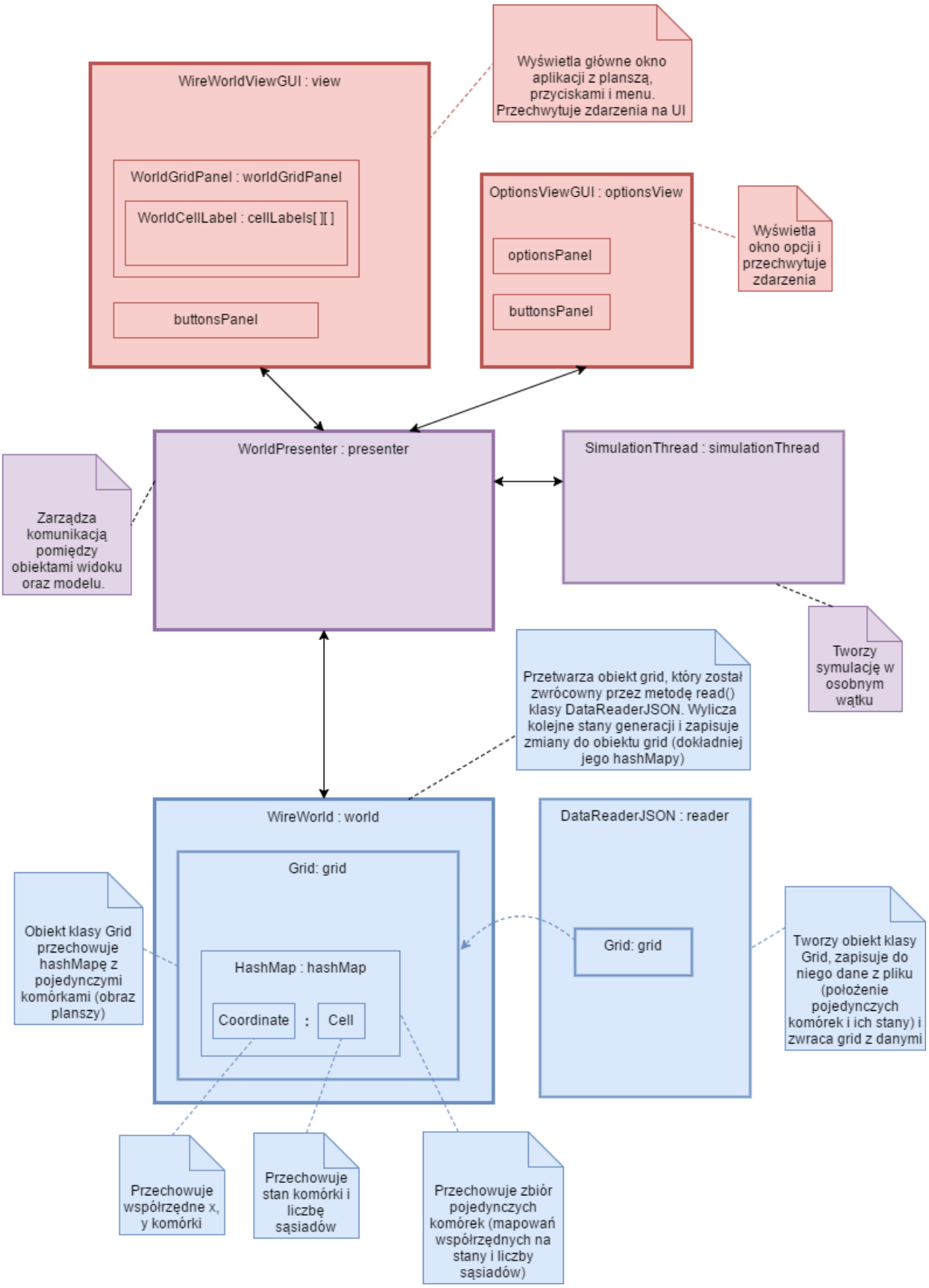
    * OrGate - definicja struktury (obiektu) składającej się z kilku komórek i tworzących obramkę typu OR, zawiera współrzędne początkowe bramki
    * XorGate - definicja struktury (obiektu) składającej się z kilku komórek i tworzących obramkę typu XOR, zawiera współrzędne początkowe bramki
    * ElectronHead - lista współrzędnych komórek, które reprezentują głowę elektronu
    * ElectronTail - lista współrzędnych komórek, które reprezentują ogon elektronu

- core - folder zawiera podstawowe klasy jak np. Cell, Grid, Coordinate
  - Grid - klasa przechowująca hashMapę z komórkami (mapowania współrzędnych x, y ze stanami komórek i liczbą sąsiadów). Zawiera metody pozwalające wstawiać do gridu (hashMapy) pojedyncze komórki - insertNewCell(int x, int y, State state) oraz całe obiekty składające się z kilku komórek - insertNewObject(WorldObject worldObject).
  - WorldObject - klasa abstrakcyjna dla obiektów składających się z więcej niż jednej komórki
  - OrGate - klasa reprezentująca bramkę typu OR. Dziedziczy po klasie abstrakcyjnej WorldObject. Zwraca listę komórek należących do bramki.
  - XorGate - klasa reprezentująca bramkę typu XOR. Dziedziczy po klasie abstrakcyjnej WorldObject. Zwraca listę komórek należących do bramki.
- utils - zawiera klasy odpowiedzialne za wczytywanie i zapisywanie plików z generacjami oraz wczytywanie konfiguracji z config.json model - klasy reprezentujące model
  - World - klasa przechowuje obiekt bazowy Grid reprezentujący planszę z komórkami. Klasa World dodatkowo posiada logikę do wyliczania kolejnych stanów generacji (komórek na planszy).
- presenter - klasy reprezentujące prezentera (pośredniczą między modelem a widokiem). Ich zadaniem jest zarządzanie informacjami pochodzącymi od widoku lub modelu/logiki biznesowej.
- view - klasy reprezentujące widok (interfejs graficzny aplikacji). Klasy widoku definiują wygląd okienek. Dodatkowo przechwytują zdarzenia, które są w nich wykonywane i przekazują sterowanie do klas prezentera.
- pom.xml - plik zawiera dane konfiguracyjne Maven
- config.json (zawiera dane konfiguracyjne aplikacji)
  - rozmiary planszy poziomy i pionowy
  - prędkość animacji (w milisekundach)
  - Liczba generacji do stworzenia po uruchomieniu animacji

Aplikacja została stworzona wg modelu MVP (z MVP wynika model, presenter, view).



Uproszczony schemat aplikacji:



3.2 Przechowywanie danych w programie

Dane generacji są pobierane z pliku JSON, gdzie:

- WireCell - współrzędne x, y pojedynczych komórek reprezentujących przewodnik
- WireLine - współrzędne początkowe i końcowe struktury reprezentującej prosty, dłuższy fragment przewodnika (odcinek)
- ElectronHead - współrzędne x, y głów elektronów
- ElectronTail - współrzędne x, y ogonów elektronów
- ... - współrzędne początkowe x, y innych dodatkowych struktur (np. bramek - AndGate)

W momencie kliknięcia “Otwórz generację” i wybraniu poprawnego pliku, dane generacji są wczytywane do aplikacji z pliku config.json, konwertowane i przechowywane w klasie Configuration.java w strukturze HashMap¡Coordinate, Cell¿. Dzięki temu czas dostępu do komórek o zadanych współrzędnych ma średnią złożoność czasową O(1).

3.3 Dane wejściowe

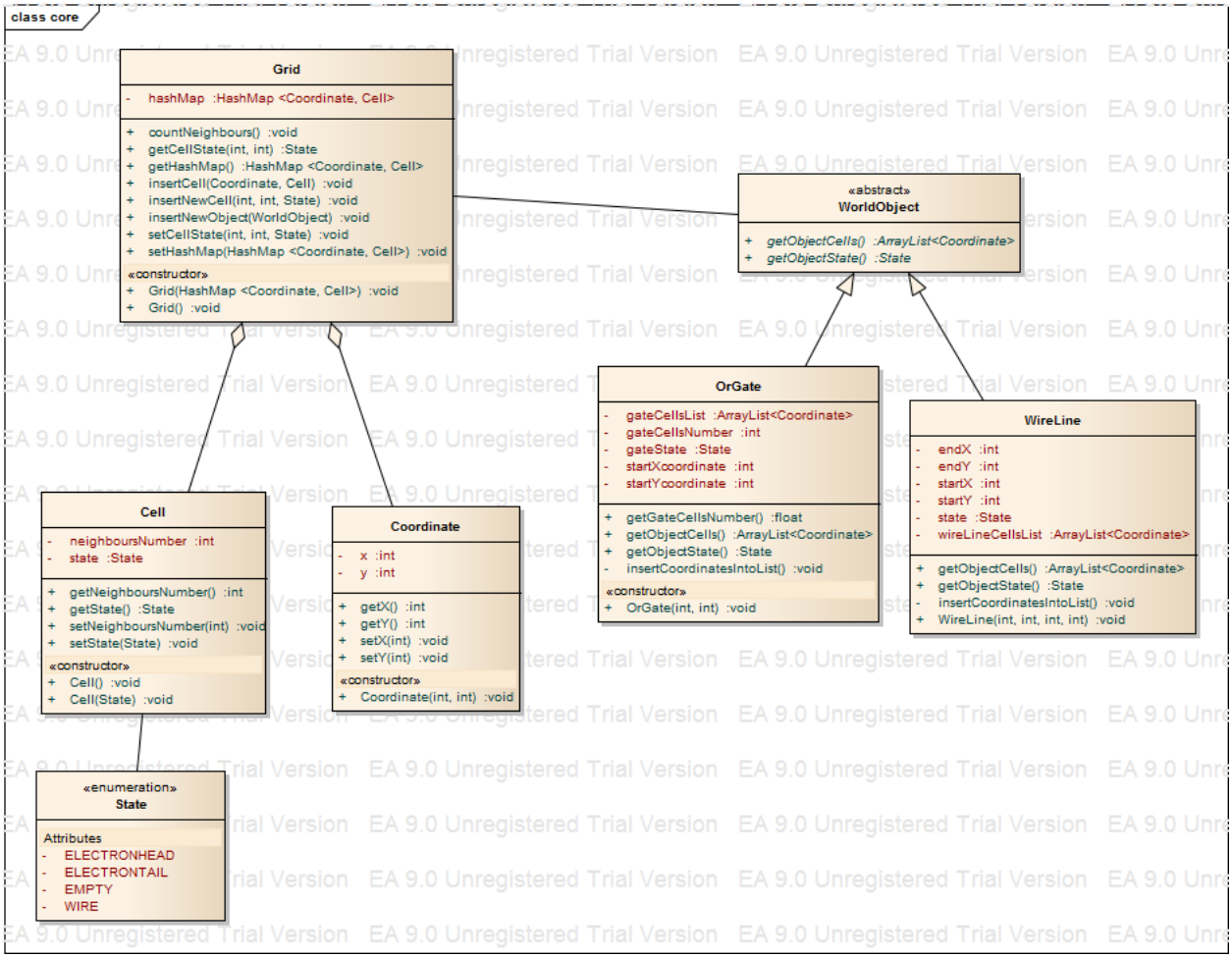
- Liczba P generacji do stworzenia
- Prędkość tworzenia generacji w wizualizacji
- Plik z konfiguracją początkową - config.json

3.4 Dane wyjściowe

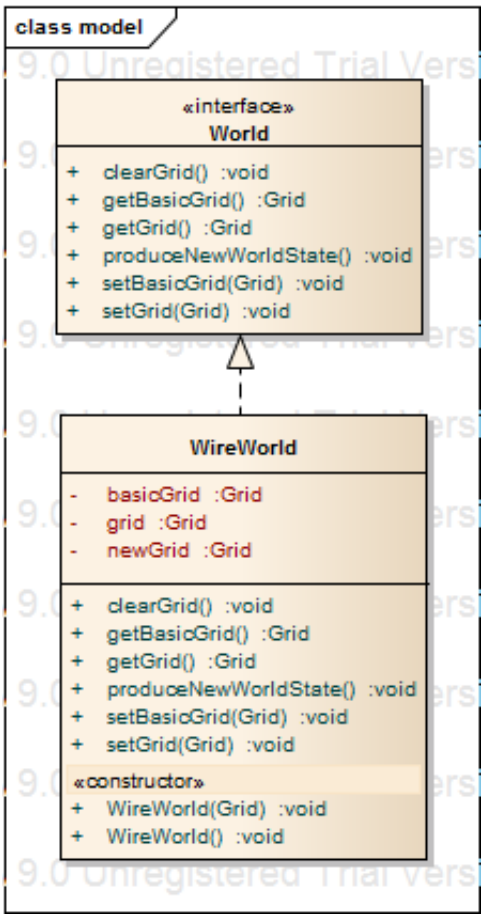
- Wizualizacja “na żywo” aktualnego stanu planszy
- Plik z opisem generacji wyjściowej

3.5 Diagramy klas

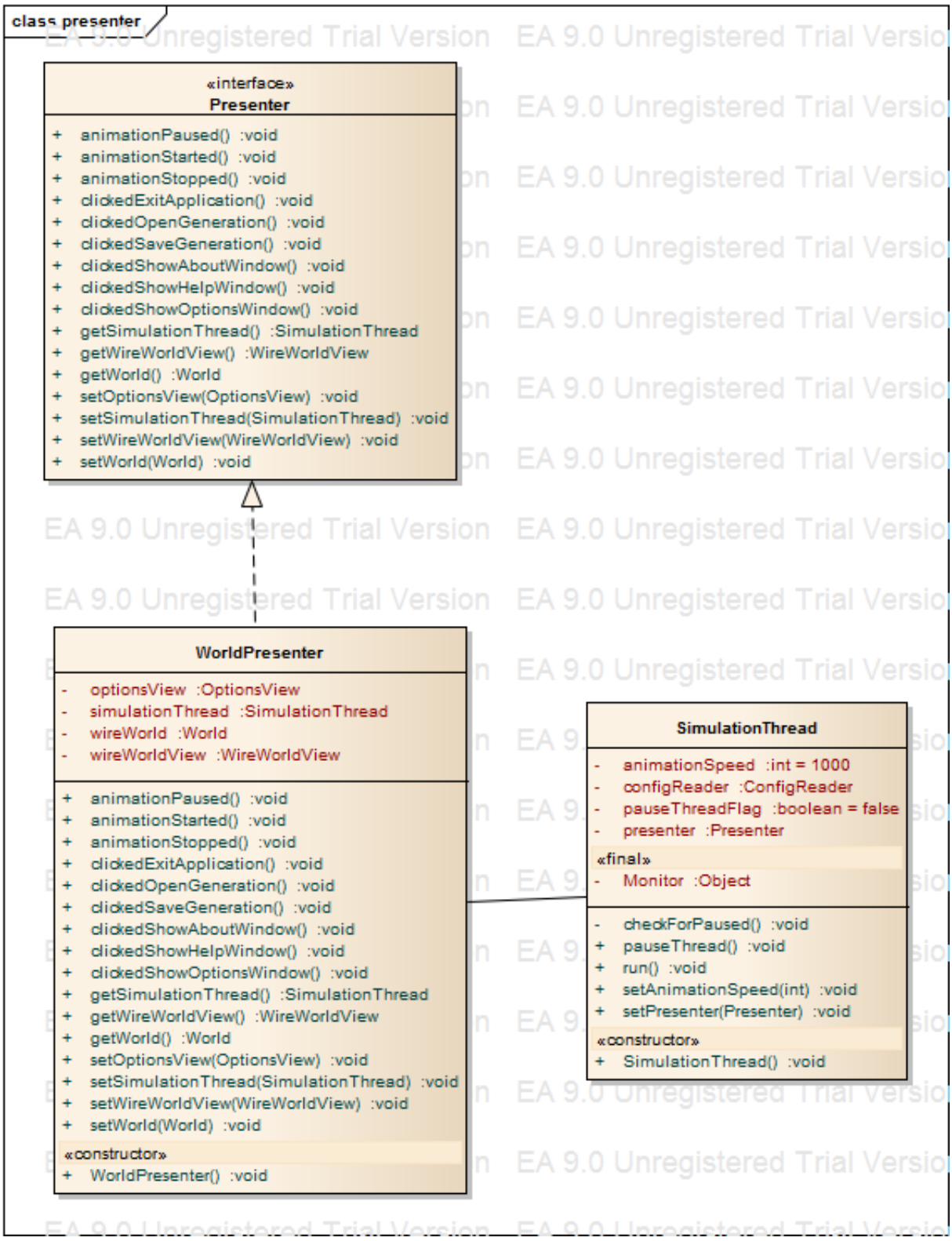
Pakiet core:



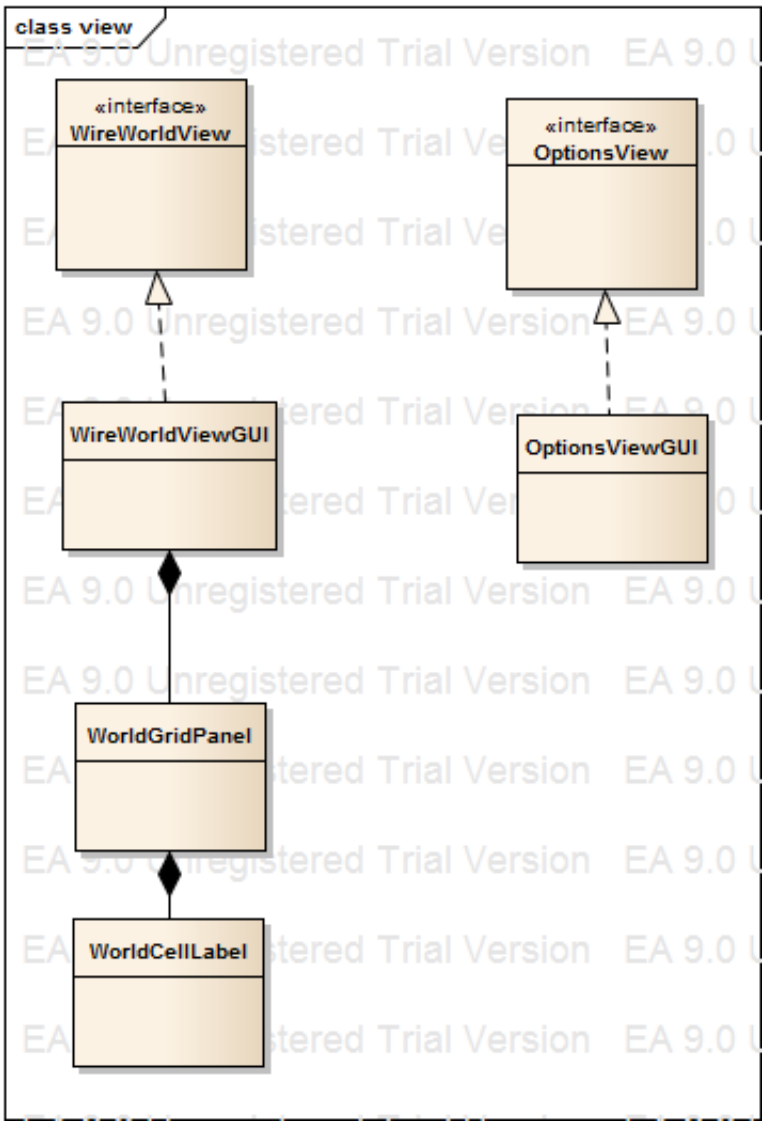
Pakiet model:



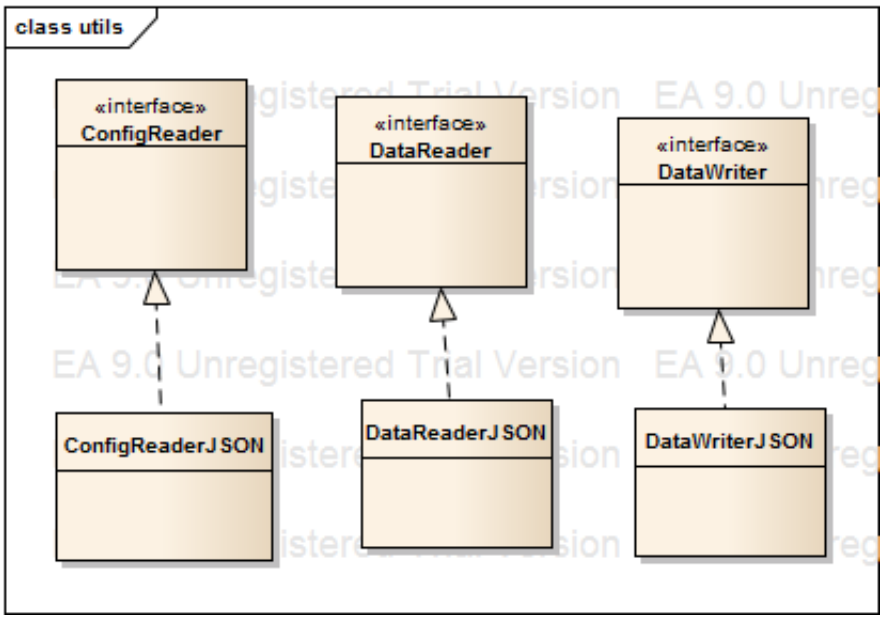
Pakiet presenter:



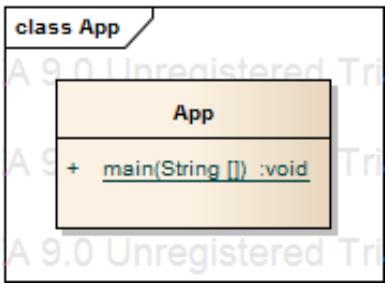
Pakiet view:


class view
«interface»
WireWorldView
«interface»
OptionsView
WireWorldViewGUI
OptionsViewGUI
WorldGridPanel
WorldCellLabel
EA 9.0 Unregistered Trial Version

Pakiet utils:



Główna klasa aplikacji:



4 Testowanie

Program został przetestowany poprzez wprowadzanie zmian w plikach z danymi oraz wybieranie wszystkich dostępnych opcji w GUI. Powstały trzy klasy testowe wykorzystujące Junit do testów jednostkowych.