GrawitexFX – Sprawozdanie

Szymon Masłowski, Adam Olękiewicz

13 czerwca 2017

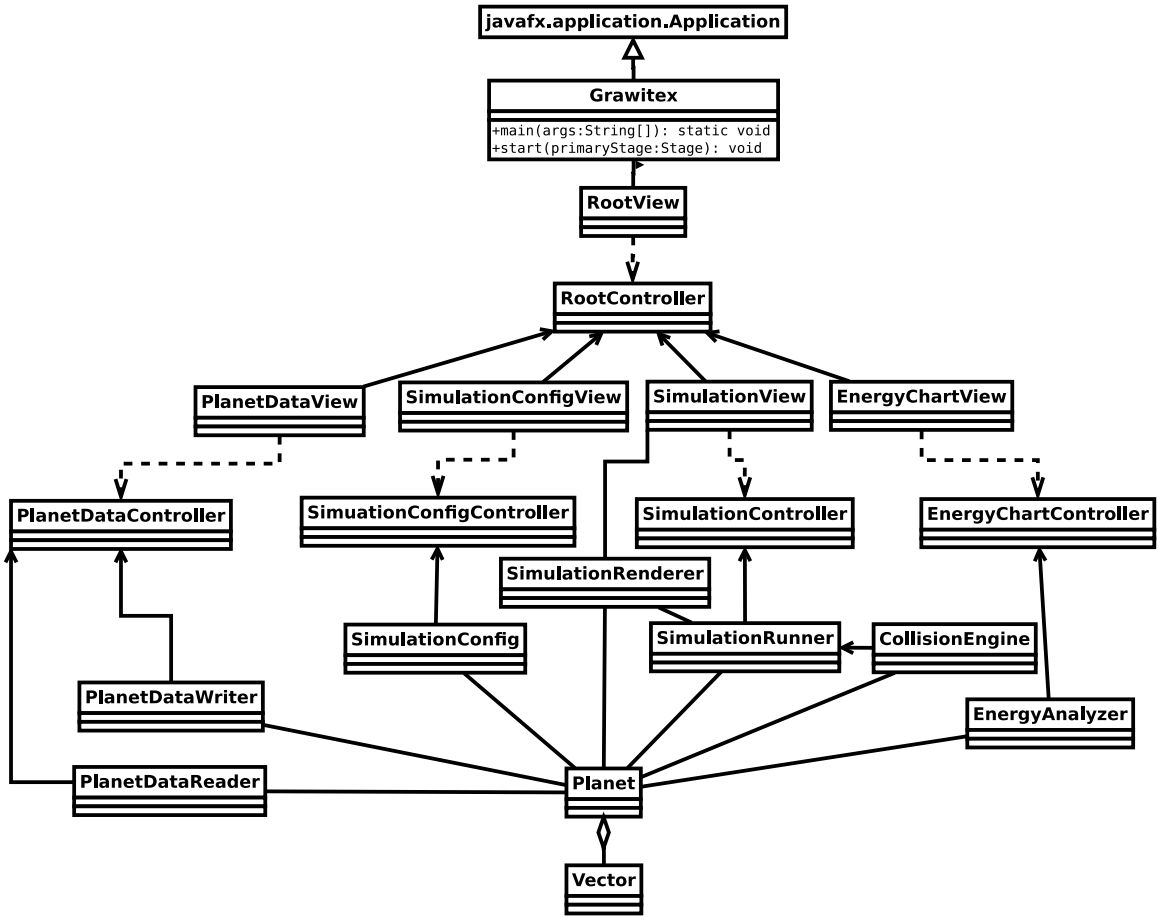
Spis treści

1 Informacje ogólne

GrawitexFX jest programem symulującym wpływ wzajemnych oddziaływań grawitacyjnych na stan układu n ciał - planet o zadanych parametrach. Dokonuje symulacji położenia obiektów w kolejnych krokach czasowych zgodnie z zadanymi danymi. Program został napisany w języku **Java** z wykorzystaniem biblioteki **JavaFX**. Dzięki Java Virtual Machine można z niego korzystać na wszystkich platformach wspierających Javę.
Aplikacja jest przeznaczona wyłącznie do użytku dydaktycznego, a jej kod został udostępniony pod adresem **http://www.github.com/Dragerus/GrawitexFX**.

2 Dynamika specyfikacji implementacyjnej

2.1 Specyfikacja początkowa



Założeniem tej struktury był model **MVC** – Model View Controller. Każdy element dziedziny(planety, symulacja, dane dotyczące energii planet) miały swoją klasę widoku, która miała zajmować się odpowiadaniem na akcje odbywające się przez GUI. To kierowane było by do odpowiedniego kontrolera, który przetworzył by dane i sięgnął bezpośrednio do obiektów dziedziny.

2.2 Specyfikacja końcowa

2.2.1 dataIO

Pakiet odpowiedzialny za możliwość importu i eksportu danych, z i do wybranego przez użytkownika pliku. Składa się z dwóch klas:
PlanetDataWriter i PlanetDataReader.

2.2.2 simulator

To mózg i centralna część projektu i logiki. W klasie SimulationRunner znajduje się główna pętla, w której zlecane jest obliczenie położenia planet i ich energii.
SimulationConfig, SimulationRunner, ColisionEngine

2.2.3 gui

Chcąc zachować odrębność samego symulatora od warstwy prezentacji, przenieśliśmy większość ciał metod do klasy Grawitex, zostawiając w RootView tylko wywołania metod obsługujących odpowiednie zdarzenia. RootControler, Grawitex(uruchamia)

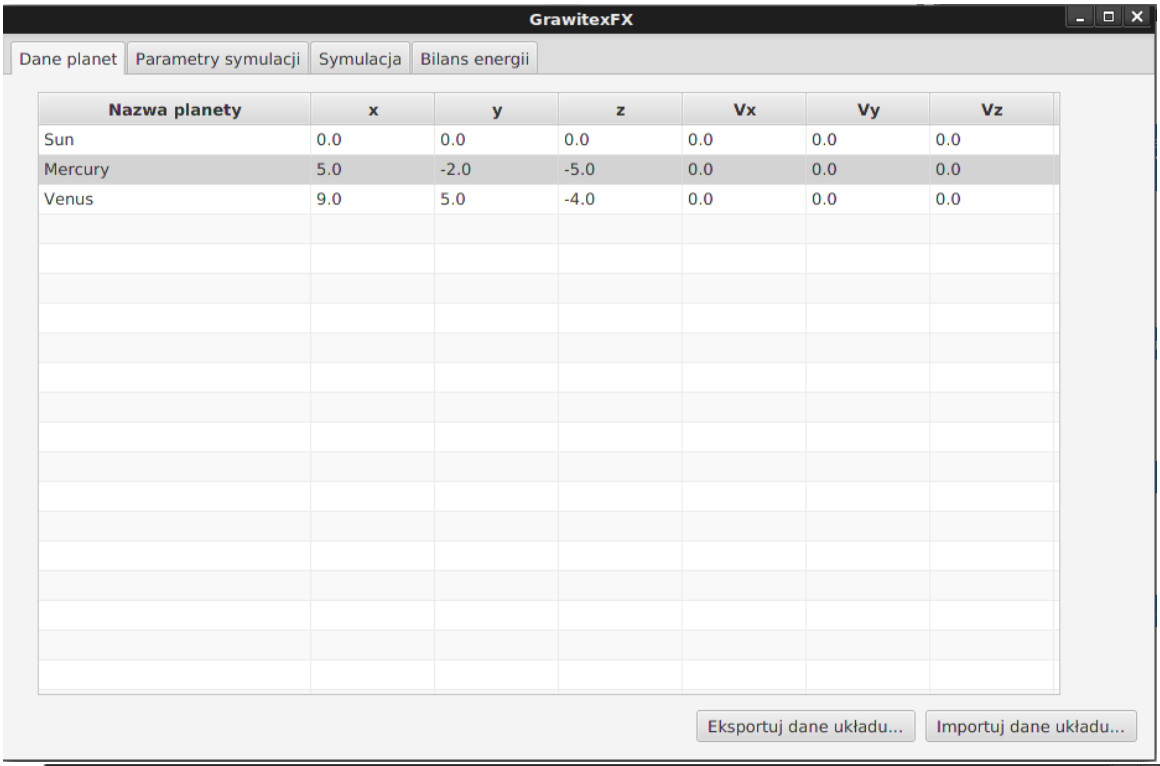
2.2.4 fancyPresenters

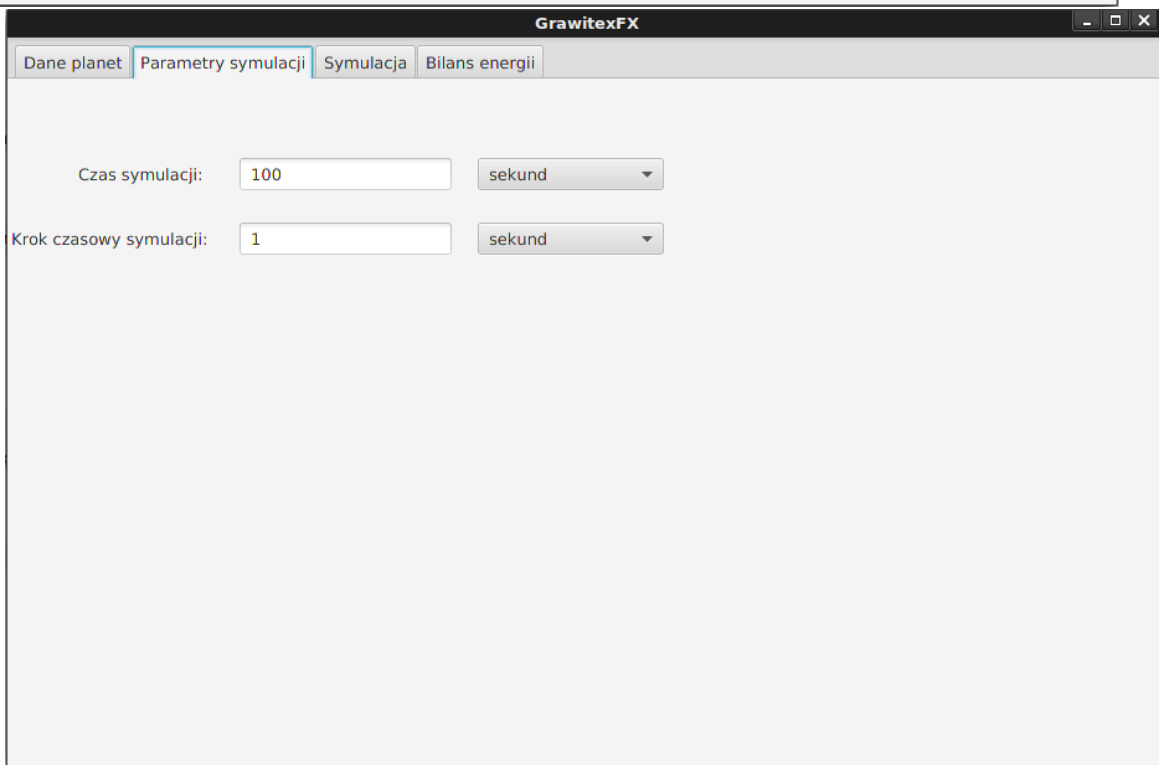
Klasy zajmujące się prezentacją dancych z głębi programu. Energy pozwala na sprawdzenie, na ile zaimplementowana metoda spełnia założenia fizyki – energia w układzie jest stała. Do tego dochodzi klasa zajmująca się przygotowaniem danych oraz prezenter położenia planet w zakładce "Symulacja".
EnergyChart, EnergyAnalyzer, SimulationRenderer

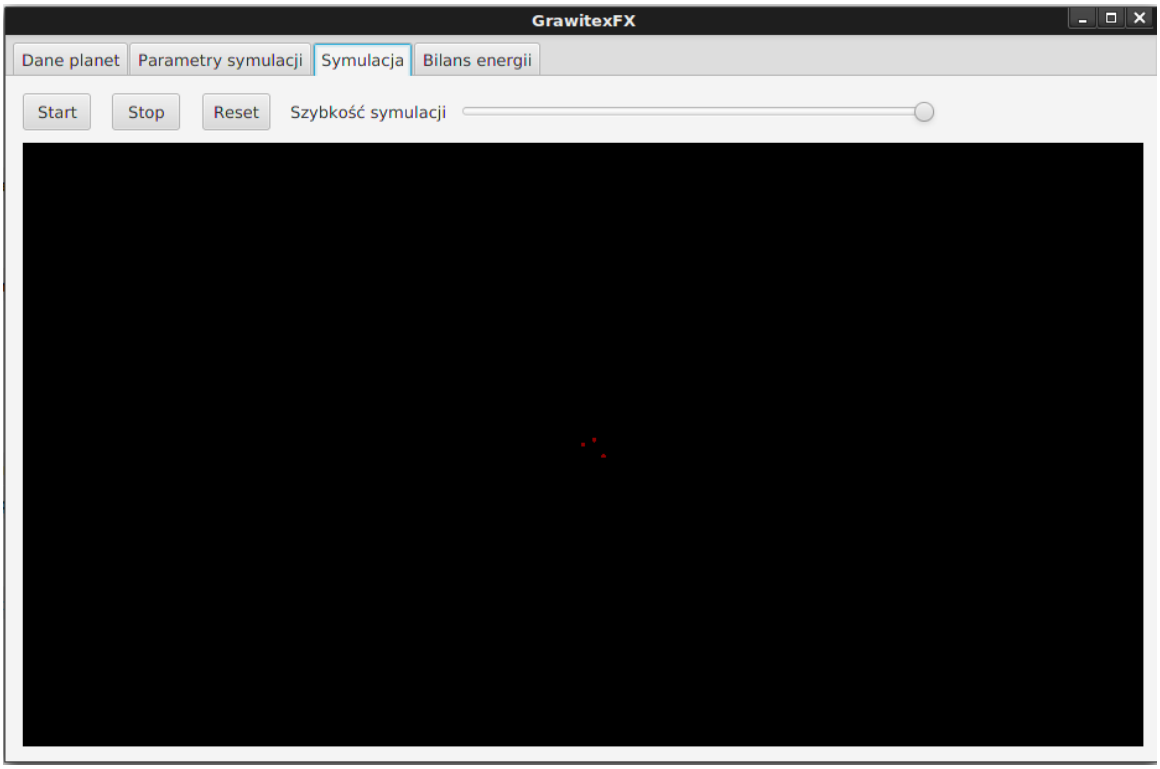
2.3 Podsumowanie i uzasadnienie różnic

Dużym powodem dla zmian w strukturze klas było to, że wbrew domysłom technologia FX realizowała całą kontrolę zdarzeń z GUI w jednym kontrolerze. Dlatego klasy (PlanetData, SimulationConfig, Simulation, EnergyChart)Controller zostały zastąpione jedną klasą RootController. Oboje nie jesteśmy entuzjastami języka Java, więc struktura też nie była od razu przystosowana do użytego języka, powodowało to konieczność pewnych zmian w trakcie tworzenia aplikacji.

3 Interfejs graficzny







4 Testowanie

4.0.1 Użyte narzędzia

Utworzone klasy zostały poddane testowaniu

- jednostkowemu przy użyciu wbudowanego mechanizmu do budowania testów – jUnit,
- nieformalnych testów sprawdzających podczas powstawania systemu,

Przypadki testowe nazywane są tak, aby wiadomo było od razu było wiadomo, jakiego wyniku się spodziewać, bądź jakie zachowanie jest testowane, np. @Test public void shouldGravitatePlanetsTowardsEachOther().

5 Podsumowanie projektu i wnioski

Realizacja projektu była dużą nauką praktyki. Począwszy od zapoznania się z dziedziną, którą później należało modelować, przez projekt funkcji i budowy systemu, do implementacji i współpracy przy kodzie. Artefakt z wykonania projektu – program w Javie, został wykonany, jednak to nie jedyna wartość tego przedsięwzięcia. Nabyte zostały umiejętności podziału zadań i wspólnego wytyczania wizji, tego jak ma wyglądać ostateczny "produkt". Dodatkowym plusem jest możliwość przećwiczenia w praktyce stosowania systemu kontroli wersji, a tam rozwiązywania konfliktów w kodzie, oraz dotyczących zamysłu na konkretne rozwiązania.
W trakcie tworzenia aplikacji, stale następowały zmiany w jej budowie i sposobie funkcjonowania, to dowodzi ogromnej potrzeby wcześniejszego zaznajomienia się z technologią, używanymi bibliotekami oraz podobnymi już istniejącymi rozwiązaniami. Niedopilnowanie tego, powoduje konieczność wymyślania rozwiązań, na etapie, kiedy część aplikacji już funkcjonuje i nie zawsze jest bardzo podane na zmiany.
Zakończony projekt będzie teraz cieszył oko, prezentując się na liście repozytoriów.