Sprawozdanie końcowe
Projekt *Gra Galaxy Defender* w języku Java

Jakub Czajka (299239)
Daniel Daczko (299241)

30 maja 2019

Spis treści

1 Wprowadzenie teoretyczne

Naszym zadaniem było stworzenie gry typu *multiplayer shooter*. Nasz program nazwaliśmy *Galaxy Defender*. Wykorzystuje ona grafikę dwuwymiarową. Gra inspirowana jest dwoma klasycznymi tytułami:

- *Space Invaders*, z których czerpie sposób eliminacji wrogów, poruszanie się gracza oraz zbieranie punktów,
- *Tetris*, z których czerpie kształt wrogów oraz ich sposób poruszania się.

Rozgrywka rozpoczyna się od pojawienia się na ekranie dwóch statków przypisanych do graczy. Gracze mogą poruszać się horyzontalnie i ustawiać kąt działka laserowego. Nie mogą oni jednak zamienić się miejscami (tzn. gracz, który zaczął grę po lewej stronie nie może przejść na prawo za gracza, który rozpoczął grę po prawej i na odwrót). Wraz z upływem czasu pojawia się pierwsza fala wrogów.

Zadaniem graczy jest zestrzelenie klocków w jak najkrótszym czasie. Za każde zbicie użytkownik otrzymuje punkty. Wraz z każdą kolejną minutą gry gra ulega utrudnieniu. Klocki zmniejszają się, zaczynają poruszać się szybciej i tworzyć bardziej złożone formy.

Jeżeli wróg dotrze do linii statków, zostają odjęte punkty od wyników obydwu graczy. Gra kończy się automatycznie, kiedy jeden z wyników wskaże liczbę ujemną. Dodatkowo gracze w dowolnej chwili mogą zakończyć grę i zapisać jej wynik w celu późniejszej kontynuacji.

Rysunek 1: Ikona programu

2 Jakie założenia wstępne udało się spełnić?

Udało nam się zaimplementować w programie wszystkie przewidywane funkcjonalności, jest to:

1. Możliwość rozpoczęcia nowej gry.
2. Wczytywanie z plików domyślnej konfiguracji programu.
3. Możliwość wczytania gry z pliku tekstowego o własnym rozszerzeniu *Galaxy Defender Game (*.gdg)*.
4. Możliwość wprowadzania imion graczy.
5. Możlwiość sterowania statkami kosmicznymi.
6. Wyświetlanie na bieżąco zdobytych przez graczy punktów.
7. Wyświetlanie czasu gry.
8. Możliwość zatrzymania gry.
9. Możliwość wznowienia gry.
10. Możliwość zapisania aktualnego stanu gry do pliku tekstowego w rozszerzeniu programu *Galaxy Defender Game (*.gdg)*.
11. Obsługa błędów powstałych na skutek wczytania niepoprawnej zawartości pliku tekstowego lub problemem z dostępem do niego.
12. Obsługa błędów powstałych na skutek wprowadzania niepoprawnych ciągów znaków w pole *imię gracza*.
13. Możliwość wyświetlenia instrukcji do gry.
14. Zaprogramowanie utrudnienia gry postępującego w czasie.

3 Jakich zadań nie udało się zrealizować?

Niestety nie wszystkie założenia i cele zostały przez nas w pełni zrealizowane. Jest to przede wszystkim:

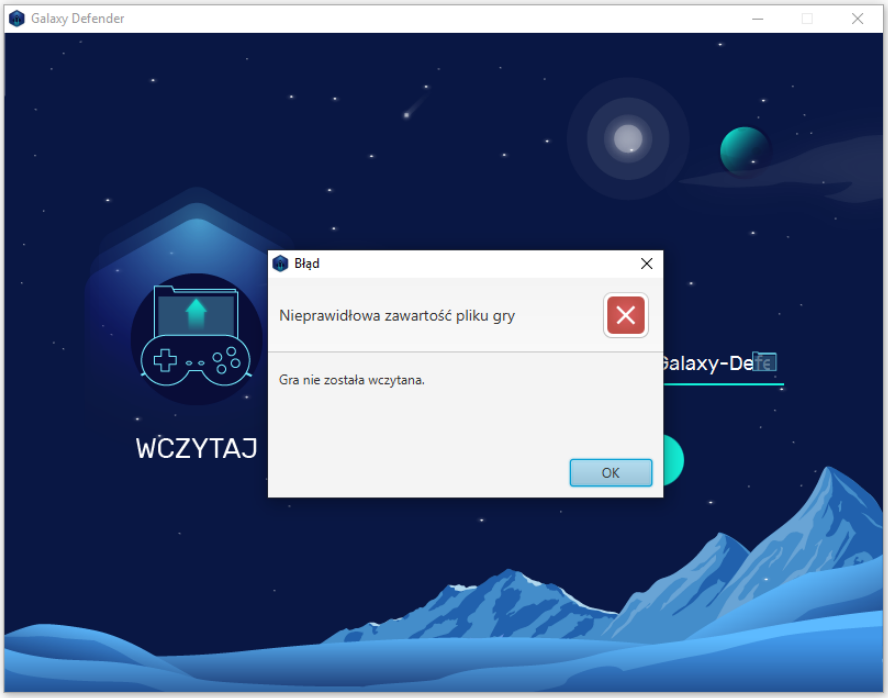
1. Zachowanie systematyczności zamieszczania testów programu w repozytorium.
2. Uniknięcie zmian podczas kodowania w stosunku do specyfikacji implementacyjnej.
3. Przetestowanie działania programu w systemie operacyjnym MacOS X.
4. Dodanie animacji podczas niszczenia klocków.

5. Dodanie animacji podczas poruszania statkami.
6. Usytuowanie miejsca startowego laserów pod statkami.
7. Unormowanie prędkości laserów.
8. Dopracowanie graficznego przedstawienia kolizji statków.
9. Dodanie efektów dźwiękowych do programu.
10. Dodanie możliwości zatrzymania gry klawiszem ESC.

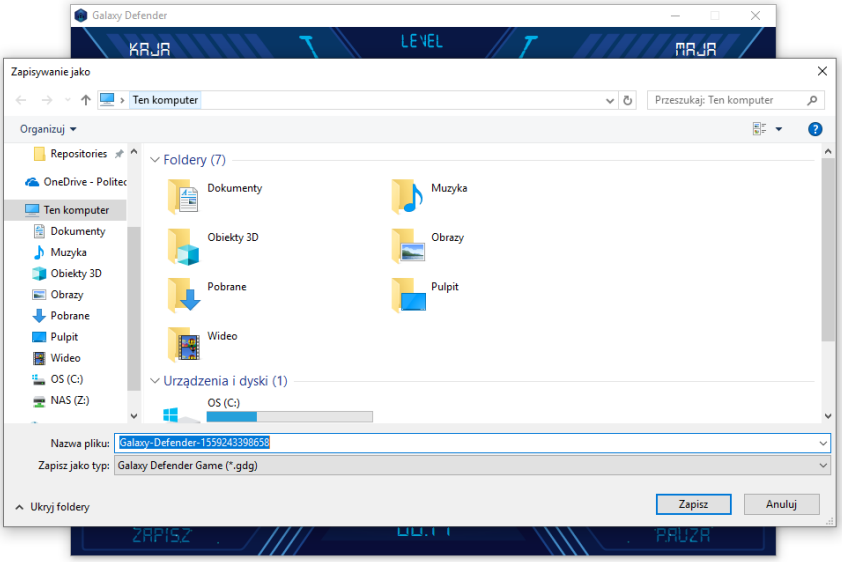
4 Przykładowe uruchomienia

Poniższe ilustracje przedstawiają przykładowe ekrany działania programu:

Rysunek 2: Ekran gry



Rysunek 3: Ekran wczytywania i alert błędu

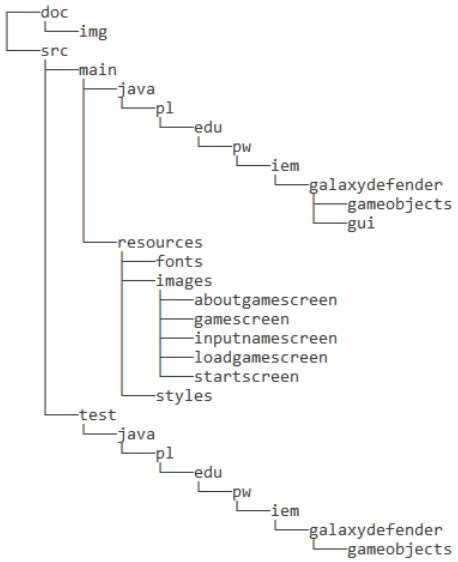


Rysunek 4: Okno dialogowe zapisu gry

5 Rozbieżności względem specyfikacji

5.1 Specyfikacja funkcjonalna

Delikatnym modyfikacjom uległa **struktra katalogów**, która obecnie wygląda w następujący sposób:



Rysunek 5: Struktura katalogów w projekcie

5.2 Specyfikacja implementacyjna

1. Rozszerzenie klas o dodatkowe metody oraz pola, tak aby kod był napisany w sposób jak najbardziej optymalny i uniwersalny.
2. Zmiana planowanych metod w klasach spowodowana nieznajomością mechanizmów Javy.
3. Zmiana w klasie `AnimationThread`:
   (a) zmiana nazwy na `GameAnimation`, aby klasa nabrała nazwy znaczącej,
   (b) rozszerzanie klasy `AnimationTimer` spowodowane poznaniem plusów podanej klasy. W odróżnieniu do Threada, klasa ta działa dokładnie raz na klatkę wykonaną w scenie, co przynosi wiele korzyści przy rozpatrywaniu trafienia klocka.
4. Zmiana układu budowy GUI.
5. Wykorzystanie klasy Button do utworzenia przycisków spowodowane zapoznaniem się z mechanizmem plików .css współpracujących z JavaFX.

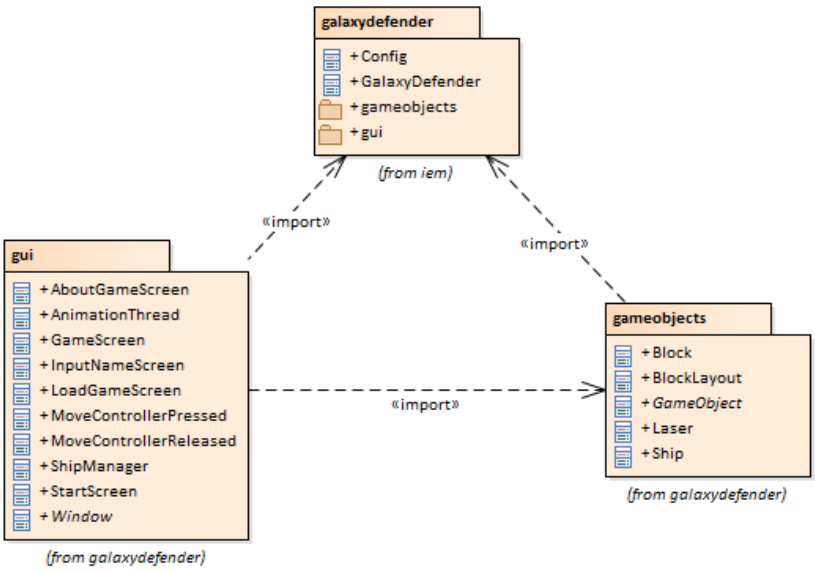
6. Wykorzystanie biblioteki JUnit.
7. Zmiana konwencji pisania testów. Klasy testujące mają nazwy klas, które testują z dopiskiem Test.

6 Mechanizmy Java 1.8 zastosowane w projekcie

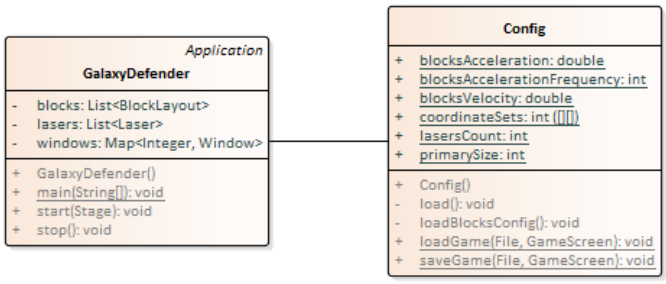
Jedynym mechanizmem pochodzącym z Javy w wersji 1.8 jest metoda `sizeToScene` klasy Stage (`javafx.stage`). Ustawia ona szerokość i wysokość obiektu klasy `Window`, aby dopasować rozmiar zawartości tej sceny do okna.

7 Aktualna wersja diagramu klas

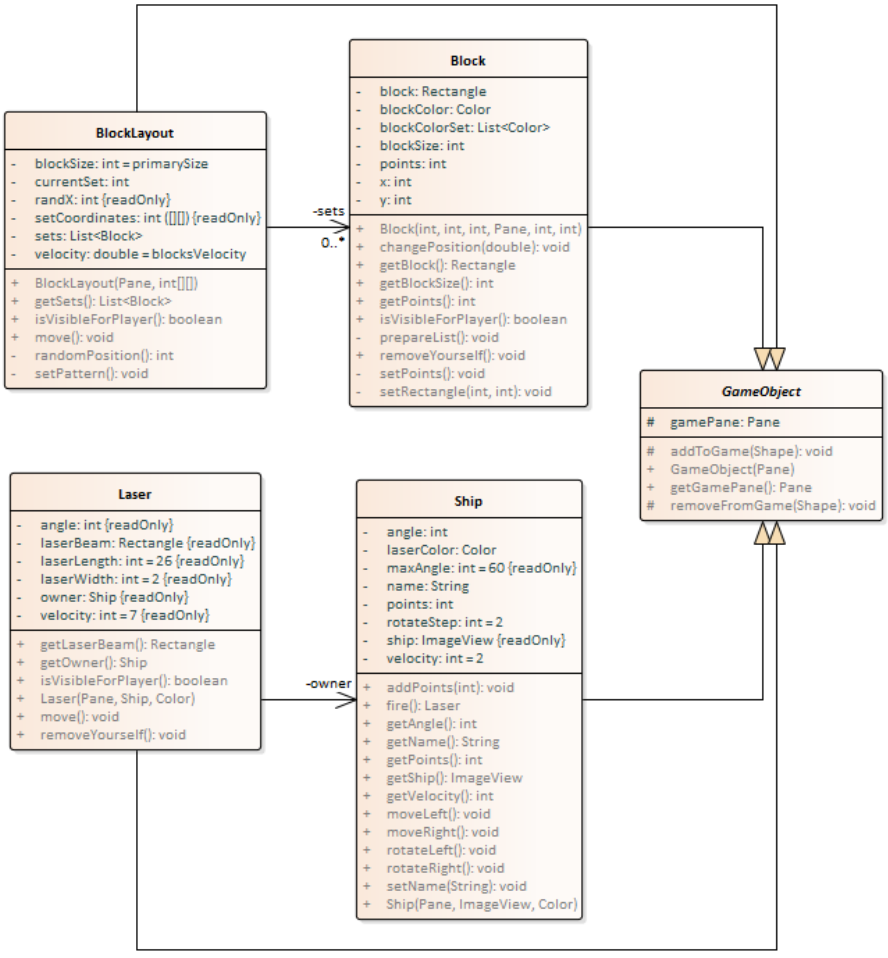
Ze względu na rozmiar diagramu klas rozbiliśmy go na trzy diagramy dla poszczególnych pakietów:



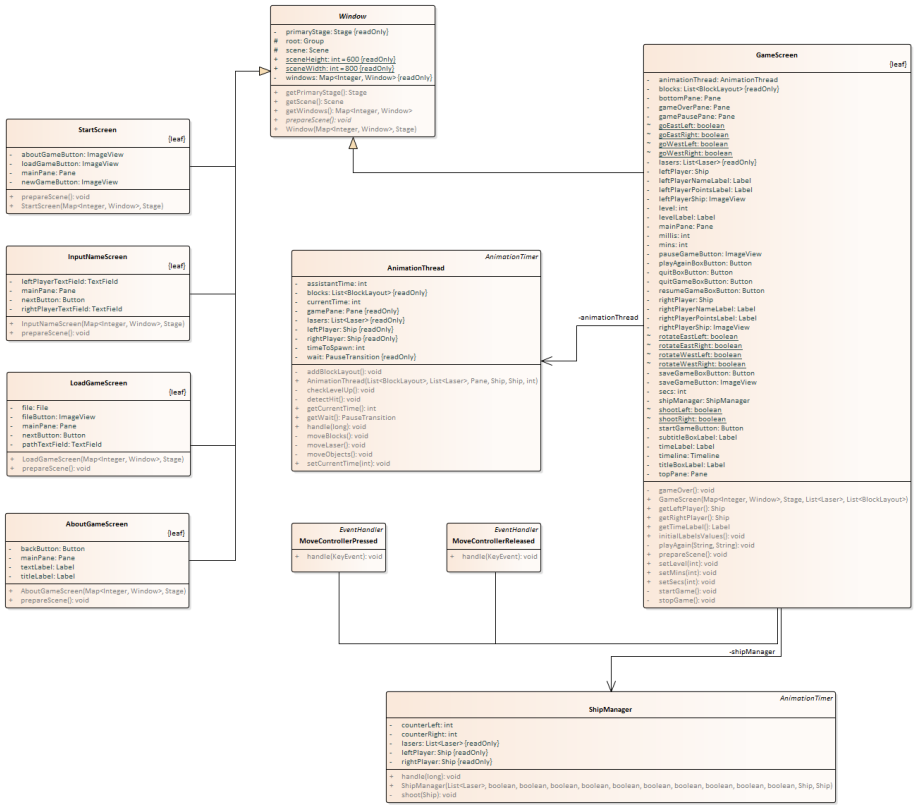
Rysunek 6: Diagram pakietów



Rysunek 7: Diagram klas w pakiecie główny *galaxydefender*



Rysunek 8: Diagram klas w pakiecie *galaxydefender.gameobjects*



Rysunek 9: Diaram klas w pakiecie *galaxydefender.gui*

8 Testy programu

Funkcjonalność poszczególnych elementów programu sprawdzaliśmy na bieżąco podczas implementacji. Jeżeli dany komponent działał zgodnie z założeniami, dołączaliśmy go do programu. W tym celu pomocne okazało się wykorzystanie testów białej skrzynki. Dodatkowo dla klas obiektów gry napisaliśmy testy jednostkowe.

Główne testy nastąpiły po zaimplementowaniu podstawowych mechanik rozgrywki. W tym momencie nasze testy polegały na przeprowadzeniu rozgrywki i sprawdzaniu zachowań programu podczas różnych okoliczności.

Ostatecznie program oddaliśmy w ręce znajomych, których subiektywna ocena oraz uwagi pozwoliły nam na naprawienie i ulepszenie działania gry. Wszystkie testy można znaleźć w katalogu `src/test/java` w naszym repozytorium.

9 Podsumowanie i wnioski

Otrzymane zadanie od początku staraliśmy się wykonać z należytym zaangażowaniem i precyzyjnością. Czyniliśmy wszelakie zabiegi, aby spełnić wszystkie postawione nam wymagania wobec działania programu. Po dokładnej analizie doszliśmy do wniosku, że udało nam się sprostać zadaniu. Program działa poprawnie i jest przenośny. Został on przez nas rozszerzony w rozmaitych detalach.

Największą trudność sprawiło nam napisanie specyfikacji i przewidzenie struktury kodu, czego przyczyną mogła być krótka znajomość programowania w języku Java. Niestety nie udało nam się uniknąć zmian w stosunku do wcześniej przygotowanych dokumentów.

Nasza współpraca przebiegała pomyślnie. Dopełnialiśmy się na każdym etapie wykonywania projektu. Pozytywnie oceniamy systematyczność naszej pracy i atmosferę wspólnego działania.

Programowanie zdecydowanie ułatwiło nam korzystanie z rozproszonego systemu kontroli wersji Git i zintegrowanego środowiska programistycznego NetBeans. Bardzo przydatnym narzędziem automatyzującym budowę oprogramowania okazał się Apache Maven.

W trakcie realizacji zadania zrozumieliśmy kwintesencję pisania specyfikacji przed rozpoczęciem samego programowania, co na pewno przyczyni się do bardziej profesjonalnego rozwiązywania kolejnych problemów programistycznych, które staną na naszej drodze podczas studiów i później w pracy zawodowej.