Specyfikacja implementacyjna
Projekt *Gra Galaxy Defender* w języku Java

Jakub Czajka (299239)
Daniel Daczko (299241)

14 maja 2019

Spis treści

1 Informacje ogólne

1.1 Ogólne parametry uruchomieniowe programu

Język programowania: *Java*.

Środowisko uruchomieniowe: *dowolny system operacyjny obsługujący Javę*.

Wielkość okna: *800 x 600 px*.

Położenie okna po uruchomieniu: *środek ekranu*.

1.2 Konwencja pisania kodu

Kod programu będzie pisany zgodnie z poniższymi zasadami:

- Kod pisany jest z poszanowaniem zasad czystego kodu.
- Nazwy są nazwami znaczącymi.
- Nazwy są pisane w języku angielskim.
- Nazwy klas są pisane w konwencji *CamelCase*.
- Nazwy metod i zmiennych są pisane w konwencji *lowerCamelCase*.
- Zawsze należy pisać nawiasy klamrowe, nawet jeśli ciało zawiera tylko pojedynczą linię.

2 Opis modułów i danych

2.1 Moduły w programie

Nasz program jest podzielony na cztery odrębne moduły:

2.1.1 Graficzny interfejs użytkownika

Moduł ten odpowiedzialny jest za interaktywną i graficzną komunikację z użytkownikiem programu. Jego głównym zadaniem jest umożliwienie korzystania z oprogramowania w jak najłatwiejszy sposób, tak aby uniknąć błędów powstałych na skutek podania niewłaściwych argumentów. Z uwagi na charakter projektu jest on również odpowiedzialny za wyświetlania elementów kluczowych dla gry. Moduł ten jako główny połączony jest ze wszystkimi innymi modułami wspierającymi.

2.1.2 Animacje

Moduł ten odpowiedzialny jest za wyświetlanie animacji związanych z interakcją użytkownika, a dokładniej funkcjonalnością oddanie strzału przez gracza. Posiada on również elementy logiki gry. Jest połączony z modułem graficznego interfejsu użytkownika i sterowaniem programu.

2.1.3 Sterowanie programem

Moduł ten odpowiedzialny jest za komunikację z urządzeniami wejściowymi komputera jakimi są mysz i klawiatura. Powiązany jest z modułem graficznego interfejsu użytkownika i animacjami.

2.1.4 Konfiguracja

Moduł ten odpowiedzialny jest za wczytywanie i zapisywanie danych do pliku. W szczególności za wczytywanie konfiguracji programu i wcześniej zapisanej gry w celu jej kontynuacji lub zapisu niedokończonej rozgrywki. Moduł ten powiązany jest z modułem graficznego interfejsu użytkownika.

3 Opis klas

3.1 GalaxyDefender

Pakiet: pl.edu.pw.iem.galaxydefender

Dziedziczy: `Application`

Konstruktor: bezargumentowy, inicjujący kontener przechowujące obiekty.

Pola:

- `List<Laser> lasers` – lista przechowująca wiązki laserowe znajdujące się na ekranie,
- `List<BlockLayout> blocks` – lista przechowująca formy klocków znajdujących się na ekranie,
- `Map<Window> windows` – mapa przechowująca obiekt zawierające sceny.

Metody:

- **Prototyp:** `public void start()`
  Zadanie: przygotowanie ramki okna aplikacji oraz wyświetlenie jej.
- **Prototyp:** `public void stop()`
  Zadanie: zatrzymanie pracy programu oraz wszystkich utworzonych wątków.
- **Prototyp:** `public void main()`
  Zadanie: uruchomienie programu, wywołanie metod inicjujących kontenery przechowujące obiekty.

3.2 GameObject

«klasa abstrakcyjna»

Pakiet: pl.edu.pw.iem.galaxydefender.gameobjects

Konstruktor: otrzymuje zmienną typu `Pane`, ustawia wartość pola `gamePane` na wartość otrzymanej zmiennej.

Pola:

- `Pane gamePane` – schemat układu ekranu na którym wyświetlane są animacje.

Metody:

- **Prototyp:** `public void addToGame(Shape gameObject)`
  Zadanie: dodanie obiektów typu `Shape` do panelu gry.
- **Prototyp:** `public void removeFromGame(Shape gameObject)`
  Zadanie: usunięcie obiektów typu `Shape` z panelu gry.

3.3 Ship

Pakiet: pl.edu.pw.iem.galaxydefender.gameobjects

Dziedziczy: `GameObject`

Konstruktor: otrzymuje zmienną typu `int` przechowującą horyzontalną pozycję obiektu na planszy, zmienną typu `Image` przechowującą graficzną reprezentację statku i zmienną typu `Pane`, wywołuje konstruktor nadklasy i inicjuje zmienne oraz dodaje je do sceny.

Pola:

- `int angle` – wartość kąta obrotu statku,
- `int maxAngle` – graniczna wartość kąta obrotu statku,
- `int points` – liczba punktów,
- `ImageView ship` – obiekt umożliwiający wyświetlenie pliku graficznego,
- `Image shipImage` – obiekt reprezentujący plik graficzny,
- `int velocity` – wartość wektora prędkości statku.

Metody:

- **Prototyp:** `public void addPoints()`
  Zadanie: dodanie lub odjęcie określonej liczby punktów do zmiennej `points`.

- **Prototyp:** `public void addShipToGame(ImageView ship)`
  Zadanie: dodanie graficznej reprezentacji statku do planszy.
- **Prototyp:** `public Laser fire()`
  Zadanie: utworzenie obiektu typu `Laser` o wskazanych parametrach.
  Wartość zwracana: obiekt typu `Laser`.
- **Prototyp:** `public void moveLeft()`
  Zadanie: przesunięcie obiektu w lewo.
- **Prototyp:** `public void moveRight()`
  Zadanie: przesunięcie obiektu w prawo.
- **Prototyp:** `public void rotateLeft()`
  Zadanie: obrócenie obiektu w lewo.
- **Prototyp:** `public void rotateRight()`
  Zadanie: obrócenie obiektu w prawo.

3.4 Laser

Pakiet: pl.edu.pw.iem.galaxydefender.gameobjects

Dziedziczy: `GameObject`

Konstruktor: otrzymuje zmienną typu `Ship` wskazującą na obiekt strzelający i zmienną typu `Pane`, wywołuje konstruktor nadklasy i inicjuje zmienne oraz dodaje je do sceny.

Pola:

- `int angle` – wartość kąta wystrzału wiązki laserowej,
- `Rectangle laserBeam` – obiekt reprezentujący graficznie wiązkę laserową,
- `Color laserColor` – kolor lasera,
- `Ship owner` – właściciel wiązki laserowej,
- `int velocity` – wartość wektora prędkości wiązki laserowej.

Metody:

- **Prototyp:** `public void move()`
  Zadanie: przesuwanie graficznej reprezentacji wiązki laserowej po ekranie.
- **Prototyp:** `public boolean isVisible()`
  Zadanie: sprawdzenie czy graficzna reprezentacja wiązki ma zostać przesunięta.
  Wartość zwracana: `true` w przypadku, gdy wiązka nie wyszła poza granice ekranu; `false` w przeciwnym przypadku.

3.5 BlockLayout

Pakiet: pl.edu.pw.iem.galaxydefender.gameobjects

Dziedziczy: `GameObject`

Konstruktor: otrzymuje zmienną typu `Pane`, wywołuje konstruktor nadklasy, losuje kształt obiektu spadającego i dodaje go do sceny.

Pola:

- `List<Block> sets` – lista przechowująca klocki znajdujące się na ekranie,
- `int velocity` – wartość wektora prędkości spadania form klocków.

Metody:

- **Prototyp:** `public void move()`
  Zadanie: przesuwanie graficznej reprezentacji konstrukcji klocków po ekranie.
- **Prototyp:** `public void setPosition()`
  Zadanie: losowanie ułożenia konstrukcji klocków w graficznej reprezentacji.

3.6 Block

Pakiet: pl.edu.pw.iem.galaxydefender.gameobjects

Dziedziczy: `GameObject`

Konstruktor: otrzymuje zmienne typu `int` przechowujące pozycje w konstrukcji obiektu spadającego i długość krawędzi, inicjuje pola oraz obiekt typu `Rectangle`.

Pola:

- `int x` – współrzędna horyzontalna położenia klocka w konstrukcji,
- `int y` – współrzędna wertykalna położenia klocka w konstrukcji,
- `int blockSize` – wartość długości krawędzi klocka,
- `Color blockColor` – kolor klocka,
- `Rectangle block` – obiekt reprezentujący graficznie pojedynczy klocek.

Metody:

- **Prototyp:** `public void setRectangle()`
  Zadanie: utworzenie obiektu typu `Rectangle` o wskazanych parametrach.
- **Prototyp:** `public void changePosition()`
  Zadanie: przesuwanie graficznej reprezentacji klocka po ekranie.

3.7 MoveControllerPressed

Pakiet: pl.edu.pw.iem.galaxydefender.gui

Implementuje: `EventHandler<KeyEvent>`

Metody:

- **Prototyp:** `public void handle()`
  Zadanie: rozpoznawanie wciskania klawiszy.

3.8 MoveControllerReleased

Pakiet: pl.edu.pw.iem.galaxydefender.gui

Implementuje: `EventHandler<KeyEvent>`

Metody:

- **Prototyp:** `public void handle()`
  Zadanie: rozpoznawanie zwolnienia klawiszy.

3.9 AnimationThread

Pakiet: pl.edu.pw.iem.galaxydefender.gui

Dziedziczy: `Thread`

Konstruktor: otrzymuje kontenery przechowujące obiekty typu `Laser` oraz `BlockLayout` i inicjuje je w obiekcie.

Pola:

- `List<Laser> lasers` – lista przechowująca wiązki laserów znajdujących się na ekranie,
- `List<BlockLayout> blocks` – lista przechowująca formy klocków znajdujących się na ekranie.

Metody:

- **Prototyp:** `public void detectHit()`
  Zadanie: ocenienie czy dana wiązka laserowa uderzyła w klocek, wywołanie metod przyznających punkty i usuwających obiekty z planszy.
- **Prototyp:** `public void moveObjects()`
  Zadanie: przesuwanie graficznej reprezentacji obiektów po ekranie.
- **Prototyp:** `public void moveLaser()`
  Zadanie: przesuwanie graficznej reprezentacji wiązki laserowej po ekranie.

- **Prototyp:** `public void moveBlocks()`
  Zadanie: przesuwanie graficznej reprezentacji konstrukcji klocków po ekranie.
- **Prototyp:** `public void removeObjects()`
  Zadanie: usunięcie obiektów z panelu gry.
- **Prototyp:** `public void run()`
  Zadanie: ocenienie czy dana wiązka laserowa uderzyła w klocek, wywołanie metod przyznających punkty i usuwających obiekty z planszy.

3.10 Config

Pakiet: pl.edu.pw.iem.galaxydefender

Konstruktor: wywołuje metodę wczytującą dane z pliku i inicjującą zmienne.

Pola:

- `int blocksAcceleration` – wartość przyspieszenia klocków,
- `int blocksAccelerationFrequency` – częstotliwość przyspieszenia klocków,
- `int blocksVelocity` – prędkość spadania klocków,
- `int primarySize` – wartość początkowej długości boku kwadratu reprezentującego pojedynczy klocek.

Metody:

- **Prototyp:** `public void load()`
  Zadanie: wczytanie danych konfiguracyjnych z pliku tekstowego.
- **Prototyp:** `public void loadGame()`
  Zadanie: wczytanie zapisanej gry z pliku tekstowego.
- **Prototyp:** `public void saveGame()`
  Zadanie: zapisanie rozpoczętej rozgrywki do pliku tekstowego.

3.11 Window

«klasa abstrakcyjna»

Pakiet: pl.edu.pw.iem.galaxydefender.gui

Konstruktor: wywołuje metodę tworzącą scenę o określonych właściwościach.

Pola:

- `Group root` – obiekt niezbędny do utworzenia sceny,

- `Scene scene` – obiekt reprezentujący graficznie interfejs użytkownika,
- `int sceneHeight` – wysokość okna,
- `int sceneWidth` – szerokość okna.

Metody:

- **Prototyp:** `public void prepareScene()`
  Zadanie: przygotowanie okna do wyświetlenia.

Klasa `Window` jest klasą abstrakcyjną, po której dziedziczą wymienione klasy, odpowiadające poszczególnym ekranom:

- StartScreen,
- InputNameScreen,
- LoadGameScreen,
- AboutGameScreen,
- GameScreen,
- PauseGameScreen,
- WinnerGameScreen,
- GameOverScreen.

Powyższe klasy nie posiadają żadnych pól i różnią się jedynie implementacją odziedziczonej metody `prepareScene()`.

4 Budowa GUI

4.1 Opis pól i słuchaczy

4.1.1 Ekran startowy

- `ImageView newGameButton` – przechwytywacz zdarzenia typu kliknięcie lewego przycisku myszy, zmiana sceny.
- `ImageView loadGameButton` – przechwytywacz zdarzenia typu kliknięcie lewego przycisku myszy, zmiana sceny.
- `ImageView aboutGameButton` – przechwytywacz zdarzenia typu kliknięcie lewego przycisku myszy, zmiana sceny.

4.1.2 Ekran wprowadzania imion graczy

- `TextField playerOneTextField` – przechwytywacz zdarzenia typu wykonana akcja, ustawienie zmiennej w obiekcie Ship.
- `TextField playerTwoTextField` – przechwytywacz zdarzenia typu wykonana akcja, ustawienie zmiennej w obiekcie Ship.
- `ImageView nextButton` – przechwytywacz zdarzenia typu kliknięcie lewego przycisku myszy, zmiana sceny.

4.1.3 Ekran wczytywania gry

- `ImageView nextButton` – przechwytywacz zdarzenia typu kliknięcie lewego przycisku myszy, wywołanie metody odpowiedzialnej za wczytanie stanu gry i zmiana sceny.

4.1.4 Ekran o grze

- `ImageView backButton` – przechwytywacz zdarzenia typu kliknięcie lewego przycisku myszy, zmiana sceny.

4.1.5 Ekran gry

- `ImageView saveGameButton` – przechwytywacz zdarzenia typu kliknięcie lewego przycisku myszy, wywołanie metody odpowiedzialnej za zapis stanu gry.
- `ImageView pauseGameButton` – przechwytywacz zdarzenia typu kliknięcie lewego przycisku myszy, zawieszenie działania programu i zmiana sceny.

4.1.6 Ekran zatrzymania gry

- `ImageView resumeGameButton` – przechwytywacz zdarzenia typu kliknięcie lewego przycisku myszy, wznowienie pracy programu i zmiana sceny.
- `ImageView saveGameButton` – przechwytywacz zdarzenia typu kliknięcie lewego przycisku myszy, wywołanie metody odpowiedzialnej za zapis stanu gry.
- `ImageView quitGameButton` – przechwytywacz zdarzenia typu kliknięcie lewego przycisku myszy, zmiana sceny.

4.1.7 Ekran wygranej

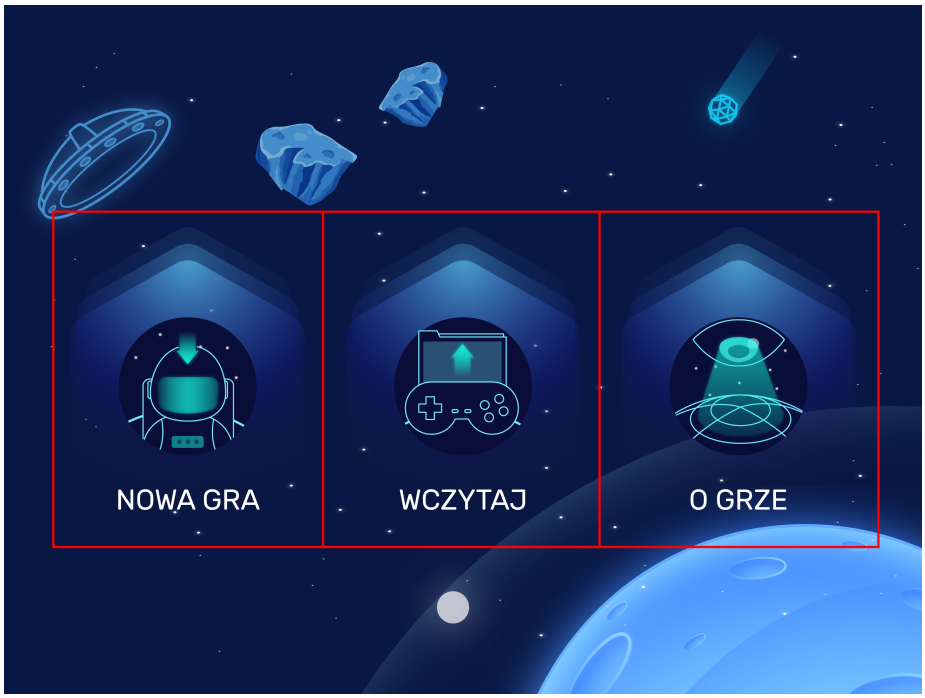
- `ImageView playAgainButton` – przechwytywacz zdarzenia typu kliknięcie lewego przycisku myszy, zmiana sceny i rozpoczęcie nowej rozgrywki.
- `ImageView quitButton` – przechwytywacz zdarzenia typu kliknięcie lewego przycisku myszy, zakończenie działania programu.

4.1.8 Ekran przegranej

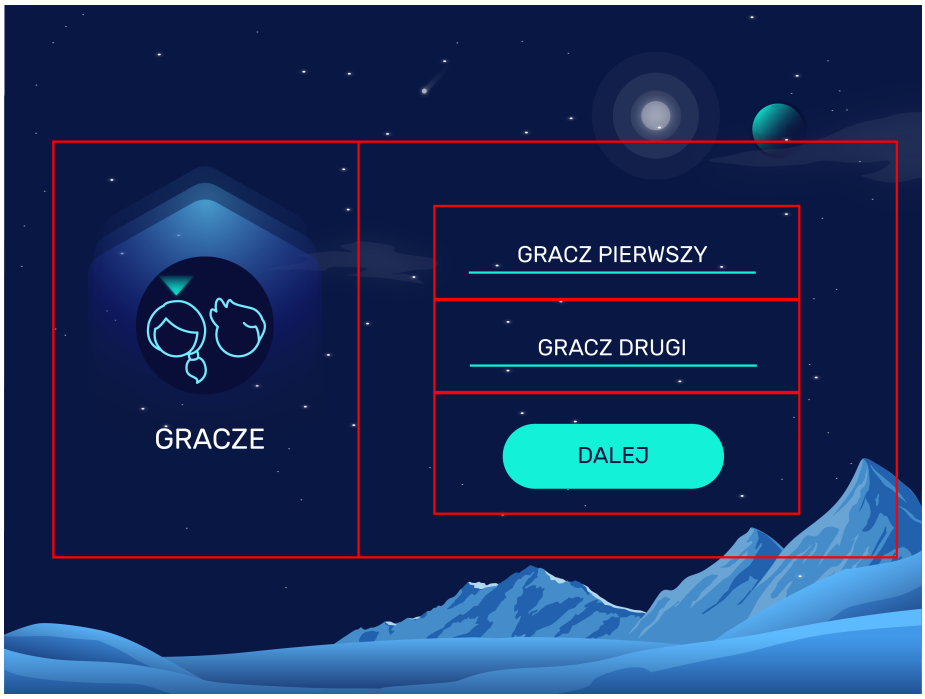
- `ImageView playAgainButton` – przechwytywacz zdarzenia typu kliknięcie lewego przycisku myszy, zmiana sceny i rozpoczęcie nowej rozgrywki.
- `ImageView quitButton` – przechwytywacz zdarzenia typu kliknięcie lewego przycisku myszy, zakończenie działania programu.

4.2 Budowa wewnętrzna GUI

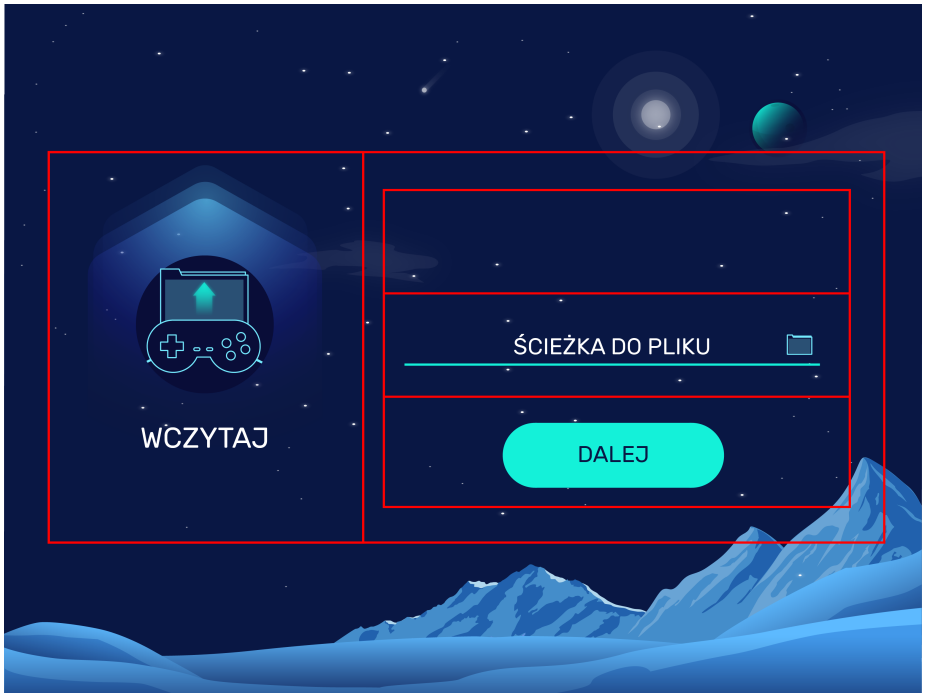
Na poniższych rysunkach przedstawiających główne ekrany programu można zaobserwować podział poszczególnych scen na panele i stopień ich zagnieżdżenia.



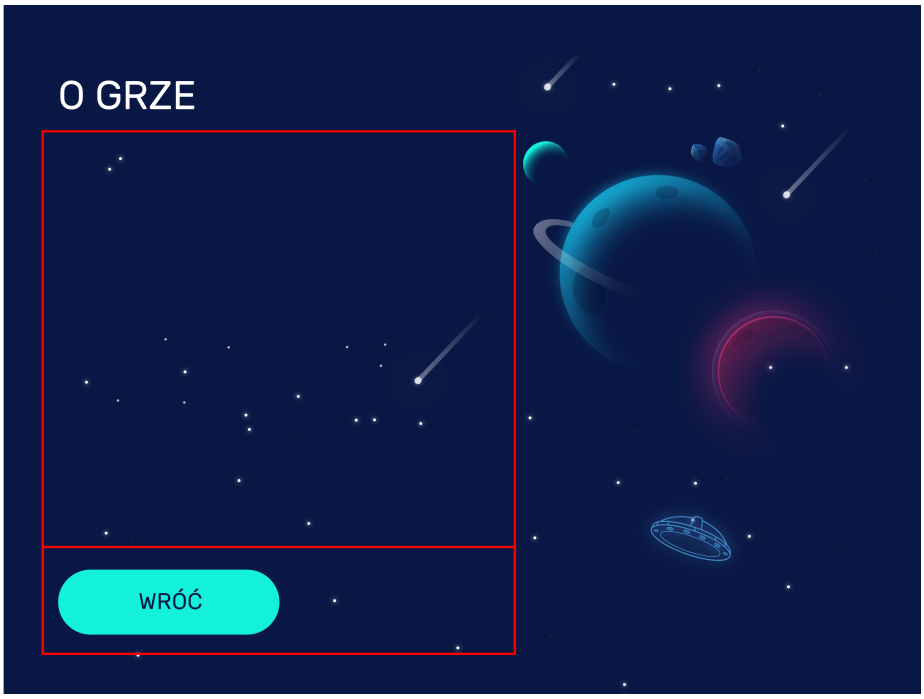
Rysunek 1: Ekran startowy



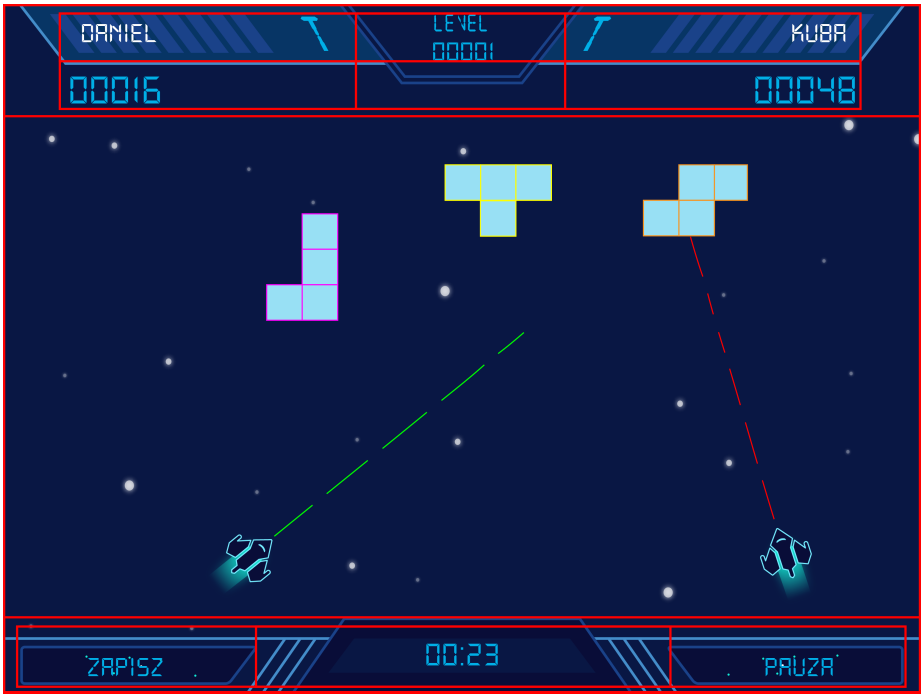
Rysunek 2: Ekran wprowadzania imion graczy



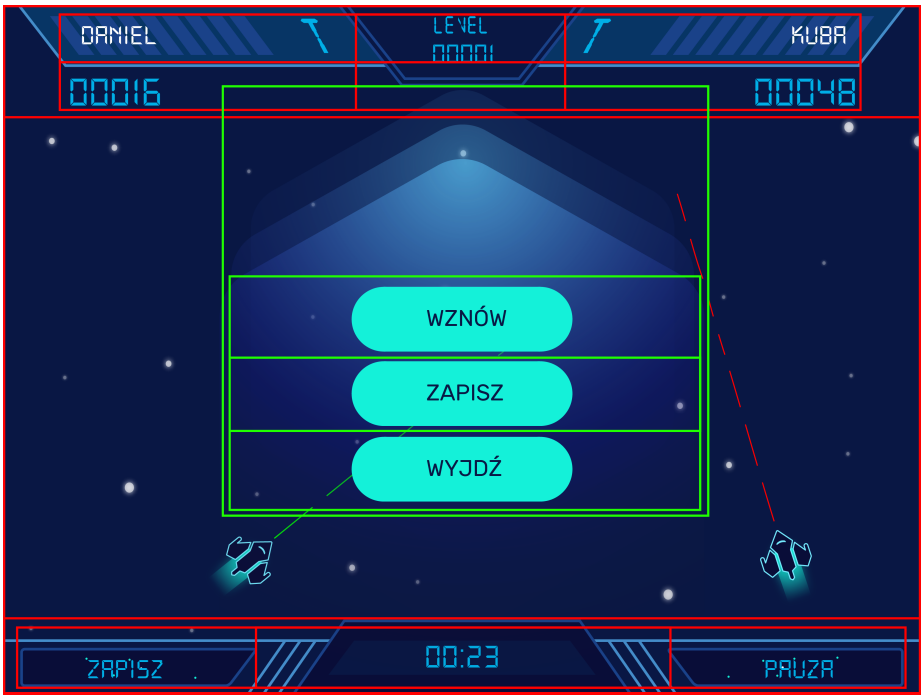
Rysunek 3: Ekran wczytywania gry



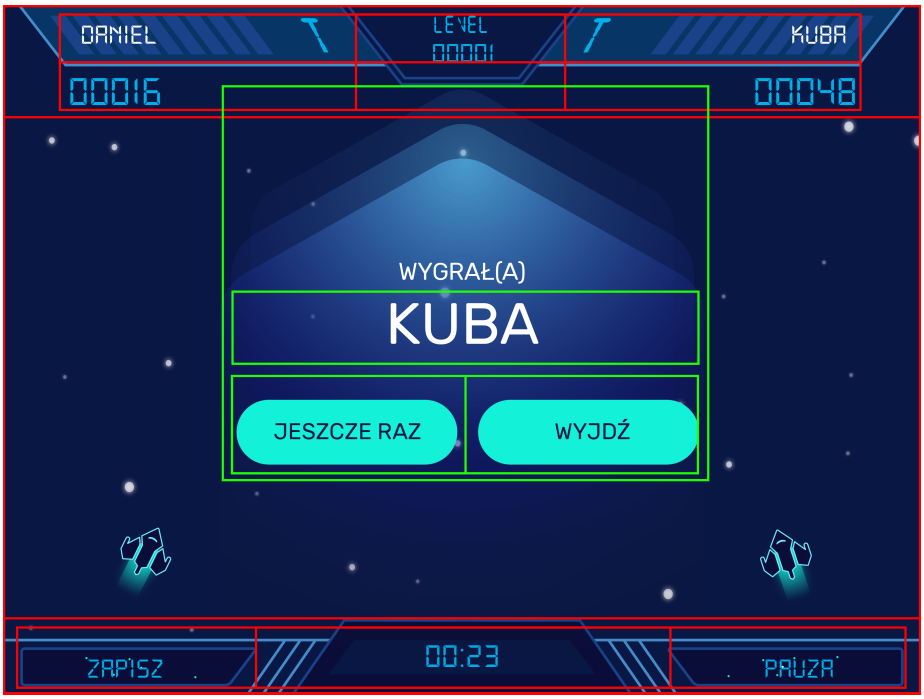
Rysunek 4: Ekran o grze



Rysunek 5: Ekran gry



Rysunek 6: Ekran zatrzymania gry



Rysunek 7: Ekran wygranej

Rysunek 8: Ekran przegranej

5 Testowanie

5.1 Użyte narzędzia

Do przetestowania kodu użyjemy narzędzi z biblioteki AssertJ. Natomiast GUI przetestujemy ręcznie podczas tworzenia aplikacji. Dodatkowo gotowy program przetestujemy sami, grając w niego oraz przekażemy go do testów znajomym.

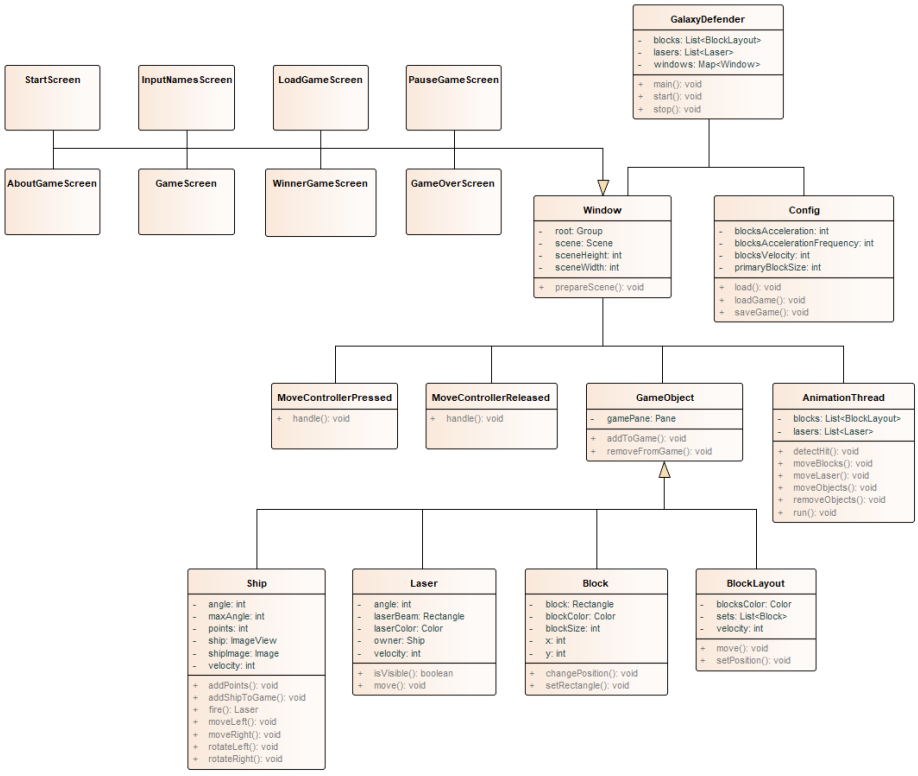
5.2 Konwencja

Metody testujące przyjmą nazwy przypominające równoważniki zdań, aby w jednoznaczny sposób przekazać informację o badaniu konkretnej funkcjonalności.

5.3 Warunki brzegowe

- Możliwość otwarcia pliku konfiguracyjnego.
- Możliwość otwarcia i zapisu pliku stanu gry.
- Odpowiednie działanie programu w przypadku spotkania się wiązki laserowej i spadającego klocka.
- Odpowiednie działania programu w kontekście zliczania punktów gracza.
- Odpowiednie działanie programu w przypadku spotkania się statków.

6 Diagram klas



Rysunek 9: Diagram klas