Sprawozdanie końcowe Projektu „Game_of_tanks”

Andrzej Czechowski, Bartosz Zakrzewski

Data utworzenia: 30.05.2020
Data ostatniej modyfikacji: 03.06.2020

1 Podsumowanie projektu

- Program ma około 1600 linii.
- Projekt na 21 klas (w tym jedną testującą).
- Czas trwania projektu to około 2 miesiące (01.04.2020 - 03.06.2020).
- Udało nam się wykonać całą funkcjonalność zaplanowaną w Specyfikacji Funkcjonalnej. Jednakże niektóre funkcje mogłyby zostać ulepszone ulepszyć, np. użyć Timerów i TimerTasków albo oprzeć projekt na JavieFX.

2 Wykonana funkcjonalność

2.1 Gracze

- Mamy dwóch graczy, którzy poruszają się swoimi czołgami po lewej i prawej stronie planszy.
- Strzelają ograniczoną ilością pocisków naraz z obracanej lufy.
- Prędkość pocisków zwiększa się po określonym czasie.

2.2 Komórki

- Kolory odpowiadające maksymalnej ilość punktów za ustrzelenie komórki pozostały takie jak w dokumentacji, czyli:

caption 1: Kolory komórek i ich maksymalne wartości

- Komórki mogą urodzić dzieci, zwiększyć wartość, zmniejszyć swój rozmiar oraz pojawiać się na planszy (wszystko po określonym czasie).
- Komórki mają losowo przypisane punkty bonusowe oraz mają szansę na zostanie komórką „Armageddon”.

2.3 Koniec gry

- Gra kończy się po zdobyciu maksymalnej ilości punktów (które pozwalają na wygraną).
- Ustrzelenie komórki Armageddon dodaje graczowi maksymalną ilość punktów, co sprawia, że wygrywa on grę.
- Po minięciu określonego czasu gra się kończy.

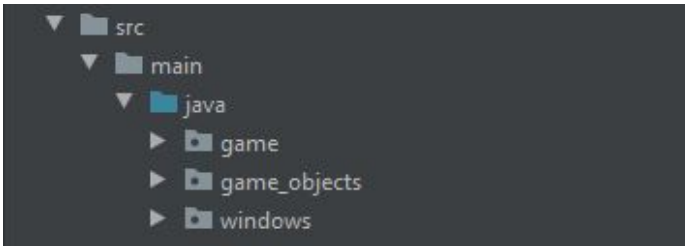
2.4 Pliki

- Użytkownik może wczytywać tylko pliki z folderu data, które będą załadowane do jara (przez opcję package w Mavenie).
- Po ukończeniu gry (wygraniu lub remisie) użytkownik otrzymuje plik PNG („zdjęcie”) który przedstawia planszę w momencie wygrania.

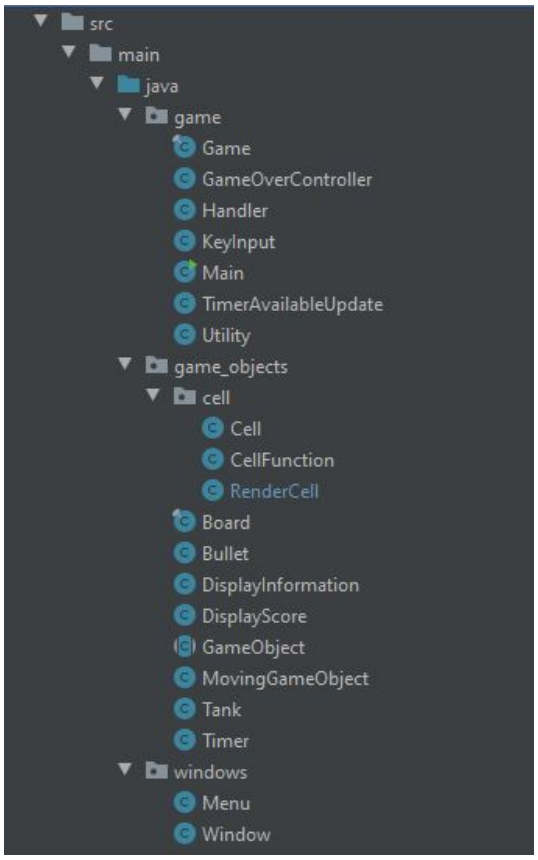
3 Struktura programu

3.1 Struktura pakietów

Do Proof of Concept mieliśmy jeden pakiet, ale postanowiliśmy podzielić nasze klasy na trzy pakiety: game, game_obejcts, windows. W game_objects są klasy dziedziczące po abstrakcyjnej klasie GameObject. W windows są klasy dziedziczące po JFrame, a w game pozostałe.



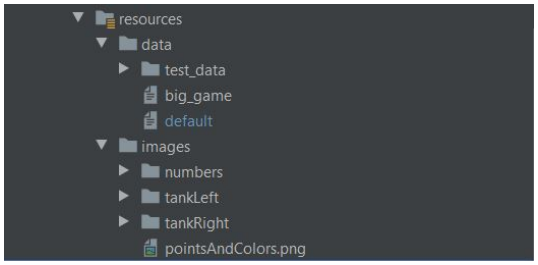
caption 2: Zewnętrzna struktura pakietów



caption 3: Zawartość trzech głównych pakietów

Pod-pakiet pakietu game_objects o nazwie cell zawiera klasy zawierające funkcje potrzebne w Cell (pozwoliło to na zmniejszenie ilości linii kodu w tej klasie).

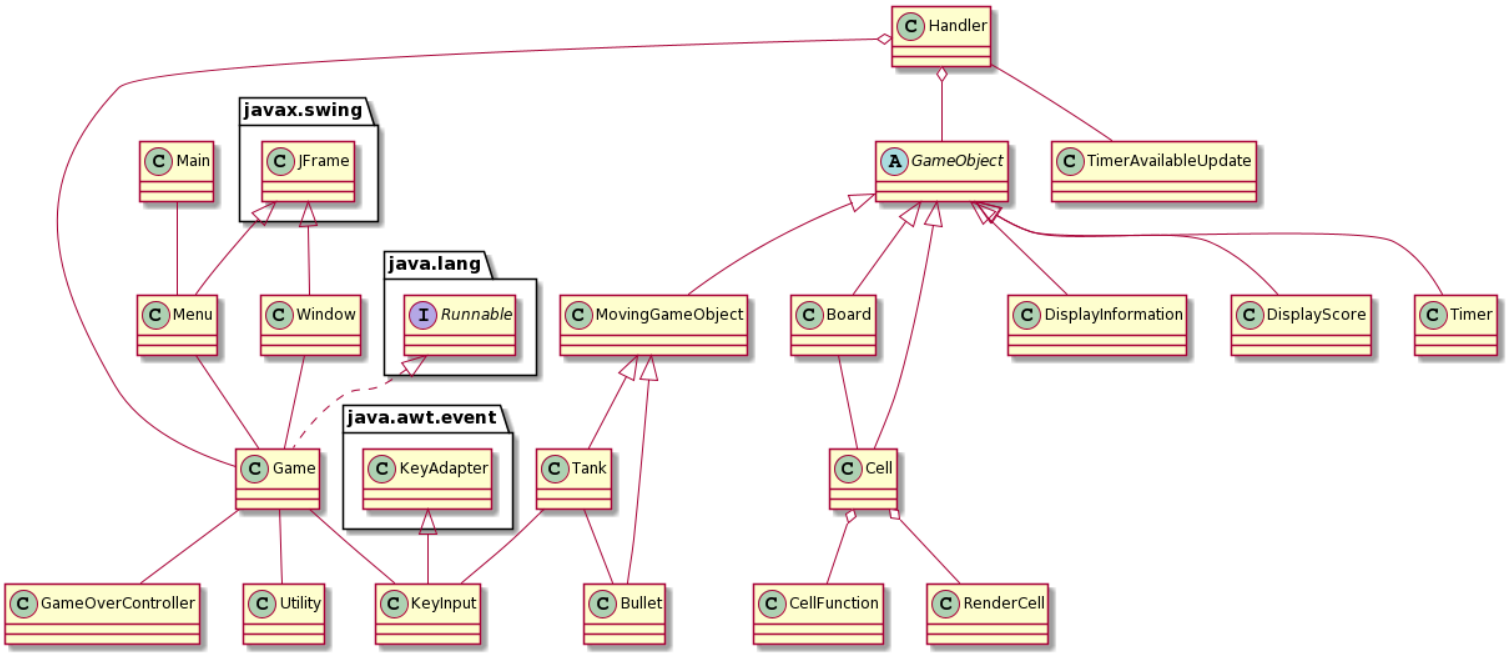
Dane są zapisane w folderze resources:



caption 4: Zawartość folderu resources

3.2 Diagram klas

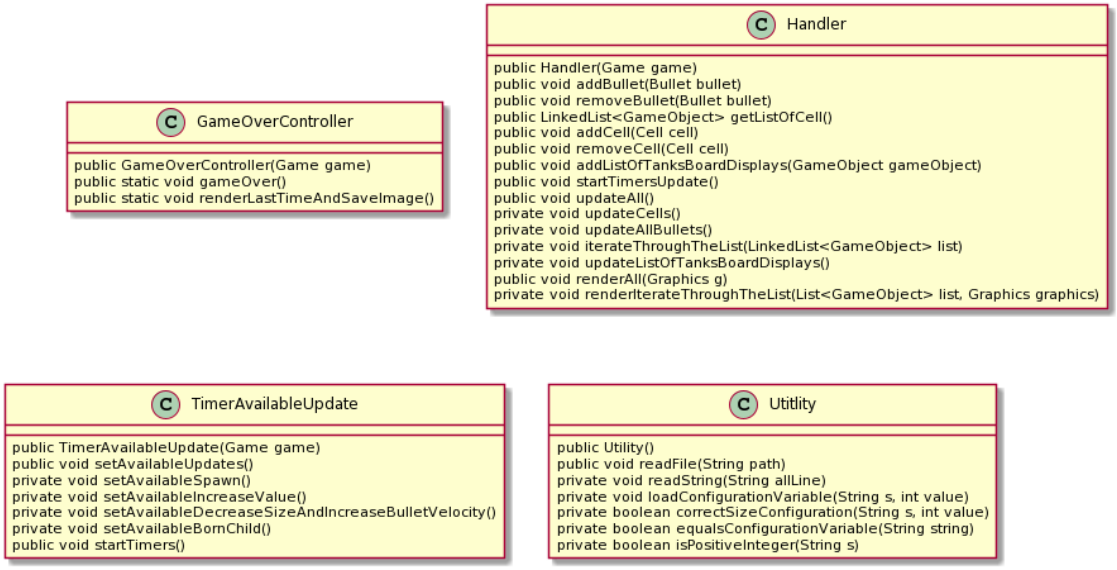
Uproszczony diagram klas, który zawiera same połączenia:



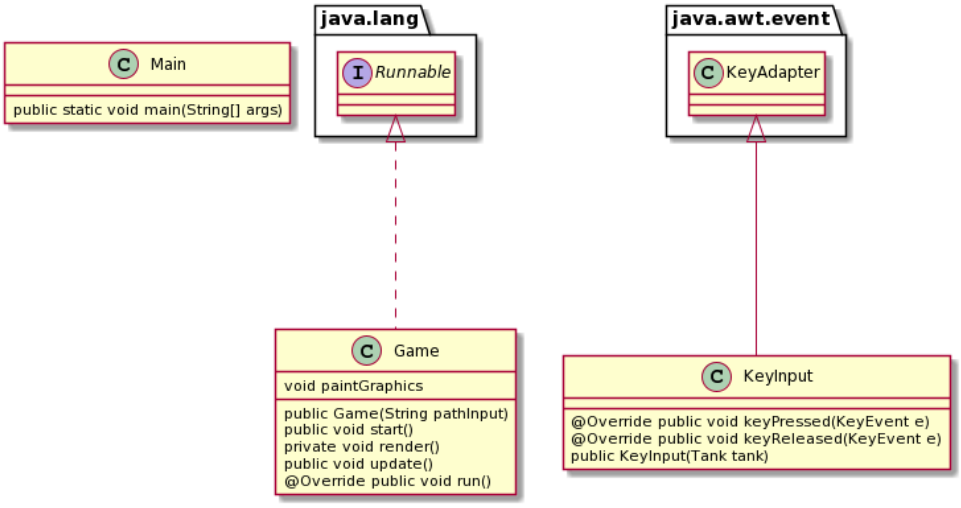
caption 5: Diagram klas - połączenia

Zmienne statyczne w Game (np. P, H, ..., HEIGHT, WIDTH) są używane przez wiele klas: Utility, TimerAvailableUpdate, Cell, Board, Bullet, Tank, Timer. Podobna sytuacja jest z Board.x, Board.y, Board.width. Istnieją także ciągi wywołań np. Main tworzy Menu, Menu tworzy Game. To wszystko tworzy dużą liczbę połączeń, dlatego zdecydowaliśmy się diagram uprościć.

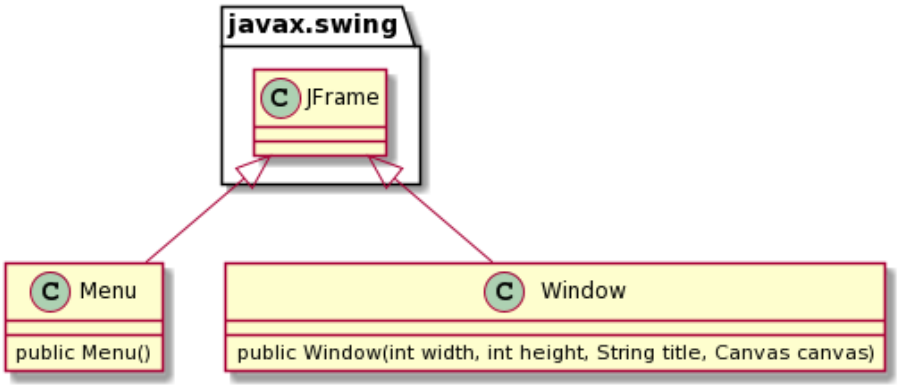
Przedstawimy także jakie metody ma poszczególna klasa (uwzględniając atrybuty wykres robi się nieczytelny):



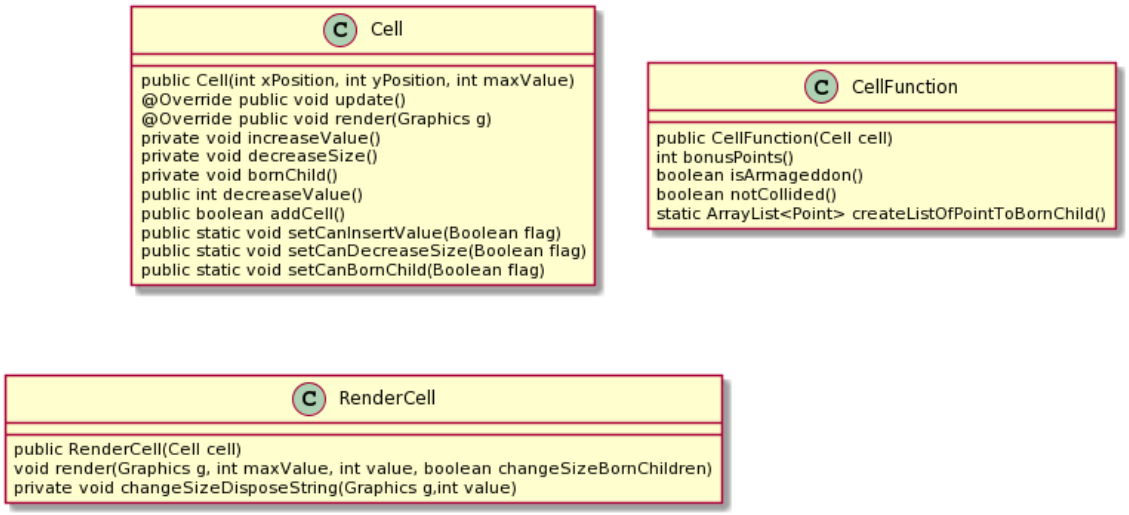
caption 6: Diagram klas - pakiet game



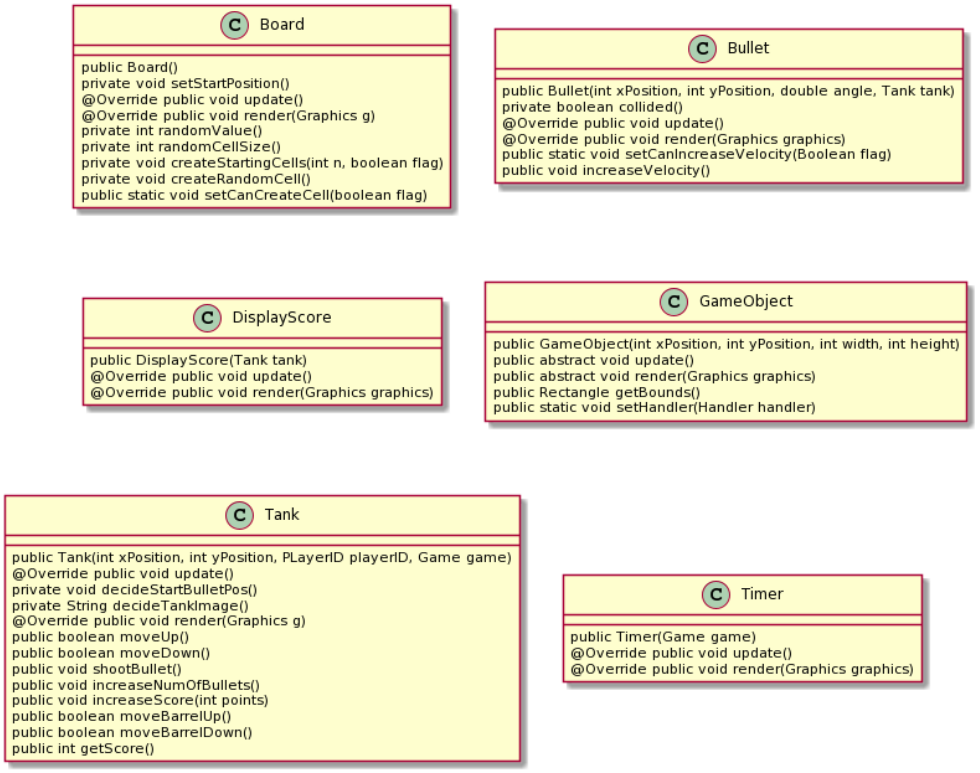
caption 7: Diagram klas - pakiet game



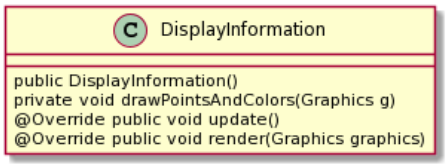
caption 8: Diagram klas - pakiet window

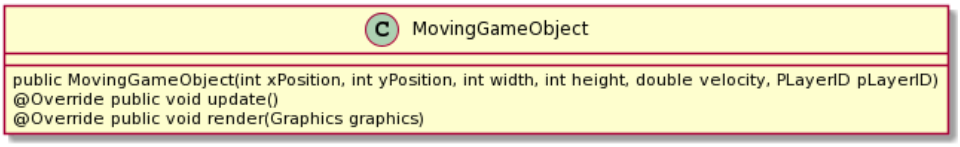


caption 9: Diagram klas - pakiet cell



caption 10: Diagram klas - pozostałe elementy game_objects





caption 11: Diagram klas - pozostałe elementy game_objects

3.3 Użyte importy

Korzystaliśmy głównie z bibliotek java.awt i javax.swing. Większość importów powtarza się w klasach, dlatego tutaj przedstawione są wszystkie różne importy.

- import java.awt.Canvas;
- import java.awt.Color;
- import java.awt.Graphics;
- import java.awt.image.BufferStrategy;
- import java.io.IOException;
- import javax.imageio.ImageIO;
- import javax.swing.JOptionPane;
- import java.awt.image.BufferedImage;
- import java.io.FileOutputStream;
- import java.util.ArrayList;
- import java.util.LinkedList;
- import java.util.List;
- import java.awt.event.KeyAdapter;
- import java.awt.event.KeyEvent;
- import java.io.BufferedReader;
- import java.io.InputStreamReader;
- import java.awt.Point;
- import java.util.Collections;
- import java.util.Random;
- import java.awt.Font;
- import java.awt.FontMetrics;
- import java.awt.Rectangle;
- import javax.swing.JButton;
- import javax.swing.JFrame;
- import javax.swing.JPanel;

- import javax.swing.JTextField;
- import java.awt.event.ActionEvent;
- import java.awt.event.ActionListener;

Importy w klasie UtilityTest:

- import org.assertj.core.api.Assertions;
- import org.junit.Before;
- import org.junit.Test;

3.4 Testy

Utworzyliśmy jeden plik testowy UtilityTest (testujący klasę Utility). Używamy w nim Assertions.assertThat (import org.assertj.core.api.Assertions). Importujemy także:

- org.junit.Before
- org.junit.Test
- org.junit.Assert.assertFalse

Dlatego w naszym pom.xml (Mavenowym) mamy:

```
<dependencies>
    <dependency>
        <groupId>org.assertj</groupId>
        <artifactId>assertj-core</artifactId>
        <version>3.11.1</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

3.5 Plik wejściowy

Zawsze wczytujemy na początku plik wejściowy default:

```
Dane wczytywane domyslnie:
        WIDTH 640
        HEIGHT  480
        X 10 //born
        H 7  //spawn
        Y 9  //increase value
        Z 5  //decrease size and increase bullet velocity
        L 10 //size
        K 20 //bullet velocity
        P 12 // number of bullets
        TIME 120
        WIN_SCORE  100
```

caption 12: Plik default

Następnie jeżeli użytkownik poda ścieżkę do innego pliku z parametrami gry, zostaną one zmienione. Przykładowo:

```
WIDTH 1280
HEIGHT   800
```

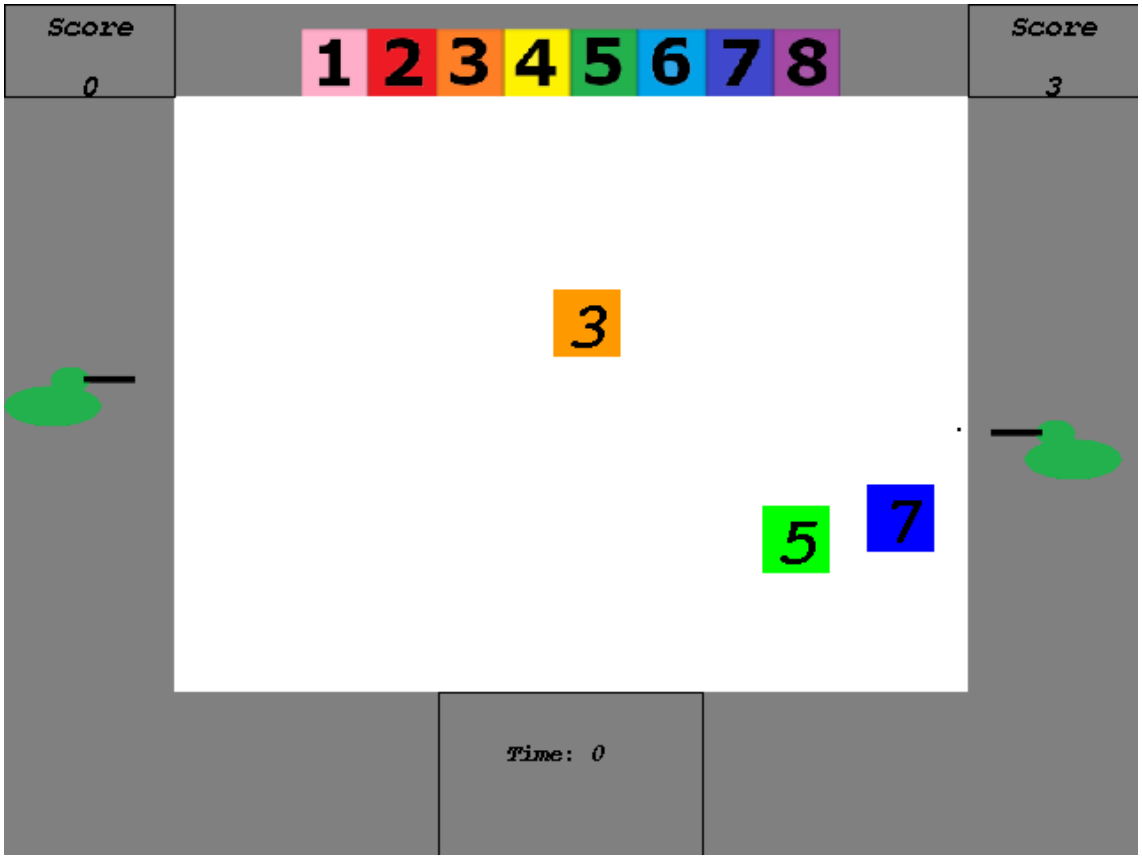
caption 13: Plik big_game

Rozwiązanie to nie zadowoli w pełni użytkownika, ponieważ musi on podać ścieżkę do istniejącego już pliku (który znajduje się folderze data).

Zmienne, które użytkownik może ustawić to: WIDTH, HEIGHT, X, H, Y, Z, L, K, P, TIME, WIN_SCORE. Ograniczamy WIDTH i HEIGHT do rozmiarów (kolejno): <480, 1440> oraz <360, 1080>.

3.6 Plik wyjściowy

Po zakończeniu gry poprzez koniec czasu lub wygraną gracza, gra zapisuje stan planszy w postaciu pliku PNG.



caption 14: Koniec gry przez upłynięcie czasu

Zapisuje się on w miejscu: Game_of_tanks /kod/GameOverScreen.png przez uruchomienie gry przez środowisko (Intellij'a) lub w: Game_of_tanks /kod/ target/GameOverScreen.png przez uruchomienie gry za pomocą jara.

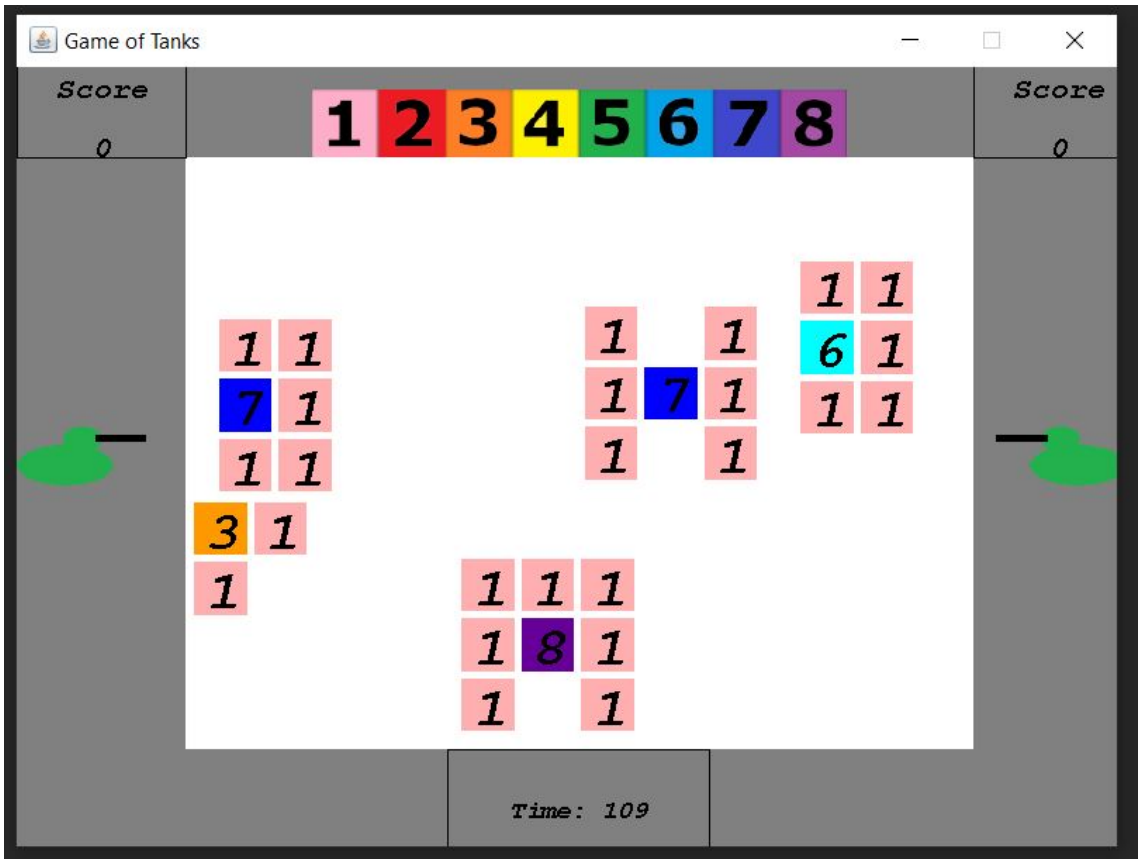
4 Podsumowanie współpracy

4.1 Środowisko pracy

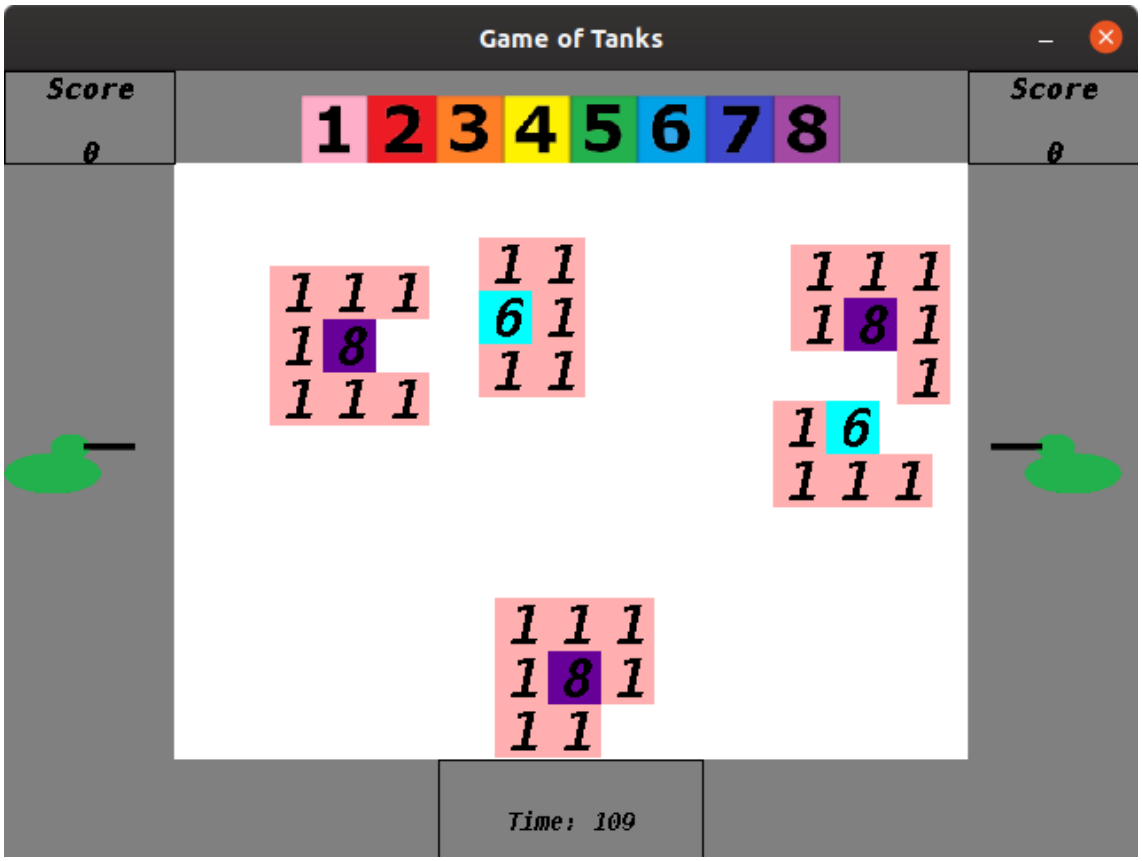
Nie zmienialiśmy naszych środowisk pracy, pozostaliśmy przy IntelliJ IDEA 2020.1. Używaliśmy javy 14 w projekcie w Intellij'u i na naszych komputerach (np. przy java -jar przez terminal).

Pozostaliśmy przy pisaniu gry na dwóch różnych systemach operacyjnych: Ubuntu 19.10 i Windows 10. Powoduje to małe różnice w wyglądzie JFrame'a - aplikacji z grą.

4.1.1 Różnice w grze



caption 15: Gra działająca na Windowsie



caption 16: Gra działająca na Ubuntu

4.1.2 Praca z gitem

Utworzyliśmy 9 gałęzi (nie licząc gałęzi 10., którą utworzymy aby wrzucić to sprawozdanie) o nazwach (jedna gałąź została nazwana przypadkowo po polsku zamiast po angielsku, nie udało nam się tego zmienić):

- 1_Specifications
- 2_runLoop
- 3_Change_of_Folder_Structure
- 4_creating_Board
- 5_Adding_Maven_and_Tests_to_Project
- 6_Tanks_and_bullets

- 7_Second_Tank_and_Display
- 8_Utility_and_Menu
- 9_Jar_and_project_structure

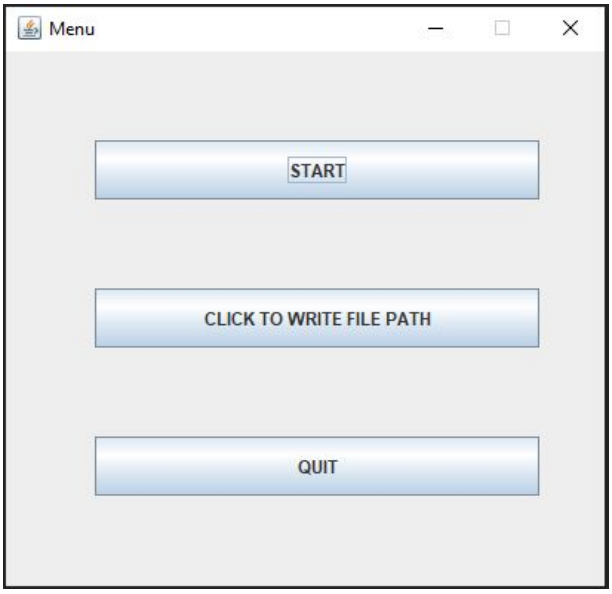
Finalna wersja znajduje się na gałęzi master.

Jako, że robiliśmy dwa projekty na jednym repozytorium to pozostałe gałęzie oraz folder „PROJEKT GRA W ZYCIE” odnosi się do pierwszego projektu (Gra w Życie w języku programowania C).

5 Uruchomienie programu

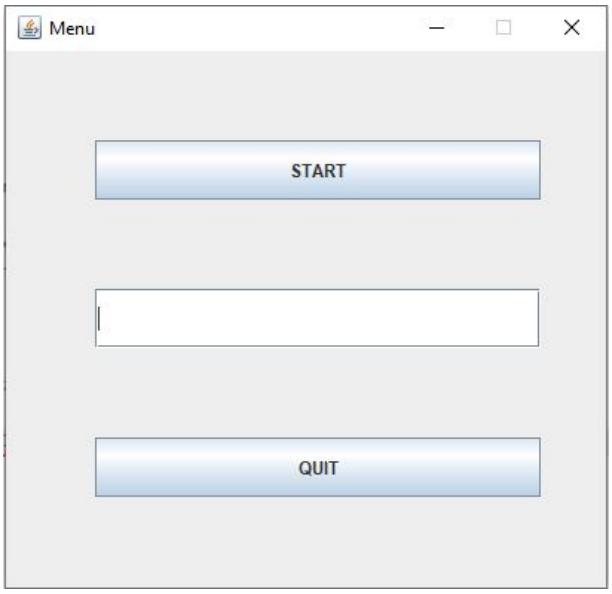
5.1 Menu

Pierwsze co włącza się po uruchomieniu gry to Menu z trzema przyciskami:

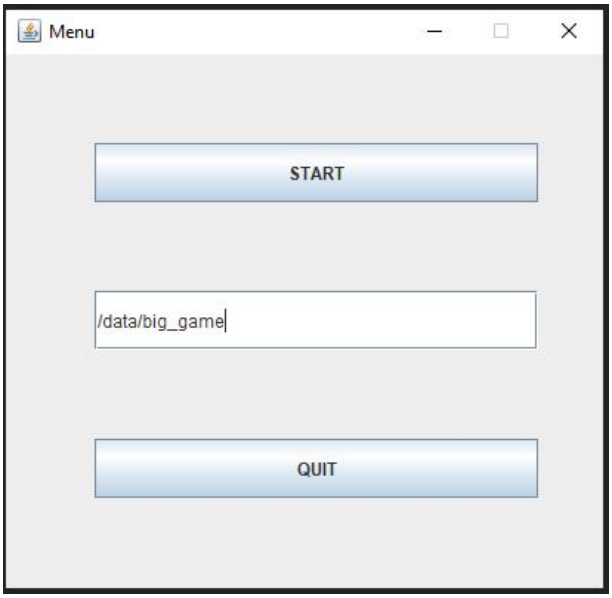


caption 17: Menu

- Przycisk „QUIT” wychodzi z Menu (także z całej gry).
- Przycisk „START” staruje grę.
- Po naciśnięciu przycisku „CLICK TO WRITE FILE PATH” pojawia się pole, w którym możemy wpisać ścieżkę do pliku. Dodatkowo musi ona wyglądać tak: „/data/nazwa_pliku” (plik jest w folderze data).



caption 18: Przycisk CLICK TO WRITE FILE PATH



caption 19: Wpisanie ścieżki

5.2 Jak włączyć grę

1. Wpisanie „java -jar Game_of_tanks-1.0-SNAPSHOT-jar-with-dependencies.jar„ w konsoli (będąc w folderze target projektu).
2. Kliknięcie 2 razy na plik jar.

5.3 Przedstawienie reakcji na błędy

Jeżeli ścieżka do pliku będzie wskazywać na nieistniejący plik lub nie będzie poprzedzona „/data/” to program uruchomi się z wartościami z pliku domyślnego.
Aplikacja najpierw wczytuje dane domyślne dane (/data/default). Jeśli w pliku wejściowym poprawnie będzie podana dana konfiguracja (np. P 20) to ją nadpisuje. Inne połączenia/znaki niż nazwy zmiennych z liczbami ignoruje.

5.4 Sytuacje wyjątkowe

Błędy, na które nie mamy wpływu to:

- Jeżeli program nie wczyta pliku konfiguracyjnego domyślnego (default) to na konsoli wypisze się błąd i należy uruchomić ponownie program (jedynie przez uruchomienie za pomocą Intellij'a - niestety ten błąd będzie niezauważony podczas włączenia gry przez jara).
- Jeśli nie pojawi się czołg lub obraz z punktacją to na konsoli wypisze się błąd, ale będzie można kontynuować grę (bez wyświetlanego czołgu - pozycja pocisków czy zdobywanie punktów nadal działa).

Dodatkowo przy dużej ilości komórek na planszy gra może się zacinać.

6 Wnioski po wykonaniu projektu

Projekt udało się wykonać mimo braku doświadczenia w Javie (zdobywaliśmy go w trakcie trwania projektu).
Do poprawy jest wczytywanie pliku konfiguracyjnego (aby użytkownik mógł wprowadzić plik ze swojego komputera - taki który nie jest załadowany do jara) oraz napisanie więcej testów jednostkowych.
Lepiej by było zacząć projekt od razu z Mavenem oraz lepszym rozwiązaniem graficznym byłby javaFx, z którym też byśmy zaczęli projekt (nowy projekt w Intellij'u za pomocą Mavena i JavyFx).

6.1 Wnioski z Proof of Concept

Wszystkie zadania wyznaczone w sprincie trwającym do Proof of Concept udało nam się zrealizować oprócz pliku jara, który wykonaliśmy w następnym sprincie. Także zadania z drugiego sprintu udało nam się wszystkie zrealizować.

6.2 Co można by usprawnić w działaniu programu

- Pracować z javaFx.
- Użyć Timerów i TimerTasków zamiast implements Runnable (brak funkcji run i w niej pętli while).
- Renderować tylko konkretne komponenty zamiast całej planszy od nowa.

6.3 Organizacja w zespole

Przynajmniej raz w tygodniu rozmawialiśmy na temat projektu (przez platformę Teams) i ustalaliśmy zadania na kolejny tydzień, do których sami się do nich zgłaszaliśmy. Obydwoje nie ma mamy zastrzeżeń do współpracy.

7 Źródła

- Diagramy klas:
  https://www.p-programowanie.pl/uml/diagramy-klas-uml/
  http://zasoby.open.agh.edu.pl/ 09sbfraczek/diagram-klas%2C1%2C11.html
- Diagram klas wykonany za pomocą https://plantuml.com/class-diagram
- Opis teoretyczny został przedstawiony przez dr. Pawła Zawadzkiego
- Ten dokument został utworzony w LaTeX'ie za pomocą strony https://www.overleaf.com
- Jest to trzeci z kolei i ostatni dokument dotyczący projektu „Game_of_tanks”
- Początkowa inspiracja:
  https://marcusman.com/ oraz „Java Programming: Let's Build a Game by RealTutsGML”