Specyfikacja Implementacyjna Projektu „Game of tanks”

Andrzej Czechowski, Bartosz Zakrzewski

Data utworzenia: 03.04.2020
Data ostatniej modyfikacji: 19.04.2020

1 Cel projektu

Celem projektu jest stworzenie gry zręcznościowej za pomocą języka programowania Java. Gra otworzy się w okienku z elemetami graficznymi. Będzie w niej dwóch graczy, którzy będą mogli sterować własnymi czołgami i strzelać za pomocą klawiszy na klawiaturze. Gra będzie sprawdzać czy warunki wygranej zostaną spełnione i oznajmiać zwycięzcę. Użytkownik będzie mógł podać plik wejściowy, który zmieni wartości liczb opisujących zasady gry oraz zobaczyć stan planszy (jako plik PNG) po ukończeniu gry.

2 Środowiska pracy

Będziemy używać programu IntelliJ IDEA wersji 2020.1 jako domyślnego środowiska programistycznego (IDE):

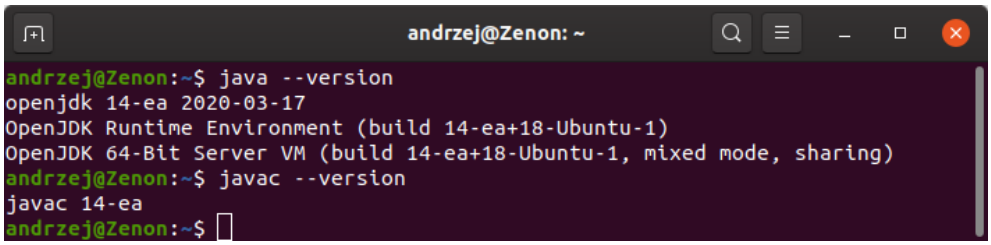
caption 1: IntelliJ 2020.1

Najnowsza wersja JDK (Java Development Kit) na dzień 15.04.2020 to 14 i jej będziemy używać.

```
C:\Users\barza>java -version
java version "14" 2020-03-17
Java(TM) SE Runtime Environment (build 14+36-1461)
Java HotSpot(TM) 64-Bit Server VM (build 14+36-1461, mixed mode, sharing)

C:\Users\barza>javac -version
javac 14
```

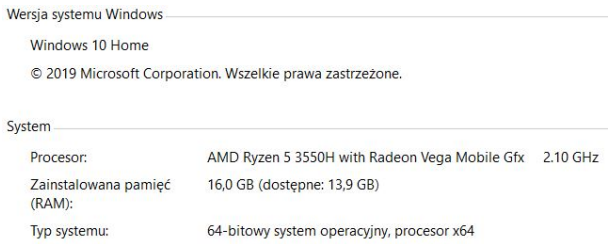
caption 2: Wersja javy



caption 3: Wersja javy

Praca będzie odbywać się na systemach operacyjnych Linux (Ubuntu) oraz Windows (10).

caption 4: Środowisko pracy Andrzeja Czechowskiego



caption 5: Środowisko pracy Bartosza Zakrzewskiego

3 Zasady komunikacji

Stworzyliśmy oddzielną grupę na messengerze, gdzie będziemy komunikować postępy prac, zamieszczać screeny, zdjęcia i robocze dokumenty.

W kodzie będziemy pisać po polsku komentarze robocze - o postępach, rzeczach do zrobienia, błędach itp.

Dodatkowo przynajmniej raz w tygodniu będzie mieli wideokonferencję, aby omówić postępy pracy oraz podzielić się zadaniami.

4 Zasady pracy

4.1 Używanie systemu kontroli wersji

Kolejne wersje programu będą umieszczane i zapisywane za pomocą systemu kontroli wersji git. Kod oraz dokumentacja będzie trafiać na repozytorium o nazwie „Git_proj” w folderze Game_of_tanks.

Zasady tworzenia i działania na gałęziach:

- Na każdej gałęzi dodajemy funkcjonalność, która po przetestowaniu trafia na mastera.
- Na gałąź master trafia tylko kod przetestowany i dający się skompilować.
- Gałęzie będą numerowane i ich nazwy będą po angielsku, zgodnie z schematem n_NameBranch, gdzie n jest numerem gałęzi.
- Nazwa gałęzi będzie starać się opisywać dodaną funkcjonalność.

Zasady tworzenia komentarzy przy commitach:

- Komentarz ma opisywać dodaną funkcjonalność, (opcjonalnie) możliwe rzeczy do poprawy oraz błędy.
- Komentarze piszemy po polsku.

Przykładowy commit: „tworzenie losowych komórek, nieefektywne tworzenie dzieci”.

Tagi:

- Wersja, którą będziemy chcieli oddać jako wersję finalną oznaczymy tagiem FINAL (lub podobnym - określonym przez osobę sprawdzającą projekt np. FINAL_RELEASE_1).

Kod przed commitem ma być sformatowany (za pomocą opcji w Intellij'u).

4.2 Zasady nazewnictwa

Nazewnictwo w programie będzie w języku angielskim oraz zgodne z notacją Javy („camelCase” oraz „UpperCamelCase”) np.

- GameObject,
- gameState,
- createRandomCell().

Nazwy folderów, dokumentów czy pakietów mogą być oddzielone znakiem „_”. Mogą być one także po polsku. np.

- game (pakiet)
- Specifications (folder)
- Specyfikacja_Implementacyjna.pdf (dokument)
- Implementation_Specification.pdf (dokument)

Nazwy funkcji testujących również mogą (ale nie muszą) zawierać znak „_”. np.

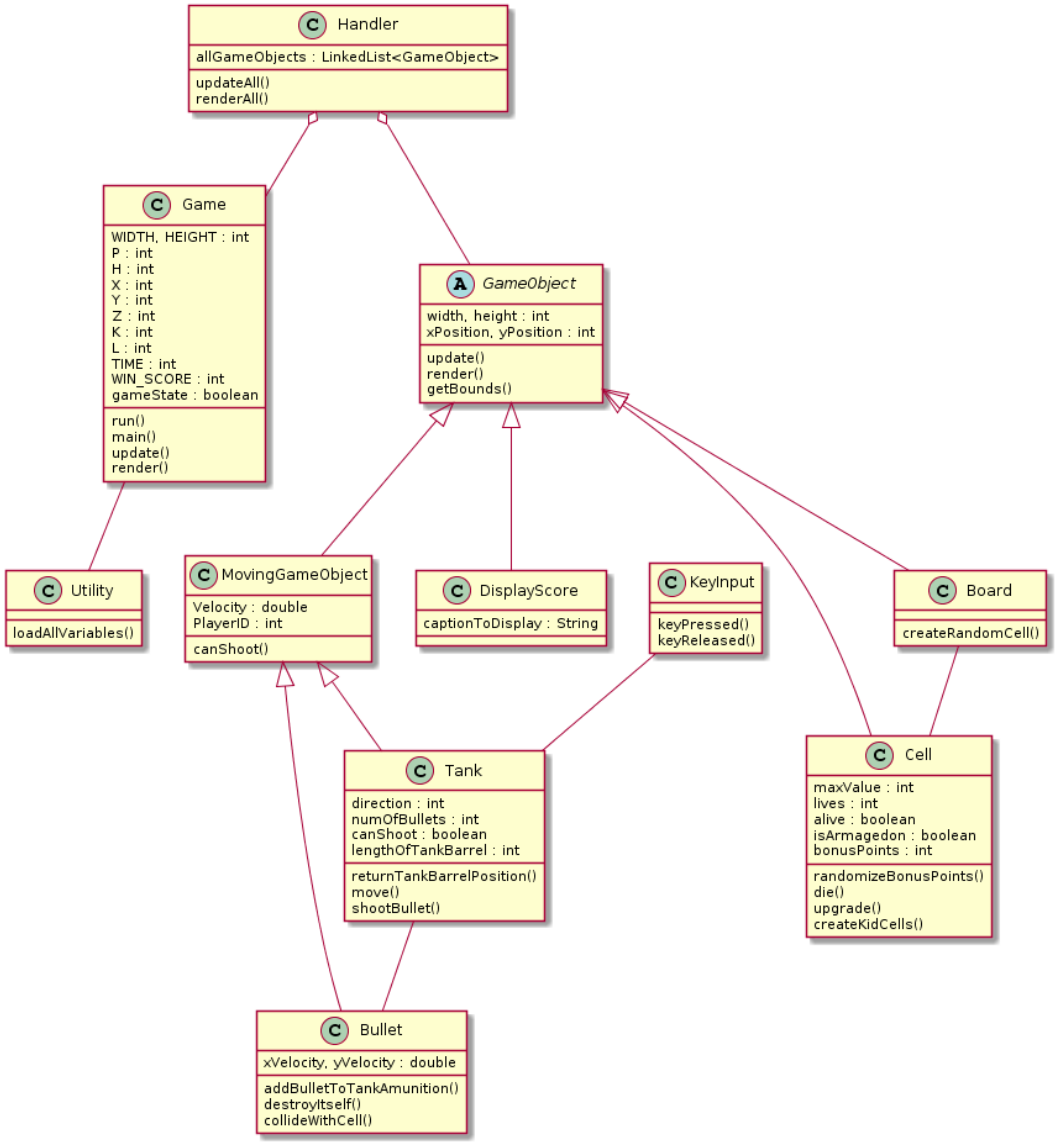
- wrongFormattedVariable_makeItDefault()
- exceedMaxPointsToWin()
- collisionWithCell_shouldDestroyBullet()

Komentarze w kodzie „roboczym” (wersje dla członków zespołu) (aby łatwiej było się komunikować oraz mieć rozpisane co trzeba poprawić) mogą być po polsku.

Jeżeli kod będzie zawierał komentarze dla osoby z poza zespołu (np. wyjaśniające wartość nieintuicyjnej zmiennej) będą po angielsku.

5 Struktura programu

5.1 Diagram klas



caption 6: Diagram klas

A - klasa abstrakcyjna, C - klasa

5.2 Importy

Potencjalne importy: (tutaj też korzystamy z: https://www.youtube.com/watch?v=JrSjwQbTldg, marcusman.com)

- import java.awt.Graphics (renderowanie)
- import java.awt.Rectangle (do getBounds() - obsługa kolizji)
- import java.awt.Canvas (pozwala na rysowanie)
- import java.awt.image.BufferStrategy (mechanizm „ładowania” i wyświetlania grafiki)
- import java.util.LinkedList
- import java.awt.event.KeyListener
- import java.awt.event.KeyEvent
- import javax.swing.JFrame
- import java.awt.Color;

Pobieranie i wyświetlanie obrazów:

- import java.awt.image.BufferedImage
- import java.io.File
- import java.io.IOException
- import javax.imageio.ImageIO
- import javax.swing.JPanel

Pozostała importy to wyświetlenie tekstu (numeru - wartości komórki) oraz inne, które będziemy poznawać podczas implementacji.

Podczas implementacji ustalimy czy potrzebujemy kilku pakietów, na początku wszystkie klasy umieścimy w jednym o nazwie game.

Folder images będzie przechowywał obrazki, a folder data będzie plik konfiguracyjny.

6 Kolejność działania

Prace podzielimy na dwa sprinty.
Pierwszy będzie trwał do Proof of concept (13.05.2020):

1. Pętla odtwarzająca program (funkcja run()) oraz okienko za pomocą JFrame).
2. Dodawanie obiektów (GameObject) do okienka aplikacji.
3. Plansza składająca się z losowych komórek (klasa Board).
4. Rodzenie się komórek-dzieci (klasa Cell).
5. Ruch pocisku (przy ustalonym punkcie początkowym i kącie strzelania).
6. Kolizja pocisku z komórką.
7. Dodanie schematu czołgu, w którym można poruszać lufą.
8. Strzelanie pocisków przy możliwości zmiany kąta.
9. Zdobywanie punktów.
10. Tworzenie pliku JAR oraz uruchamianie programu.

Drugi sprint do 03.06.2020:

1. Dodanie drugiego gracza.
2. Tworzenie klasy DisplayScore (okienka timer i score).
3. Wczytywanie z pliku wejściowego danych konfiguracyjnych.
4. Obsługa punktów spadkowych.
5. Zmniejszanie się rozmiaru komórek.
6. Inne możliwości wygrywania.
7. Tworzenie pliku wyjściowego PNG.

Zakres prac czy kolejność wykonywania zadań może być lekko zmieniona podczas implementacji.
Po pierwszym sprincie będziemy chcieli osiągnąć grę, która będzie działać, ale nie będzie musiała spełniać wszystkich funkcjonalności.
Natomiast drugi sprint będzie trwał do terminu oddania projektu.

7 Istotne Algorytmy w programie

7.1 Skalowanie rozmiarów obiektów

Użytkownik ma mieć możliwość zmiany rozmiarów planszy (tutaj zamiast planszy z komórkami - Board mamy jednak na myśli okienko JFrame z całą grą (jego WIDTH oraz HEIGHT)).
Aby móc przeskalować inne elementy gry (aby ich rozmiar był zgodny z rozmiarem aplikacji) utworzymy dwie zmienne:

- $p_w = \frac{WIDTH}{defaultWidth}$
- $p_h = \frac{HEIGHT}{defaultHeight}$

gdzie:
WIDTH, HEIGHT - zmienne wejściowe,
defaultWidth, defaultHeight - domyślny rozmiar okienka JFrame.

Następnie wysokość i szerokość każdego obiektu będzie skalowana zgodnie ze schematem:
w, h - domyślne rozmiary obiektu,
$w \cdot p_w$, $h \cdot p_h$ - rozmiary obiektu po przeskalowaniu.

7.2 Obliczanie aktualnego położenia posicku

Pocisk będę miał własną prędkość (V) (która będzie się zwiększała podczas gry). Aby pocisk mógł się poruszać po planszy w zależności od miejsca wystrzelenia (kąta lufy) nadamy mu dwie prędkości składowe Vx i Vy.
Algorytm zmiany położenia pocisku:

V - prędkość wystrzelonego pocisku wyrażona w ilościach pikseli na t
t - czas odświeżania (1/60 s) (nadamy grze 60 FPS'ów (60 klatek na sekundę))
α - kat wystrzelonego pocisku
x, y - polozenie pocisku przed odświeżeniem
$x + V\cos(\alpha) \cdot t$, $y + V\sin(\alpha) \cdot t$ - położenie pocisku po odświeżeniu

7.3 Funkcja run() i jej opis

```
private boolean gameState = true;

public void run() {

    long lastTime = System.nanoTime();
    double nanoSecondFpsConversion = 1000000000.0 / 60;
    double deltaTime = 0;

    while (gameState) {
        long now = System.nanoTime();

        deltaTime += (now - lastTime) / nanoSecondFpsConversion;
        while (deltaTime >= 1) {
            update();
            deltaTime--;
        }

        render();
        lastTime = now;
    }
}

public static void main(String[] args) {
    Game game = new Game();
    Thread gameThread = new Thread(game);
    gameThread.start();
}
```

Klasa Game implementuje interfejs Runnable. Następnie wątek gameThread, a wraz z nim funkcja run() będzie zajmować się obsługą naszej gry. Ustawiliśmy nasze zmienne tak, aby co 1/60 sekundy (czyli 60 klatek/odświeżeń/update'ów na sekundę) odświeżał się stan gry. Sprawi to, że niezależnie na jakim komputerze, gra będzie „chodziła” tak samo szybko.

7.4 Pozostałe algorytmy

Inne istotne algorytmy będę musiały zostać opracowane podczas pisania kodu (np. podczas wideo-rozmów). Są to np.

- Częstotliwość i prawdopodobieństwo punktów spadkowych.
- Działanie równoległe programu (podział na wątki).
- Obsługa wyjątków (Exceptions).
- Log programu (gdzie wypisujemy komunikaty - błędy).
- Animacje (poruszanie lufy itp.).

Również możemy napotkać ograniczenia i rzeczy, których nie będziemy potrafili napisać. Będzie musiało to zostać udokumentowane w Proof of Concept lub w Sprawozdaniu Końcowym.

8 Testowanie

Testy jednostkowe będzie pisać z wykorzystaniem biblioteki AssertJ. Planujemy robić testy na poszczególnych gałęziach a dopiero następnie mergować kod na mastera.

9 Źródła

- Tobiasz Siemiński - opis specyfikacji implementacyjnej:
  https://sortris.blogspot.com/2010/08/jak-napisac-specyfikacje.html?m=1
- Ten dokument został utworzony w LaTeX'ie za pomocą strony
  https://www.overleaf.com
- Jest to drugi z kolei dokument dotyczący projektu „Game_of_tanks”
- Początkowa struktura gry, pętla run() oraz diagram klas na podstawie:
  https://marcusman.com/ oraz
  Java Programming: Let's Build a Game by RealTutsGML
  (https://www.youtube.com/watch?v=1gir2R7G9ws)
- 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
- Importy do wyświetlania obrazków:
  https://javastart.pl/baza-wiedzy/grafika_awt_swing/pobieranie-i-wyswietlanie-obrazow