Specyfikacja implementacyjna

War Of Tanks

Aleksandra Michalska, Natalia Olszewska

24.04.2021

Spis treści

1 Informacje ogólne

1.1 Opis dokumentu

Dany dokument koresponduje ze *Specyfikacją funkcjonalną*.

1.2 Środowisko implementacyjne

Projekt zostanie zaimplementowany w środowisku *IntelliJ IDEA Community Edition 2021.1* oraz napisany w języku *Java 1.8*.

Będzie przeznaczony do środowisk Microsoft Windows, Linux (dystrybucje Mint oraz Ubuntu), Mac OS.

W celu archiwizowania postępów pracy oraz sprawnego kontrolowania projektu zostanie wykorzystany system kontroli wersji: *projektor.ee - system kontroli wersji Politechniki Warszawskiej*.

1.3 Narzędzia i programy towarzyszące tworzeniu

W czasie trwania projektu będzie konieczne skorzystanie z pomocy dodatkowych narzędzi oraz programów:

- **projektor.ee** - system kontroli wersji,
- **LaTeX** - język pisania dokumentów projektu,
- **Overleaf** - platforma do sporządzania dokumentacji w języku LateX,
- **IntelliJ IDEA Community Edition 2021.1** - środowisko implementacyjne,
- **www.piskelapp.com** - strona internetowa do tworzenia grafiki pikselowej,
- **GIMP 2.10.20** - edytor graficzny,
- **ecrettmusic.com/play** - strona internetowa do generowania muzyki,
- **VLC media player** - odtwarzacz oraz konwerter multimedialny,
- **Swing** - biblioteka graficzna używana w języku programowania Java,
- **Dysk Google** - dysk do przechowywania multimediów projektowych.

Wszelkie dodatkowe narzędzia niewymienione powyżej zostaną dodane w dokumentacji uzupełniającej po zakończeniu implementacji projektu.

1.4 Konwencja wprowadzania zmian w kontroli wersji

Wprowadzane zmiany w kodzie będą zatwierdzane zgodnie z konwencją zaproponowaną na stronie www.conventionalcommits.org.

1.5 Określenie kamieni milowych

W celu zobrazowania postępów faz projektu została przyjęta konwencja kamieni milowych jako tagów w systemie kontroli wersji.

W momencie gdy implementacja fazy projektu zostanie zakończona, w systemie kontroli wersji należy oznaczyć tag: *MILESTONE_COMPLETED_NoXX*, gdzie *XX* należy zamienić na odpowiedni numer kamienia milowego (01, 02, 03, ..., 12).

Projekt będzie posiadał dwanaście kamieni milowych:

- 01 - zakończenie implementacji wyglądu okien *StartFrame*, *GameFrame*, *SettingsFrame*, *HelpFrame*,
- 02 - skolekcjonowanie grafiki oraz ścieżek dźwiękowych dla projektu,
- 03 - opracowanie podstawowego poruszania się czołgów,
- 04 - zakończenie implementacji życia komórek oraz kolonii,
- 05 - opracowanie struktury zachowania komórki bomby,
- 06 - implementacja zarządzania kolizjami,
- 07 - opracowanie podstawowej struktury projektu,
- 08 - zakończenie wstępnej fazy testowania,
- 09 - zakończenie nakładania poprawek,
- 10 - zakończenie ostatecznej fazy testowania,
- 11 - zakończenie refaktoryzacji kodu,
- 12 - zakończenie obowiązkowej części implementacji projektu.

2 Opis klas i ich zależności

2.1 Diagram klas

W celu uproszczenia procesu implementacji został sporządzony diagram klas, jednak z przyczyn wizualnych jest on dostępny pod danym adresem.

2.2 Klasy

2.2.1 MainOfWar

MainOfWar będzie klasą początkową. To właśnie w niej można będzie znaleźć metodę *main*, odpowiedzialną za uruchomienie programu. Będzie posiadać jeden atrybut *start* będącym obiektem klasy **StartFrame**.

2.2.2 StartFrame

Klasa, której zadaniem będzie wyświetlenie podstawowego menu otrzyma możliwość uruchomienia ustawień dźwięku, pokazania pomocy, załączenia pliku konfiguracyjnego lub reset ustawień otrzymanych z poprzednich plików konfiguracyjnych. Pojawi się również opcja uruchomenia samej gry.

Klasa będzie dziedziczyć po **JFrame**, a wszelkie jej ustawienia będą możliwe do uruchomienia poprzez kliknięcie w odpowiedni przycisk – obiekty typu **JButton**.

Do atrybutów tej klasy będą należeć obiekty typu **HelpFrame**, **GameFrame**, **SettingsFrame**, klasa posiadać będzie również obiekt przechowujący ustawienia konfiguracyjne typu **ConfObject**. Dodatkowo, co zostało wspomniane powyżej, menu zostanie zaimplementowane w postaci obiektów typu **JButton**.

Metody podzielą się na *manageButtons()*, odpowiedzialną za kontrolę kliknięć menu, oraz *readFile()*, która będzie nadpisywała i uaktualniała obiekt przechowujący ustawienia konfiguracyjne.

2.2.3 HelpFrame

Klasa będzie miała za zadanie wyświetlić podstawową instrukcję rozgrywki oraz obsługi gry. Będzie dziedziczyć po **JFrame**.

Atrybutem tej klasy najprawdopodobniej stanie się pole tekstowe *help* typu **JTextField** zawierające wszelkie potrzebne informacje dla graczy.

2.2.4 ConfObject

Klasa **ConfObject** będzie miała za zadanie przechowywać wszelkie parametry potrzebne do skonfigurowania pojedynczej rozgrywki. Każdy jej atrybut otrzyma określone wartości domyślne opisane w *Specyfikacji funkcjonalnej*.

Atrybuty tej klasy będą typu **Integer**.

Do jej metod wliczą się funkcje *get* oraz *set* dla każdego atrybutu. Dodatkowo zostanie zaimplementowana funkcja *setPrimitiveParameters()* określająca domyślne parametry, która będzie prywatna – dostępna tylko dla konstruktora tej klasy.

2.2.5 GameFrame

Klasa **GameFrame** będzie tzw. *sercem rozgrywki*. To właśnie ona skontroluje przebieg całej gry. Sama klasa będzie dziedziczyć po **JFrame**, dodatkowo będzie posiadać implementacje zdarzeń kolizji, ruchu czołgów, wystrzeliwania pocisków oraz struktury funkcjonowania komórek i kolonii. Jej konstruktor przyjmie argument typu **ConfObject** otrzymany od klasy **StartFrame**.

Do atrybutów tej klasy zaliczą się:

- prawy czołg - typu **TankRight**,
- lewy czołg - typu **TankLeft**,
- panele punktacji - typu **JPanel**,
- panele informacji o pociskach - typu **JPanel**,
- panel czasu gry - typu **JPanel**,

- bomba - typu **Bomb**,
- lista spadających komórek - typu **List<Cell>**,
- lista kolonii różnej liczby komórek - typów **List<Colony2>**, **List<Colony3>**, **List<Colony4>**, **List<Colony5>**,
- lista pocisków lewego czołgu - typu **List<BulletL>**,
- lista pocisków prawego czołgu - typu **List<BulletR>**,
- obiekt konfiguracyjny - typu **ConfObject**,
- czasomierz - typu **Timer**.

Do metod będą się zaliczać:

- konstruktor - przyjmujący argument typu **ConfObject**,
- *manageKeyboard* - obsługujący wejście klawiatury,
- *addPoints()* - dodający punktację do odpowiedniego panelu w momencie zdobycia punktów przez któregoś z graczy, przyjmuje argumenty *points* oraz *which* typu **Integer**.
- *manageColisionOnLeft()* - sprawdzający kolizje pocisków lewego czołgu,
- *manageColisionOnRight()* - sprawdzający kolizje pocisków prawego czołgu,
- *setPanels()* - określający wygląd paneli,
- *setObstacle()* - określający podstawowe ustawienia kolonii oraz komórek,
- *setBullet()* - przygotowujący odpowiednią strukturę pocisków dla czołgów,
- *setConf()* - ustawiający obiekt konfiguracyjny zgodnie z tym przekazanym przez konstruktor, przyjmuje argument typu **ConfObject**,
- *saveScreenShoot()* - zapisujący obraz po zakończeniu rozgrywki do pliku o rozszerzeniu graficznym.

2.2.6 SettingsFrame

Klasa będzie dziedziczyć po **JFrame**. Stanie się odpowiedzialna za zmiany ustawień głośności ścieżek dźwiękowych. Do możliwych opcji będą dostępne:

- ściszenie,
- podgłośnienie,
- wyłączenie,
- włączenie.

Każda z tych opcji będzie dostępna w postaci przycisków przy każdej kategorii dźwiękowej:

- muzyka startowa,
- dźwięk wystrzału pocisków,
- muzyka w czasie rozgrywki.

Do atrybutów tej klasy zostaną przydzielone:

- przyciski - typu **JButton**,
- wartość głośności muzyki startowej - *volumeOfMusicStart*,
- wartość głośności muzyki podczas gry - *volumeOfMusicPlaying*,
- wartość głośności muzyki wystrzału pocisku - *volumeOfBullets*.

Do metod należeć będą:

- *turnOnMusic()* - włączająca muzykę,
- *turnOffMusic()* - wyłączająca muzykę,
- *volumeUp()* - podgłośniająca muzykę,
- *volumeDown()* - ściszająca muzykę,
- *getVolumeOfMusicStart()* - zwracająca wartość głośności muzyki startowej,
- *getVolumeOfBullets()* - zwracająca wartość głośności wystrzału pocisków,
- *getVolumeOfMusicPlaying()* - zwracająca wartości głośności muzyki podczas rozgrywki,
- *setButtons()* - ustawiająca podstawowy wygląd przycisków,
- *manageButtons()* - kontrolująca operacje na przyciskach.

2.2.7 Colony

Klasa **Colony** będzie klasą abstrakcyjną, służącą do implementacji klas **Colony2**, **Colony3**, **Colony4**, **Colony5**. Klasa ta będzie odpowiedzialna za funkcjonowanie struktury kolonii złożonej z pewnej (od 2 do 5) liczby komórek.

Do atrybutów tej klasy abstrakcyjnej należeć będą:

- liczba komórek, z jakiej składa się kolonia - typu **Integer**,
- lista komórek - typu **List<Cell>**,
- liczba punktów możliwa do uzyskania po likwidacji kolonii - typu **Integer**,
- prędkość poruszania się kolonii - typu **Integer**.

Do metod klasy należeć będą:

- *setPrimaryParameters()* - ustawiająca początkowe wartości kolonii,
- *getPoints()* - zwracająca liczbę możliwych punktów,

- *getVelocity()* - zwracająca wartość prędkości kolonii,
- *getHowMany()* - zwracająca liczbę wszystkich komórek w kolonii,
- *isDead()* - zwracająca wartość typu **Boolean** czy kolonia została zlikwidowana,
- *setPoints()* - zmieniająca wartość punktów możliwych do uzyskania, przyjmuje argument typu **Integer**,
- *setVelocity()* - zmieniająca wartość prędkości kolonii, przyjmuje argument typu **Integer**,

2.2.8 Cell

Klasa będzie miała za zadanie kontrolować strukturę pojedynczej komórki.
Do jej atrybutów będą należeć:

- lista kolorów jakie przyjmować będzie komórka zależnie od liczby żyć - typu **List<Color>**,
- liczba żyć komórki - typu **Integer**,
- rozmiar boku komórki - typu **Integer**,
- prędkość komórki - typu **Integer**,
- liczba punktów za likwidację komórki - typu **Integer**.

W skład metod będą wchodzić funkcje *set* do zmiany parametrów komórki oraz *get* do otrzymywania odpowiednich wartości atrybutów. Dodatkowo klasa będzie posiadać metody:

- *getShoot()* - wywoływaną przy kolizji z pociskiem,
- *setPrimaryCellParameters()* - ustawiającą podstawowe wartości dla komórki,
- *isDead()* - sprawdzającą czy komórka została zlikwidowana, zwraca wartość typu **Boolean**.

2.2.9 Bomb

Klasa będzie odpowiadała za strukturę komórki bomby.
Będzie posiadać dwa atrybuty. Jeden określający czy została uderzona, a drugi przechowujący grafikę samej bomby.
Do metod tej klasy będą się wliczać: pobranie i ustawienie wartości atrybutu czy została uderzona oraz ustawienie wstępnych wartości komórki bomby.

2.2.10 Bullet

Klasa abstrakcyjna służąca późniejszej implementacji obiektów **BulletR** oraz **BulletL**, będzie posiadać atrybuty określające promień i prędkość pocisku.
Dodatkowo do jej metod będą się zaliczać *get* oraz *set* dla każdego z atrybutów.

2.2.11 TankLeft oraz TankRight

Klasy bliźniacze będą implementowały struktury odpowiednio czołgu lewego oraz prawego.

W atrybutach znajdą się panele odpowiadające korpusom czołgów oraz lufom, pozycja czołgów na osi x oraz kąt pod jakim będzie się aktualnie znajdować lufa.

W metodach zostaną zaimplementowane opcje poruszania czołgiem oraz lufą, a także możliwości uzyskania wartości atrybutów pozycji i kąta (*getPosition()*, *getAngle*).

2.2.12 BodyL oraz BodyR

Klasy bliźniacze dziedziczące po **JPanel**. Będą odpowiadały za strukturę korpusu czołgu, a dokładniej zmianę odpowiedniej grafiki.

W atrybutach będą się znajdować dwie grafiki korpusu. Ich zmiana w momencie poruszenia czołgiem będzie symulować ruch.

Do metod będą należeć:

- *changeBody()* - zmieniająca grafikę korpusu,
- *setBody()* - ustawiająca podstawowe parametry korpusu.

2.2.13 BarrelL oraz BarrelR

Bliźniacze klasy będą odpowiadały za struktury luf. Będą dziedziczyć po **JPanel**.

Ponieważ lufa będzie posiadać wiele grafik odpowiadających kątom, zostaną one umieszczone w liście **List<Image>**.

Klasa będzie posiadać trzy metody:

- *changeBarrelUp()* - zmieniająca grafikę lufy o kąt w górę,
- *changeBarrelDown()* - zmieniająca grafikę lufy o kąt w dół,
- *setBarrels()* - ustawiająca podstawowe parametry panelu lufy.

2.3 Zależności między klasami

W projekcie mamy na celu wykorzystanie trzech rodzajów relacji: agregacji (kompozycji), dziedziczenia, asocjacji. Duży nacisk zostanie nałożony na niski poziom dziedziczenia oraz na jak największą modularność.

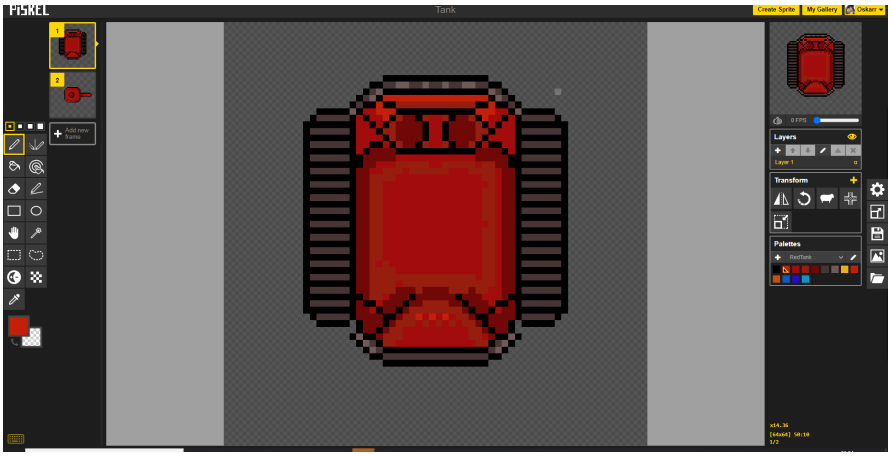
2.4 Wzorce projektowe

W późniejszych etapach projektu może zaistnieć potrzeba wykorzystania pewnych wzorców projektowych, np. fabryki abstrakcji (dla implementacji czołgów).

3 Multimedia

3.1 Grafika

Wszelkie grafiki zostały lub zostaną wykonane na stronie *www.piskelapp.com*, a następnie dopracowane w *GIMP* (patrz *Rysunek 1*).



Rysunek 1: Proces wytwarzania grafiki

3.2 Muzyka

Ścieżki dźwiękowe zostaną wygenerowane i dopracowane na stronie *ecrettmusic.com*. Będą one edytowane w *VLC media player*.

4 Testowanie

4.1 Wygląd testów

Testowanie zostanie zaimplementowane w trakcie tworzenia pojedynczych klas oraz w fazach testowania. Kod będzie przetestowany za pomocą automatycznych testów jednostkowych (JUnit).

4.2 Najważniejsze elementy do przetestowania

W testach nacisk zostanie nałożony na przetestowanie wszelkich wyjątków dotyczących przekazywanych oraz zapisywanych parametrów.

4.3 Opis testów kluczowych

1. Dodawanie punktów za zabicie komórki przez odpowiedni czołg.
2. Odpowiednia liczba wystrzelonych pocisków.
3. Poprawna obsługa odczytu pliku konfiguracyjnego.
4. Poprawna obsługa zmiany parametrów po czasie.
5. Prawidłowy zapis obrazu okna po zakończeniu gry.
6. Obsługa wejścia klawiatury.
7. Prawidłowa obsługa zabiegu hermetyzacji atrybutów oraz metod klas.
8. Poprawna obsługa zmian ustawień dźwięku.