

Sprawozdanie końcowe dla projektu w javie pt.
„BattleZone”

...

Wykonali: Franciszek Wysocki, Bartosz Zdybel
Sprawdzający: mgr inż. Paweł Zawadzki
Data: 01.06.2020

Spis treści

1	Cel projektu	3
2	Struktura projektu	3
3	Różnice między specyfikacjami	5
4	Testowanie	6
5	Reakcje na błędy	7
5.1	Błędy zwykłe	7
5.2	Błędy krytyczne	8
6	Prawidłowy sposób uruchomienia programu	9
7	Poprawny format danych w pliku wejściowym	13
8	Wykorzystywane narzędzia oraz podział prac	15
9	Podsumowanie i wnioski	15

1 Cel projektu

Celem projektu było stworzenie programu w języku Java, który będzie posiadał graficzny interfejs użytkownika (Graphical User Interface, GUI) i umożliwi rywalizację dwóm graczom poprzez poruszanie się czołgami i strzelanie do komórek. Głównymi założeniami gry, którą nazwaliśmy „BattleZone” są:

- wczytywanie danych z pliku wejściowego o rozszerzeniu txt
- umożliwienie dwóm graczom poruszania się czołgami
- stworzenie planszy z komórkami
- zmniejszanie komórek co określony czas podany przez użytkownika
- stworzenie komórek dzieci, które będą powstawać według sąsiedztwa Moore’a
- stworzenie pocisków i ich przyspieszanie co określony czas
- stworzenie komórki armagedon, której zestrzelenie automatycznie kończy grę
- zapisanie stanu końcowego gry do pliku graficznego o rozszerzeniu png

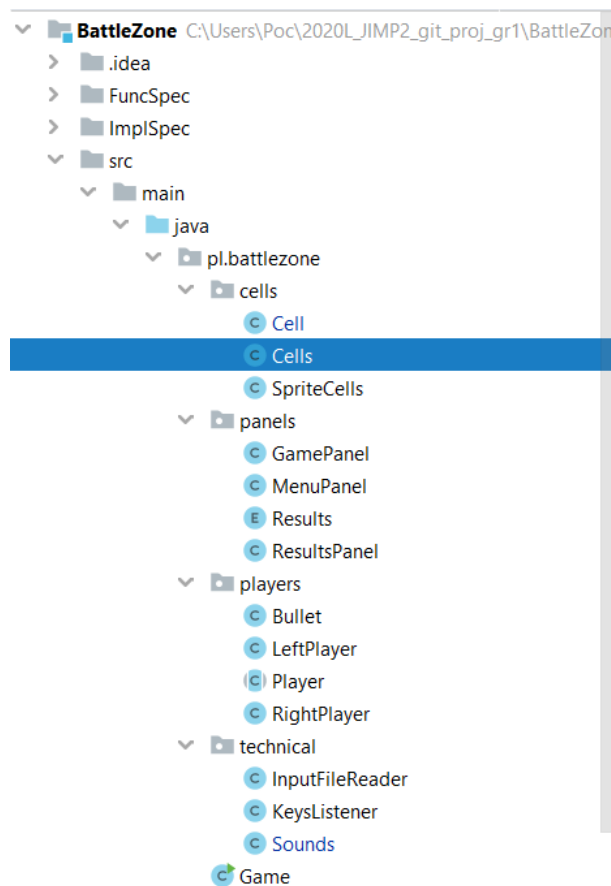
2 Struktura projektu

Główną klasą znajdującą się w naszym projekcie jest klasa Game, która dziedziczy po klasie JFrame. Klasa ta zawiera metodę main. W programie również zostały wyróżnione poniższe klasy:

- Cell - klasa reprezentująca komórkę
- Cells - klasa obsługująca i przechowująca listę komórek
- SpriteCells - klasa odpowiedzialna obsługę grafikę komórek
- GamePanel - główny panel gry
- MenuPanel - panel startowy
- Results - klasa wyliczeniowa przechowująca możliwe zakończenia programu
- ResultsPanel - panel końcowy z wyświetleniem zwycięzcy oraz możliwością zapisu do pliku graficznego
- Bullet - klasa obsługująca pociski
- Player - klasa abstrakcyjna reprezentująca gracza
- LeftPlayer/RightPlayer - klasa obsługująca lewego i prawego gracza

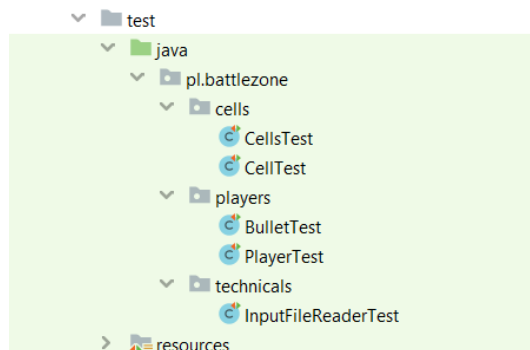
- InputFileReadeer - klasa odpowiedzialna za wczytanie danych z pliku konfiguracyjnego
- KeyListener - klasa odpowiedzialna za „nasłuchiwanie” klawiszy
- Sounds - klasa odpowiedzialna za dźwięk

Na poniższym zdjęciu (Obrazek 1) przedstawiona jest struktura projektu:



Obrazek 1

Zaimplementowaliśmy również 39 testów, które sprawdzają poprawność wybranych metod. Strukturę klas zawierającą testy znajduje się na obrazku poniżej (Obrazek 2):



Obrazek 2

3 Różnice między specyfikacjami

Program w większości powstał z głównym planem opisanym szczegółowo w specyfikacji implementacyjnej oraz specyfikacji funkcjonalnej. Niemniej jednak dokonaliśmy kilku zmian.

Pierwszą zmianą jest modyfikacja pliku wejściowego, który różni się formą wprowadzanych danych. W specyfikacji funkcjonalnej założyliśmy, że w pliku wielkość planszy podajemy w następujący sposób: 20X20, zmodyfikowaliśmy to do takiej postaci:

A=500

B=500

Było to spowodowane łatwiejszym sprawdzeniem błędów wprowadzonych przez użytkownika oraz chęcią uspoźnienia danych w pliku, dzięki czemu mogliśmy tylko raz napisać metodę odczytującą i wykorzystać ją dla każdej zmiennej (zgodnie z zasadą DRY). Drugą modyfikacją było usunięcie z pliku ostatniej zmiennej - M (maksymalna ilość punktów przewidziana na daną rozgrywkę), aby było to zgodne z wymaganiami i aby użytkownik mógł dostosowywać tę wielkość bez konieczności wchodzenia w plik konfiguracyjny.

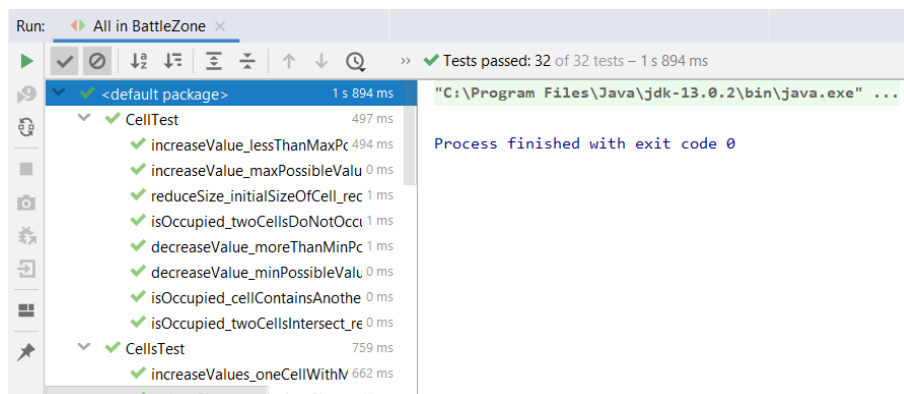
Ponadto zostały dodane klasy:

- Sound (klasa odpowiedzialna za dźwięk)
- ResultsPanel (klasa odpowiedzialna za wyświetlenie wyniku i możliwość zapisu końcowego stanu gry do pliku tekstowego)
- Results (typ wyliczeniowy (enum) przechowujący możliwe zakończenia gry)
- SpriteCells (klasa zajmującej się obsługiwaniem graficznym komórek)

Stworzyliśmy również nowy algorytm, który tworzy według sąsiedztwa Mor'a komórki dzieci wokół komórki rodzica. Algorytm sprawdza po kolei czy miejsca wokół komórek rodzica są wolne miejsca oraz czy wstawiona komórka-dziecko zmieści się w planszy i jeśli tak to tworzy tę komórkę.

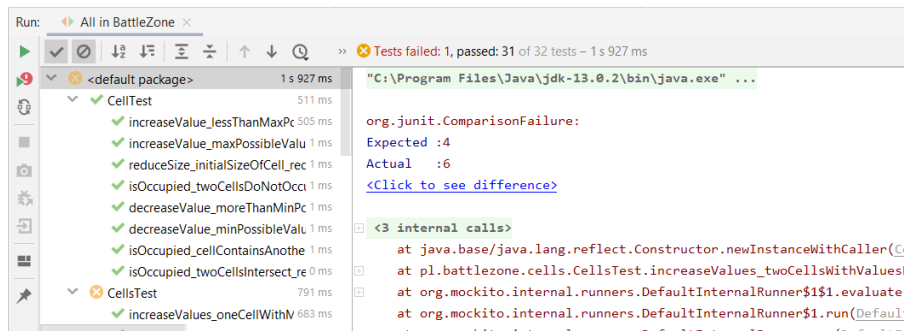
4 Testowanie

Testy jednostkowe można np. uruchomić zarówno za pomocą środowiska IntelliJ lub Mavena. Efekt działania zaprezentowany na zdjęciu poniżej (Obrazek 3):



Obrazek 3

W przypadku wprowadzenia modyfikacji do jednego testu, by działał niepoprawnie otrzymujemy informację o błędzie (Obrazek 4):



Obrazek 4

Testy korzystają z bibliotek: JUnit4, AssertJ oraz Mockito, a ich szkielet jest zgodny z schematem given/when/then. Nazwy testów są wzorowane konwencją „nazwaMetody_podaneDane_oczekiwanyResultat()”. Testy były pisane równolegle do powstawania kodu.

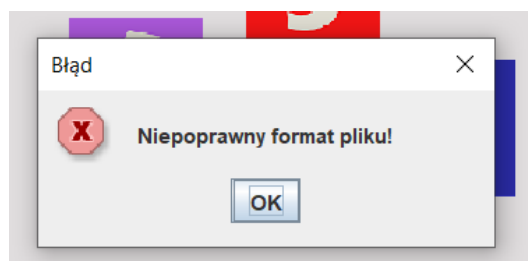
Poza testami jednostkowymi program został przetestowany także manualnie, a ewentualne poprawki były korygowane na bieżąco. Działanie to pozwoliło zabezpieczyć program przed ewentualnymi błędami użytkownika o czym więcej w sekcji „Reakcja na błędy”.

5 Reakcje na błędy

W projekcie zostały wyróżnione 2 rodzaje błędów: zwykłe i krytyczne. Te pierwsze pozwalają na korektę ze strony użytkownika, te drugie natomiast kończą działanie programu.

5.1 Błędy zwykłe

W przypadku gdy użytkownik chce wprowadzić plik z błędnym formatem, program mu na to nie pozwoli i wyświetli stosowny komunikat (Obrazek 5):

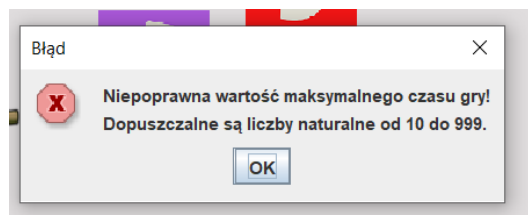


Obrazek 5

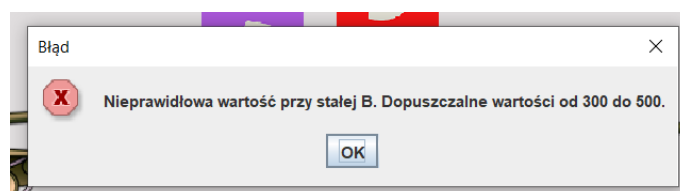
Program zadziała podobnie w odpowiedzi na:

Pozostawienie pustych miejsc w nazwie gracza, maksymalnej liczbie punktów i czasie rozgrywki - pola te zostają one wypełnione wówczas wartościami domyślnymi.

Błędnie wprowadzone dane: (Obrazek 6 i 7):

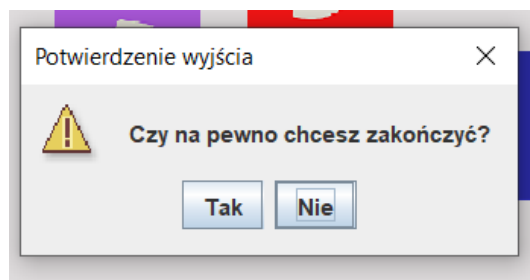


Obrazek 6



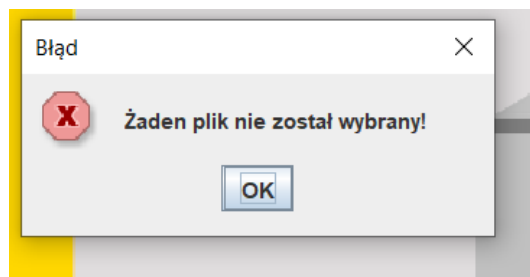
Obrazek 7

Przypadkowe zamknięcie programu - wyświetlenie potwierdzenie (Obrazek 8):



Obrazek 8

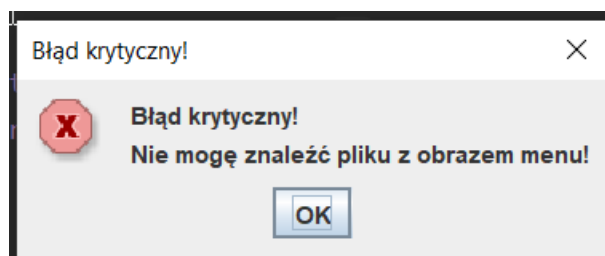
Niewybranie własnego pliku konfiguracyjnego, pomimo odznaczenia checkboxa z domyślnym plikiem (Obrazek 10):



Obrazek 9

5.2 Błędy krytyczne

Błędy krytyczne pojawiają się wtedy, gdy kontynuowanie gry nie jest możliwe. np. Zostanie usunięty obraz tła (Obrazek 10).



Obrazek 10

We wszystkich wyżej wymienionych przykładach poza komunikatami pojawia się także dźwięk sygnalizujący błąd.

6 Prawidłowy sposób uruchomienia programu

W celu prawidłowego uruchomienia aplikacji wystarczy na nią dwukrotnie nacisnąć, gdyż został stworzony odpowiedni Manifest. Można także skorzystać z polecenia „java -jar”, lecz program został napisany w taki sposób, aby wszystkie komunikaty pojawiały się w formie wyskakujących okienek, a nie informacji w konsoli, dlatego sposób ten nie jest to konieczny. Po wykonaniu jednej z tych operacji wyświetli się Menu, gdzie użytkownik może załączyć plik konfiguracyjny (lub grać z domyślnym), wybrać imiona graczy, zdecydować, czy chce grać z dźwiękiem czy bez, wybrać maksymalną ilość punktów oraz czas rozgrywki (Obrazek 11).

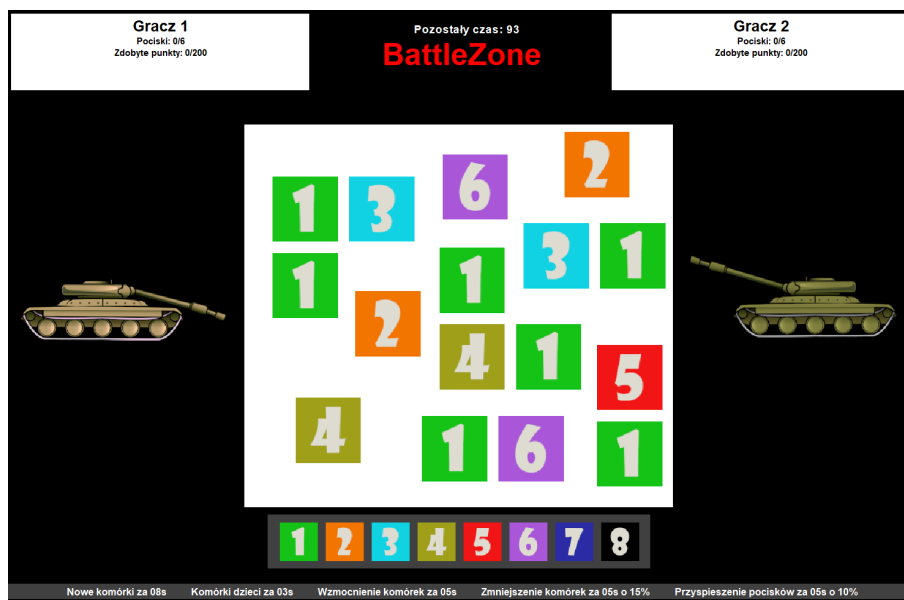


Obrazek 11

Następnie należy nacisnąć przycisk „Rozpocznij” i rozgrywka się zaczynie. Tak jak założyliśmy w specyfikacji funkcjonalnej użytkownicy mogą poruszać się swoimi czołgami:

- Prawy użytkownik strzałką w górę i w dół, a lufą strzałkami w prawo i w lewo
- Lewy użytkownik porusza się klawiszami W - w górę, S - w dół oraz do sterowania lufą służą klawisze A i D

Przykład gry znajduje się na rysunku poniżej (Obrazek 12):



Obrazek 12

W prawym i lewym górnym rogu gry znajduje się kolejno imię/nick gracza, jego ilość aktualnie wystrzelonych pocisków oraz zdobytych punktów (Obrazek 13):



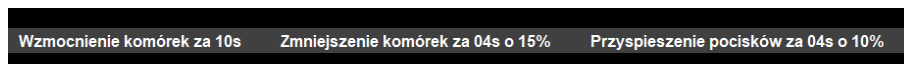
Obrazek 13

Na samej górze po środku znajduje się czas jaki pozostaje do końca rozgrywki oraz nazwa gry (Obrazek 14):



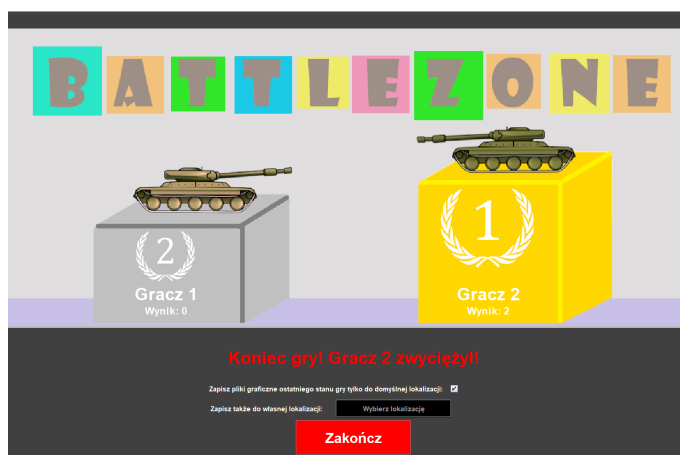
Obrazek 14

Na dole planszy znajduje się dane podane w pliku konfiguracyjnym jak np. czas do powstania nowych komórek, komórek dzieci itd. Fragment poniżej (Obrazek 15):



Obrazek 15

Istnieje kilka możliwości zakończenia programu, np. Prawy gracz wygrywa poprzez większą ilość punktów (Obrazek 16) albo remis (Obrazek 17):

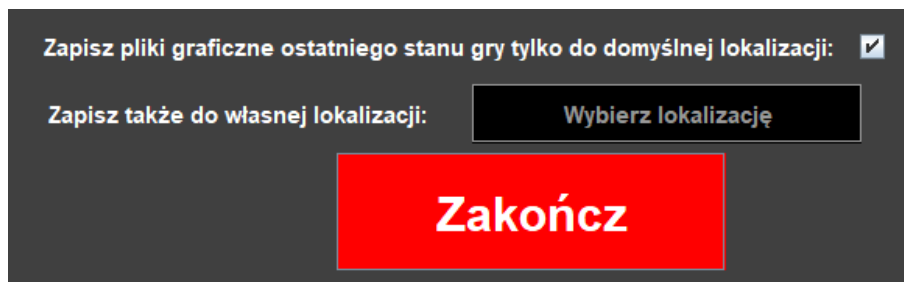


Obrazek 16

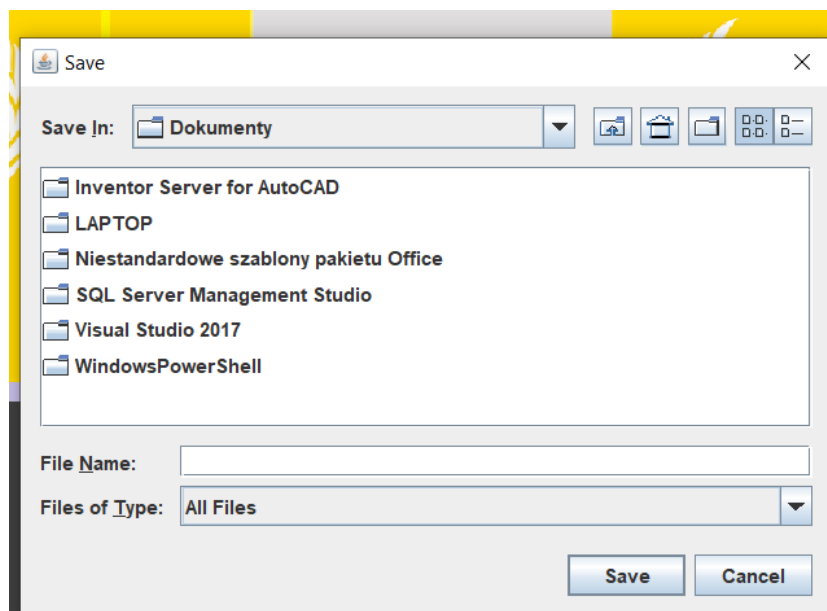


Obrazek 17

Nad przyciskiem „Zakończ” znajduje się możliwość zapisu planszy do pliku o rozszerzeniu png, w przypadku niewybrania lokalizacji, zdjęcia zostaną zapisane w lokalizacji domyślnej - katalogu, w którym znajduje się aplikacja (Obrazek 18), natomiast w przypadku chęci zapisu do własnej lokalizacji pojawi się okno zapisu (Obrazek 19):



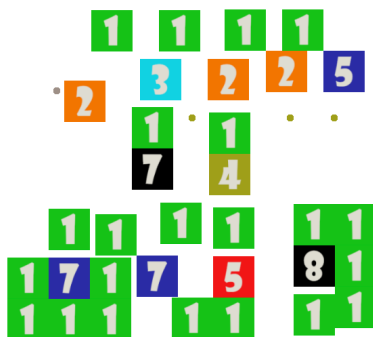
Obrazek 18



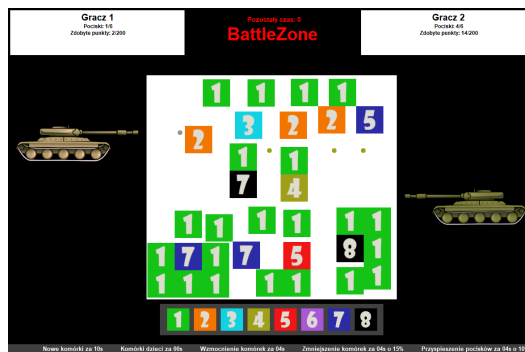
Obrazek 19

Jeżeli użytkownik się pomyli i poda nazwę pliku z rozszerzeniem innym niż png, program sam dostosuje to rozszerzenie.

Efektem wybrania lokalizacji przez użytkownika/skorzystania z lokalizacji domyślnej jest stworzenie dwóch obrazków, jeden z planszą na której znajdują się komórki (Obrazek 20) oraz drugi na którym znajduje się cały końcowy panel gry (Obrazek 21):



Obrazek 20



Obrazek 21

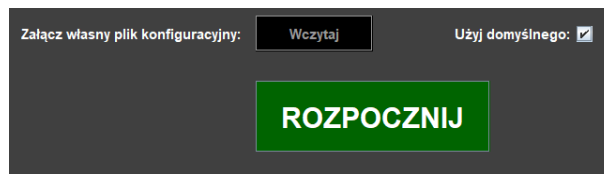
7 Poprawny format danych w pliku wejściowym

Wszystkie dane w pliku powinny być zapisane w następujący sposób w pliku z rozszerzeniem txt (Obrazek 22):

```
A=560
B=500
P=6
H=15
X=5
Y=12
Z=6
K=10
L=15
```

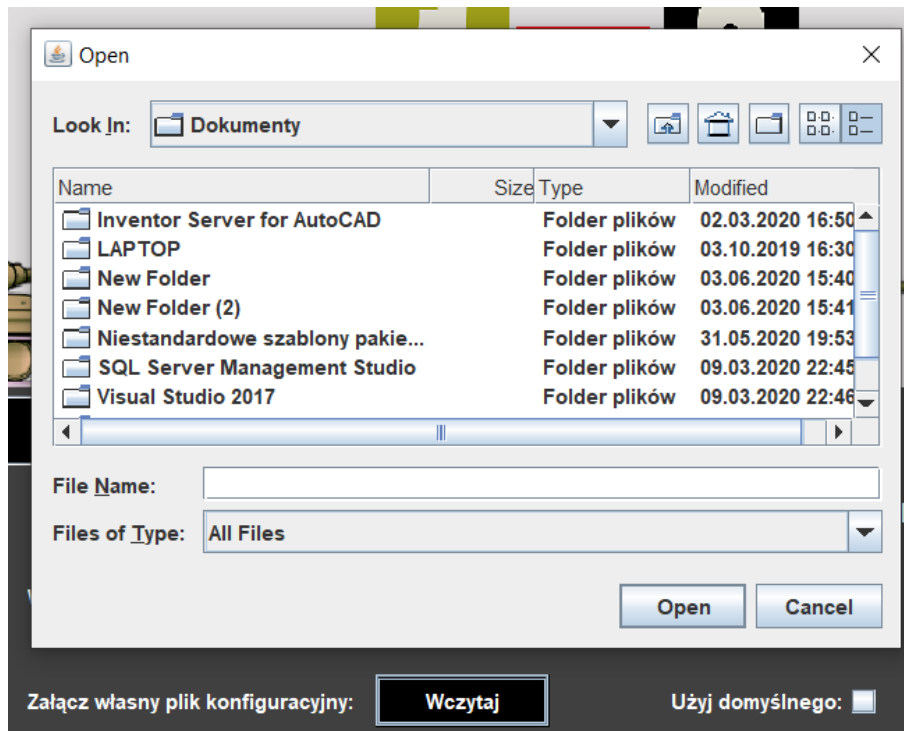
Obrazek 22

W przeciwnym razie program nie pozwoli na rozpoczęcie gry z tym plikiem wejściowym (Obrazek 5). Istnieje możliwość uruchomienia gry z domyślnym plikiem tekstowym. W tym celu należy pozostawić pola tak jak na obrazku poniżej (Obrazek 23):



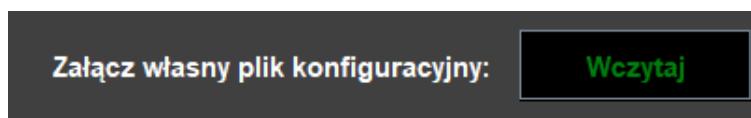
Obrazek 23

Gdy użytkownik będzie chciał załączyć własny plik pojawi się poniższe okno (Obrazek 24):



Obrazek 24

Gdy plik będzie poprawny, a dane będą w odpowiednich zakresach to przycisk zmieni kolor na zielony (Obrazek 25), w przeciwnym razie wyświetli się komunikat o błędzie (Obrazek 9).



Obrazek 25

8 Wykorzystywane narzędzia oraz podział prac

- Maven: Narzędzie to okazało się bardzo pomocne. Maven posłużył nam w automatyzacji budowania aplikacji, wykorzystania zewnętrznych bibliotek oraz uruchamiania testów.
- Git: Repozytorium gita posłużyło nam do pracy równoległej na wielu branchach, testowania nowych rozwiązań i kilkakrotnie do powrotu do wcześniejszych.
- IntelliJ: Powyższe środowisko bardzo ułatwiło pracę nad kodem. Korzystaliśmy z wielu skrótów klawiszowych, pluginów i innych udogonień tego programu, dzięki czemu praca była szybsza i przyjemniejsza.
- Gimp: Stworzenie grafiki (sprite) czołgu, komórek, tła.
- Paint: Wstępna wizualizacja gry.
- Texture Packer GUI: Pakowanie grafik (sprites) do jednej (spritesheet).
- Launch4U: Przekonwertowanie pliku jar do exe. (Oba te pliki przygotowaliśmy w podkatalogu FINAL_RELEASE wraz z przykładowymi danymi konfiguracyjnymi).
- Bardzo często sięgaliśmy także do dokumentacji Javy ze strony Oracle i szukaliśmy przydatnych rozwiązań. Wykorzystaliśmy także strony z darmową licencją takie jak: OpenGameArt.org do znalezienia dźwięków do gry oraz favpng.com do znalezienia grafiki czołgu, którą w Gimpie przerobiliśmy na dwa czołgi o różnych ułożeniach lufy.

Podział prac:

Bartosz odpowiadał za wczytanie danych z pliku wejściowego, komórki, zapis do pliku.

Franciszek odpowiadał za klasę odpowiedzialną za pociski, panele, graczy oraz grafiki.

Wspólnie opracowaliśmy plan oraz sposób działania naszego programu, stworzyliśmy silnik gry, testy, dobraliśmy dźwięki i kolory do programu, wykonaliśmy sprawozdanie końcowe oraz obie specyfikacje.

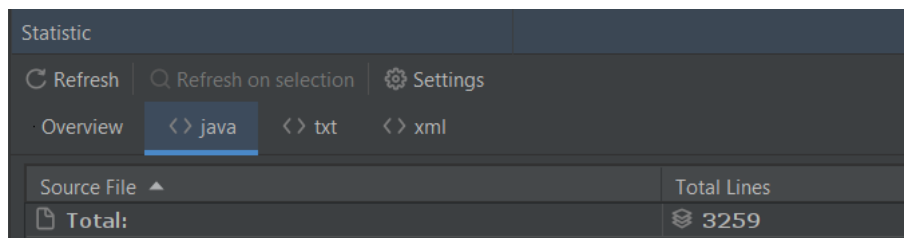
Przy następnym projekcie zadbalibyśmy lepiej o częstsze wymienianie się pomysłami oraz konsultowanie wprowadzanych zmian.

9 Podsumowanie i wnioski

Różnice programu końcowego, między tym co założyliśmy wynikają z nowych pomysłów i udoskonaleń. Przykładem jest stworzenie nieplanowanej klasy Sounds odpowiadającej za dźwięki w programie. Stworzyliśmy wiele udoskonaleń i uproszczeń wykorzystując GUI, dzięki czemu (subiektywnie) program jest estetyczny, a interfejs prosty do użytku.

Staraliśmy się napisać kod czysty i czytelny, dlatego wielokrotnie go refaktoryzowaliśmy. W rezultacie metody są krótkie, konkretne metody odpowiadają za konkretne funkcjonalności, nazwy zmiennych i metod (a także treści commitów) są pisane w języku angielskim zgodnie z ogólnymi konwencjami i ograniczyliśmy ilość powtórzeń. Zrezygnowaliśmy także z pisania komentarzy.

Po za tym duży nacisk położyliśmy na tym, aby program odpowiednio reagował na błędy ze strony użytkownika. Finalna wersja została wielokrotnie przetestowana. Łącznie w programie zostało napisanych 3259 linijek Javy (według plugina Statistics) (Obrazek 26). Co składa się na 17 klas (w tym 5 testowych).



Statistic	
Refresh	Refresh on selection
Settings	
Overview	<> java <> txt <> xml
Source File ▲	Total Lines
Total:	3259

Obrazek 26

W programie wyróżniliśmy także 5 dźwięków: wystrzał, koniec gry, tykanie zegara, zestrzelenie komórki oraz dźwięk wyskakującego błędu z Windowsa XP.

Większość czasu, który poświęciliśmy na projekt, poświęciliśmy faktycznie na naukę. Bardzo przydatne okazały się fora, gdzie znajdowaliśmy odpowiedzi na nasze wątpliwości, jak i tutoriale na YouTube.

Obydwu nam praca nad tym projektem sprawiła przyjemność.