1. ISTOTA PROJEKTU

Gra Battleships to klasyczna gra w statki. Polega na jednej aplikacji serwera i dwóch aplikacji klienckich komunikujących się za jego pośrednictwem. Grać może jednocześnie dwóch graczy, po skończonej grze można jeszcze raz uruchomić dwie aplikacje klienckie i zagrać jeszcze raz (ten proces można powtarzać dowolną ilość razy). Graficzny interfejs użytkownika zrealizowany jest za pomocą JavaFX.

2. SPOSÓB URUCHOMIENIA APLIKACJI

Program widoczny w systemie kontroli wersji zawiera (oprócz folderu z projektem, który można normalnie uruchomić za pośrednictwem IDE) również folder z uruchamialnymi plikami JAR (działającymi wg testów na Windowsie i Ubuntu, inne systemy nie były testowane). Zawierają one wszystkie potrzebne zależności. Wystarczy z poziomu terminala/PowerShella/cmd uruchomić plik serwerowy i dwie instancje pliku klienckiego, żeby przetestować jak program działa (zwyczajne kliknięcie też działa, ale warto zwrócić uwagę wówczas na to, że serwer działać będzie w tle).

3. SCHEMAT GRY

1. Logowanie
2. Faza ustawiania statków (jeden 4-masztowiec, dwa 3-masztowce, trzy 2-masztowce i cztery 1-masztowce). Zgodnie z zasadami gry w statki, statki nie mogą się stykać, czego pilnuje program.
3. Potwierdzenie gotowości (odpowiednim przyciskiem)
4. Faza gry (standardowe strzelanie w pola przeciwnika do momentu chybienia)
5. Koniec gry (wygrana lub przegrana z uwagi na zestrzelenie wszystkich statków lub wyjście przeciwnika). Koniec występuje też szybciej, gdy serwer zostanie zamknięty.

4. STRUKTURA PROGRAMU

Program podzielony jest na moduły-Client oraz Server oraz nadzorujący je moduł Battleships.

5. UŻYTE TECHNOLOGIE DODATKOWE

- SQL
  W ramach SQL zaimplementowaliśmy logowanie się do bazy użytkowników (za pomocą SQLite). Jeśli użytkownik użyje nazwy, która już istnieje, wymagane jest wpisanie poprawnego hasła, tj. Podanego przy pierwszym logowaniu (o czym w razie czego program informuje). Jeśli użytkownik użyje jeszcze nieistniejącej nazwy, wprowadzone przez niego hasło zostanie przypisane do tejże nazwy.

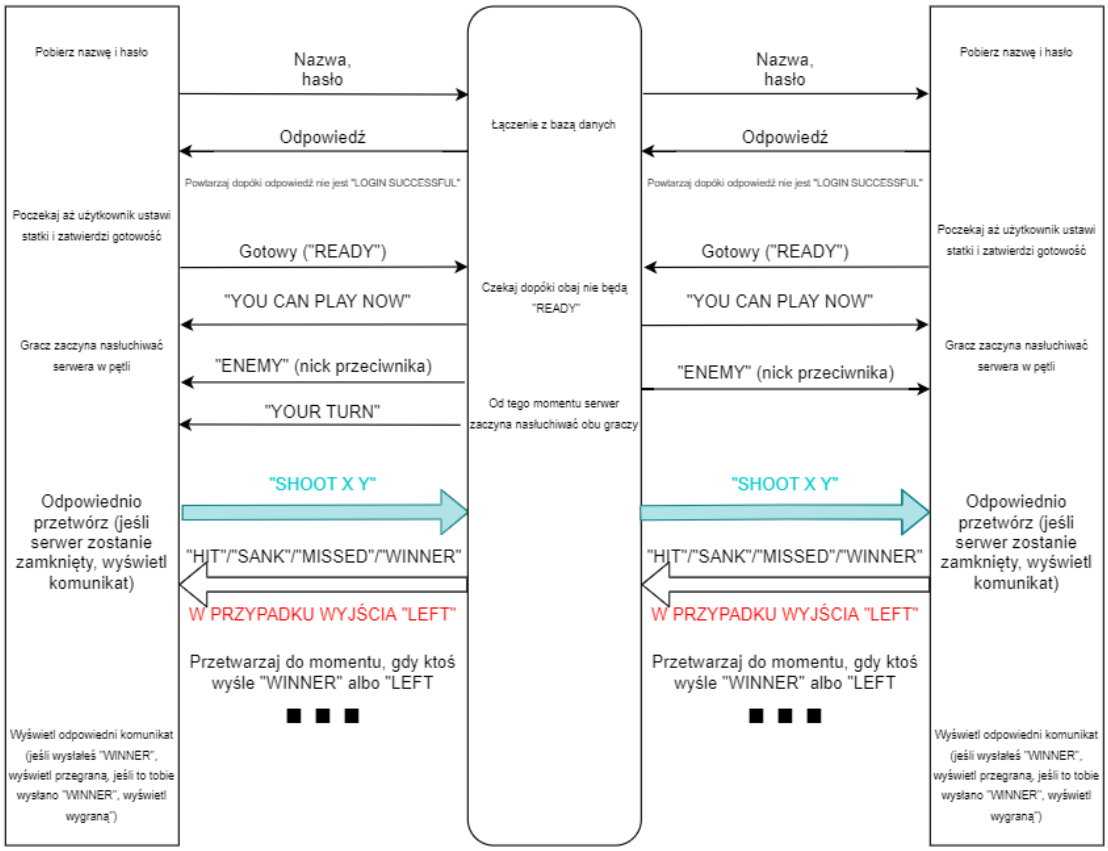
  Hasła są przechowywane w formie hashowanej (za pomocą biblioteki JaSypt), z kolei wszelkie funkcjonalności konieczne do łączenia się z bazą i operacji na danych w niej się znajdujących są zaimplementowane przez obiekt stworzonej przez nas klasy DatabaseConnector.

- JSON
  Serwer dodaje komendy graczy do tablicy JSON (za pomocą JSONSimple), a na koniec swojego działania je zapisuje w odpowiednim pliku JSON. Stworzony plik pozwala prześledzić wysyłane przez serwer komunikaty z ostatniej gry.

- JavaFX
  Graficzny interfejs użytkownika wykonaliśmy przy użyciu JavaFX. W tym celu wygenerowaliśmy dwa pliki .fxml w narzędziu SceneBuilder oraz napisaliśmy do nich odpowiednie klasy kontrolerów.

6. PROTOKÓŁ KOMUNIKACJI KLIENT-SERWER

W naszym projekcie serwer działa jak przekaźnik, a to programy klientów decydują co zrobić z otrzymaną wiadomością. Wszystkie wiadomości obsługiwane są automatycznie i użytkownicy na bieżąco widzą zmiany będące ich wynikiem (decydują tylko gdzie strzelić). Poniżej zamieszczona wizualizacja przybliża działanie zastosowanego protokołu komunikacyjnego.



Po wyjściu obu graczy serwer jest znów gotowy przyjmować kolejną parę

7. TESTY JEDNOSTKOWE

Plik TestPlayerBoard.java z testami jednostkowymi opartymi na JUnit testuje plik PlayerBoard.java, będący głównym trzonem logiki gry. Inne klasy nie były testowane, bo znajdują się w jednej z poniższych kategorii:

-komunikują się przez sieć

-są klasami związanymi z GUI

-są klasami trywialnymi (znajdują się w nich tylko gettery, settery lub metody analogiczne do już przetestowanych)

-wywołują praktycznie tylko polecenia z biblioteki zewnętrznej