**• Wymagania oraz zasoby, jakie program musi spełniać lub posiadać, żeby mógł się uruchomić i działać (np. login i hasło itp.):**

Aplikacja nie wymaga żadnych szczególnych uprawnień oraz zasobów żeby się uruchomić. Do programu została dołączona biblioteka gson do serializacji i deserjalizacji danych przed wysłaniem między klientem a serwerem.

**• Krótki opis działania programu (dostarczana funkcjonalność):**

Gra multiplayer Bomberman przeciw drugiemu człowiekowi i przeciw AI.

W zamyśle generalnym, jest to system gracz-serwer. Serwer zarówno obsługuje proces matchmakingu oraz mechanikę gry.

Do realizacji wykorzystaliśmy odrębny interfejs dla gracza, oraz dla serwera działający w oparciu o zasadę kto pierwszy ten lepszy.

W ramach wizualizacji gry użyliśmy mapy z wyznaczonymi ścianami, cegłami oraz alejkami, rysowanymi przy pomocy biblioteki Graphics2D.

Do kontroli postaci:

W – w górę

A – w lewo

D – w prawo

S – w dół

P – pauzuje

Esc – wychodzi się z gry

Spacja – bomba

Drugi gracz ma takie samo sterowanie nie ma odbicia lustrzanego.

**• Opis w jaki sposób spełniono wymagania techniczne (zob. Implementacja):**

Ponad 25 własnych klas z polami i metodami o różnym dostępie, dziedziczących po sobie. Klasa Game to rozszerzony JFrame. Klasa postaci, która jest klasą abstrakcyjną faktycznie - AbstCharcter ma własne abstrakcyjnej metody.

Składają się na nią dwie klasy player i monster, która sprawdzają m.in. koordynaty X Y, czy żyje, czy nie dostał bombą. Zastosowano rzutowanie typów z game na JFramea w obu kierunkach przed parametrem w nawiasie jest nowa wartość typu danych. Rysowanie postaci odbywa się za pomocą funkcji.

• interfejs Swing:
Po stronie klienta mamy wykorzystany interfejs okienkowy w Swingu korzystający z menadżera układów CardLayout i komponentów takich jak buttony, labele, imageicon.

• obsługa wyjątków:
W klient serwer przy łączeniu i rozłączaniu wykorzystujemy obsługę wyjątków. W ramach tego wywołuje się np. metodę kill.

• obsługa zdarzeń:
Implemtacja pause po wciśnięciu P - stan gry zmienia się na zapauzowany i przykrywa grę napis GAME PAUSED. Obsługa klawisza ESC w trakcie gry, która zakańcza grę. Card Layout opiera się na zdarzeniach niestandardowych, po naciśnięciu przycisku jest przeniesie do innych kart. W klient serwer Zachowanie przycisku i Card Layout zależy od innego zewnętrznego parametru isMaster.

• użycie kontenerów danych:
Kontenerami danych poza sztandarowymi są obiekty klas np. tile block , sterta stack, oraz w pewnych etapach projektu używaliśmy tez Arraylist, która ostatecznie została zamieniona na stertę.

• czytanie i pisanie z/do plików:
Cała architektura klient serwer jest zrobiona na stream reader(buffered reader) i writer, które mogą być używane do zapisu pików ale u nas skorzystaliśmy z nich do obsługi klient serwer( korzystając jednak z tych samych typów danych). Z uwagi na brak czasu nie zdecydowaliśmy się na import export ale działa to dokładnie tak samo w komunikacji klient serwer – porozumiewa się jako przepływ danych w pliku. Priorytetem dla nas była architektura klient serwer i działająca mechanika gry.

• wątki
Po pierwsze pętla gry jest zbudowana na wątku. Wykorzystaliśmy wątki tam gdzie użyty jest update i draw i czekanie w czasie tam gdzie są fps. Po drugie cala architektura klient serwer jest oparta na watkach. Sesja ma jedno wystąpienie. I klient jest serwerem i serwer jest klientem żeby input i output nie stanowił problemów. Ze względu na problemy z jednoczesnym połączeniem wielu klientów z serwerem i input i output wielu obiektów w tym samym

momencie. Komunikacja jest jednostronna od serwera do klienta, po tym pierwszym polaczeniu tworzy się nowe połączenie ale stare nadal istnieje w celu komunikacji dwustronnej.

• model klient – serwer (komunikacja przy pomocy socketów)
- parowanie klientów

Single player:
Single player po uruchomieniu generuje losową mapę, po wylosowaniu mapy zbiera zwartość, przeszukuje gdzie są puste pola i potem w te puste pola, które znajdzie generuje przeciwników ale nie w ścianach.
Przy zderzeniu się ze ścianą przeciwnika jest losowany jego kolejny kierunek ruchu – jest to random zrobiony na enum vector i on generuje kierunki jak się zderzy ze ścianą. Jest checker napisany który sprawdza czy postać zbliżyła się do ściany i wtedy wywoływane jest to losowanie w przypadku postaci gracza nie ma odwrócenia kierunku tylko gracz nie może pójść ani piksel dalej. Każdy obiekt abstrakcyjny jak postać ma swój „twardy środek", czyli to czym nie może przeniknąć ściany. Twarda cześć postaci zapewnia ten luz ponieważ checker umożliwiający ten luz jest relatywny do tej twardej części postaci.

Multiplayer:
Wyjątek w lewym górnym rogu i prawym dolnym rogu: wolne miejsca bez bloków żeby można było postawić bombę i nie zostać przez nią zabitym.
W trybie multiplayer w momencie zderzenia ze sianą następuje obrót zgodnie z ruchem wskazówek zegara . Klient może być jednocześnie serwerem. W trybie multiplayer w momencie inicjacji komunikacji jest generowana od serwera do klienta mapa w formacie JSON i wysyłana do klienta. Klient to odbiera i na tej podstawie generuje identyczna mapę. Tak samo informację o wrogach serwer generuje i wysyła do klienta. Dzięki temu zapewnia się, że stan mapy i wrogów jest identyczny zarówno dla gracza jak i serwera.

**• Inne zastosowane mechanizmy języka, funkcje, dodatkowe biblioteki (jeśli użyto) itp.;**

Do osiągniecia celów wyżej określonych posłużyła biblioteka gson. Użycie własnego typu danych enum do określania kierunku poruszania się postaci. Użyto sterty, zaimportowano typ danych java stack.

• **Największe problemy:**

Największym wyzwaniem okazała się cała obszerna forma projektu. Zarówno mechanika gry jak i implementacja architektury klient serwer stanowiły nie lada wyzwanie spowodowane m.in złożonością struktury klas jaki i ich liczbą.

• **Największym powodem do dumy:**

Największym powodem do dumy był moment w którym po uruchomieniu programu na drugim komputerze obydwaj gracze mogli poruszać się w tym samym momencie i widzieć się nawzajem .

• Do projektu została załączona dokumentacja techniczna kodu wygenerowana za pomocą JavaDoc.