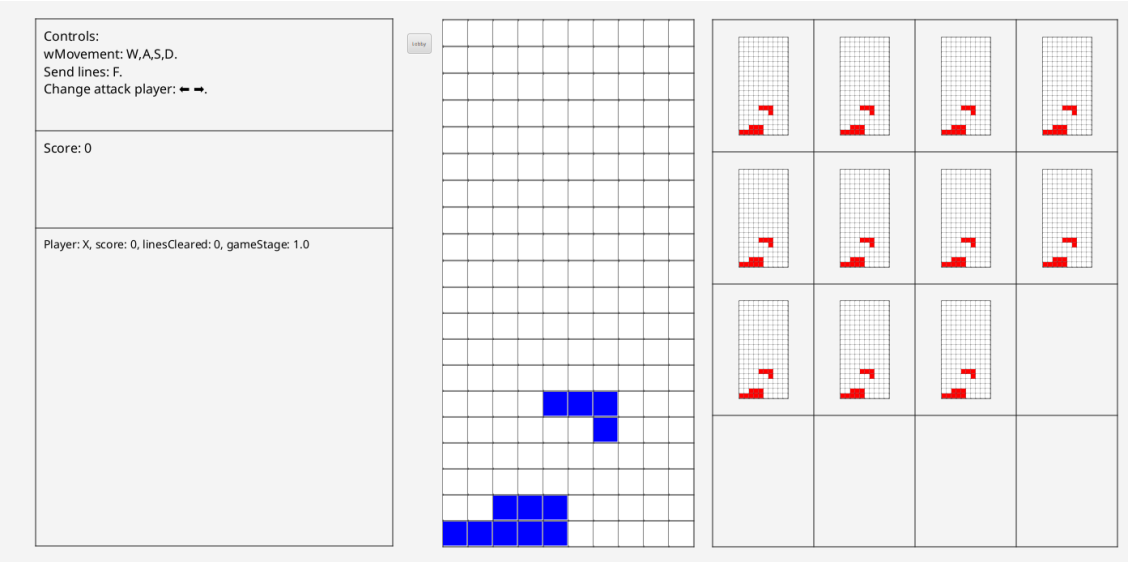
Tetris Battle Royal

Łukasz Kołakowski 198000

1. Wstęp

Celem projektu jest utworzenie gry wieloosobowej opartej na grze *Tetris 99* od Nintendo Switch. Gra od strony klienta będzie napisana w języku Java z wykorzystaniem JavaFX do wyświetlania obrazu użytkownikowi. Server zostanie napisany w języku C z wykorzystaniem tylko podstawowych bibliotek standardowych, oraz tych do obsługi wątków (pthread.h) i gniazdek sieciowych (arpa/inet.h), protokół TCP będzie odpowiedzialny za przesyłanie informacji pomiędzy graczami.



Rysunek 1. Poglądowy rysunek interfejsu gry

2. Działanie z punktu widzenia użytkownika

Użytkownik na początku będzie dołączać do lobby w którym będzie czekał na połączenie się wszystkich graczy, jak to nastąpi na środku ekranu wyświetli się plansza na której każdy z graczy będzie grać swoją rozgrywkę w *Tetris*, po prawej będą wyświetlane plansze przeciwników które będą aktualizowane w czasie rzeczywistym, po lewej stronie znajdować się będzie tabela wyników w której pojawią się wyniki innych graczy. Gracz może poruszać bloczkiem za pomocą klawiszy W, A, S i D, jaki obracać go za pomocą Q i E. Modyfikacją względem tradycyjnego Tetris'a jest możliwość wysyłania przeciwnikowi linii na spód jego planszy z jednym brakującym klockiem do uzupełnienia. Gracz może zaznaczyć gracza do którego wyśle linie za pomocą strzałek ← →, i wysłać do niego linie za pomocą klawisza F. Gracz dostaje linie do wysłania tylko, gdy samemu uda mu się wypełnić linię u siebie.

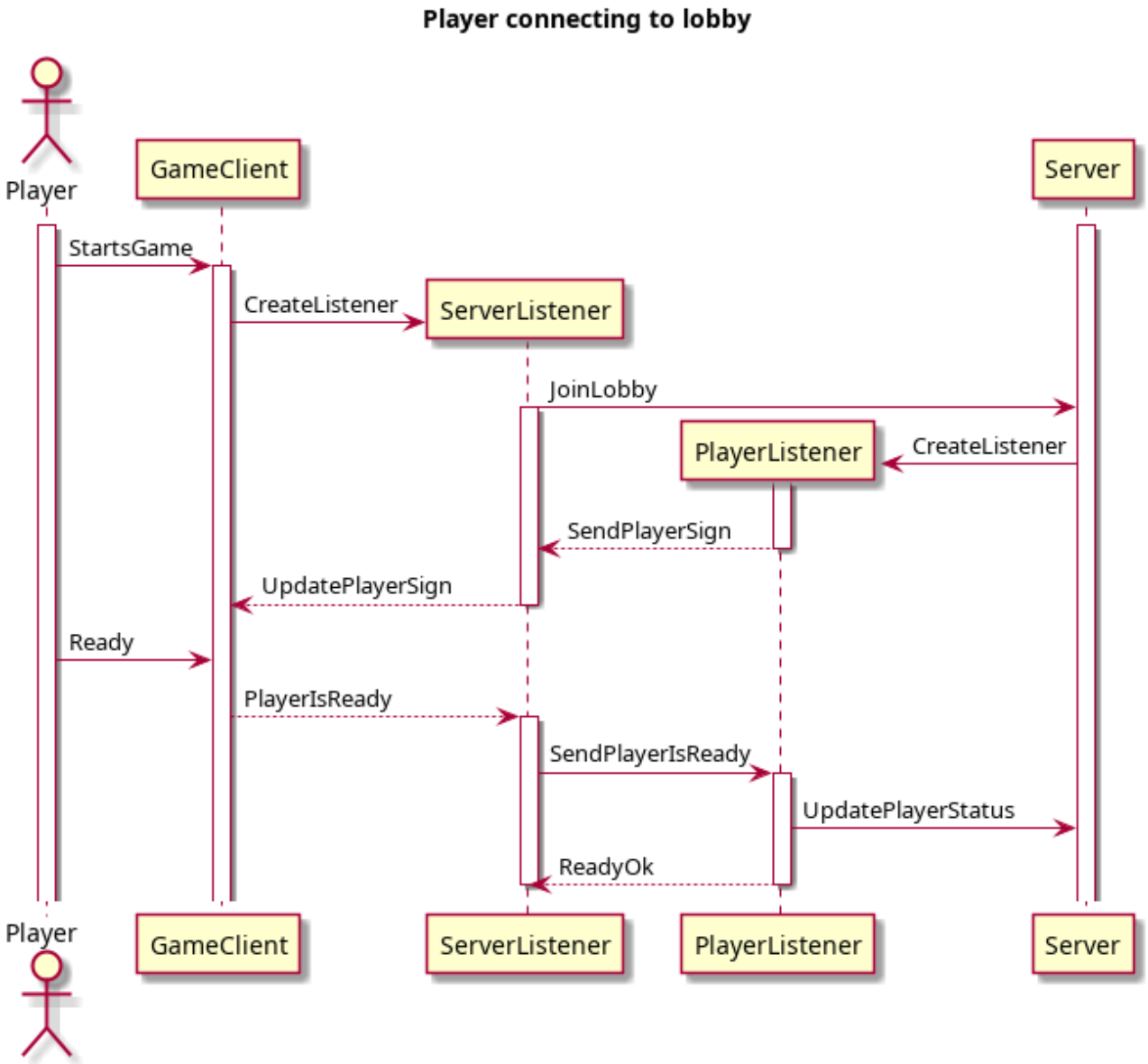
3. Działanie z punktu widzenia serwera

Serwer ma za zadanie przesyłanie informacji pomiędzy graczami, synchronizacja jest na drugorzędnym miejscu, ponieważ wpływ jednego gracza na drugiego jest mały. Na początku jest odpowiedzialny za dołączenie wszystkich graczy do jednej poczekalni. Po czym wysłanie komunikatu o starcie gry. W trakcie trwania gry serwer powinien zapamiętywać i przesyłać:

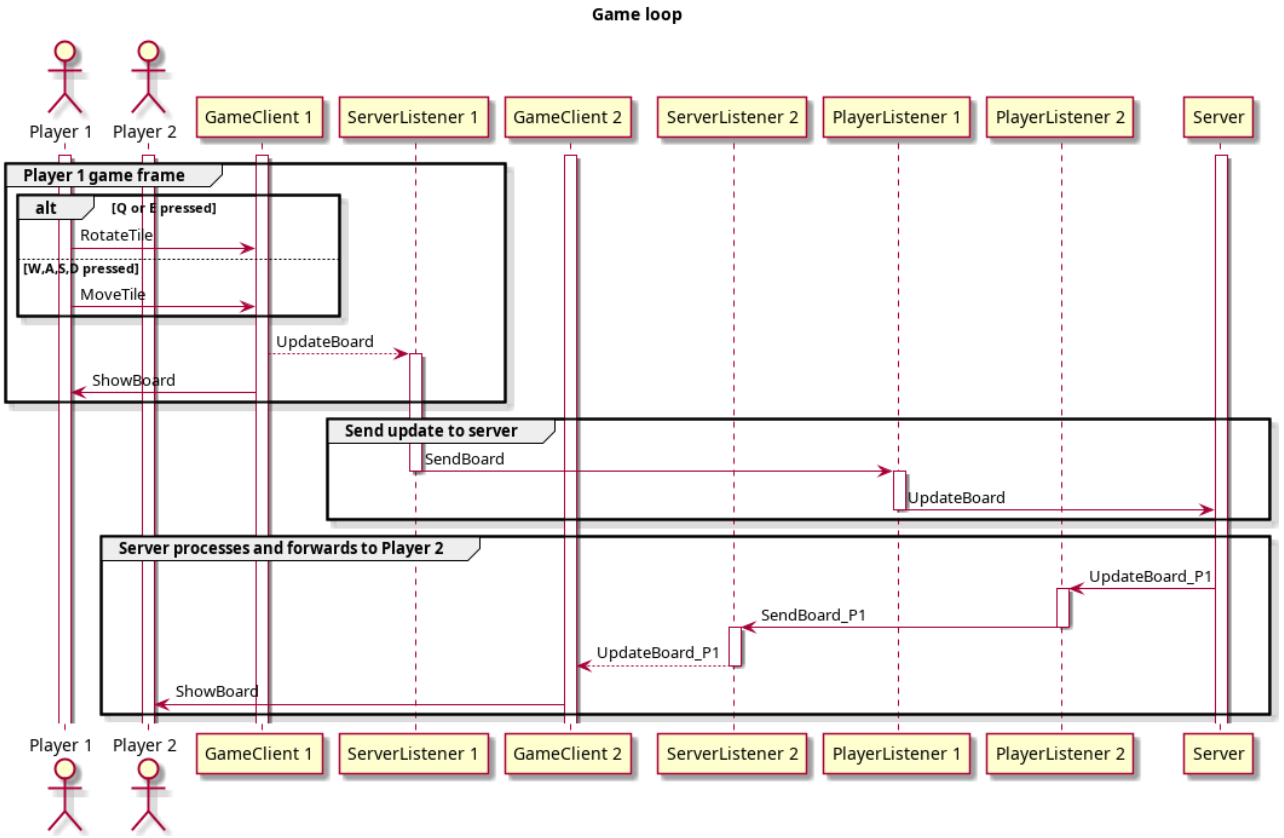
- zaktualizowane pozycje planszy gracza do reszty użytkowników;
- aktualizowanie tablicy wyników;
- informacje jaki gracz wysłał jakiem innemu graczowi daną liczbę linii.

4. Diagramy Sekwencyjne

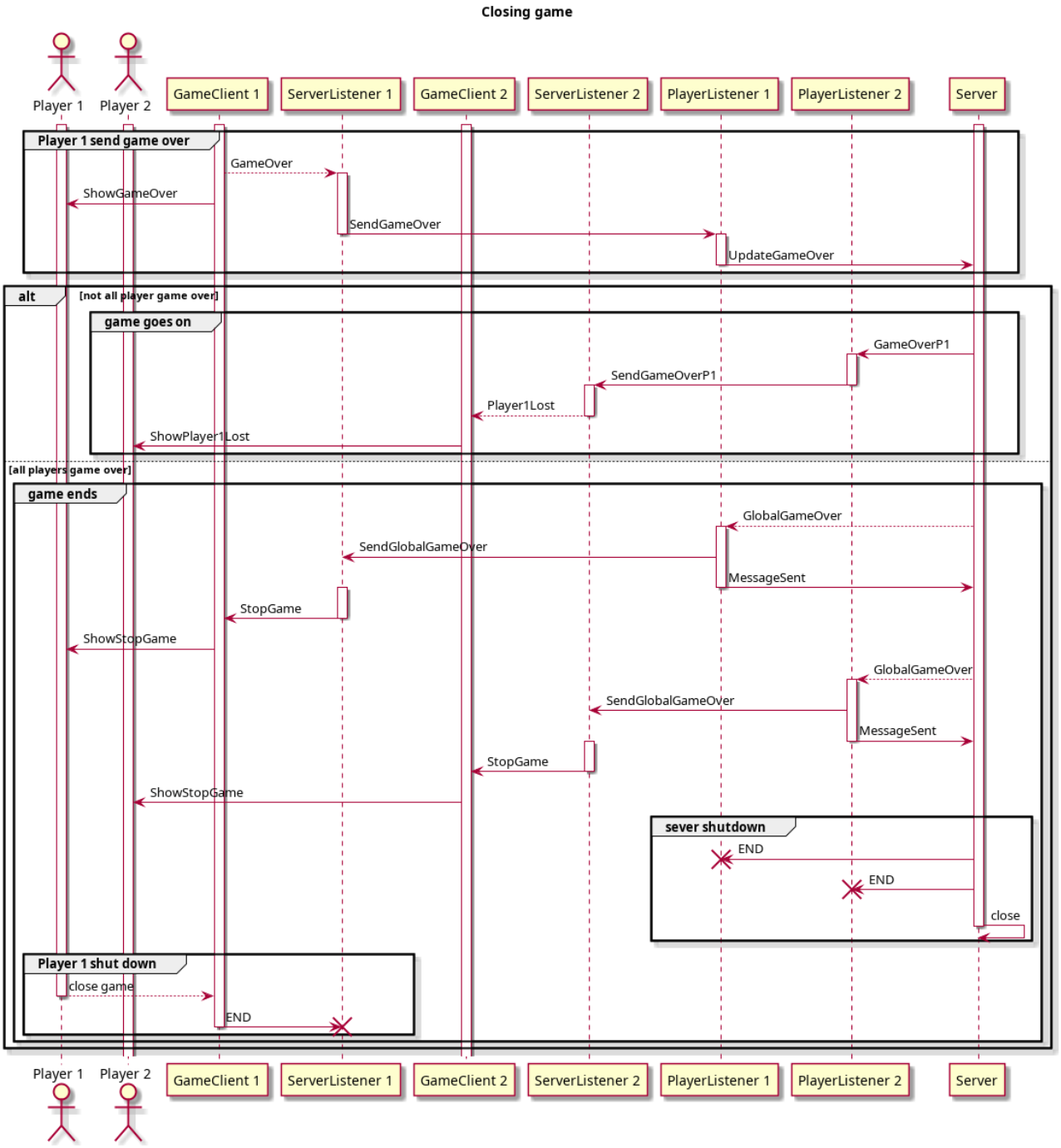
Połączenie gracza do poczekalni, proces przebiega analogicznie dla każdego z graczy.



Rysunek 2. Przedstawia sekwencję czasową łączenia gracza do poczekalni na serwerze

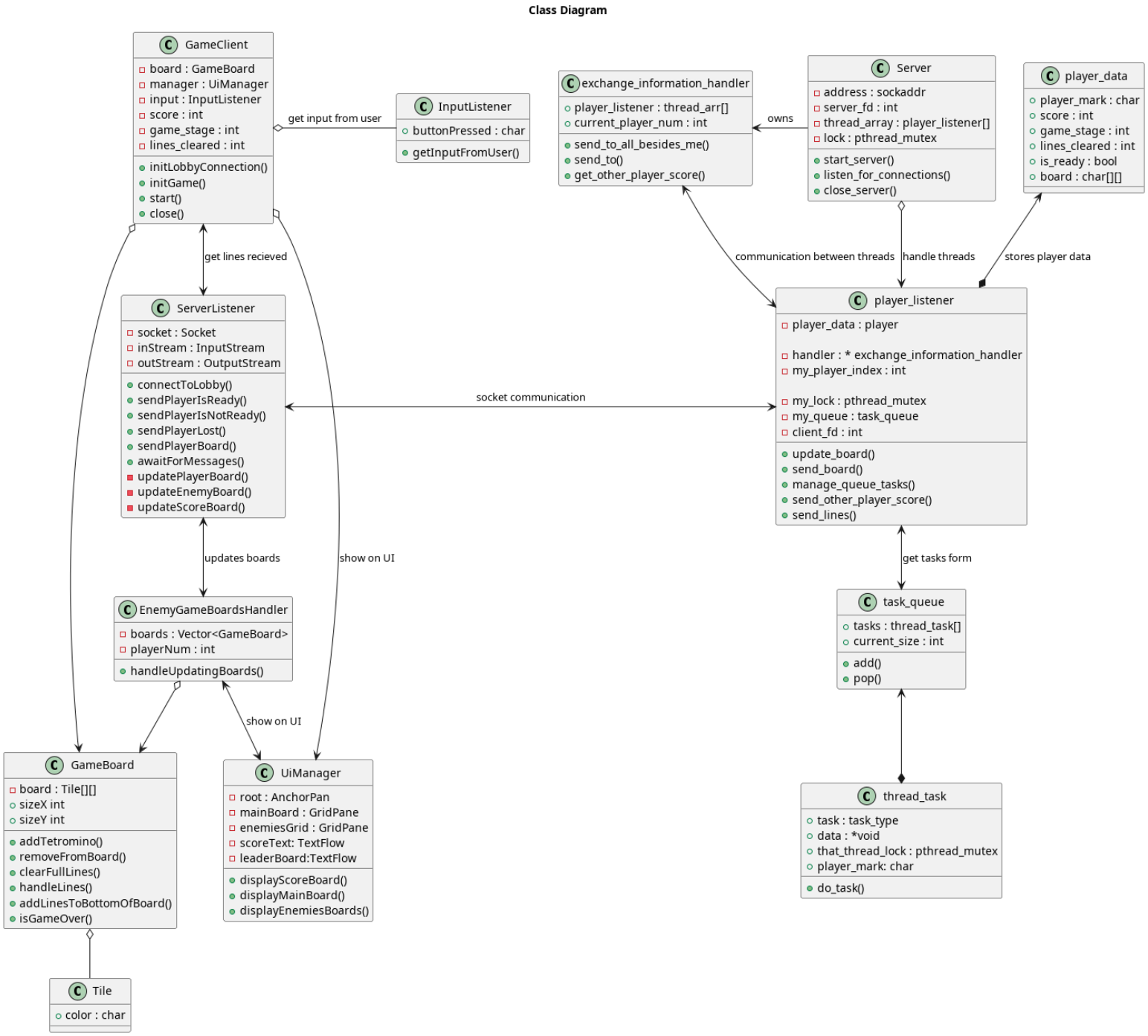


Rysunek 3. Przedstawia zachowanie systemu w pętli gry, gdy nastąpi jakaś zmiana na planszy



Rysunek 4. Przedstawia proces zamykania się programu

5. Diagram klas



Rysunek 5. Przedstawia diagram klas dla naszej gry

6. Opis diagramu klas

1. GameClient – Główny obiekt odpowiedzialny za rozpoczęcie gry po stronie klienta, obsługuje on całą grę.
2. InputListener – Odpowiada za nasłuchanie wciskanych klawiszy i zapisywanie tych znaków.
3. ServerListener – Osobny wątek odpowiadający na obsługę komunikatów z serwera oraz wysyłanie wiadomości GameClienta na serwer.
4. EnemyGameBoardsHandler – Pomocniczy obiekt potrzebny do wypisywania plansz przeciwników.
5. GameBoard – Obiekt odpowiedzialny ze logikę gry *Tetris*: przesuwanie klocków, spadek klocka, czyszczenie linii itd.
6. UiManager – Zajmuje się wyświetlaniem w oknie aktualnego stanu gry.
7. Tile – Obiekt reprezentujący pojedynczą kratkę w planszy.
8. Server - Obiekt będący serwerem naszej gry, odpowiada za utworzenie hosta, oraz gniazdek dla przychodzących klientów.
9. exchange_information_handler – Struktura odpowiedzialna ze przesyłanie zadań pomiędzy wątkami.
10. player_listener – Zajmuje się komunikacją z ServerListenerm, odbiera komunikaty i wysyła komunikaty do konkretnego gracza.
11. player_data – Struktura przechowująca informacje o danym graczu na serwerze.
12. task_queue – Struktura przechowująca tablicę w której znajduje się kolejka zadań do wykonania, osobna dla każdego wątku.
13. thread_task – Zawiera informacje o tym jakie zadanie jest do wykonania, gdzie znajdują się dane, oraz semafor i znak gracza, którego będzie dotyczyć to działanie.

7. Ogólna zasada działania programu

Na początku użytkownik uruchamia program i łączy się z serwerem. Po czym czeka w poczekalni na resztę graczy, aż wszyscy dołączą. Gdy wszyscy gracze dołączą do poczekali i zasygnalizują swoją gotowość, gra zacznie się.
W trakcie trwania gry gracz zmienia położenie klocka za pomocą strzałek, lub zmienia się na skutek samoczynnego jego spadku. Gdy to nastąpi do serwera wysyłany jest sygnał o zmianie statusu planszy naszego gracza. Obiekt ServerListenera jest osobnym wątkiem i działa niezależnie od głównego programu, ma za zadanie wysłać informację do player_listenera. Wątek po stronie serwera odbiera te informację, zapisuje zmiany w odpowiednim miejscu, oraz zawiadamia inne wątki poprzez exchange_information_handler, że powinny one przesłać do swoich graczy zmienioną wersję planszy danego gracza. Analogicznie wygląda sytuacja z przesyłaniem wyników gracza.

Na każdego gracza przypada osobny wątek i są one jednoznacznie związane. Wątek ten posiada wszystkie przesłane od gracza dane i przechowuje je u siebie w strukturze player_data i tylko on powinien je modyfikować. Reszta wątków też ma do nich dostęp, robi to za pośrednictwem obiektu exchange_information_handler. Będzie on wywoływany kiedy jeden wątek wysyła informacje do wszystkich pozostałych, że nastąpiła zmiana, wtedy to on prześle te informacje to odpowiednich kolejek.

Ogólny algorytm działania wątku:

```
while (!koniec_gry) {
        zadanie_klienta = pobierz_zadanie_od_gracza()
        wykonaj_zadanie_u_siebie(zadanie_gracza)
        powiadom_resztę_wątków(zadanie_gracza)
        sprawdź_własną_kolejkę_zadań(zadania_od_wątków)
        wykonaj_zadania(zadania_od_wątków)
}
```

Wątek będzie aktywnie oczekiwać na zadania do wykonania, przez pewien czas będzie oczekiwać na informację od gracza, jeżeli ona nie przyjdzie. Sprawdzi swoją kolejkę zadań i jeżeli jest tam zadanie od innego wątku (np. z wysłaniem do gracza zaktualizowanych danych o wyniku innego gracza) to wykona to zadanie.

Zadaniem głównego wątku serwera jest na początku nasłuchiwanie na graczy próbujących dołączyć do poczekalni. Gdy wszyscy gracze połączą się i gra wystartuje, rolą jego jest sprawdzanie, czy gra się nie zakończyła (gra kończy się jak każdy gracz przegra), wtedy informuje on wszystkie wątki o tym, że gra dobiegła końca, więc powinny one zmienić swój tryb działania. W podstawowej implementacji będzie to sygnał do zakończenia działania programu i odpowiedzialny będzie za zamykanie całego programu po stronie serwera będzie też główny wątek.

8. Sekcje krytyczne w programie

1) GameClient i ServerListener

W GameClientcie znajdować się będzie tablice z informacjami o planszach przeciwników jak i gracza, które muszą zostać wyświetlone co klatkę gry. Dostęp do tych plansz będzie miał też ServerListener, który jak przyjdzie odpowiednia wiadomość uaktualni daną planszę, w trakcie gdy to następuje dostęp do zasobu musi zostać zablokowany do momentu przepisania całej planszy. Podobnie jeżeli GameClient chce przeczytać zawartość planszy, ServerListener powinien poczekać, aż GameClient skończy.

2) player_listener i Server

Player_listener zawiera w sobie informacje o graczu, które samemu aktualizuje oraz udostępnia innym wątkom. Możliwość zapisu i odczytu będzie kontrolowana przez semafor pthread_mutex. Każdy wątek posiada swój i będzie on kontrolował możliwość odczytu danych. Nasz program zawiera parę małych sekcji krytycznych w player_data do których dostęp jest kontrolowany niezależnie między jedną a drugą.

3) task_queue

Player_listener zawiera również prywatną kolejkę zadań przychodzących z innych wątków, ten zasób także powinien zostać objęty semaforem, by żadne zadanie nie zostało niepoprawnie nadpisane.

9. Inne problemy jakie mogą wystąpić

1) Przepełnienie kolejki zadań dla wątków

Rozmiar kolejka do zadań przeznaczona dla wątku będzie ograniczona, i w momencie gdy się ona wypełni reszta zadań będzie odrzucana, by nie powodować zakleszczeń. Nie jest to wielkim problemem z poziomu rozgrywki, ponieważ nie licząc informacji o wysyłaniu przeciwnikowi linii, informacje nie są niezbędne do rozgrywki, pełnią tylko rolę informacyjną.

2) Odłączenie się gracza w trakcie trwania rozgrywki

Jeżeli nastąpi problem z połączeniem między graczem a serwerem, wątek poinformuje o tym inne wątki graczy oraz główny wątek. Nie powinno mieć to w pływu na przebieg rozgrywki.