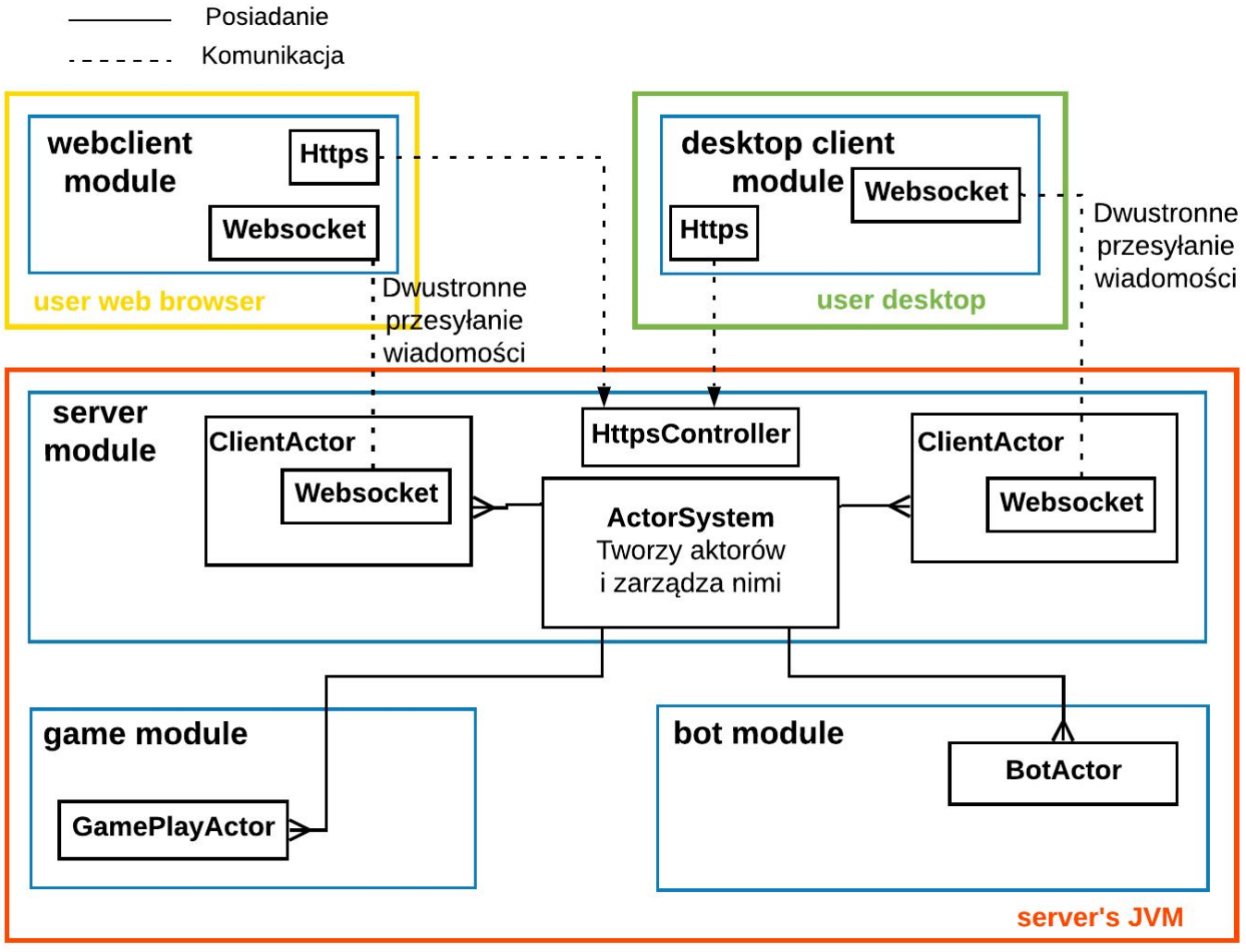
Projekt: Trach Game
Data modyfikacji: 08.04.2019 r.

Zespół:
Mateusz Kobak,
Krzysztof Piesiewicz,
Tomasz Grześkiewicz,
Karolina Gabara

Specyfikacja techniczna

Podział projektu

1. game module (logika gry)
2. server module (serwer)
3. webclient module (klient przeglądarkowy)
4. desktop client module (klient desktopowy)
5. bot module (biblioteka do botów)



Projekt: Trach Game
Data modyfikacji: 08.04.2019 r.

Moduł game

Moduł odpowiada za modelowanie elementów gry oraz przeprowadzanie rozgrywek.

Moduł napisany jest w języku *Scala* z wykorzystaniem frameworka *Akka* (w wersji ≥ 2.5). Interfejs modułu oparty jest na modelu aktorów (*Akka → Actor*).

Każda rozgrywka jest osobnym serwerem zaimplementowanym w postaci aktora, udostępnionym jako JVM-owa klasa `GamePlayActor`. Aktor odbiera i wysyła wiadomości w formacie JSON. Aktor udostępnia usługę serializacji stanu gry do formatu JSON.

Moduł server

Moduł odpowiada za serwer systemu, z którym użytkownicy łączą się za pomocą aplikacji klienta WWW lub klienta desktopowego. Serwer obsługuje zapytania klientów związane z rejestracją i uwierzytelnianiem użytkowników przez protokół HTTPS. Natomiast uwierzytelnionych użytkowników obsługuje przez websockety za pomocą protokołu WSS. Serwer posiada bazę danych, w której przechowuje informacje o użytkownikach, prowadzonych rozgrywkach wraz z serializowanymi stanami gry.

Moduł napisany jest w języku *Scala* przy użyciu *PlayFramework* (w wersji ≥ 2.6) oraz *Akka* (w wersji ≥ 2.5).

Serwer uruchamia rozgrywki (aktorów `GamePlayActor`) w swoim systemie aktorów (*Akka → ActorSystem*) na tej samej JVM, na której jest uruchomiony.

Serwer uruchamia boty (aktorów `BotActor`) w swoim systemie aktorów na tej samej JVM, na której jest uruchomiony.

Moduł webclient

Moduł odpowiada za aplikację WWW klienta na przeglądarki: Firefox, Google Chrome, które obsługują JavaScript w wersji 6. Klient umożliwia rejestrację i uwierzytelnienie użytkownika przez protokół HTTPS. Po pomyślnym uwierzytelnieniu łączy się z serwerem i komunikuje przez websocket za pomocą protokołu WSS.

Moduł desktop client

Moduł odpowiada za aplikację klienta desktopowego na platformę Linux ze środowiskiem graficznym obsługującym Qt5. Klient umożliwia rejestrację i uwierzytelnienie użytkownika przez protokół HTTPS. Po pomyślnym uwierzytelnieniu łączy się z serwerem i komunikuje przez websocket za pomocą protokołu WSS.

Moduł bot

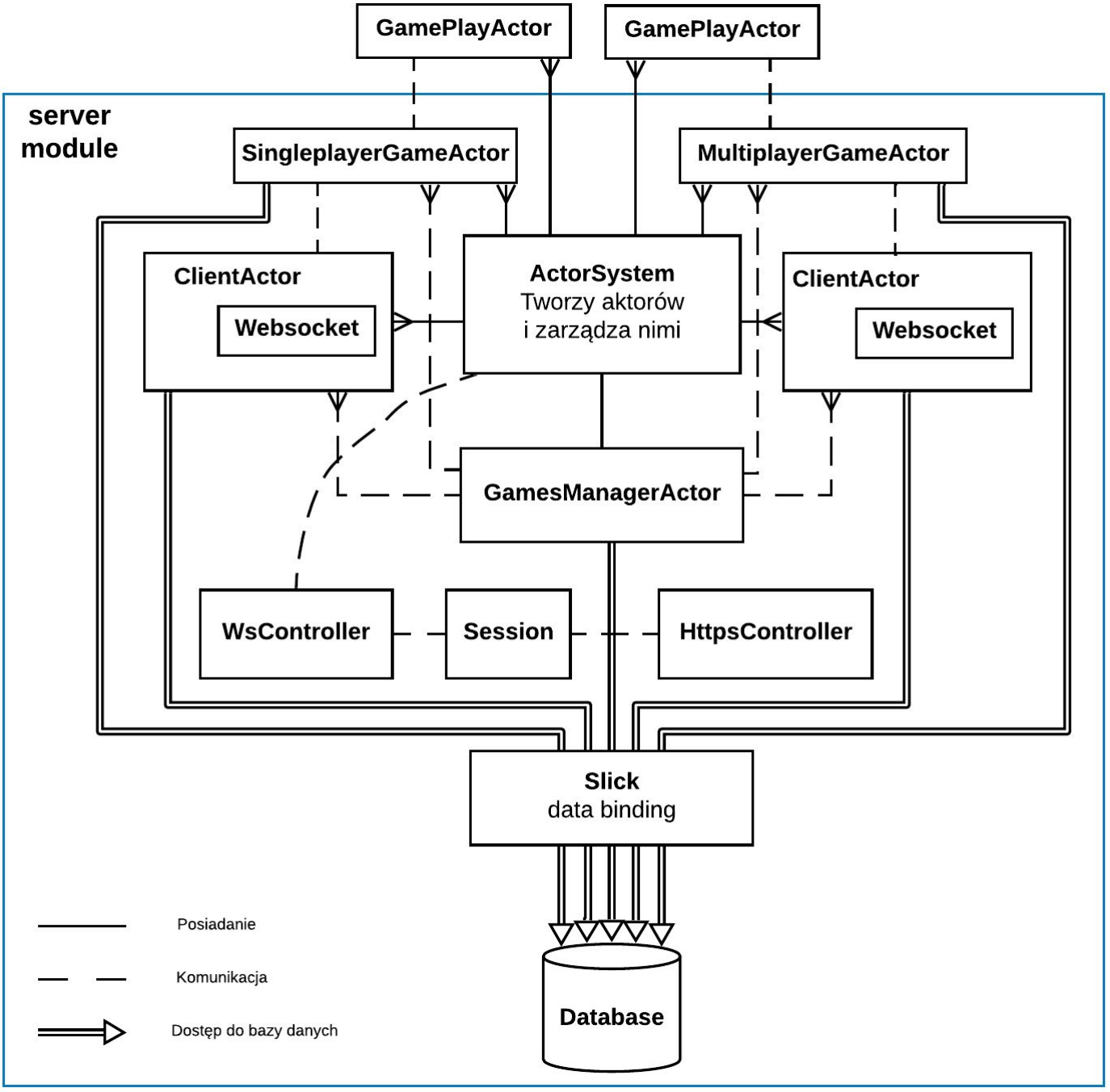
Bot jest aktorem udostępnionym jako JVM-owa klasa `BotActor`. Symuluje akcje gracza, używając swojej strategii. Aktor odbiera i wysyła wiadomości w formacie JSON.

Modele i komunikacja

Modele gry i systemu oraz komunikacja są opisane w dokumencie „Modele i komunikacja”.

Projekt: Trach Game
Data modyfikacji: 08.04.2019 r.

Moduł server



ActorSystem

System aktorów *Akka* ActorSystem zarządza pulą wątków serwera. Tworzy obiekty aktorów, zarządza nimi, synchronizuje ich pracę, przydziela dostęp do zasobów.

ClientActor

Każdy połączony klient ma przyporządkowanego aktora `ClientActor`, który:

- obsługuje komunikację z klientem przez websocket,
- uwierzytelnia w systemie użytkownika korzystającego z klienta,

- pozwala na przeglądanie danych użytkownika i ich modyfikację,
- pośredniczy w komunikacji klienta z innymi elementami systemu; np.:
  - umożliwia użytkownikowi tworzenie rozgrywek jedno- i wieloosobowych (zleca aktorowi `GamesManagerActor`),
  - umożliwia użytkownikowi dołączenie do rozgrywki wieloosobowej (zleca aktorowi `GamesManagerActor`),
  - umożliwia użytkownikowi wznowienie jednej ze wstrzymanych rozgrywek jednoosobowych (zleca aktorowi `GamesManagerActor`),
  - przekazuje do rozgrywki otrzymane od klienta polecenia sterowania graczem (przekazuje aktorowi `SingleplayerGameActor` lub `MultiplayerGameActor`),
  - przekazuje klientowi komunikaty z rozgrywki (od aktora `SingleplayerGameActor` lub `MultiplayerGameActor`),
  - informuje rozgrywkę o utracie połączenia z klientem lub o jego wznowieniu (powiadamia aktora `SingleplayerGameActor` lub `MultiplayerGameActor`)

HttpsController

Kontroler *PlayFramework* tworzony przy starcie serwera. Obsługuje zapytania https. Odpowiada za rejestrację i uwierzytelnianie użytkowników oraz obsługę sesji.

WsController

Kontroler *PlayFramework* tworzony przy starcie serwera. Przyjmuje przychodzące przez protokół WSS połączenia klientów. Kontroler weryfikuje czy połączenie pochodzi od uwierzytelnionego użytkownika, sprawdzając ciasteczka żądania i sesję. Dla każdego nowego, zweryfikowanego połączenia zleca systemowi aktorów utworzenie aktora `ClientActor`, który ma odpowiadać za komunikację z klientem.

SingleplayerGameActor

Aktor `SingleplayerGameActor` zarządza rozgrywką jednoosobową.

- Pośredniczy w komunikacji między aktorami: `ClientActor` i `GamePlayActor`.
- Inicjuje utworzenie i uruchomienie aktora `GamePlayActor` z modułu game (opcjonalnie, przekazując mu zapisany stan gry).
- Przechowuje informacje o rozgrywce jednoosobowej użytkownika (o stanie gry) i zapisuje do bazy danych.
- Zarządza botami i pośredniczy w komunikacji między nimi a aktorem `GamePlayActor`.

MultiplayerGameActor

Aktor `MultiplayerGameActor` zarządza rozgrywką wieloosobową.

- Pośredniczy w komunikacji między aktorami: `ClientActor` i `GamePlayActor`.
- Inicjuje utworzenie i uruchomienie aktora `GamePlayActor` z modułu game (opcjonalnie, przekazując mu zapisany stan gry).
- Przechowuje informacje o rozgrywce wieloosobowej (stan gry, uczestniczący użytkownicy wraz odpowiadającymi graczami) i zapisuje do bazy danych.
- Zarządza botami i pośredniczy w komunikacji między nimi a aktorem `GamePlayActor`.

GamesManagerActor

Aktor `GamesManagerActor` jest uruchamiany przy starcie serwera. Odpowiada za tworzenie i uruchamianie aktorów `SingleplayerGameActor` i `MultiplayerGameActor`.

- Tworzy (wznawia) ww. aktorów na podstawie informacji przechowywanych w bazie danych (na polecenie aktora `ClientActor`).
- Tworzy ww. aktorów, zapisując informacje w bazie danych (na polecenie aktora `ClientActor`).
- Znajduje użytkownikowi rozgrywkę wieloosobową, do której może dołączyć (na polecenie aktora `ClientActor`).

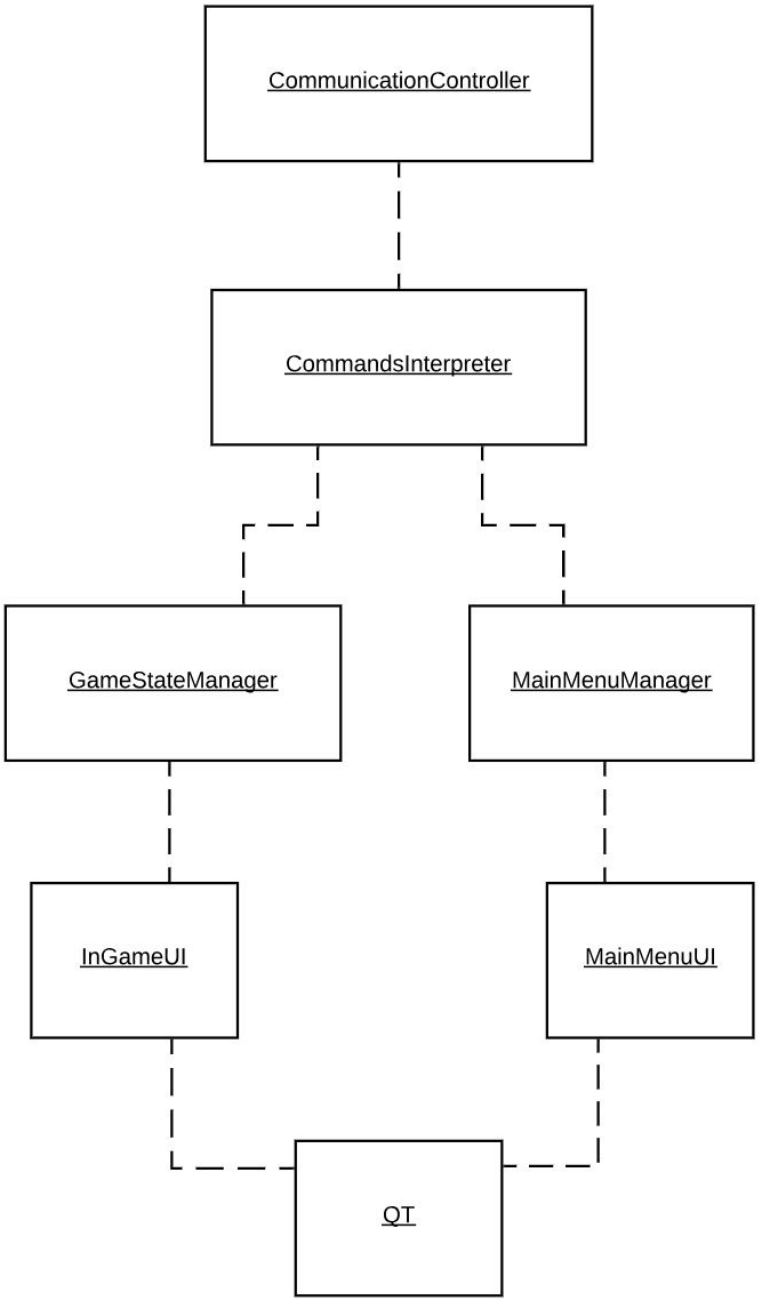
Informacje zapisane w bazie danych służą do uruchomienia rozgrywek po awarii serwera (gdy serwer zostanie ponownie uruchomiony) lub do uruchomienia rozgrywek wstrzymanych przez użytkowników.

Database

Serwer posiada relacyjną bazę danych *SQLite* (w wersji ≥ 3.0). Baza danych jest obsługiwana za pomocą wiązania *Slick* (w wersji ≥ 3.2.3).

Projekt: Trach Game
Data modyfikacji: 08.04.2019 r.

Moduł desktop client



CommunicationController

Zarządza połączeniem z serwerem i przesyłaniem danych otrzymanych z `CommandsInterperter` przez websocket oraz z serwera do `CommandsInterpreter`.

CommandsInterpreter

Parsuje do JSON'a dane otrzymane z managerów oraz wysyła do managerów informacje o otrzymanych wydarzeniach.

Projekt: Trach Game
Data modyfikacji: 08.04.2019 r.

GameStateManager

Dba o to, żeby UI właściwej gry było aktualne, aktualizując je na podstawie otrzymanych danych. odbiera sygnały z `InGameUI`, reagując na akcje gracza.

MainMenuManager

Zajmuje się tworzeniem UI Menu głównego oraz odbiera sygnały z `MainMenuUI` będące skutkami interakcji gracza z UI.

InGameUI

Tworzy `QWidgety` zlecone przez `GameStateManager`, takie jak drzewo kart, awatary graczy, rękę gracza itp., używa Qt do tworzenia interface'u.

MainMenuUI

Udostępnia funkcjonalność tworzenia menu używając do tego frameworku qt oraz odbiera sygnały z `Qt` przekazując odpowiednie informacje `GameStateManagerowi`.