

# Checkers

ISTE.442.01. - Web Application Development

Roman Pusec

## Abstract

This manual describes the entire internal structure and the purpose of my final project from Web Application Development. We were assigned to develop a turn-based multiplayer game where at least two players can interact with one another. My version of the game was built with PHP working on the backend (handling user validations, and sending game states to the client) and JavaScript (plus external libraries) on the frontend.

In terms of technology, meekrodb was used for handling all sorts of database calls, and the rest architecture was built with pure PHP. As for the frontend, I've used the CreateJS, which offers abstractions for the visual aspect for the canvas, already preprogrammed animation algorithms, and other things that were not required for this project (e.g. sound, preloading assets).

The software architecture (on the backend) is based upon a slightly amended version of the MVC paradigm (includes Views, Controllers, Business classes, and database utility classes). The architecture of the frontend is diverged into Game Components, AJAX calls, and Business classes.

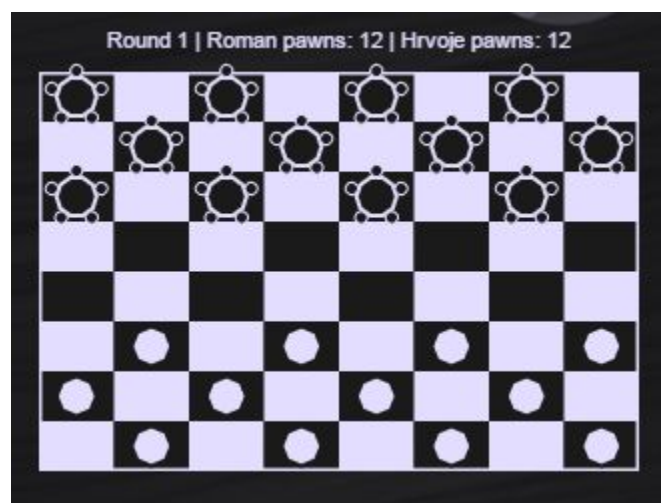
All of these features will be discussed in detail in this document.

## Game Info

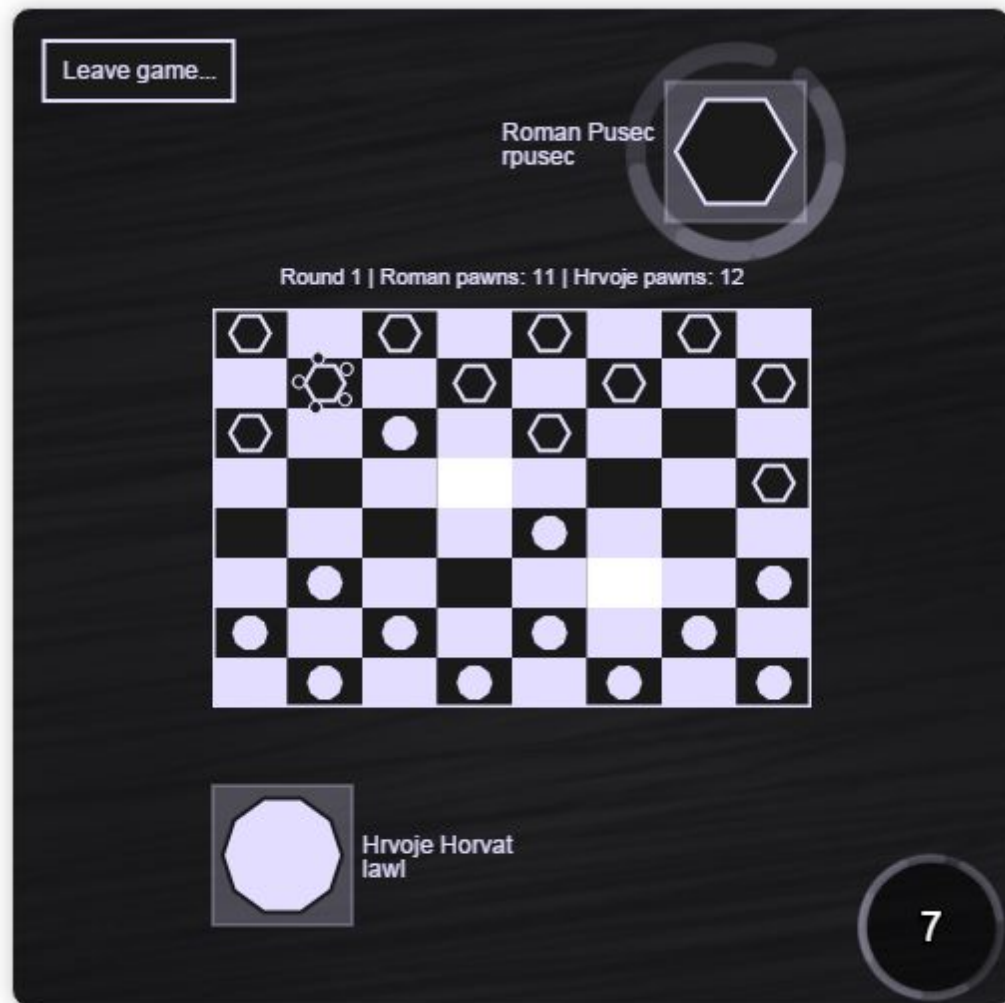
The game is called Checkers. It is a two player turn-based game and its goal is to be first to eliminate all of the pawns of your opponent.

## Gameplay

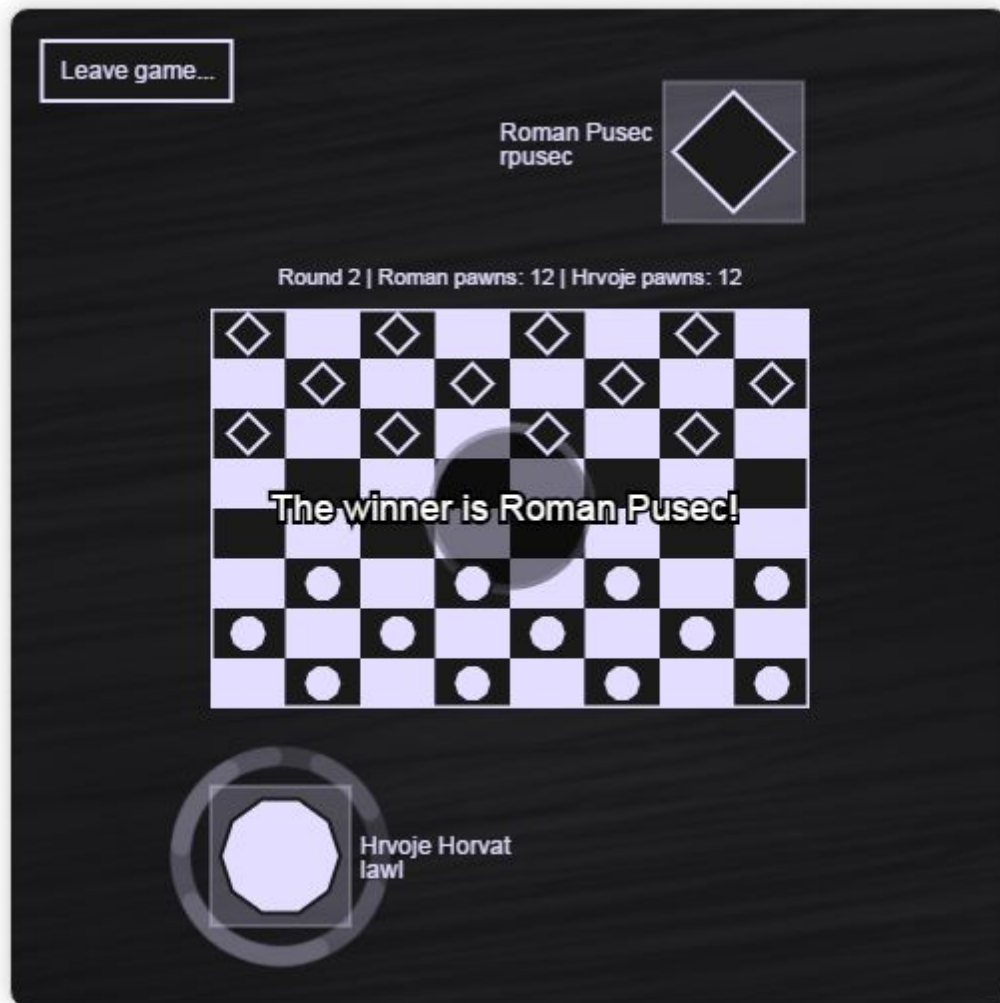
The pawns can be moved diagonally only, and the pawn cannot be placed on a black square. To eliminate an opponent pawn (or multiple opponent pawns), a player pawn has to be positioned right beside their opponent's pawn, then the player can move their pawn over their opponent's pawn, which would then eliminate the opponent's pawn. The pawns can be killed off recursively if multiple opponent pawns are positioned diagonally, with one square of space between them, the player can then eliminate multiple opponent pawns with a single turn - by placing their pawn after the last opponent pawn.



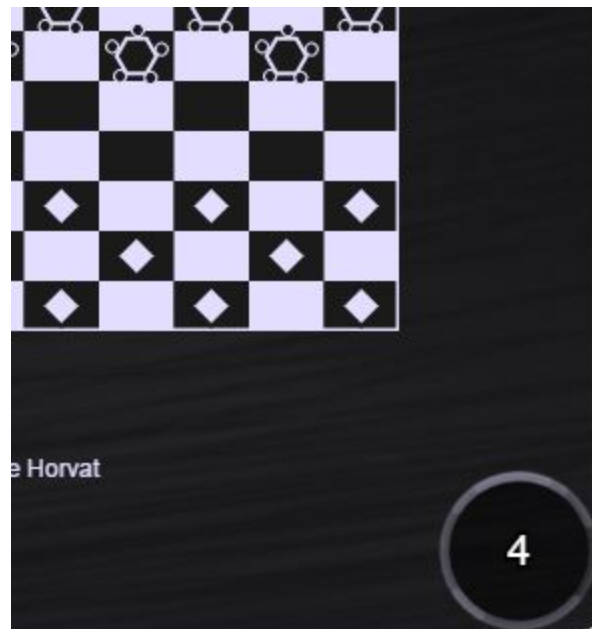
Both player's and opponent's pawns are different in shape and color. And the pawns that the player can choose are highlighted (tiny circles appear around them).



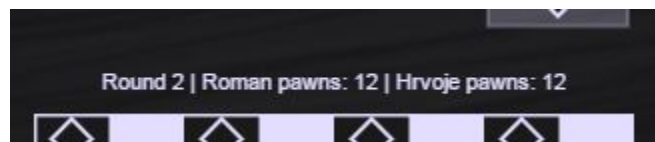
The screenshot shows a portion of the gameplay. The white squares indicate where the player can move their pawns. The diamond shaped pawns (the first twelve from above) belong to the user **rpusec**, and that player chose the pawn that's located on the second column and second row. Since there's a gap between two of the **lawl**'s pawns, **rpusec** can eliminate two of them by clicking on the second white square (the one that's located at the sixth column and sixth row). Lastly, since it is **rpusec**'s turn, their avatar is highlighted with transparent lines that circle around.



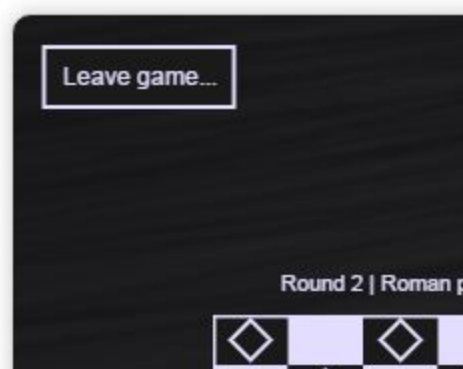
The winner of a round is displayed on the screen and the pawns are reassigned to both players.



Each player turn lasts for 10 seconds, in case the player was immobile for more than 10 seconds, the turn is then passed to the opponent. There's a small timer appearing on the bottom right corner which displays the time.



Right above the board, you can see the amount of your pawns, your opponent's pawns, and which round it currently is.



On the top-right corner is a button which is quite self-explanatory, the player can leave the current game by clicking it.



Once you enter a game room (discussed below), you'll be seeing a screen which displays "Waiting for second player...".

## Main Menu

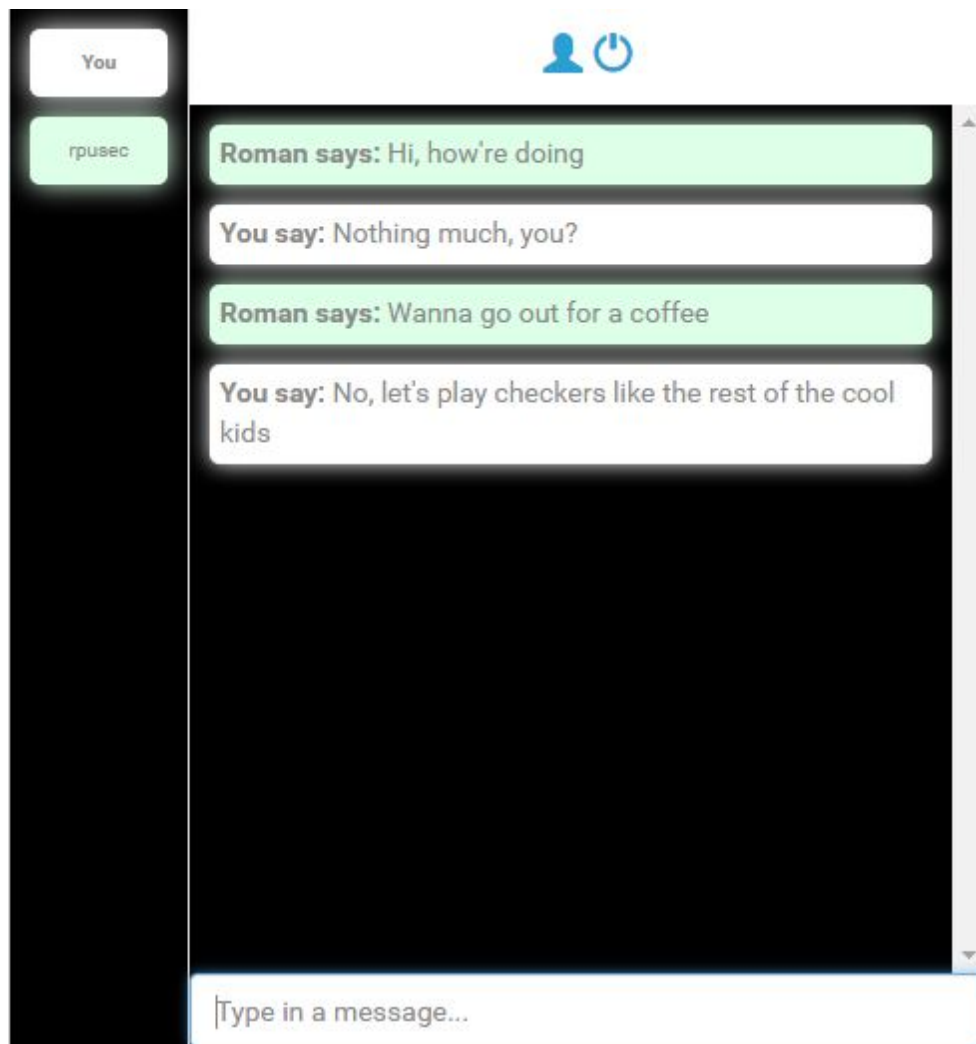


The main menu is composed of six game rooms. A game room represents a separate place where two players can play the game. Since there are 6 game rooms, a maximum of 12 players can use the application.

Users can also see if other users are waiting for their opponent from the main menu, as in the example above, the user **rpusec** is waiting for their opponent.

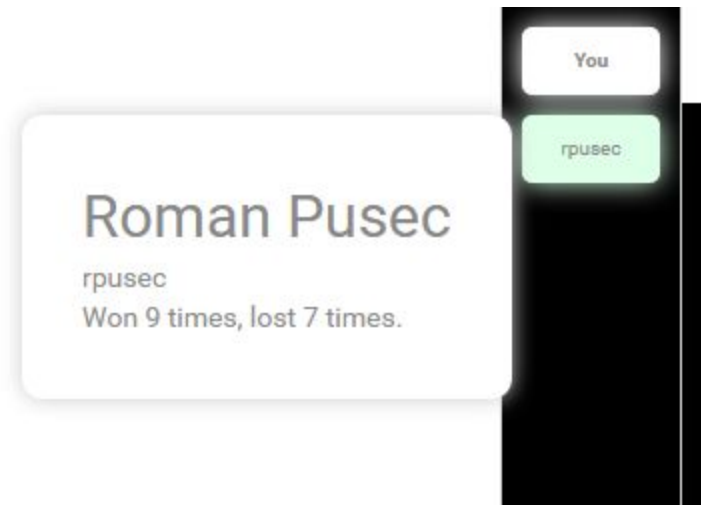


## Chat Menu



The chat menu is pretty self-explanatory - it is composed of the list of users on the left side, the messages on the right side, and the message input field below.

The first icon (person) redirects you to the account settings, and the second icon logs you off the game. More on these options later.

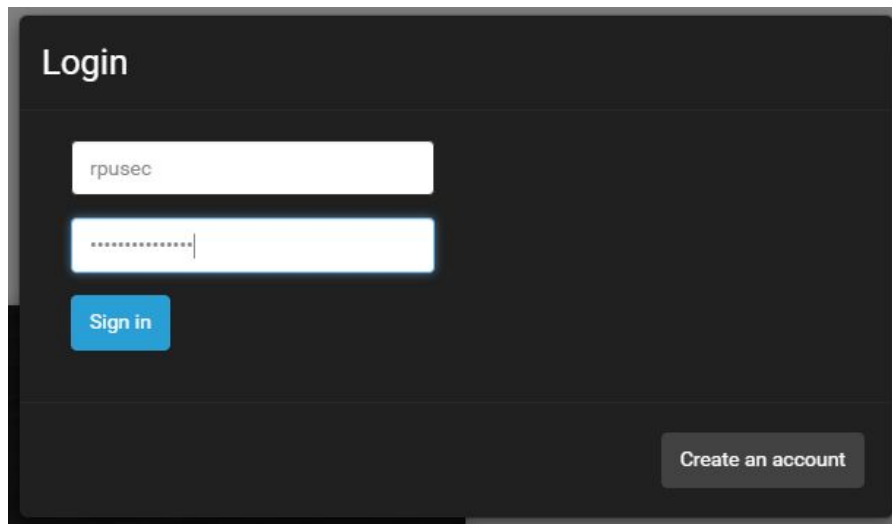


If you hover over a particular user on the chat list, you'd see more of their information, such as their first and last name, username, the amount of times they have won and lost.

The chat in the main menu is public, so everyone can chat with everybody. However during gameplay, the chat between the player and their opponent is private, so it's only between them.

You should also keep in mind that the color of your game interface matches your chat bubbles. Basically, if your canvas interface is red, then your chat bubbles will also be red. That color is assigned upon login.

## Login Screen



Login

rpusec

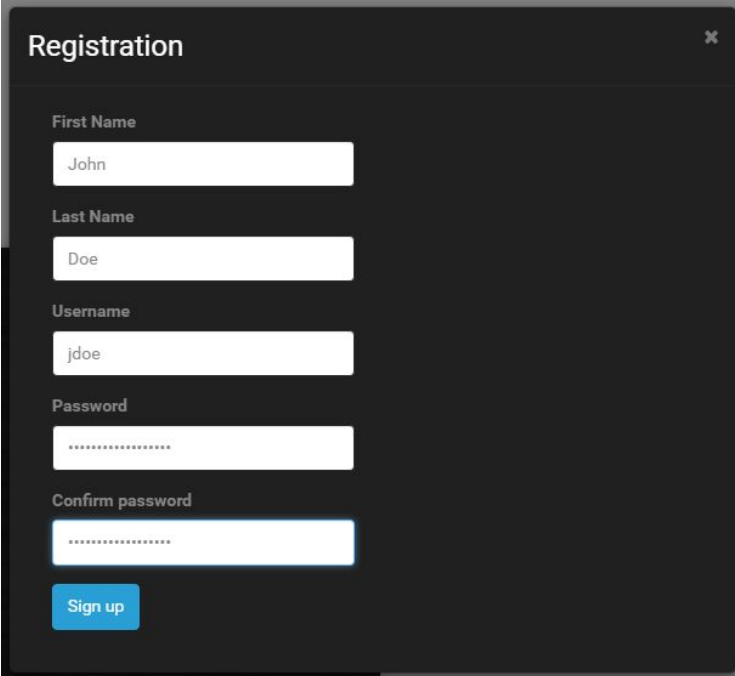
.....

Sign in

Create an account

This is a screenshot of the login screen. The user can insert their credentials and sign in or signup by using the "*Create an account*" option.

## Registration Screen

A screenshot of a registration form titled "Registration" with a close button (X) in the top right corner. The form contains five input fields: "First Name" with the value "John", "Last Name" with the value "Doe", "Username" with the value "jdoe", "Password" with masked characters "\*\*\*\*\*", and "Confirm password" with masked characters "\*\*\*\*\*". A blue "Sign up" button is located at the bottom left of the form.

Registration

First Name

John

Last Name

Doe

Username

jdoe

Password

\*\*\*\*\*

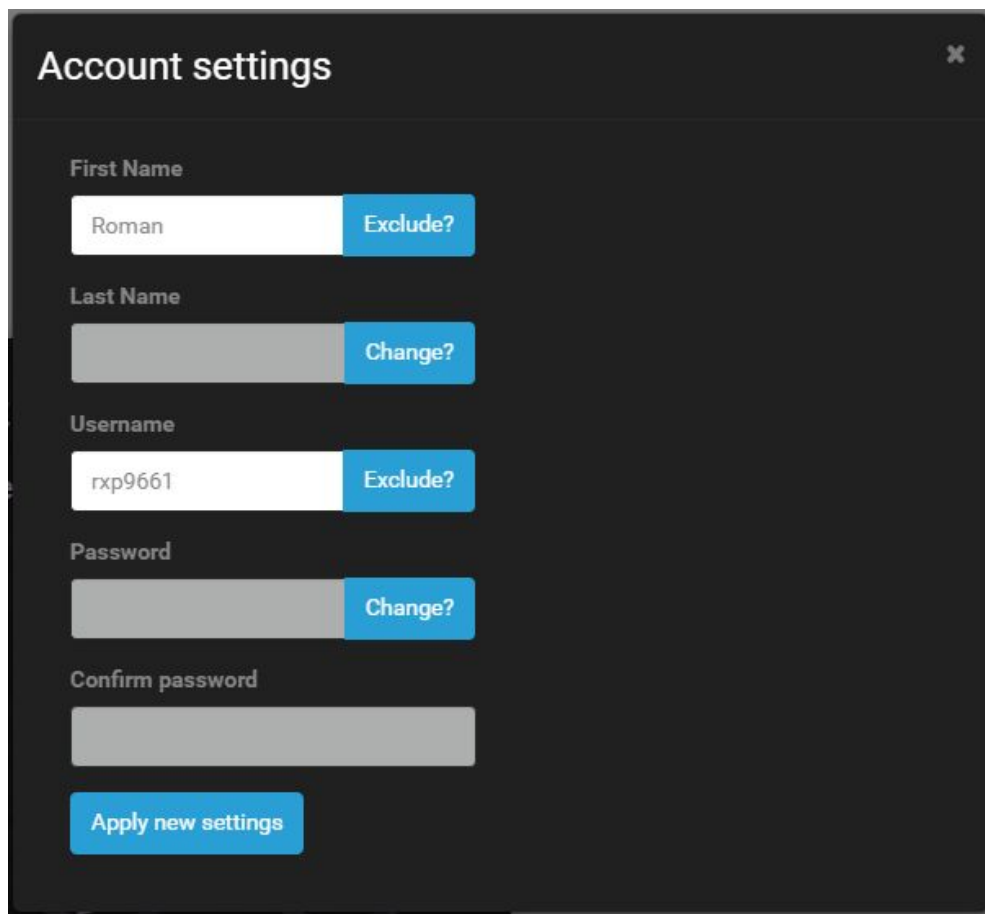
Confirm password

\*\*\*\*\*

Sign up

This is what the user registration screen looks like (it is displayed upon clicking on the "*Create an account*" option on the login frame).

## User account settings



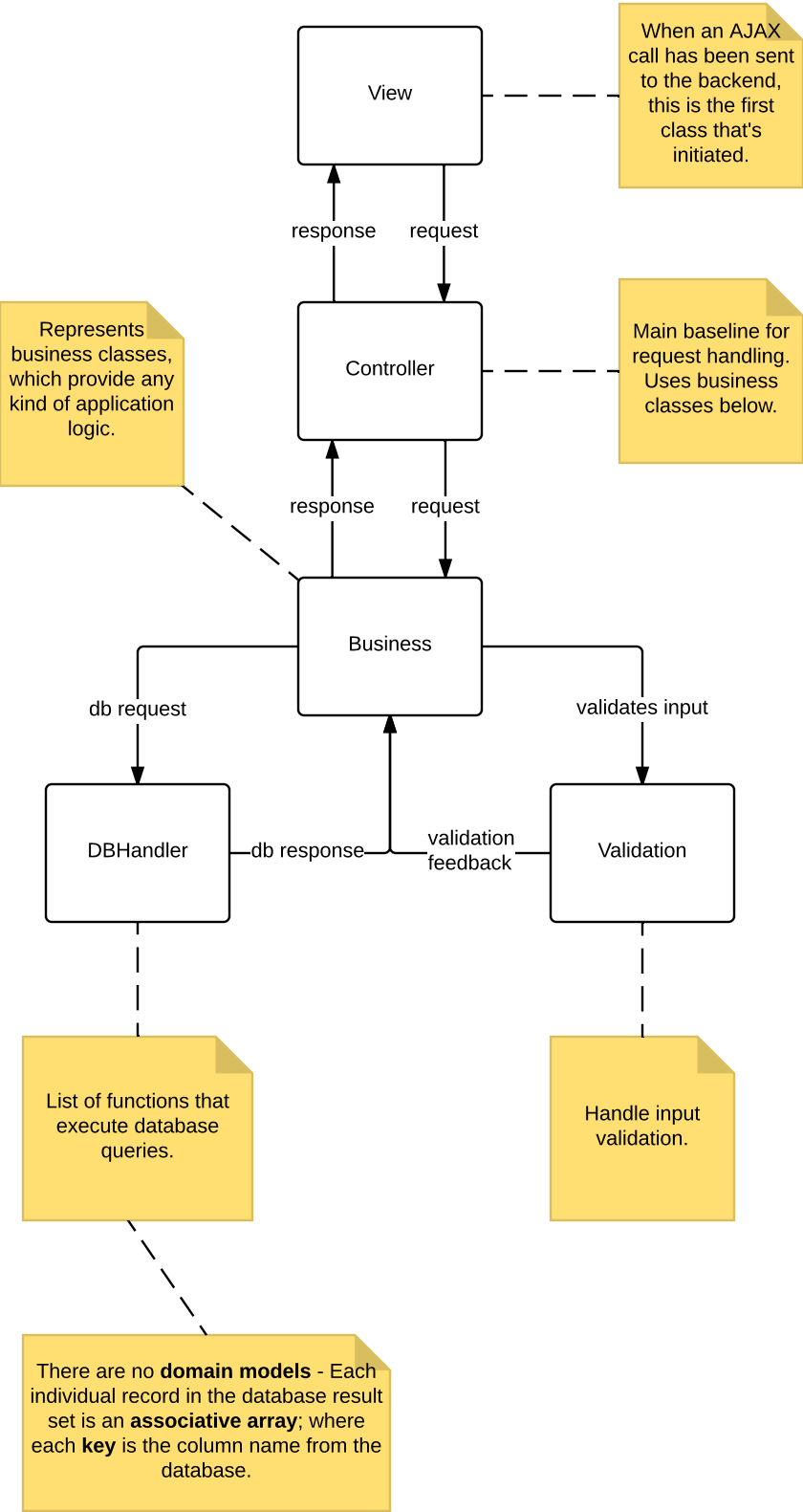
The image shows a dark-themed dialog box titled "Account settings" with a close button (X) in the top right corner. It contains five input fields, each with a label and a corresponding action button on the right:

- First Name:** The input field contains the text "Roman". The action button is labeled "Exclude?".
- Last Name:** The input field is empty. The action button is labeled "Change?".
- Username:** The input field contains the text "rxp9661". The action button is labeled "Exclude?".
- Password:** The input field is empty. The action button is labeled "Change?".
- Confirm password:** The input field is empty.

At the bottom of the dialog box is a blue button labeled "Apply new settings".

Changes the user settings. By pressing "*Change?*" on the right of an input field enables you to change the value of that field in the database, and only the fields that were changed will be updated. In case you change your mind and don't want to update a particular field that you were already amending, then simply press the "*Exclude?*" button on the right side of that particular input field. This window can be accessed by clicking on the blue **person icon** on the chat window.

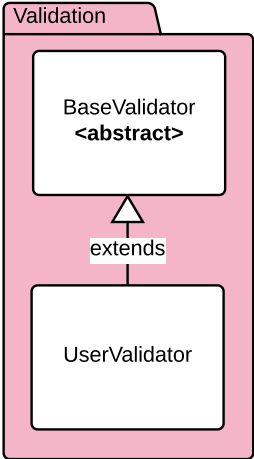
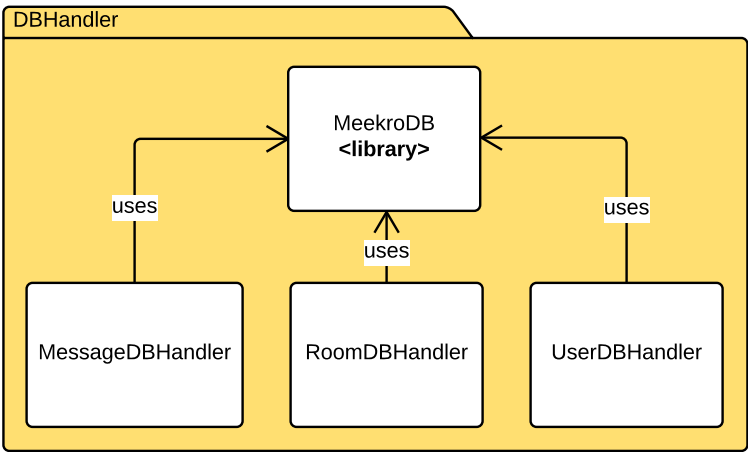
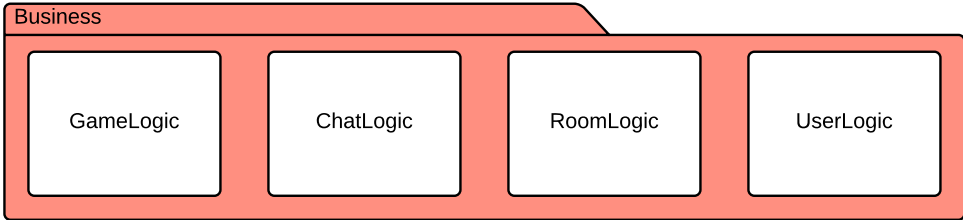
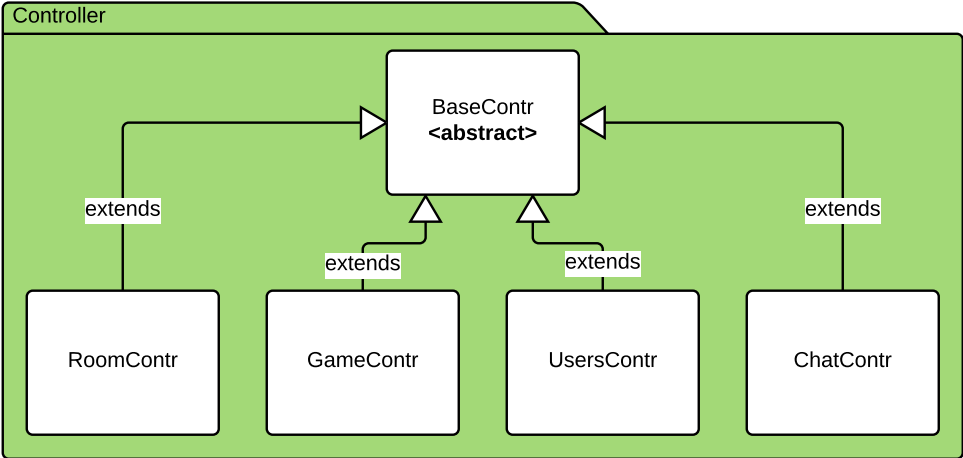
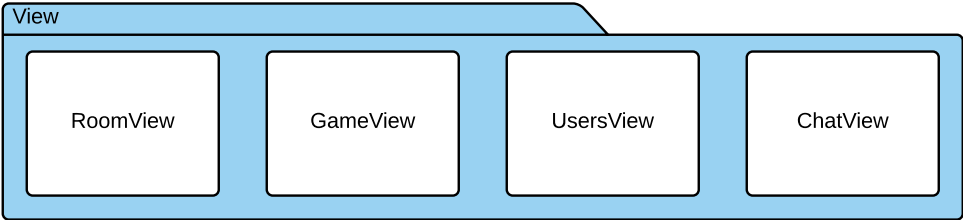
# Backend - Class Diagram (High level)



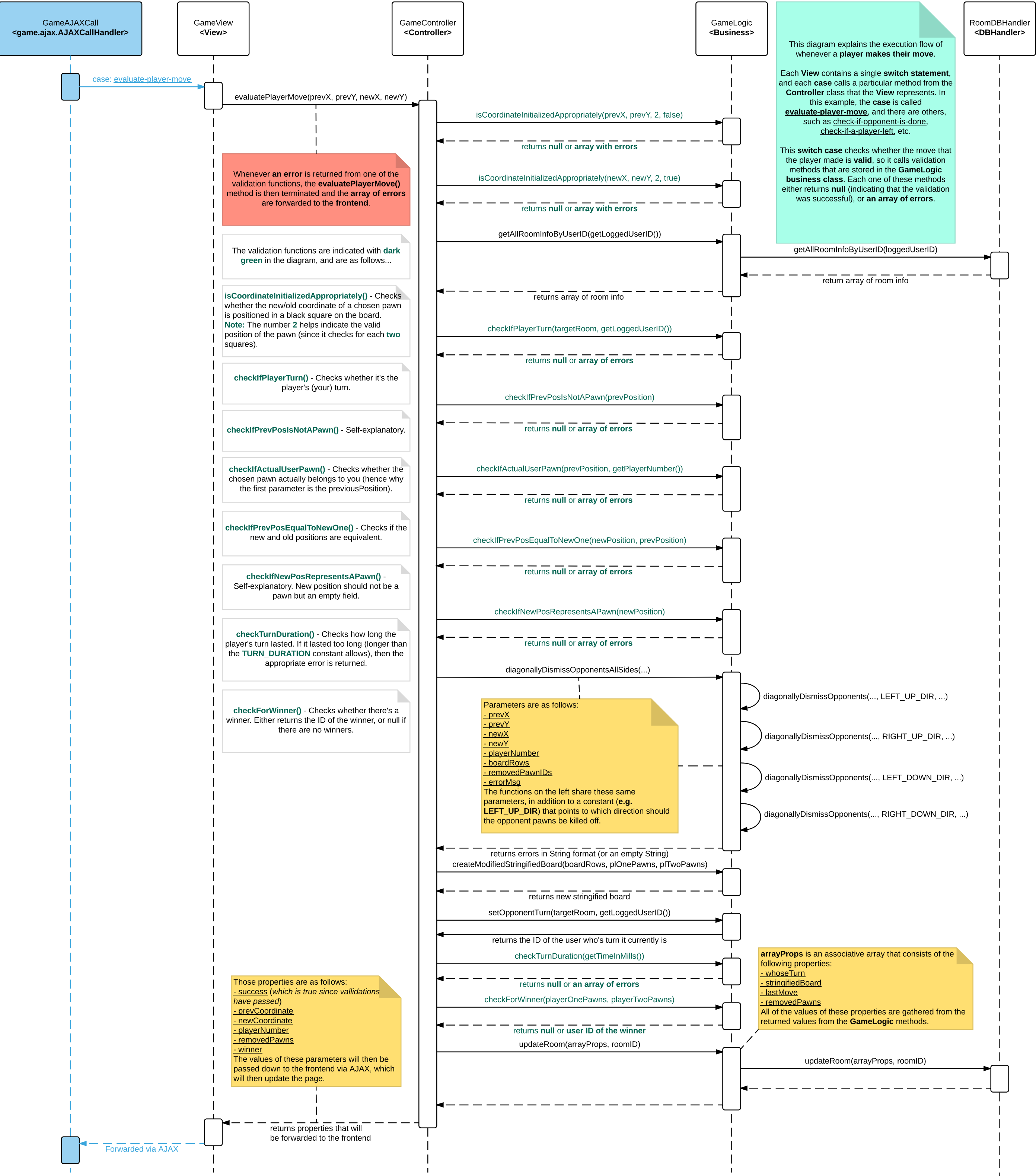
# Backend - Class Diagram

## (Low level)

These **folders** represent **packages** (in the sense of project structure)

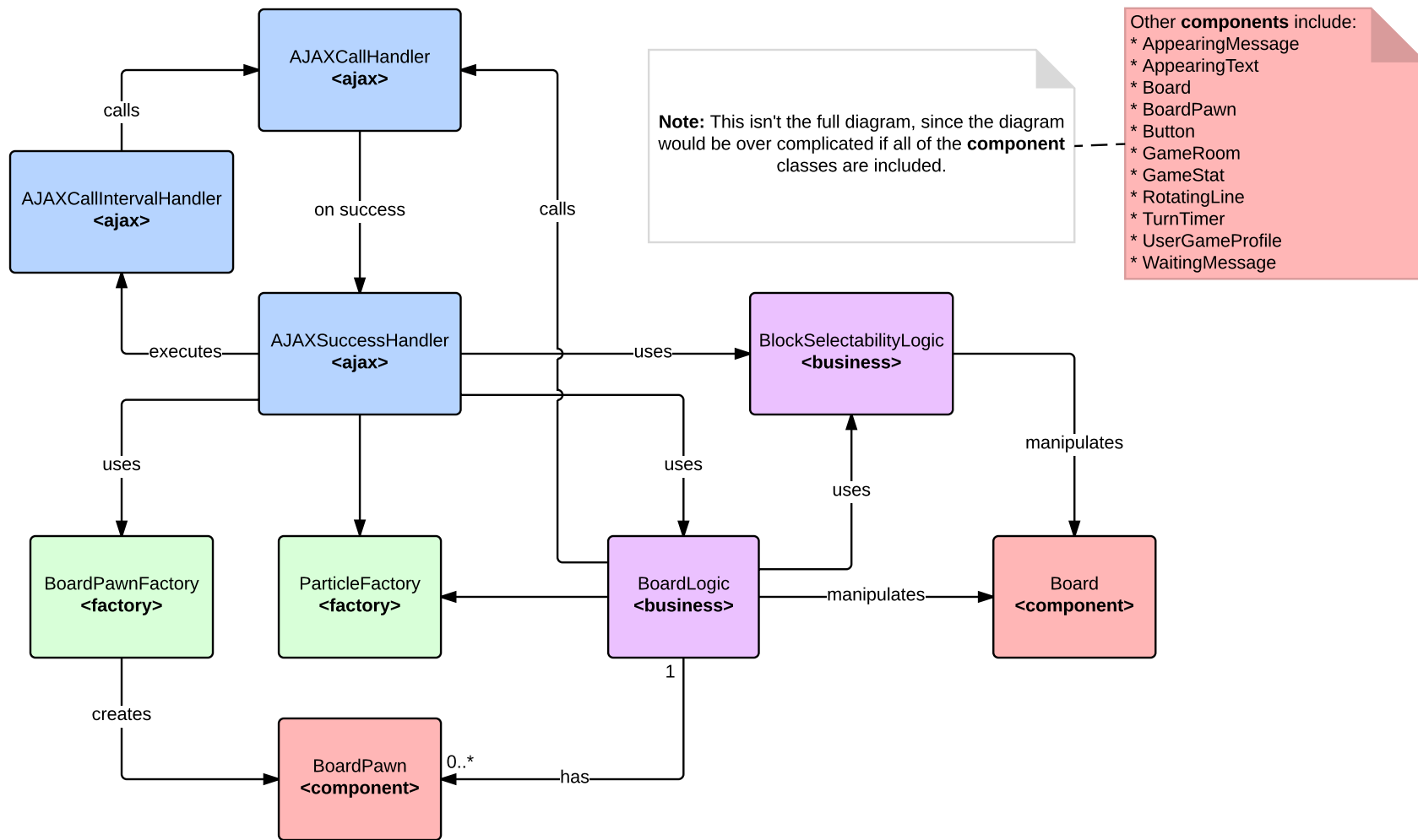


Backend - Sequence Diagram (validating player's move)





# Class Diagram (Game) - High Level



The following high level diagram displays the **game portion** of the application (there are two other portions which manage **chat** and **user** operations, they won't be covered, but they work in a similar principle (e.g. chat and user also have the AJAX classes as provided in this class diagram)).

**AJAXCallHandler** - Class which is responsible for sending out an AJAX request to the backend.

**AJAXSuccessHandler** - Class which contains a stream of methods that are executed when the AJAX request that they represent was handled successfully.

**AJAXCallIntervalHandler** - Makes certain AJAXCallHandler methods execute themselves after a certain amount of milliseconds (by using setInterval function). As an example, this is useful for repeatedly checking if your opponent made their move.

**BoardPawnFactory** - Creates BoardPawn objects.

**ParticleFactory** - Creates particles (when placing a pawn).

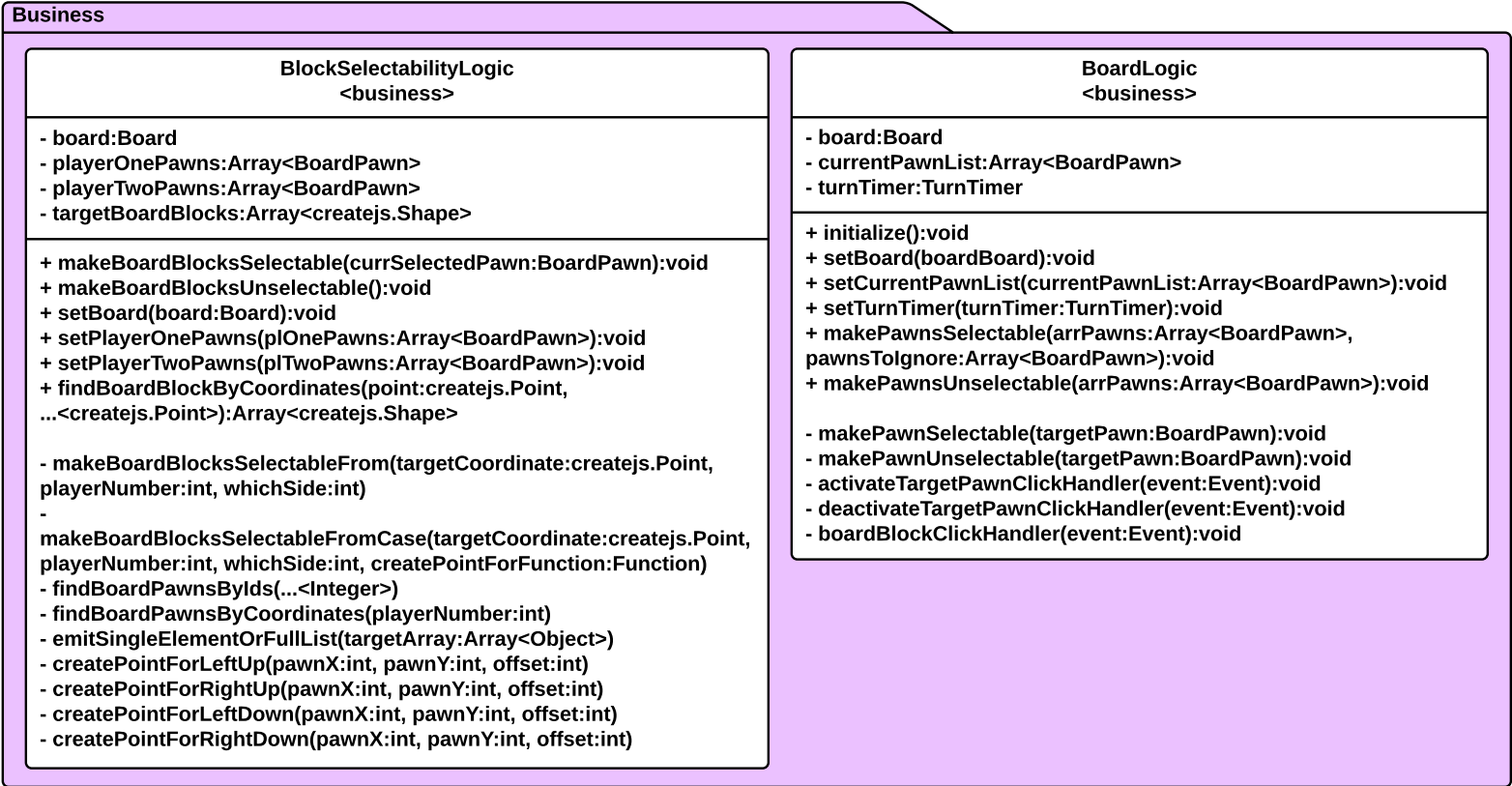
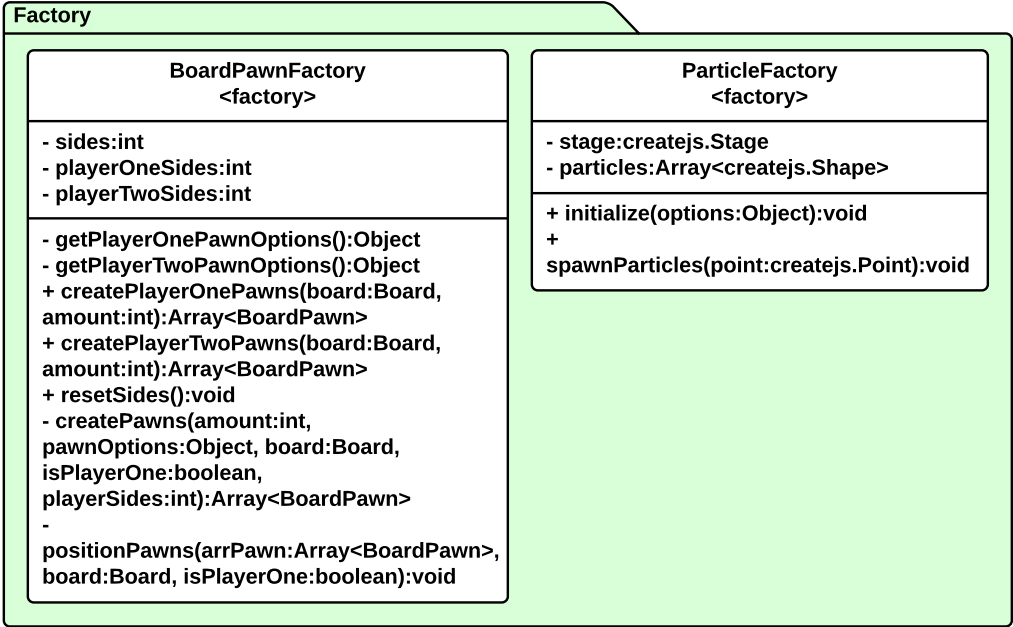
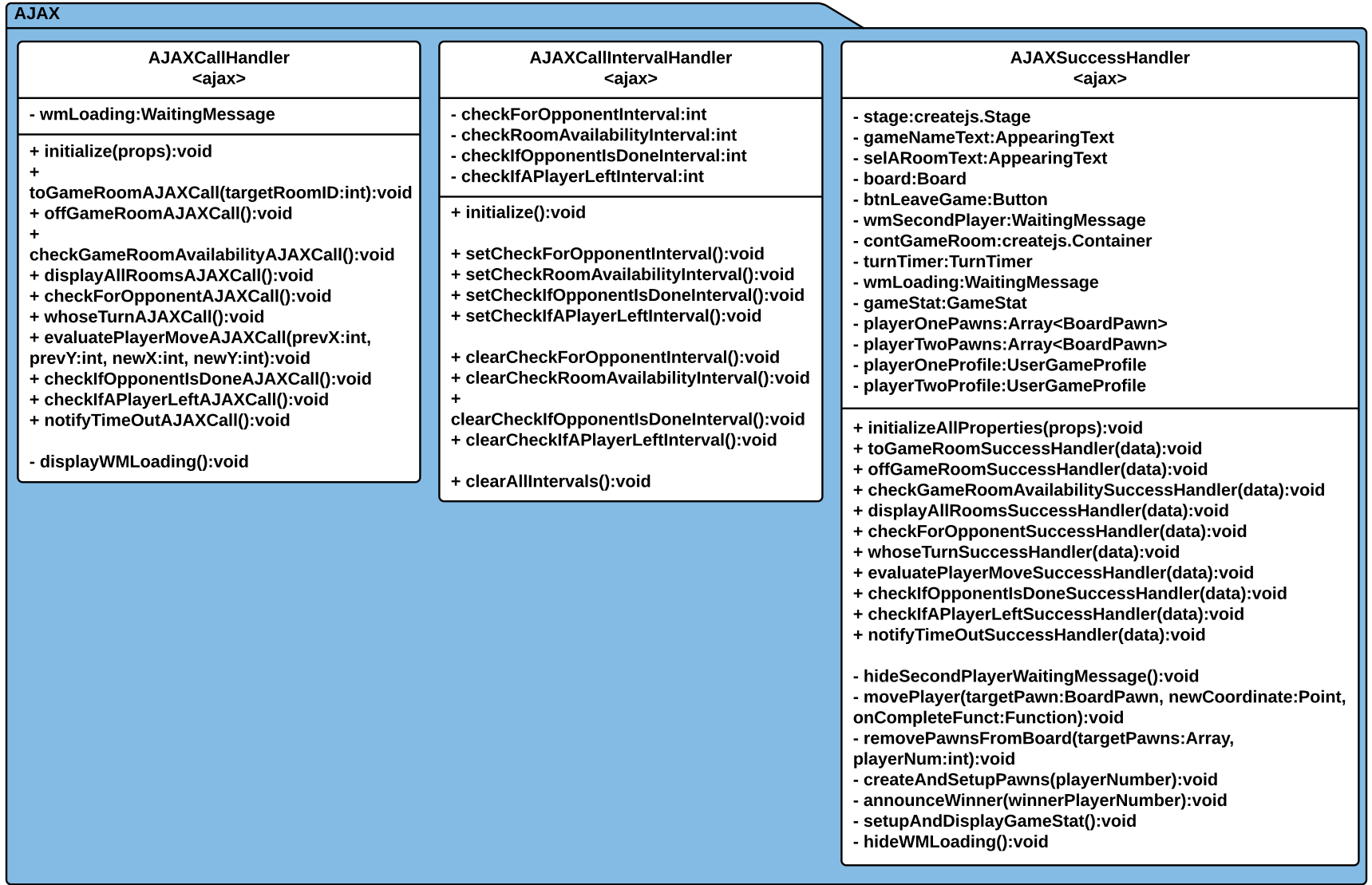
**BoardLogic** - Handles all of the logic regarding placing a pawn.

**BlockSelectabilityLogic** - Handles all of the logic regarding making board fields selectable or unselectable (in the sense that you can place a pawn there).

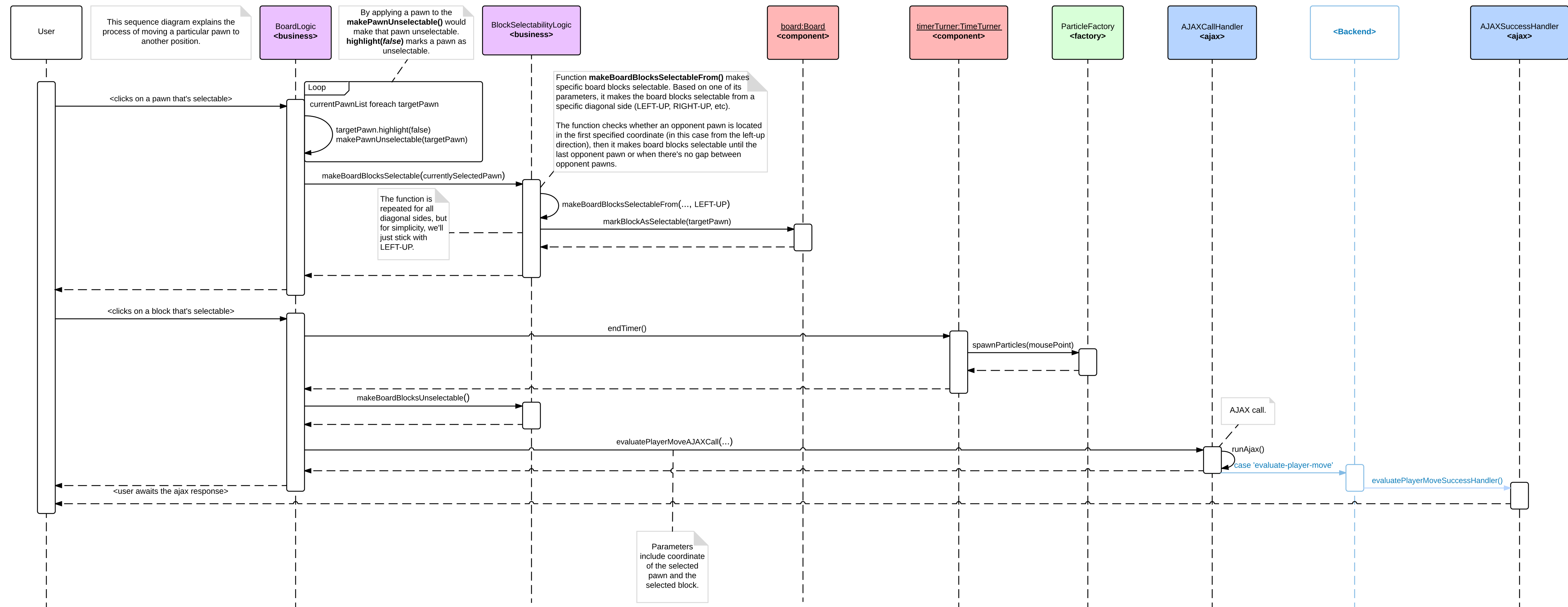
**Board** - Component which represents a board.

**BoardPawn** - Component which represents a single pawn.

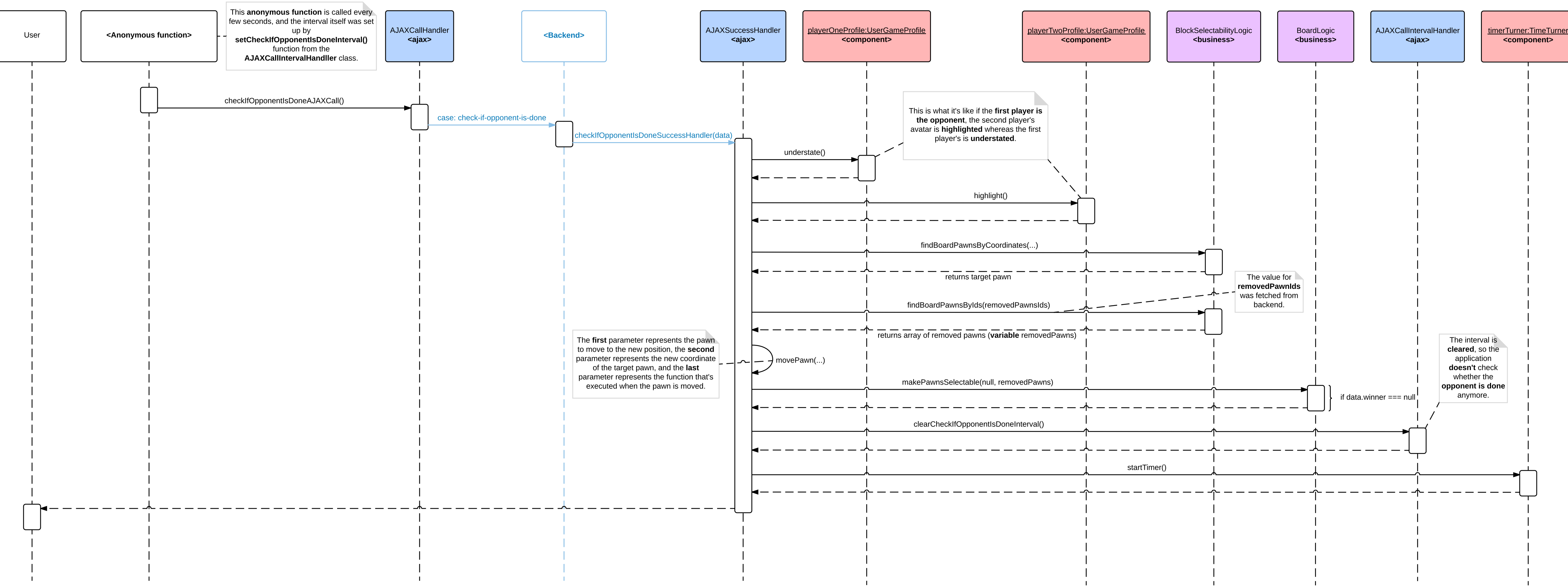
# Class Diagram (Game)- Low Level



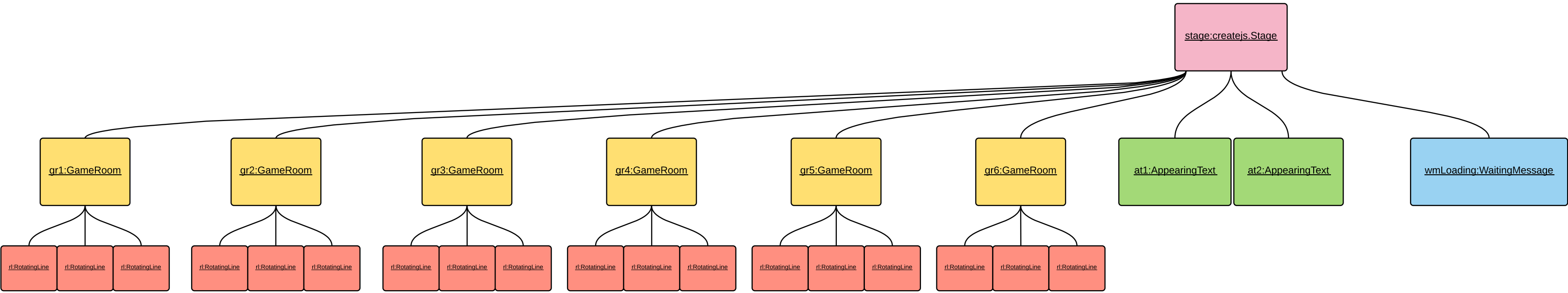
# Sequence Diagram - Moving a pawn



Sequence Diagram - Check if opponent is done



MENU



GAME

