Software Architecture Document
MicroGames
Justin Tantiongloc
Matthew Bohl

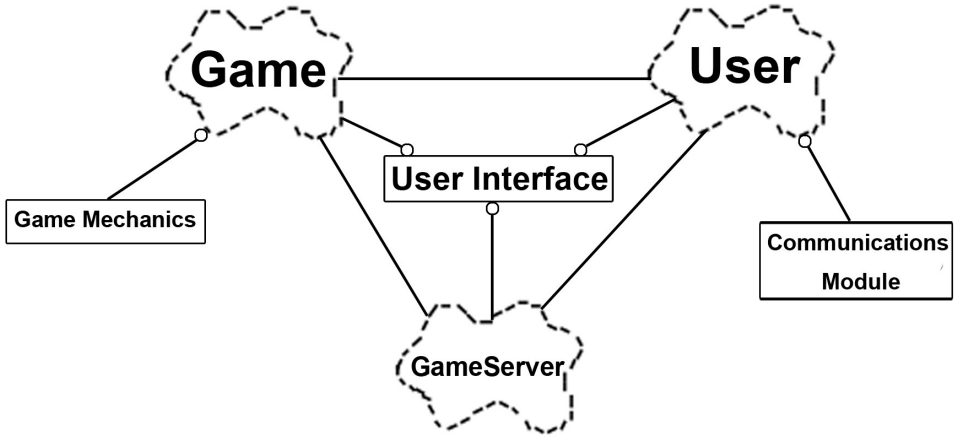
Introduction

This document provides a description of the software architecture of the MicroGames system. MicroGames provides a service allowing users to play fast, easy to understand and conveniently paced board games with each other through a web application implemented using Ruby on Rails. The goal of MicroGames is to facilitate the play of as many board games as possible without subjecting users to overwhelming user interfaces or extremely strict time limits dictating the pace of the game or how long a user has to make a move. In this way, users are given a casual, yet fun, way to enjoy their favorite board games with other users in semi-real time.

The goal of this document is to provide a high-level overview of the software and system architecture using the 4+1 Architectural View Model.

Logical Representation

The system can initially be represented as a standard games server with users associated to individual gaming sessions.



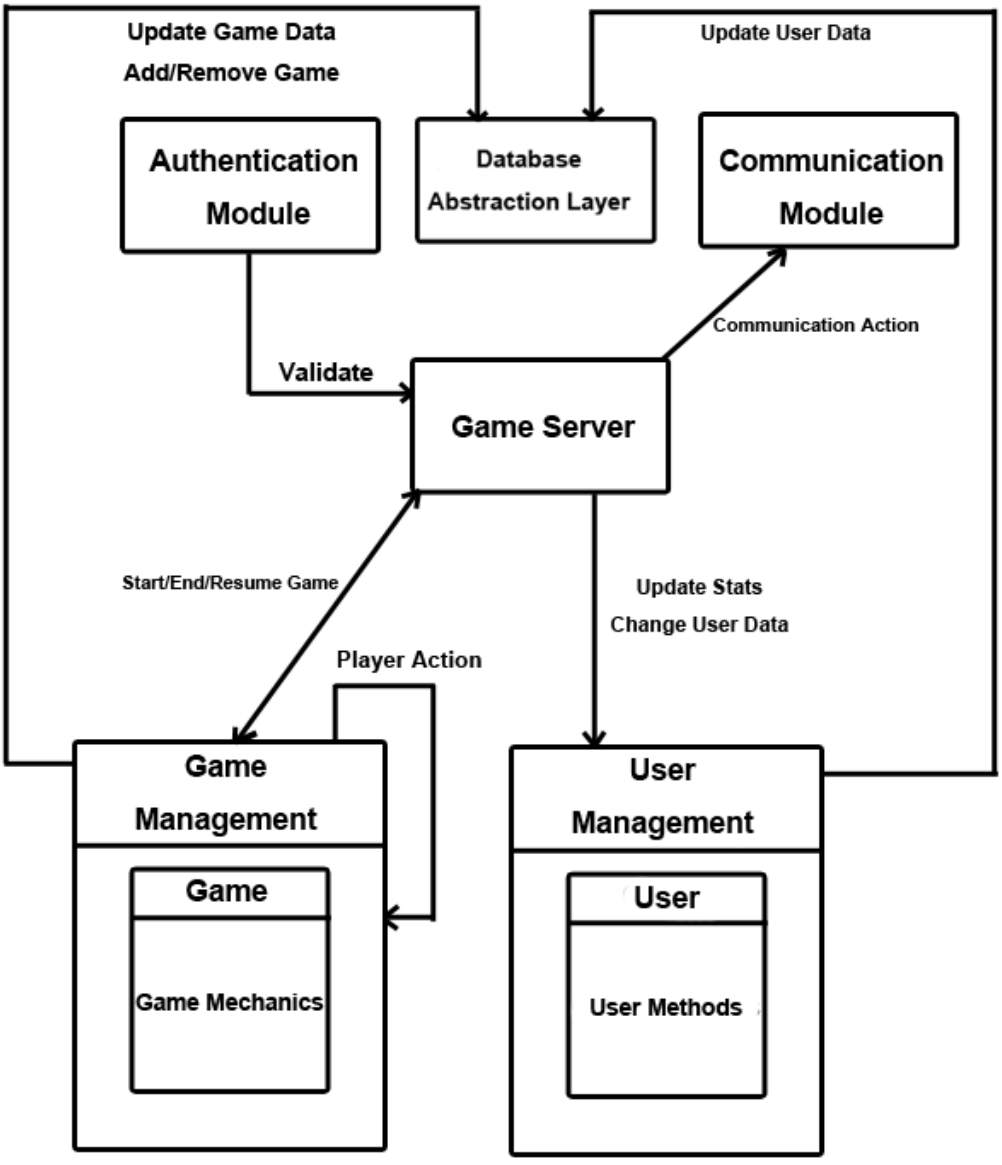
The largest piece of the system is the Game Server/Gateway class. This module represents the management system of instantiating gaming sessions, allowing users to log in and host games etc. It will provide the initial interface users will interact with, and be the starting point of all new games. In addition, it is the primary communication area between users. Associated with the Game Server/Gateway are the User and Game classes.

In this representation, the Game class represents a single instance of a game in progress, with its type dictating the specific board game it represents. Users are associated to each Game object as they join the session to play. Our initial system will only support gaming sessions involving exactly 2-players; this circumvents the need for a computer player and the need to accommodate more than 2 Users

associated with each gaming session and the synchronization it would necessitate.

Each Game object will then have access to class methods governing the rules of each game and that ensure game integrity. Users themselves will interact with the User Interface classes, which then forward player decisions and actions to the Game Mechanics module via the Game object.

Development Representation



The basic modules within the Development Representation are essentially the same as in the Logical Representation, however with a few additional components. First is the Authentication Module within the Game Server module which is responsible for authentication of new or existing users. This module will take advantage of implemented security measures to check the database for a user's credentials and

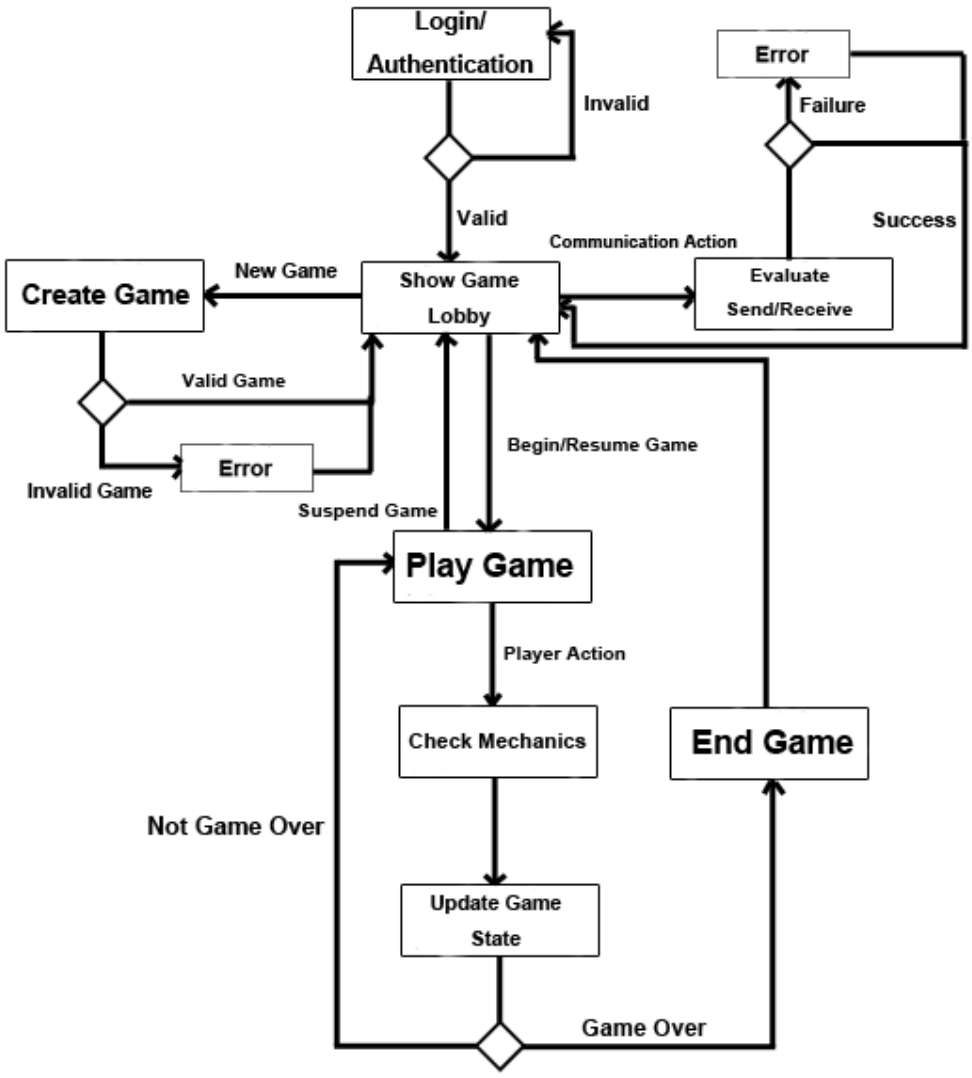
validate the login of existing users, or the creation of new ones. This module can also handle the removal of users from the database.

The Game Server module once again serves as the gateway to all other actions. If a game is to be created, the user can specify a new game type and name and a new instance of Game will be created within the database via the Game Management module. When the user wishes to start the game, they once again communicate with the Game Server which then moves control to the Game Management module. From this point, the Game Management module manipulates the corresponding Game object to the current game session. The functionality of the Game Mechanics module within each Game object includes evaluating player actions, keeping track of whose turn it is, changing the state of the game depending on what players do, and deciding when a player has won or lost the game. When a game ends, its object should return control to the Game Server and destroy itself.

If a communication action is initiated on the Game Server interface, a special communication module will be invoked. This module handles all communication and message delivery between players.

The User Management module handles logistic configuration of User settings as well as adding associations between Users and Games when new Games are created. When games end, the User Management module will also handle updates to a User's game history and evaluate gaming statistics for later use based on the latest game's result.

Process View



The Process View summarizes the functional behavior of the system. Again, engaging the system begins with authentication. In the event that a user is successfully validated, the Game Lobby will be shown on the UI for the user to interact with. The user can then create a new game or play a game

(either newly created, or already existing). When in-game, if it is the user's turn, the user can make an action consistent with available actions according to the current gaming session's mechanics. If an action is made, the game's mechanics are checked to ensure that the move is valid, and to determine what effects the action has on the state of the game. The state of the game is then updated on the server, and the game is evaluated further to determine if the game has been completed. If the game is still in progress after the latest user action, the process can repeat on the next user action (note that this illustration does not explicitly capture the concept of alternating player turns, but this can be viewed as a check against the game's mechanics to decide if it's the acting player's turn). If the game has ended, the system enters the End Game phase (i.e. show "Game Over" screen, display winner, etc.) and the user is returned to the Game Lobby.

Finally, the user can send or receive messages from the Game Lobby. Similar to other messaging systems integrated into gaming systems, a completely independent system handles the sending/receiving of messages.