## Software Design Document

Intra-network Multiplayer Scrabble® Application

Maximus Latkovski

Jy’el Mason

David Carr

Samuel Costa

Ian Boyer

## 

1. Introduction

This section gives an overview of the Scrabble® design document, in addition to the project's scope description and intended audience.

## 1.1 Purpose

The purpose of this document is to describe an online, multiplayer Scrabble® game application similar to that of the Hasbro game. This document will illustrate a mock-up representation of the user interfaces and describe the application constraints and fail safes. It will also explain the functions of which the application will be capable.

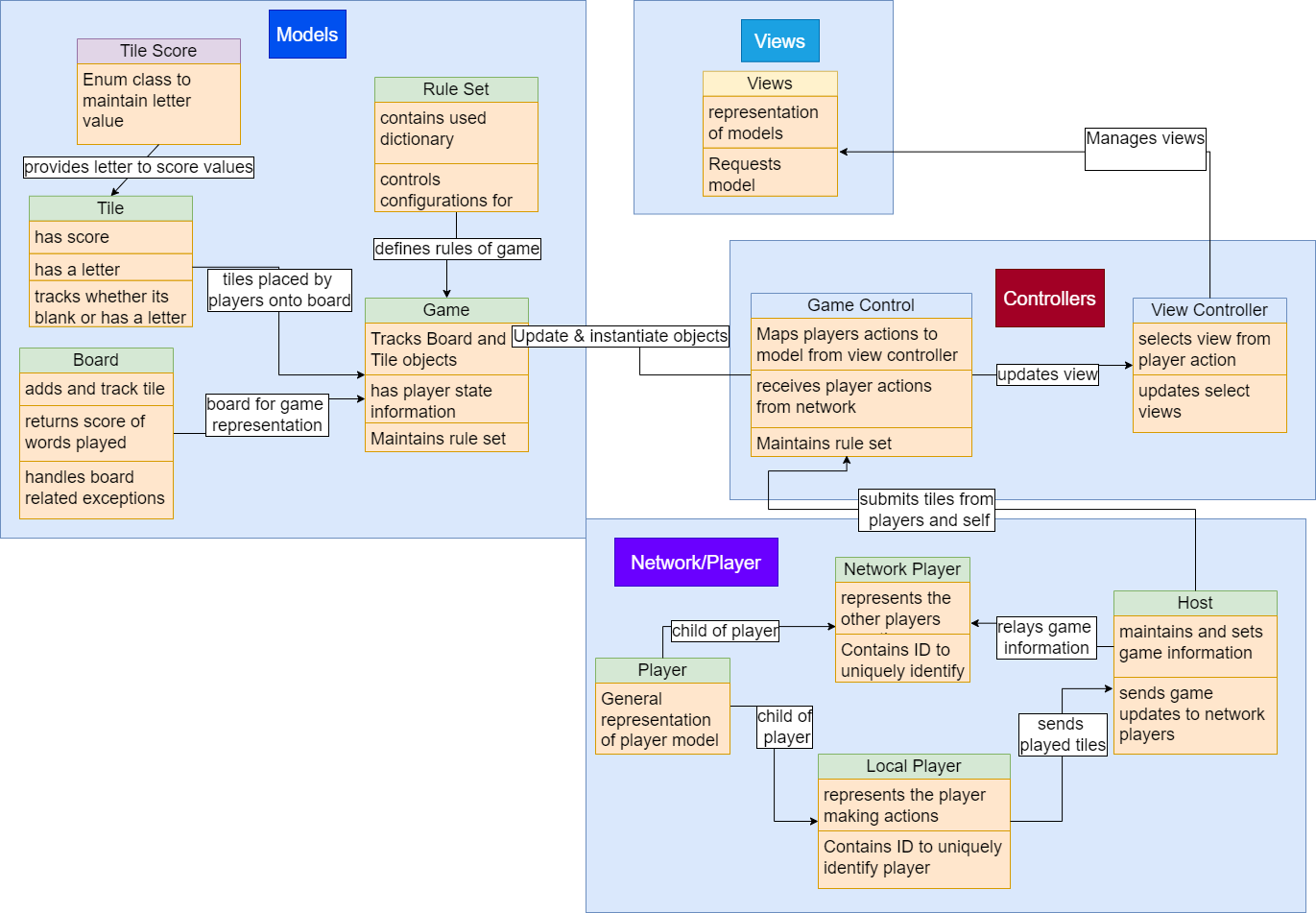
## 1.2 Scope

The Scrabble® game application is a desktop-based game application intended to connect 2 to 4 players at a time, with the host having multiple options to change how the game plays. This application is based on Hasbro’s game of Scrabble® and will include all the same rules. This application is also intended for the same audience, ages 8 and older, with the advantage of using software to eliminate the need for a physical board.

## System Architecture

The Scrabble system will comprise several objects. The game's function and representation will have multiple objects to represent the game. This includes Tile, Board, Game, Player, Host. The game object will contain and send data to both the board and the game controller, this information will further be given to the host and then sent to the rest of the players which are named network player, while local player refers to the current users' instance of the game

The game will use tile objects that have letters and associated scores and pass them to a board which will score and validate the playable words. Information is sent to and from the game and necessary objects from the game controller which will receive network information in addition to updating the views through the view controller



Our networking will rely on a single host player committing changes to the board and game and then relaying that information back to all players. The initial connection will require the players to know the chosen host's IP to then send a connection request to the host. Once ready, the host will commence the game sending all relevant information to all players. For each turn, a player will submit a validated tile set to the host to then update the board, sending relevant information back to all players.

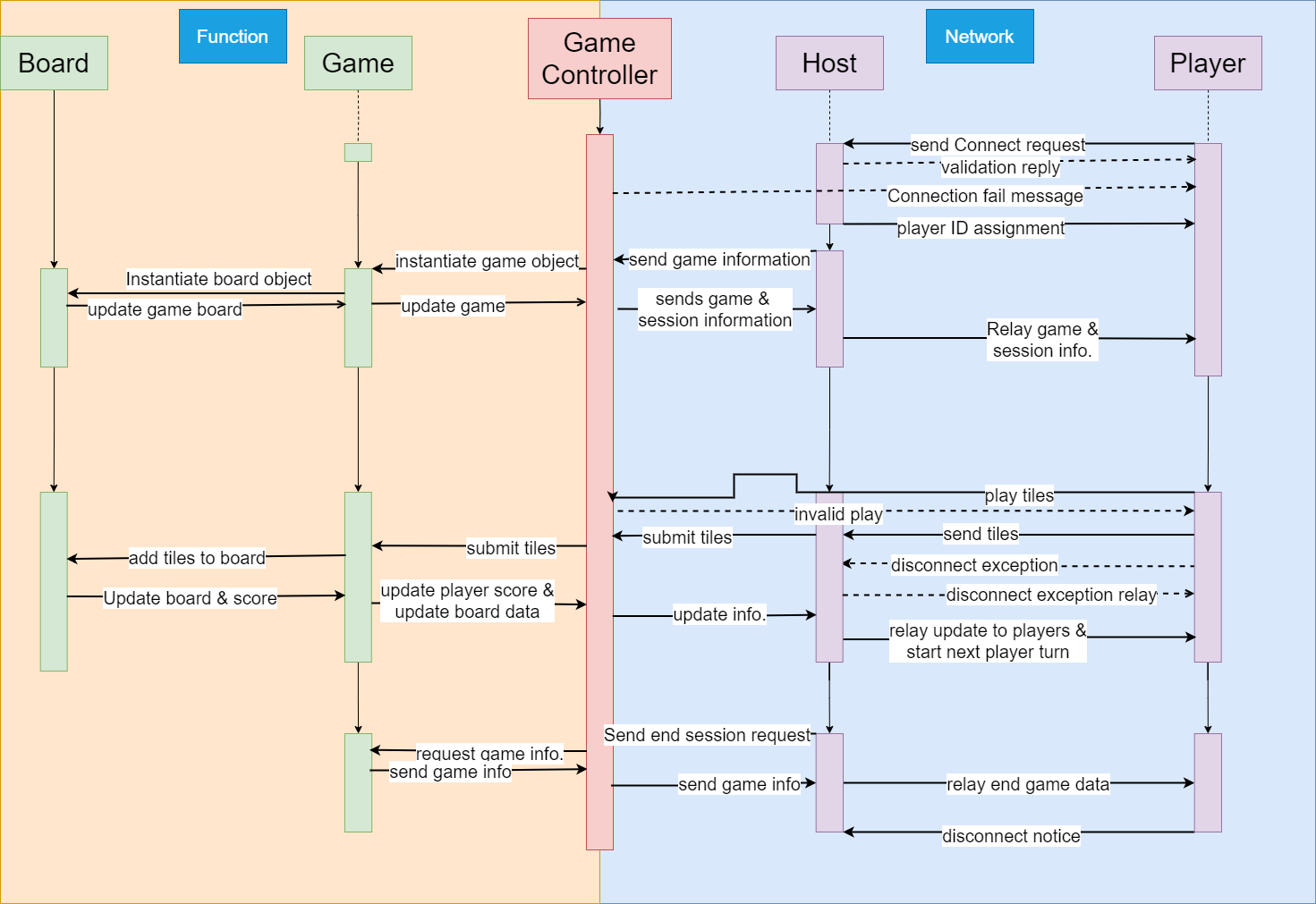
The view controller's intended purpose is to manage the total views that must be served to the user; the game controller object will update the views from user inputs and make changes to the game model.

## Data Design

Our Scrabble game will use a multitude of data structures to keep track of both player and board information. Necessary data structures include: the board represented by a 2D array of tiles, the player tile rack represented by a tile array, furthermore the board contains a modifier hashmap for tracking special cells, and lastly a string array to track words created by prior players' tile placement.

Figure 2 sequence diagram starts by describing the initial connection process between the host and the players where players will enter the host IP to then connect to and when the connection times out and does not connect we inform the player of the error. Once players enter the lobby they are assigned an ID.

Afterward, the host will submit the game settings which include the dictionary being used and the ruleset which includes game time, player time, and whether a challenge mode is enabled. This information will be used to create the game session which includes instantiating the board which will be sent back to the game controller. In addition to sending the board, the host will send the initial racks of each player to the respective player. Then, players will take their turns in order by ID number.



When a player submits a set of tiles, an exception can be thrown from the game controller to ensure all tiles are placed in a valid location. Once confirmed, the array of placed tiles is sent to the host, where they will be placed on the board. The board will then return an array of the words played and the score gained per word. When being sent back to all players the score total will be sent to the active players' ID while board placement information is sent to everyone equally.

In the case of player disconnection at the wait screen (as shown in the interface design), they will simply be removed from the game. If they disconnect while in a game session, the player’s tiles will go back into the bag, and the bag will be reshuffled; tiles already played will remain for the rest of the game. If the host leaves, the game terminates, and player screens return to the main menu. If a game only has 2 players, and one of the players disconnects from the game, the remaining player will return to the main menu. After the game is finished, the game will disconnect the players from the host, sending them to the main menu.

## Component Design

|  |  |
| --- | --- |
| **Class Name**: LocalPlayer | |
| **Brief description:** A LocalPlayer is the representation of the Player on the local machine. This player object is distinct from a network player to enforce encapsulation in the system | |
| **Attribute** | **Attribute Description** |
| rack | This array of Tile objects keeps track of which Tiles, which individual letters, the user has access to and can play |
| **Methods** | |
| LocalPlayer | **Method Description** |
| This method constructs a LocalPlayer object from their name, ID, and rack |
| **Method Signature** |
| public LocalPlayer(String name, int ID, Tile[] rack) |
| LocalPlayer | **Method Description** |
| Constructs a LocalPlayer object without setting their rack |
| **Method Signature** |
| public LocalPlayer(String name, int ID) |
| getRack | **Method Description** |
| Returns an array of the tiles which the local player can place on the board |
| **Method Signature** |
| public Tile[] getRack() |
| removeTiles | **Method Description** |
| This method removes specified tiles from the rack |
| **Method Signature** |
| public void removeTiles(Tile[] tiles) |
| addTiles | **Method Description** |
| Adds the specified tiles to the rack |
| **Method Signature** |
| public void addTiles(Tile[] tiles) |

|  |  |
| --- | --- |
| **Class Name:** NetworkPlayer | |
| **Brief Description:** A NetworkPlayer is a representation of a player who is connected via a network. The representation for a NetworkPlayer differs in that the number of tiles is listed instead of which tiles the player has access to. | |
| **Attributes** | **Attribute Description** |
| numTiles | The number of tiles which the player has. It is an integer |
| isConnected | **Attribute Description** |
| Whether the player is connected to the host. It is a boolean |
| **Methods** | |
| NetworkPlayer | **Method Description** |
| Constructs a new networkPlayer object from their name and ID. They are assigned the starting number of 7 tiles |
| **Method Signature** |
| public NetworkPlayer(String name, int ID) |
| isConnected | **Method Description** |
| Returns whether the player is connected to the host |
| **Method Signature** |
| public boolean isConnected() |
| setConnected | **Method Description** |
| Change the state of the player’s connection. A disconnection signifies that they are no longer active |
| **Method Signature** |
| public void setConnected(boolean connected) |
| getNumTiles | **Method Description** |
| Returns how many tiles the player has in their rack |
| **Method Signature** |
| public int getNumTiles() |
| setNumTiles | **Method Description** |
| Changes how many tiles are in the player’s rack |
| **Method Signature** |
| public void setNumTiles(int numTiles) |

|  |  |
| --- | --- |
| **Class Name:** Player | |
| **Brief Description:** A generalization of Local- and NetworkPlayer objects. This parent class holds information that is useful to both, such as name, ID and score. | |
| **Attributes** | **Attribute Description** |
| name | The name of the player. This name will be visible to other players. It is of type String |
| score | The total score which the player has accumulated over their turns. It is of type int |
| ID | The unique identification of the player. This field is also the order in which the player takes their turn. It is of type int |
| hasPassedLastTurn | This field tracks whether the player has passed their previous turn. This information is used to enforce the requirement on a player passing two consecutive turns. It is of type boolean |
| isActive | This field tracks whether a player is legally allowed to make plays on the board. In the case they pass two turns, this will be set to false and they may not play on the board. It is of type boolean |
| **Methods** | **Method Description** |
| Player | Constructs a new player object from their name and ID. |
| **Method Signature** |
| public Player(String name, int ID) |
| getName | **Method Description** |
| Returns the String which represents the player’s name |
| **Method Signature** |
| public String getName() |
| getScore | **Method Description** |
| Returns the integer score of the player |
| **Method Signature** |
| public int getScore() |
| setScore | **Method Description** |
| Allows the score of the player to be changed to a new set value |
| **Method Signature** |
| public void setScore(int score) |
| getID | **Method Description** |
| Returns the unique int(eger) ID of the player |
| **Method Signature** |
| public int getID() |
| isHasPassedLastTurn | **Method Description** |
| Returns the boolean of value of whether the player has passed their previous turn |
| **Method Signature** |
| public boolean isHasPassedLastTurn() |
| setHasPassedLastTurn | **Method Description** |
| Changes the state of hasPassedLastTurn to some boolean value passed in |
| **Method Signature** |
| public void setHasPassedLastTurn(  boolean hasPassedLastTurn  ) |
| isActive | **Method Description** |
| Returns the boolean value of the player’s ability to make plays |
| **Method Signature** |
| public boolean isActive() |
| setActive | **Method Description** |
| Allows the activity status of the player to be changed to some boolean value passed in |
| **Method Signature** |
| public void setActive(boolean active) |

|  |  |
| --- | --- |
| **Class Name:** Board | |
| **Brief Description:** This class will keep track of the Tiles that are placed on the board, check that they have a valid placement, make sure that the placed word is a valid word in the dictionary, and will calculate the total score for the placed word. | |
| **Attributes** | **Attribute Description** |
| private Tile[ ][ ] board | The board data structure representation |
| private Map<Point,ModifierType> boardSpecialCell | Contains the cell locations with special multipliers |
| private String[ ] lastWordsPlayed | Contains word created from tiles played by previous player |
| **Methods** | **Method Description** |
| hasAdjacentTile | checks if any of the four adjacent cells to point are occupied returns true if adjacent is occupied |
| hasDuplicates | checks if any points have same x and y value |
| sameXorY | checks that all points have either same x or y value |
| initializeModifierCells | Initializes boardSpecialCell Attribute |
| isValidWord | Checks whether a created word is within the game dictionary |
| stringBuild | Constructs words created from current tile placement |
| findOrigin | Finds the originating tiles from newly placed tiles |
| getXAndY | Returns tiles inside of given x and y locations |
| removeTiles | Removes tiles from board |
| hasAdjacentCaller | A caller method for testing purposes |
| playTiles | places tiles at positions on the board and returns the score of the play made |
| addToBoard | helper method which adds tiles to the board at specified points. does not check scoring or validity of play |
| score | helper method; calculates the score of tiles played with words and modifier cells.  Also updates lastWordsPlayed |
| validatePositions | Validates possible location of tiles on the board |
| getLastWordsPlayed | an array of the words played on the most recent board change |
| clearBoard | Clears the board of all tiles, puts null values in their place |

|  |  |
| --- | --- |
| **Class Name:** Game | |
| **Brief Description:** This class will keep track of all game information and send it Game Controller | |
| **Attributes** | **Attribute Description** |
| private Player[] players | Data Structure representing players in current game |
| private Board board | The game board |
| private Ruleset ruleset | The game ruleset |
| private LocalPlayer self | The local player representing the current instance |
| private int currentPlayerTime | how much time (in seconds) the current player has |
| private int gameTime | how much time (in seconds) remains in the game |
| private int currentPlayer | The current player identified by their ID and position in players |
| **Methods** | **Method Description** |
| playTiles | Plays tiles on the board for a player |
| getCurrentPlayerTime | Returns the current players time allowed on a play |
| setCurrentPlayerTime | Sets the current players time allowed on a play |
| getGameTime | Get total allowed game time |
| setGameTime | Set total allowed game time |
| getCurrentPlayer | Returns current player from players |
| setCurrentPlayer | Sets current player from players |
| increaseScore | Increases total score from a play |
| passTurn | Sets current player to next player in players |

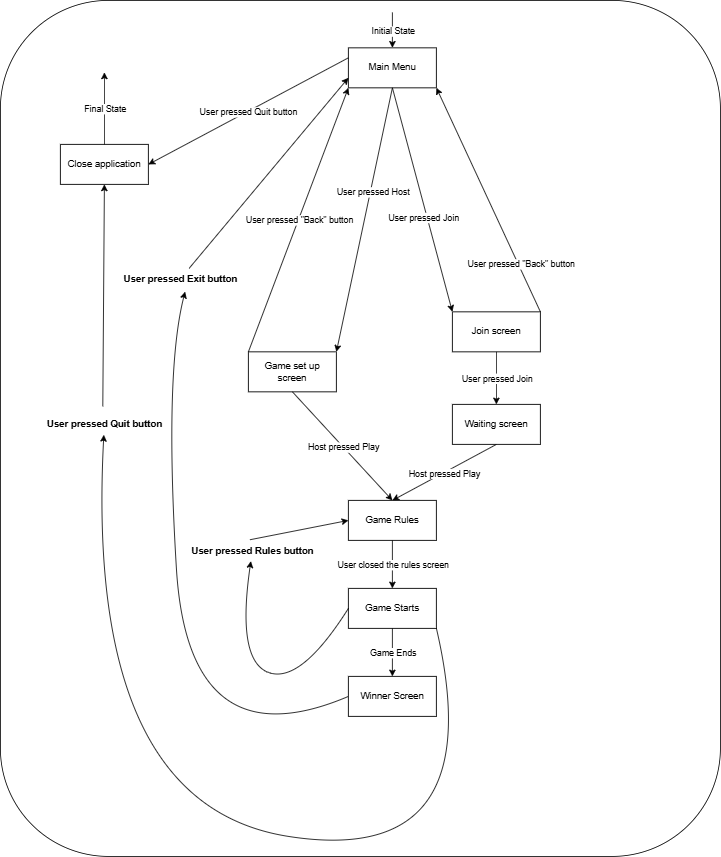
|  |  |
| --- | --- |
| **Class Name:** Ruleset | |
| **Brief Description:** contains the rules that will be used to instantiate the game object | |
| **Attributes** | **Attribute Description** |
| private final int totalTime | Total allowed time for a game |
| private final int turnTime | Total time allowed for a player to spend on a turn |
| private final boolean areChallengesAllowed | Boolean for whether the extended feature challenges is enabled |
| private final String dictionaryFileName; | File name of the dictionary being used |
| private String[] dictionary; | Array of words from dictionary file |
| **Methods** | **Method Description** |
| isWordInDictionary | Checks to see if a word is in the dictionary |
| getTotalTime | getter for total time |
| getTurnTime | getter for turn time |
| isAreChallengesAllowed | getter for challenges enabled |
| readInDictionary | reads in the words in the dictionary using the pathname given |

|  |  |
| --- | --- |
| **Class Name:** Tile | |
| **Brief Description:** representation of tile within game | |
| **Attributes** | |
| private final int score | **Attribute Description** |
| The value of the given letter |
| private char letter | **Attribute Description** |
| The letter on the tile |
| private final boolean isBlank | **Attribute Description** |
| Variable to check the tile is blank |
| private Point location | **Attribute Description** |
| Tile’s location on the board |
| private boolean isNew | **Attribute Description** |
| Variable to check if tile is newly placed |
| **Methods** | |
| setLetter | **Method Description** |
| Sets the Letter of the tile |
| **Method Signature** |
| public void setLetter(char letter) |
| getLetter | **Method Description** |
| Returns the letter associated with the tile |
| **Method Signature** |
| public char getLetter() |
| getScore | **Method Description** |
| Returns the score associated with the tile |
| **Method Signature** |
| public int getScore() |
| isBlank | **Method Description** |
| Returns if the tile is blank |
| **Method Signature** |
| public boolean isBlank() |
| setLocation | **Method Description** |
| Sets the location of the tile when it is placed on the board |
| **Method Signature** |
| public void setLocation(Point point) |
| getLocation | **Method Description** |
| Returns the location of the tile |
| **Method Signature** |
| public Point getLocation() |

## Interface Design

State Diagram:

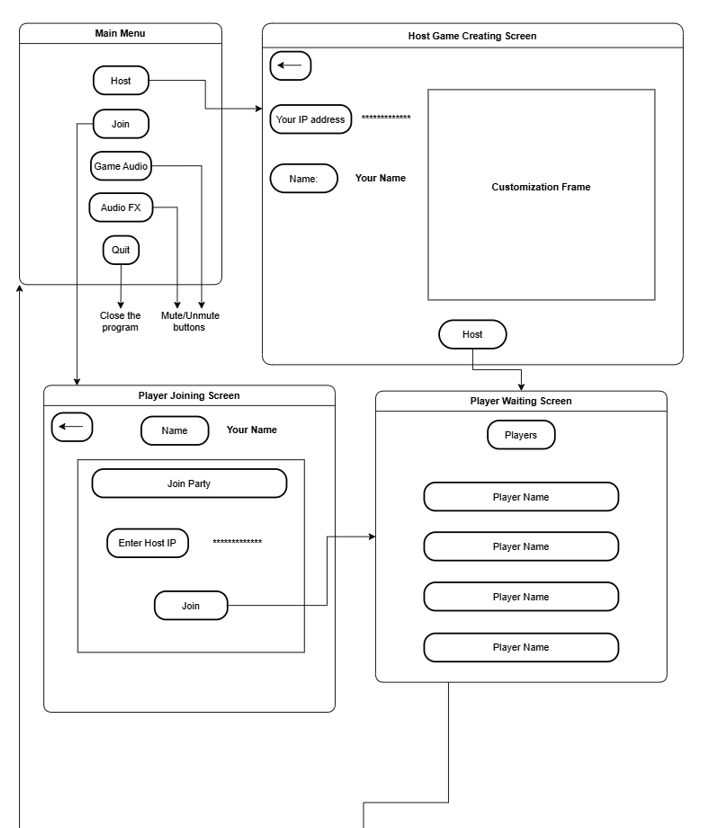
This state diagram shows the possible different screens the user will see and the different states the program will be in depending on the buttons the user chooses to press.

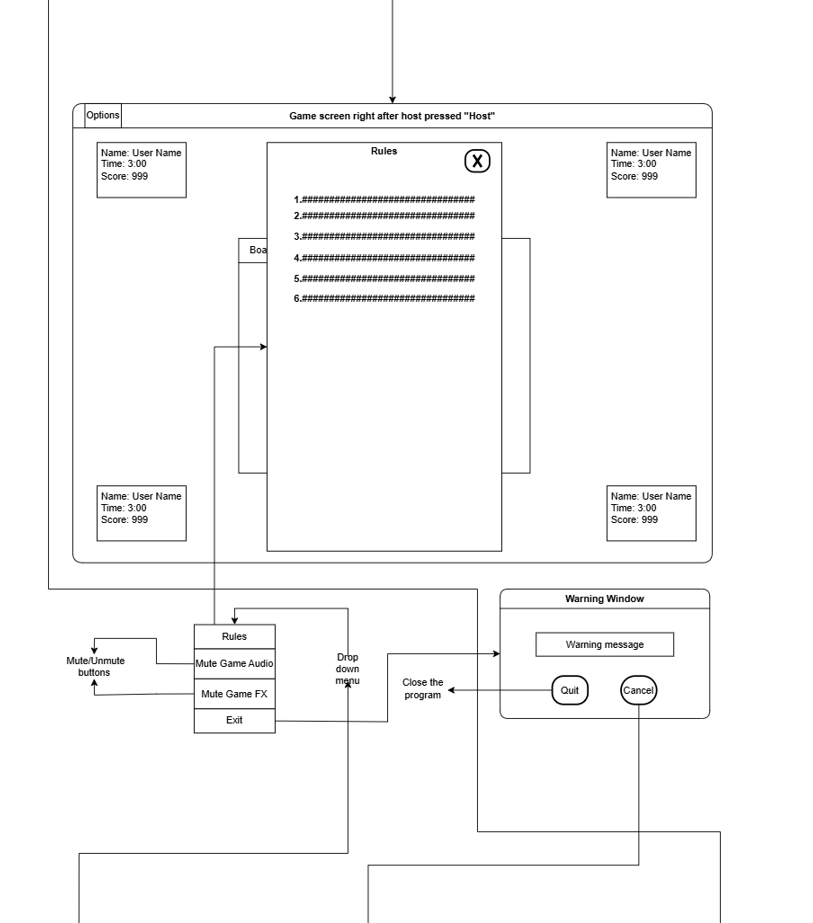


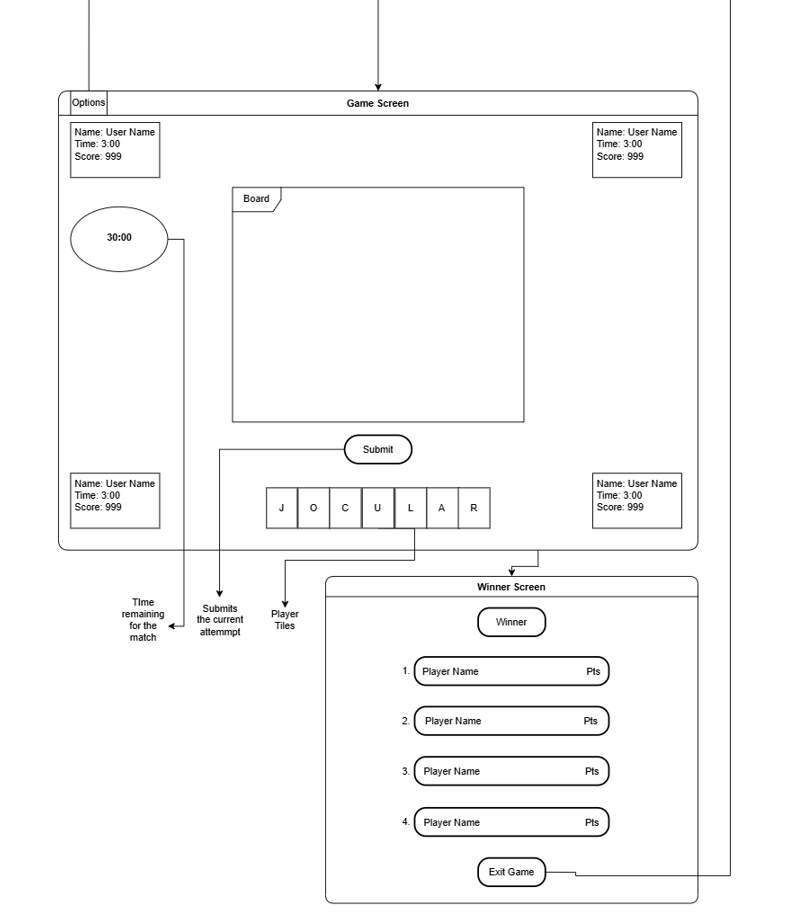
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Wireframe Roadmap:

The wireframe for this Scrabble game shows a detailed representation of how the user interfaces inside of the program will operate in relation to each other. The diagram indicates where each button will take the user. Our program should operate similarly to the diagram when it is finished.

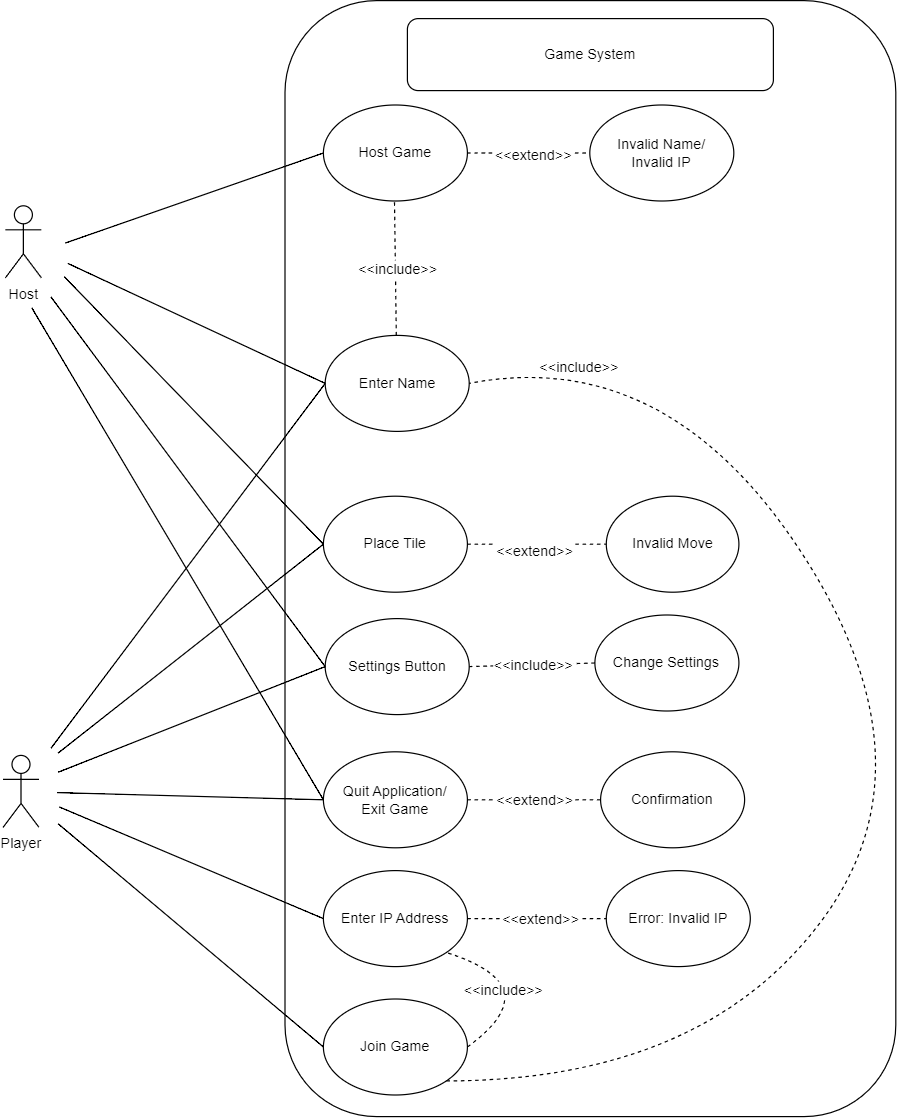






Use Case Diagram:

The use case diagram for the Scrabble game application shows the interactions between the actors (Host and Players) and the system’s functionalities. Each use case represents a specific utility that users can perform within the application with relationships to show how these functions are connected.



## Prototype

Include the names of the files which are used in the prototype. I believe (David) that all our prototype files are inside the “model” package.

## Milestones

* Completing the Model by November 9th
  + Functional Requirements:
    - Complete the Board class to handle tile placement, scoring, and board state management. (Jy’el, Max)
    - Finalize the Game class to maintain game state, manage player turns, include game timer, and enforce game rules via the Ruleset. (David, Sam)
    - Ensure the Player class and its subclasses (LocalPlayer and NetworkPlayer) accurately represent player states and actions. (Ian, David)
* Host Implementation by November 20th
  + Functional Requirements: (David, Jy’el)
    - Establish network communication protocols for NetworkPlayer to send and receive game updates.
    - Ensure that the host can relay game actions to the appropriate players.
* Networking Implementation by November 25th
  + Functional Requirements: (Sam, Max)
    - Finalize networking capabilities to allow communication between LocalPlayer and NetworkPlayer instances.
    - Ensure the Game class can send and receive player actions over the network.
* GUI Implementation by November 30th
  + Functional Requirements: (Max, Ian)
    - Design the main interface using MVC principles, ensuring that views update in response to model changes.
    - Implement UI elements for player actions, including tile placement and game settings.
    - Ensure the UI reflects the current game state, player scores, and active player information.
* Extensions by December 2nd
  + Functional Requirements:
    - Adding optional settings accessible to users relating to SFX and music volume toggles and levels.(Sam)
    - Implement Fade-in and Fade-out transitions between screens for a smoother user experience. (Max)
    - Implementation of Challenge mode, allowing players to make non-valid word plays, which other players must then challenge to remove from the game board. (David, Jy’el, Ian)

