## 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, as well as the project's scope description and intended audience.

## 1.1 Purpose

The purpose of this document is to describe a system and component design that will meet the individual requirements for the multiplayer Scrabble® game application. This document will describe the overarching system architecture, the individual components that will operate in the system, the data structures involved in maintaining the application, the interfaces for user interaction, and the development process in completing the application.

## 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.

1.3 Document Audience

This document is for the clients who must be informed of the development of the Scrabble® game application in detail. Furthermore, the document is meant to inform the developers of the required design for the models, GUIs, and network necessary for the completion of the application.

1.4 Definition of Terms

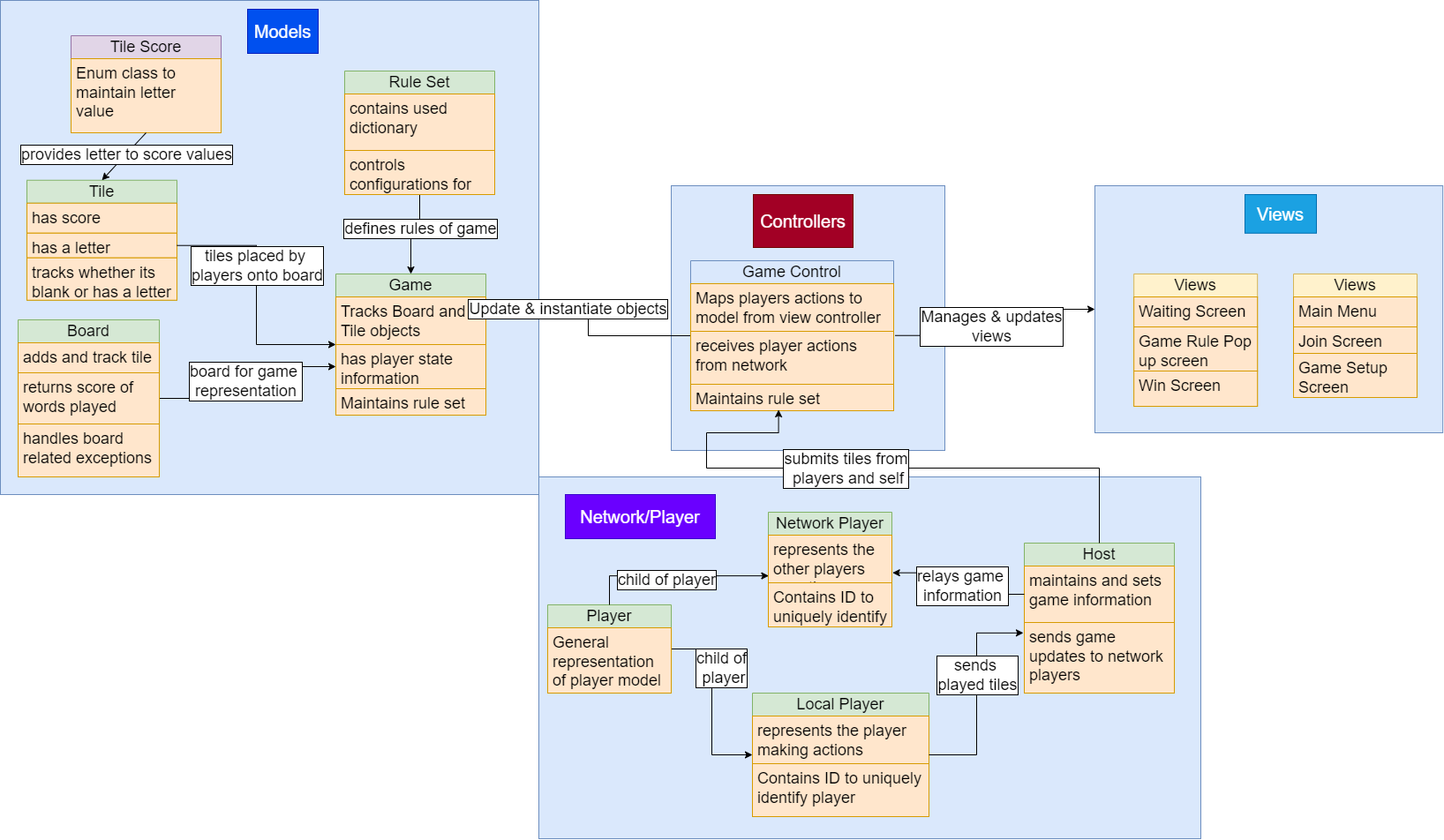
|  |  |
| --- | --- |
| **Term** | **Definition** |
| **User Terms** | |
| User | A person who is using the software described in this document |
| Player | A player is a user who is actively participating in a Scrabble® game |
| Host | The host of a Scrabble® game is the computer that manages the game, including but not limited to communication with other players, timekeeping, and Scrabble tile bag queue management. |
| Host Application | The application is designated as the host for party management. This application oversees the tile bag, timing for each player, and sending messages to non-host applications |
| **Game Terms** | |
| Board | A 15-cell by 15-cell interface upon which Scrabble® tiles are placed |
| Tile | A square containing both letter and letter value |
| Tile Bag | The data structure that holds the letters that will be played by the players during the game |
| Rack | The tiles which a player may use to make plays on their given turn. These tiles are drawn from the tile bag and will number 7 in total in a traditional Scrabble® game. |
| Cell | A spot on the playable board |
| Modifier Cells | Specific squares on the board that increase points made when words are placed on them |
| DL/Double Letter Cell | A light blue-colored cell which doubles the value of any tile placed on it |
| TL/Triple Letter | A navy blue-colored cell which triples the value of any tile placed on it |
| DW/Double Word | A pink-colored cell that doubles the value of any word on the first turn and has a tile placed on it |
| TW/Triple Word | A red-colored cell that doubles the value of any word on the first turn and has a tile placed on it |
| Letter | The character centered on the tile |
| Turn | A turn is a set period during which a player may pass, exchange either one of or all their tiles, or play tiles on the board |
| Letter Value | The numeric value on the bottom right corner of a tile |
| Word Value | The total value of a word. The individual tile scores calculate this score (see 4.5.2.5 for more details), with any modifier cells which were triggered on a turn. |
| Bingo | When a player uses all tiles on their rack |
| Turn Score | The total amount of points a player scores in each turn. This score consists of all word values, plus a 50-point bonus if a bingo is earned |
| Player Score | A player’s score is the cumulative points earned throughout their turns |
| Dictionary | List of acceptable words |
| Blitz Mode | A game of Scrabble® which has a lower timer countdown preset |

## System Architecture

The Scrabble® system will use multiple objects to represent the game. The objects will interact in a Model-View-Controller architectural pattern to maintain the organization of the application. These include Tile, Board, Game, Player (NetworkPlayer and LocalPlayer), Host, Controller and View. The Controller components have the greatest responsibility in the system, as they will communicate with the Host (described below), maintain the game model, and update the user interface as necessary. The game model (Game object) will allow for updates to be made from the controller and change its own fields as necessary to ensure the game state is consistent. Finally, the view components will be responsible for representing the data of the game model in a GUI so that users can easily understand the game state and make desired changes.

The methods of the controller can be called by the view components when events are triggered. In response, the controller will call model methods to update the game’s status, and then update view components on changes to the GUI.

For example, the game will use tile objects that have letters and associated scores and pass them to Board which will score and validate the playable words.



Now for an overview of the networking involved in the system. Each player will act as a client to a server application running in the background of a single player’s program, including the player whose system runs the server application. 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 a start game message. The initial message from the host will contain the information required to initialize the Ruleset object and set the starting tiles for each client.

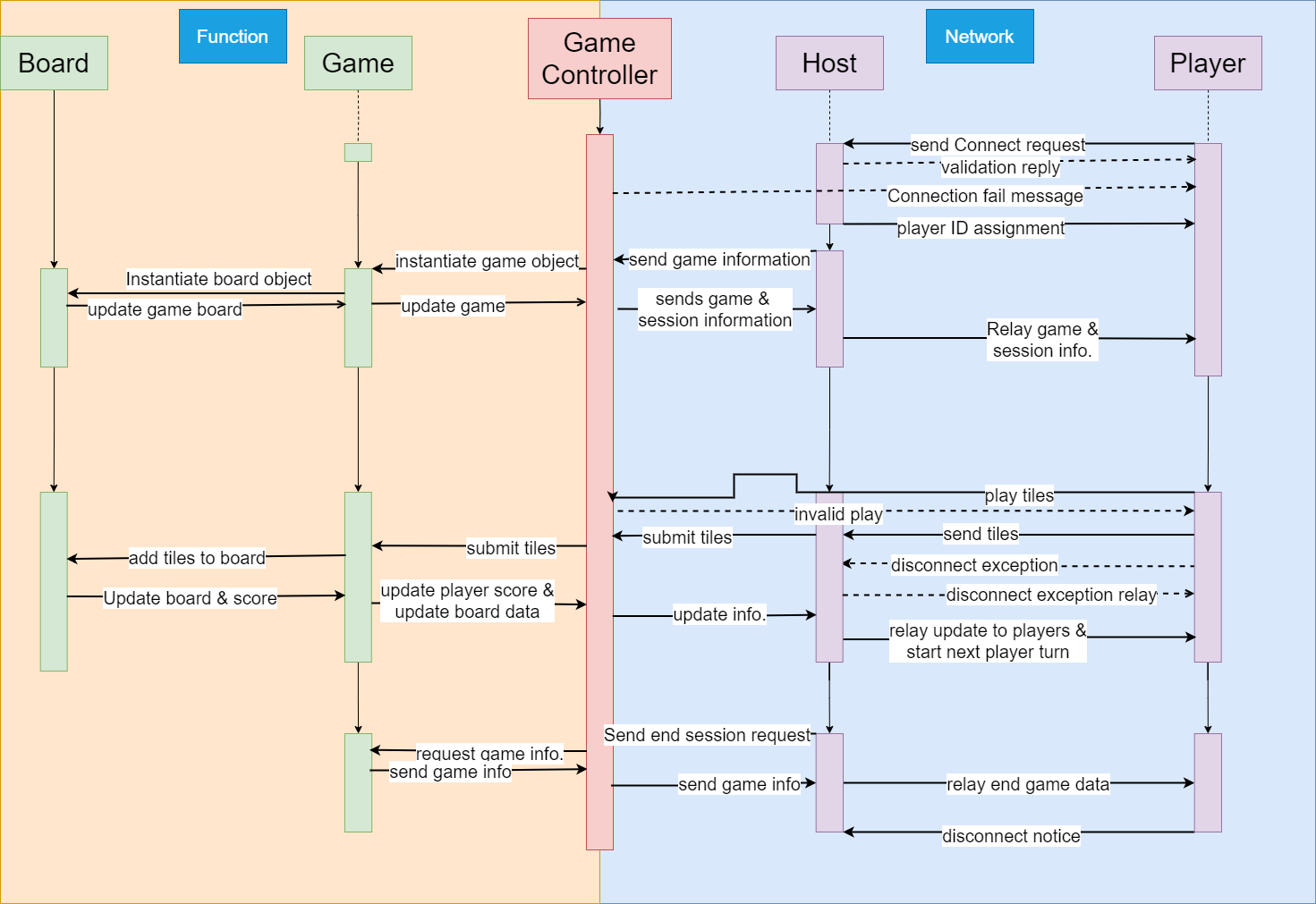
The information being sent to the other players will be in the form of message objects that will contain the necessary data to update the game model, such as the tiles being played and player IDs. For each turn, a player will submit a play turn message containing the set of chosen tiles to the host who will then distribute those played tiles to the rest of the network players. Other messages would include a message for challenging a player's word, as well as a pass message for a player to skip a turn. The game controller object will update the models and serve those changes to the GUI.

## Data Design

Our Scrabble game will use data structures to keep track of both player and board information. Necessary data structures include: the board, which is represented by a 2D array of tiles, while the player tile rack is represented by a tile array list in addition the game object also contains a bag to represent all tiles accessible to players. Furthermore, the board object contains a HashMap for tracking special cells to be used when scoring, 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 object, which includes instantiating the board, ruleset, players, and so on. The host will send the initial racks to each respective player. Then, players will take their turns in order by ID number.



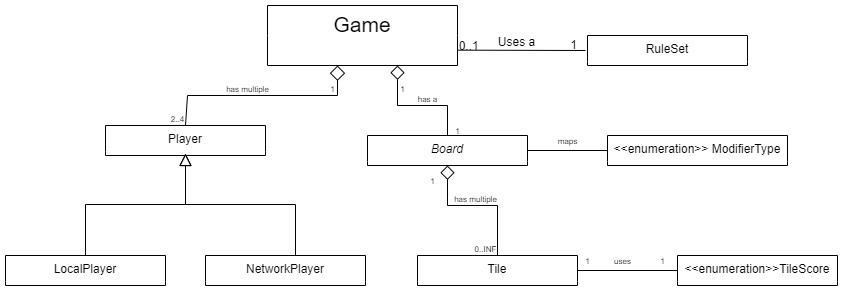
When a player submits a set of tiles, the controller calls game.playTiles(). This method will return a negative number if the word does not have valid positioning, or a positive number if the word is placed. Once confirmed, a message to play tiles will be sent to the host, as explained above; this message will contain the tiles being played, as well as the integer value of the player. The host will then relay the tiles played information to the network players. Once the players receive the play tiles message, the tiles will be added to the player’s own board. The play will allow the controller to update the score of the player who made the play via their player ID.

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.

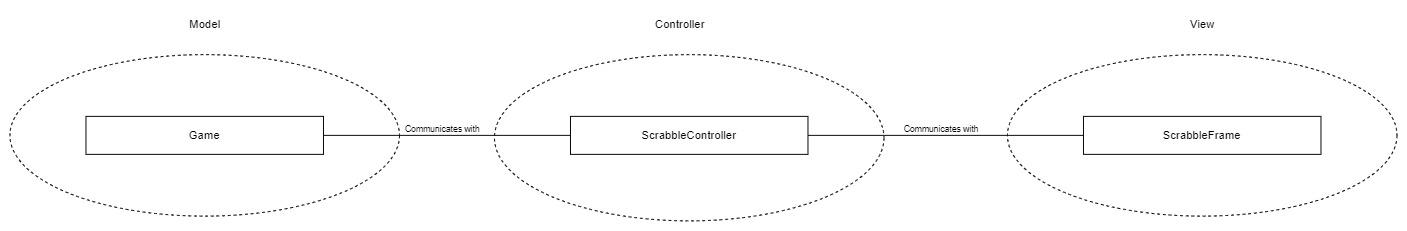
## Component Design

This section will describe the individual classes (components) which will make up the Scrabble application. The following diagrams are provided as visual overviews of the classes and interactions between classes that make up the system and subsystems of the Scrabble application. They follow the standards of UML class diagrams.

The following figure represents the game model subsystem. A Game object is made up of Players, a Board, and a RuleSet. Board maps ModifierType to cells for scoring purposes and keeps track of all tiles played on the board (marked as infinite, but up to board rows times columns). Tile uses TileScore to keep track of letter scores. Finally, Player has two subclasses for more detailed fields and methods.



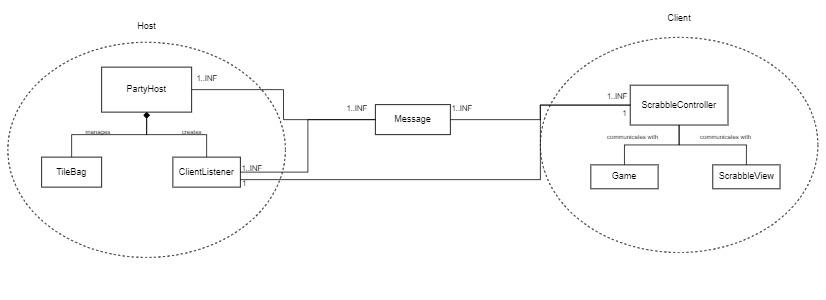
This diagram simply shows that a player’s application is made up of Model, Controller, and View. Model and View communicate with the Controller when updates are made.



This diagram shows the classes involved in the Host side of Client-Host communication. The PartyHost class is responsible for managing the TileBag and relaying messages as it receives updates from the ClientListener objects. The TileBag represents the remaining tiles that have been neither assigned to a player nor played on the board. The PartyHost selects tiles from the TileBag to send to players who need more. The ClientListener objects operate as their own Threads ([Java Thread](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html)) which listen for new messages from players (clients). The ClientListener notifies the PartyHost of new messages so that it can take appropriate action.



This diagram shows the interactions between client and host components. The ScrabbleController class sends Message objects to the ClientListener object of the host in regular play. This object then relays these Messages to the PartyHost. PartyHost only listens to ScrabbleController at the start of the game. Messages can be sent from the PartyHost to the ScrabbleController during the game, but all further listening is channeled through the ClientListener object.



Here begins the descriptions of individual classes. Each class is represented in a table, with a short description of its responsibilities, the fields (variables) used by the class, and the public methods, both static and instance.

|  |  |
| --- | --- |
| **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 that 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 |
| passedLastTurn | This field tracks whether the player has passed their previous turn. This information is used to enforce the requirement for 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() |
| passedLastTurn | **Method Description** |
| Returns the boolean of the value of whether the player has passed their previous turn |
| **Method Signature** |
| public boolean passedLastTurn() |
| setPassedLastTurn | **Method Description** |
| Changes the state of hasPassedLastTurn to some boolean value passed in |
| **Method Signature** |
| public void setPassedLastTurn(  boolean passedLastTurn  ) |
| 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**: 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 ArrayList 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:** 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** |
| getTile | Returns tile inside of given x and y location |
| removeTiles | Removes tiles from board |
| playTiles | Places tiles at positions on the board and returns the score of the play made. Returns a negative number if the play was not valid |
| 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 |
| toString | Returns a String representation of this board object. Where a tile is played, it includes the letter. If the cell is blank, it includes either two underscores or a representation of the ModifierType present in the cell |

|  |  |
| --- | --- |
| **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 HashSet<String> dictionary; | HashSet 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() |

|  |  |
| --- | --- |
| **Class Name:** ScrabbleController | |
| **Brief Description:** Mediator between Model and GUI components. Communicates with the host | |
| **Attributes** | |
| hostSocket | **Attribute Description** |
| The Socket object which connects to the host |
| input | **Attribute Description** |
| The InputObjectStream from which host Messages may be received |
| output | **Attribute Description** |
| The OutputObjectStream into which Messages to the host may be sent |
| view | **Attribute Description** |
| A reference to the high-level view component which handles user inputs |
| model | **Attribute Description** |
| A reference to the Game object which models the ongoing scrabble game |

|  |  |
| --- | --- |
| **Class Name:** GameFrame | |
| **Brief Description:** This class displays scrabble information to the player. It is responsible for providing options to change the game model and disabling options when they cannot be selected. | |
|  |  |

|  |  |
| --- | --- |
| **Class Name:** PartyHost | |
| **Brief Description:** This class is responsible for receiving and sending Message objects to and from clients. It uses an observer design pattern, with this class being the single observer and ClientHandler objects serving as observables. Therefore, PartyHost implements the PropertyChangeListener interface | |
| **Attributes** | |
| hostSocket | **Attribute Description** |
| A reference to this object’s ServerSocket |
| socketByID | **Attribute Description** |
| A Map of client sockets searchable by player id (Integer) |
| outputStreams | **Attribute Description** |
| The Map of OutputObjectStreams into which Messages to the host may be sent. Searchable by player id (Integer) |
| listeners | **Attribute Description** |
| An ArrayList of Threads which listen for input from clients |
| inGame | **Attribute Description** |
| A boolean which determines if the server socket is currently accepting clients |
| tileBag | **Attribute Description** |
| The TileBag which tracks the collection of remaining tiles for distribution to clients |

|  |  |
| --- | --- |
| **Class Name:** TileBag | |
| **Brief Description:** This class represents the bag of tiles in a game of Scrabble and randomly selects tiles as they are played on the board to refill client racks | |
| **Attributes** | |
| tiles | **Attribute Description** |
| The ArrayList of Tile objects which can be sent to clients |
| **Methods** | |
| addTiles | **Method Description** |
| This method adds tiles to the tile bag. The bag will be shuffled after these tiles are added |
| **Method Signature** |
| public void addTiles(Tile[] tiles) |
| removeTiles | **Method Description** |
| This method removes a specific number of tiles from the bag and returns them as an array |
| **Method Signature** |
| public Tile[] removeTiles(int amount) |

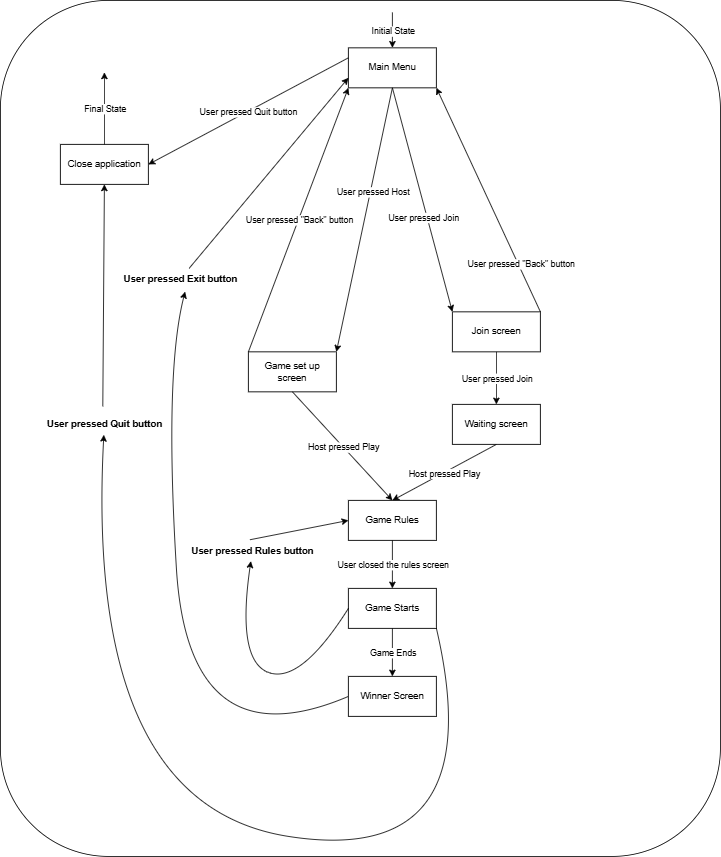
|  |  |
| --- | --- |
| **Class Name:** ClientListener | |
| **Brief Description:** This class listens to client sockets for new messages and updates the PartyHost when a new message is received. This class implements PropertyChangeSupport with PartyHost being the PropertyChangeListener (observer). | |
| **Attributes** | |
| clientSocket | **Attribute Description** |
| The Socket object to which this object should listen for messages |
| messageListener | **Attribute Description** |
| The PropertyChangeListener object which receives updates when a message is received from the client |
| input | **Attribute Description** |
| The InputObjectStream from which client Messages may be read |

|  |  |
| --- | --- |
| **Class Name:** Message | |
| **Brief Description:** This generalized class represents classes which are sent between clients and the host. Subclasses provide more detailed messages for updates to client game models | |
| **Attributes** | |
| playerID | **Attribute Description** |
| The ID of the player who is sending this message as an integer |
| timeStamp | **Attribute Description** |
| The system time at which this message is being sent to the host |
| **Subclasses** | **Subclass Description and Usage** |
| InitiateGame | This message will be sent from the host to clients when the game is starting. It will contain the Ruleset object and an array of starting Tiles. Each set of starting tiles will be unique |
| PlayTiles | This message will be sent from a client when it is their turn and they decide to play tiles on the board. It will contain the array of tiles which are being played on the board |
| PassTurn | This message will be sent from a client when it is their turn and they decide to pass their turn. It does not contain any additional information |
| Challenge | This message will be sent from a client when they decide to challenge the words played on the most recent game turn. It will contain the player ID of the player to be challenged. This message may not make it to development as it is part of possible extensions |
| ExchangeTiles | This message will be sent from a client when they decide to exchange a tile(s) on their turn. It will contain an array of the tiles to be exchanged |
| ExitGame | This message will be sent from a client when they manually close the application. It does not contain additional fields |
| NewTiles | This message will contain new tiles for a player to add to their rack after they make a play or choose to exchange tiles. It will contain an array of the new tiles to be added to the rack |

## Interface Design

State Diagram:

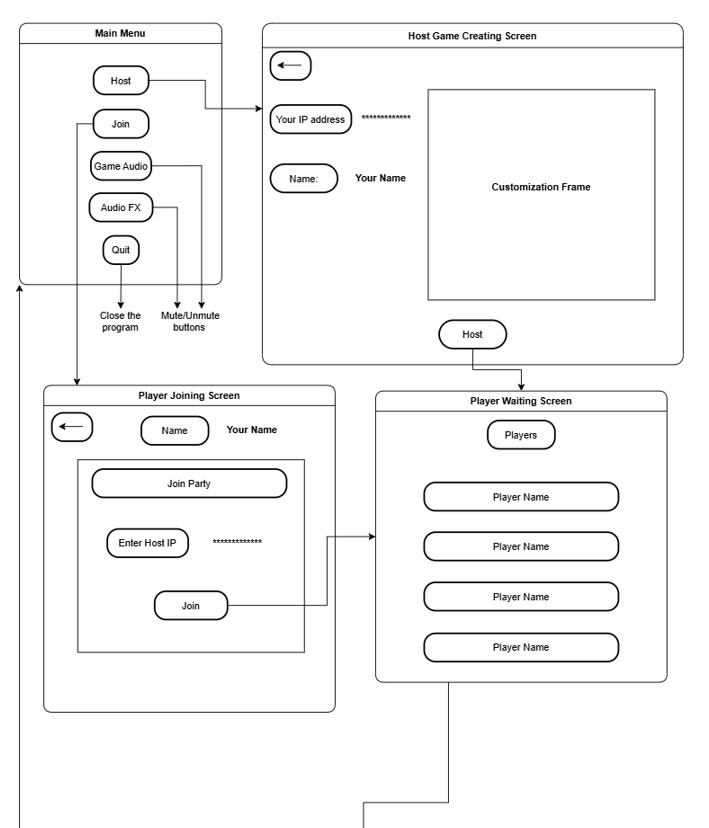
This state diagram shows the possible different screens the user will see and the different states the program will be in depending on the buttons the user chooses to press.

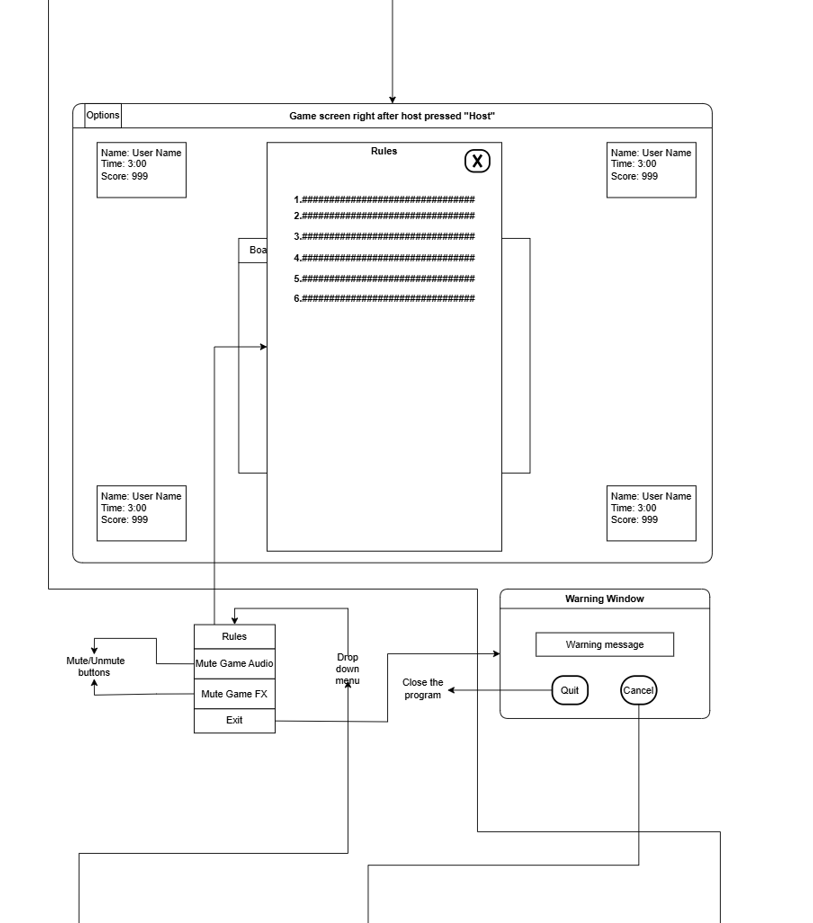


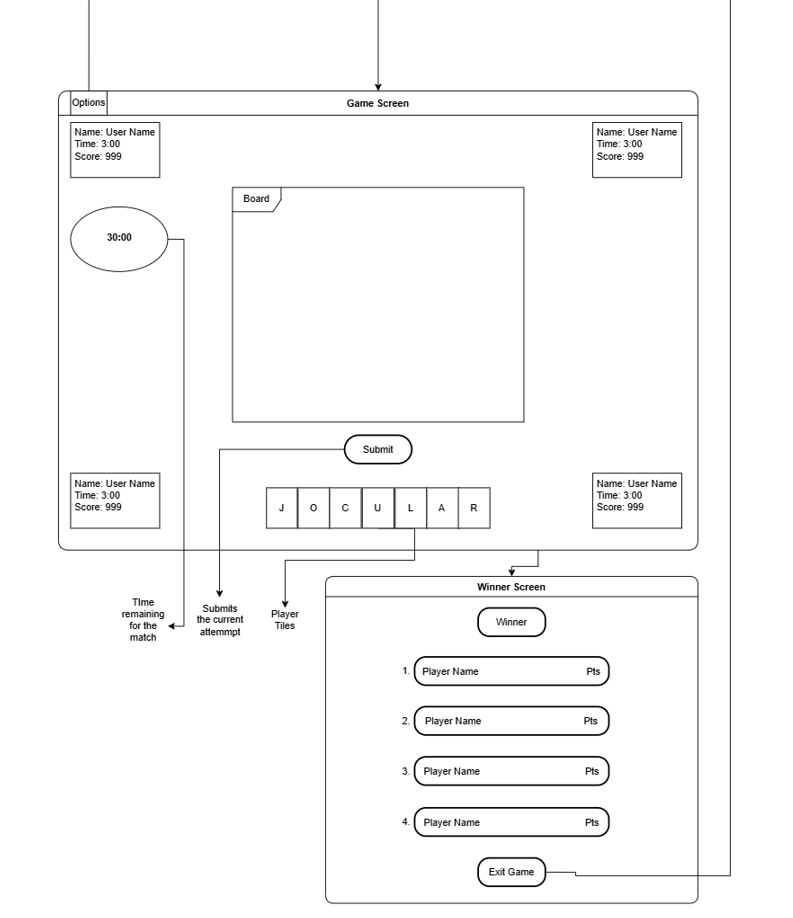
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Wireframe Roadmap:

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

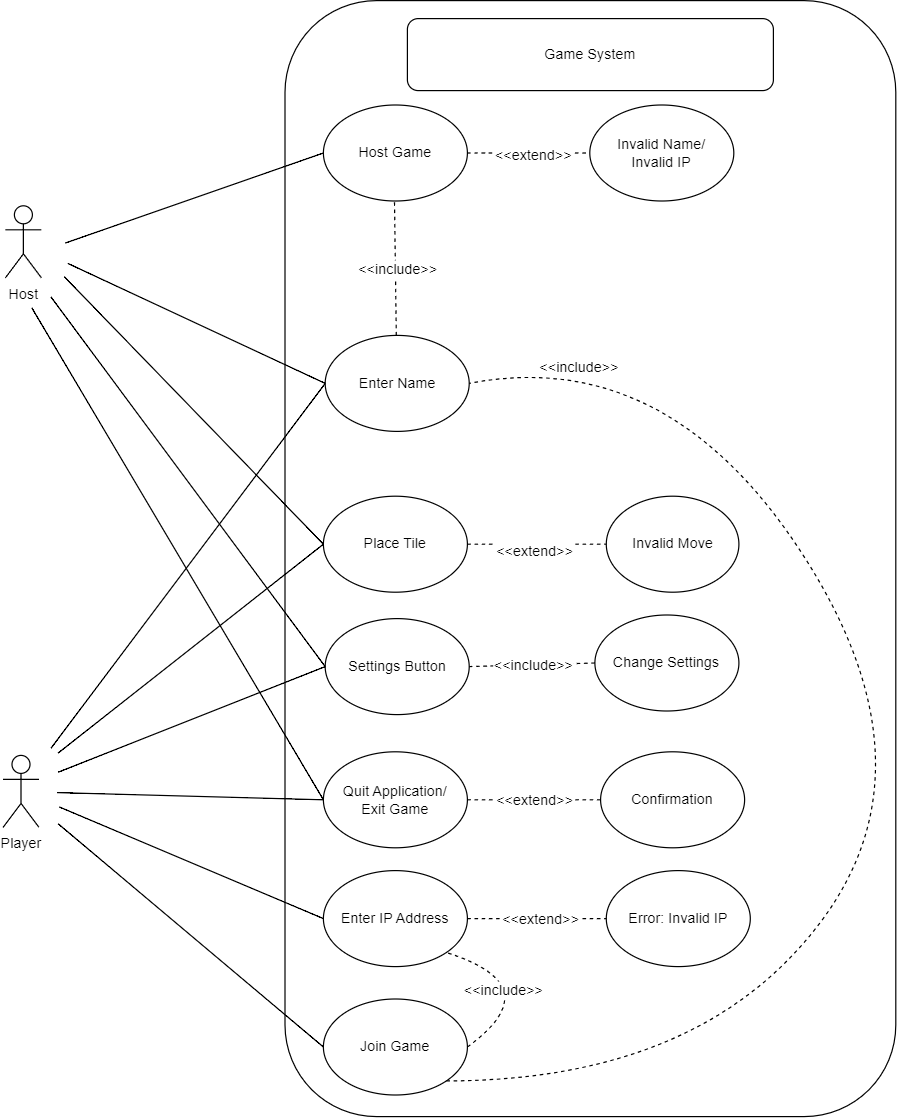






Use Case Diagram:

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

Our design has three included prototypes for different facets of the system.

1. Model testing: this prototype comes in two parts, both to explore the model portion of the application.
   1. TestBoard: this testing class has methods within it to test the core components of the model. Each method can be run individually in the IDE to show methods working in the Board class. See the description above the class.
   2. Board. main: running the main scrabble model.Board allows the user to test placing and scoring tiles with input from the console. Follow the prompts from the system.
2. Network prototype: this prototype models the networking components of the system. It involves PartyHost, ClientListener, and NetworkController. To run this prototype, start at least two instances of the Windows command line, set the path to be in the “./code” portion of the project, and then use the usage comment in PartyHost to compile and run the class. Running the class will start a server and output the host IP address and listening port. Now run NetworkController with the usage comment above the class. Make sure to use the host IP and port printed out by PartyHost. At this point, the client and server sound confirm that they are connected. Client-side console input will be printed to the server console. “End” and “Start” are special keywords that close the connection and start a game in the PartyHost object respectively.
3. GUI prototype: This prototype shows how the Main Menu, Join Screen, and Host Screen will look and function. To run this prototype, open the GameFrame.java class. Once this is open, run the main method. This will display the Main menu screen first. The “Host” button, “Join” button, and “Quit” button have been implemented. When clicking “Host”, the user will be transitioned to the Host Screen. When clicking “Join”, the user will be transitioned to the Join Screen. When clicking “Quit”, the Main Menu will close, and the program will terminate. If you press the “X” button in the top right of the Host Screen and Join Screen, the user will be transitioned back to the Main Menu.

## Milestones

* Completing the Model by October 23rd
  + Tasks:
    - Complete the Board class to handle tile placement, scoring, and board state management. (Jy’el, Max)
    - Finalize the Game class to maintain the game state, manage player turns, include the 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 October 28th
  + Tasks: (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 1st
  + Tasks: (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 7th
  + Tasks:(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
  + Tasks:
    - Adding optional settings relating to SFX and music volume toggles and levels accessible to users. (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)

