CA314 MyScrabble Analysis

Group 1

Table of Contents

Refined Requirements Specification 1.0

OVERVIEW 1.1

Based on the original overview:

- Our internet-based implementation will be a web application based on the board game scrabble.
- As mentioned, the game will take a client / server structure where the server will be the centralized component that handles information received from various clients(players) and manages / synchronizes the state of a game for all clients/players. As such, the server-side will also perform all computations and process the game logic.
- Keeping user interest : randomize where special tiles are, randomize multipliers (double, triple word & letters), randomize tile scores (A = 13 instead of 1).

Group interpretation and overview of the project scope.

1. The user will connect to a website / web app which allows them to play a game of scrabble with other users over the internet, by either creating an instance of a game which a specified number of users can then join, or by joining an existing game.
2. A game requires a minimum 2 players to begin, with a maximum of 4 players. Upon a game starting, players will then be presented with the game's interface, which will include the game board, a leaderboard, their own available letter tiles and any other key information relating to a traditional game of scrabble.
3. The game is turn-based, players place letter tiles on the board (which is a NxN grid of tile-shaped cells) with the aim to form words which will increase the players score by as much as possible.
4. The calculated value of words are based on the combination of letters used to form the words, in conjunction with the cells those letters were placed on.
5. The first letter of the first word must be played from the centre tile. The first word played must also be at least 2 letters long.
6. Following words must be played by using one or more tiles to place a word on the board. This word may use one or more tiles already on the board and must join with the cluster of tiles already on the board.
7. Any word played must be valid. It is only valid if the combination of letters form a word which exists in the English dictionary used by the game.
8. Once a valid move has been made, the player's score is increased relative to the value of the word created.
9. The player will then replace the number of tiles they used, with an equal number of new tiles randomly selected from the tile bag, while it has tiles remaining.
10. Players also have the option to pass/skip their turn. However, if 6 turns pass in a row without any tiles being placed, the game ends.
11. Additionally, players can use their turn to exchange a chosen number of their tiles randomly for an equal number of tiles in the tile bag. Their turn ends once this exchange has been made.
12. The game continues until an exit criteria has been met. The game will end when every player uses up all their available tiles, and the tile bag is empty.

13. The player with the highest score at this point is the winner.
14. Players are then prompted to either begin another game, or return to the home page.

Refined Requirement 1.2

Functional 1.2.1

- The user shall be able to connect to the game, remotely, over the internet.
- A functional UI and graphics shall be developed in order to play the game.
- The user shall enter a unique display name before creating a game, or upon joining an existing game.
- Each instance of a created game, shall generate a unique code. This code can be used by other players to join the game.
- The server must be able to manage non-unique usernames.
- The server shall allow only 1 user to make a move at a time.
- The server will manage and update the state of the game for all users/clients.
- The client will generate player data and send it to the server.
- Users will only be able to play valid words, in valid locations on the board.
- The game will have audio features to denote certain actions and events in the game.

Non-Functional 1.2.2

- Easy to run / play. Minimal resources required by the user.
- The unique code should not be too long (<= 7 characters).
- The speed it takes for the system to perform its actions must be very quick
- The request to create or join a game, should be processed quickly by the server (< 3 seconds).
- The game board must update with every turn to display the most recent state efficiently and with minimal delay (< 1 second).
- Checking the validity of words placed by the user should be done in the smallest length of time possible, so as to make the experience smooth and acceptable. E.g/ “Instant” lookup time of dictionary.
- Placing tiles on the board should be intuitive and seamless for the user.
- The UI and interface should be clear and understandable e.g being able to see all the options properly.

Scenarios (User Stories)

Accessing Webpage (www.myscrabble.ie)

Current System State:

Server is waiting.

Informal Scenario:

Marius accesses www.myscrabble.ie and the website's landing page is displayed. He is presented with the site options and UI.

Next Scenario:

Marius will input his username.

Input user name

Current System State:

Displaying home page and its functionalities: a username field, create game button, a game code field and join game button.

Informal Scenario:

Marius types the name that will be displayed during a match into the text field.

Next Scenario:

Marius will create a game.

Creating a game

Current System State:

Displaying creation page.

Informal Scenario:

Marius clicks on the create game button and is redirected to the game creation webpage. If the game is set to private then a game code is generated which Marius can share so his friends can join his match directly.

Next Scenario:

Marius invites his friend to join the game.

Joining a game

Current System State:

Displaying the home page.

Informal Scenario:

Stefan inputs the game invite code into the text field. He presses the join game button and is placed into Marius' game.

Next Scenario:

Board setup

Playing a first turn

Current System State:

The board is empty and it is Marius' turn. Both players have 7 tiles in their hand.

Informal Scenario:

Marius spells the word "SHOVEL" and places it vertically in the center of the board, with the "H" tile covering the star tile. He submits his move and it is accepted. He is given 6 tiles from the bag in order to refill his hand. His turn is over as he has no other moves.

Next Scenario:

It is Stefan's turn next.

Standard turn

Current System State:

The board is displayed as it was 1 turn prior.

Informal Scenario:

Stefan forms a word using his tiles. The word must be valid as in present in the game dictionary and must be connected to any of the pre-existing words on the board. After making a word Stefan gets back an equal amount of tiles. The score for the word is then calculated taking into account special tiles and added to Stefans score.

Next Scenario:

It is Marius' turn next.

Invalid move #1

Current System State:

It is turn 5 and it is Marius' turn. Board has 4 words around the center area.

Informal Scenario:

Marius plays the word "TROLL" in the top left corner of the board, not attached to the other words on the board. He submits the word and an "invalid word placement" error pops up on the screen, which tells him to try again.

Next Scenario:

Marius makes the word "ROLE" using the "E" from the word "ENERGY" already on the board. The system accepts his turn and his tiles are replaced.

Invalid move #2

Current System State:

It is turn 8 and it is Stefan's turn. Board has 6 words around the center area.

Informal Scenario:

Stefan plays the word "DAKSHBD" using the "D" from "DAYS" already on the board. He submits the word and an "invalid word, word does not exist" error pops up on the screen, which tells him to try again.

Next Scenario:

Stefan instead makes the word "DAB" which is valid and counts to his score.

User Tile Exchange

Current System State:

Marius has just ended his turn. It is now Stefan's turn.

Informal Scenario:

Stefan is unpleased with his hand. He presses the exchange tiles button. He selects his "E", "X" and "K" tiles to be replaced. His tiles are placed into the bag and the bag is shuffled. He receives back the same amount of tiles as he selected, these being: "A" , "F", "X". His turn ends.

Next Scenario:

Marius takes his turn.

End game

Current System State:

Players can no longer make moves. A display pops up to ask for a rematch or to return back to the home page.

Informal Scenario:

Either Stefan or Marius no longer have any tiles and the bag is empty OR Both Stefan and Marius have skipped their turns 3 consecutive times.

Next Scenario:

The game ends and the winner, Marius, is displayed on the screen as he has the most points. A rematch button appears on the screen.

Skip turn

Current System State:

It is turn 10 and it is Marius' turn. Board has 12 words around the center area.

Informal Scenario:

Marius has a mix of consonants and vowels but can not form any valid words. He presses the skip turn button in order to forfeit his turn and allow the game to move on.

Next Scenario:

It is Stefan's turn.

Primary class List

ID	Name
1.	Client
2.	Player
3.	Server
4.	Game
5.	Board
6.	Tile

Class Responsibility Collab (CRC) Descriptions

Class Name: Client	**ID:** 1	**Type:** Object
Description: The Client class will handle the communication between player and server and the initial connection. It will take the moves made by the player and send the data over to the server, once the server replies it will decode the data and send it back to the player.		**Associated Use Cases:** - 1, 2, 3, 4, 5

Responsibilities	Collaborators
sendData: The client sends data both ways between server and player	Server
connect: connects the client to the server so that the game can be played	Player
renderGame: displays the board and UI on the client side.	

encodeData: encodes data into relevant format.	
decodeData: decodes data into relevant form.	

Class Name:	**ID:**	**Type:**
Player	2	Object
Description: The class responsible for the moves a player can make during a turn and responsible for player information such as name, score etc.		**Associated Use Cases:** - 1, 2, 3, 4, 5

Responsibilities	**Collaborators**
getScore: retrieves the players current score.	Client
exchangeTiles: allows the player to swap their tiles for new ones.	
makeMove: allows the player to attempt to make a move by placing tiles on the board.	
getName: retrieves player name.	
getID: retrieves player ID.	
getTiles: retrieves how many tiles the player has in his hand.	

skipTurn: allows the player to skip their turn.	
setTiles: sets the tiles a player has in their hand.	
setScore: sets the player score.	

Class Name: Server	**ID:** 3	**Type:** Object
Description: Communicates with the client and game. It is the intermediary which handles the data being sent from the client to the game and vice-versa. It will receive/send data such as player moves and updated board states. It will also manage the connection of the client to the correct game instance.		**Associated Use Cases:** - 1, 2, 3, 4, 5

Responsibilities	**Collaborators**
encodeData: Converts the data it handles into the correct format/structure to be sent to/from either the game or client.	
decodeData: Converts data into the correct format/structure to be parsed and then utilised.	

sendData: Sends data to the client or game, packaged/structured correctly for being interpreted by the recipient class.	Client, Game

Class Name: Game	**ID:** 4	**Type:** Object
Description: Handles all the core logic of the game itself. It is the primary way in which the state of the game and board are updated. It checks the validity of moves and words. It calculates scoring, updates the board and players state, and sends the updated state back to the server.		**Associated Use Cases:** - 1, 3, 4, 5

Responsibilities	Collaborators
checkMove: Checks the validity of the move data sent by a player.	Board
updateBoard: Updates the state of the board.	Board
updateTileBag: Updates/changes the state and “contents” of the tile bag	Board

checkWord: Checks the validity of the word formed by a valid player move.	Board
updatePlayer: Updates the state of the player. Their tiles, score, etc.	Board
calculateScore: Calculates the score of the word/move made by a player.	
sendData: Sends the updated game and player state data to the server.	Server

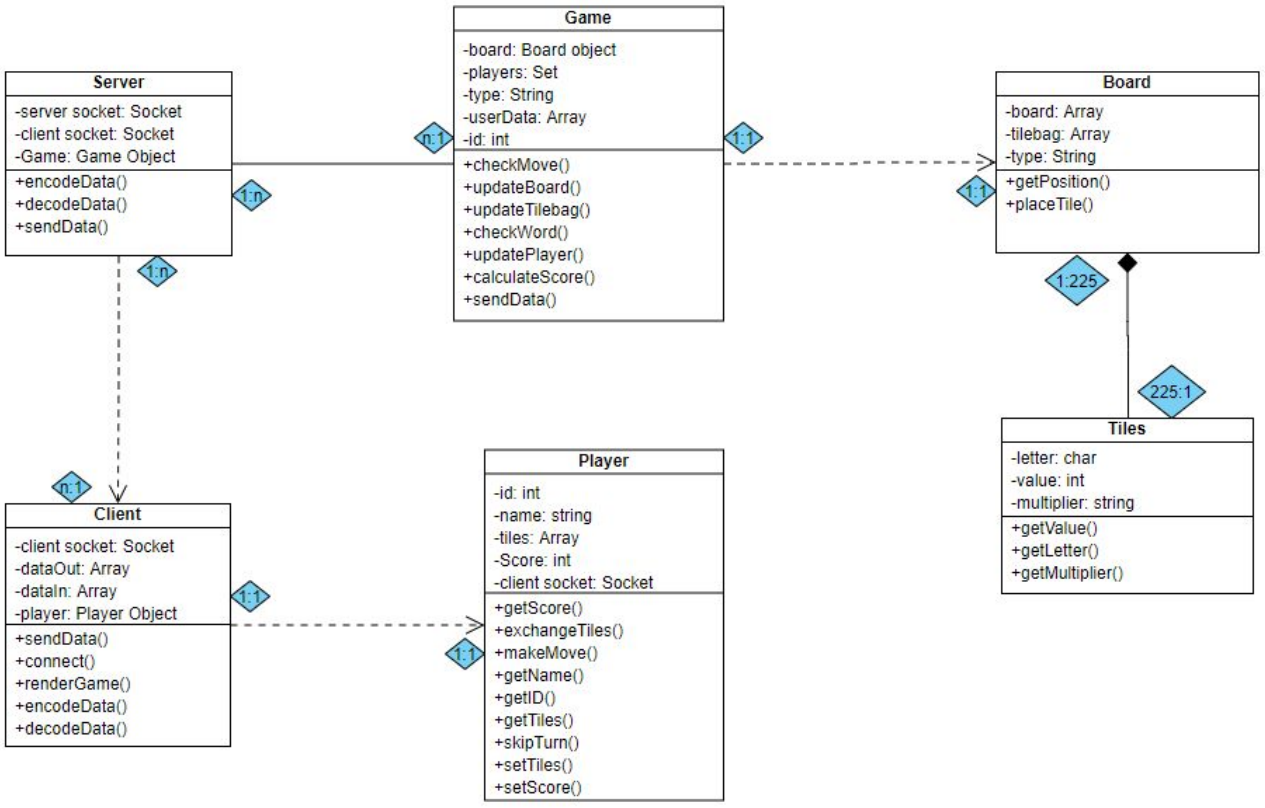
Class Name: Board	**ID:** 5	**Type:** Object
Description: This class represents the board of the game. It is responsible for getting the position of tiles on the board and placing tiles.		**Associated Use Cases:** - 1, 2, 3, 4, 5

Responsibilities	**Collaborators**
Get position: The board needs to keep track of what position letter tiles are placed on. The board can therefore retrieve the position of any tile.	Game
Place Tile: The tile a player uses needs to be placed on the board.	Game Tiles

Class Name: Tile	**ID:** 6	**Type:** Object
Description: The tile class represents the tiles of letters used in the game. Every tile object has a letter, a value and a multiplier.		**Associated Use Cases:** - 1, 2, 3, 4, 5

Responsibilities	**Collaborators**
Get Value: This returns the value associated with a tile	
Get Letter: Returns the letter that the tile represents.	
Get Multiplier: Tiles will either have a multiplier or none at all. This returns that value.	

Class Diagrams



Use Cases

Use Case 1	Creating a game
Goal in Context	Player creates a game
Scope & Level	HTTPS, Primary Task
Preconditions	Player accessed website
Success End Condition	Game is created
Failed End Condition	Game is not created
Primary, **Secondary Actors**	Player Client, Server, Game, Board, Tile
Trigger	Player presses “Create Game” button
DESCRIPTION	**Action**
1	Player enters their unique “in game name” in the “input name” text field
2	Player clicks “create game” button
3	Client checks the “input name” text field is filled
4	Client sends input name of the Player and game creation request to server
5	Server checks that there is capacity for a new game (# sockets)
6	Server instantiates an instance of Game
7	Server sends game state to Client
8	Server generates a unique code for sharing the game
9	Server returns “invite code” to Client
10	Client displays empty scrabble board to the Player
11	Client displays “invite code” in an overlay to Player until Player 2 joins
12	Server waits for Player 2 to join
EXTENSIONS	**Branching Action**
3a	Name is not provided: error message appears asking for a name to be provided
5a	There are not sufficient server resources to facilitate the creation of another game. It is declined and “all rooms full” error is shown under the create button.
VARIATIONS	

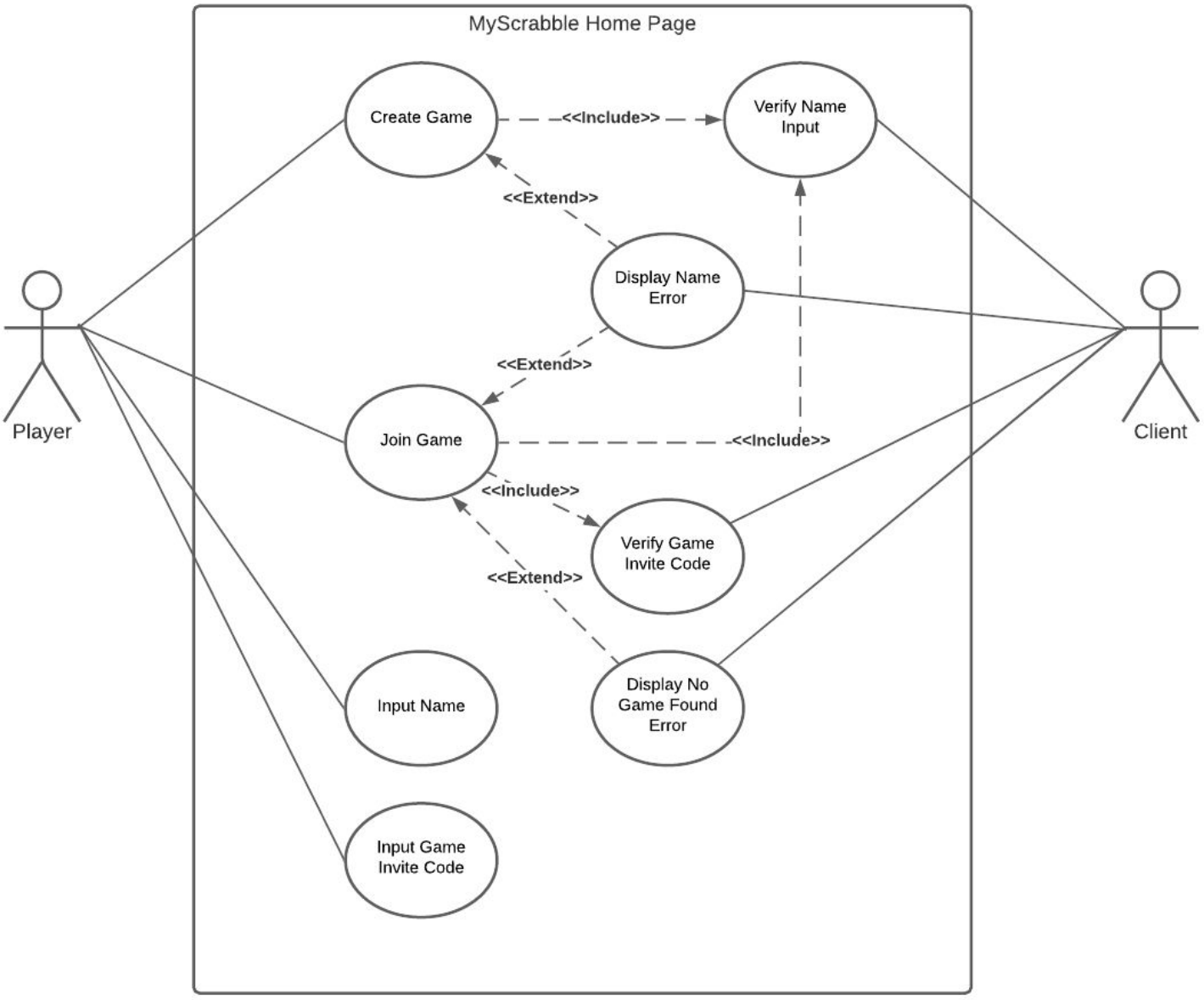
Use Case 2	Join a game
Goal in Context	Player joins a game
Scope & Level	HTTPS, Primary Task
Preconditions	Player accessed website
Success End Condition	Player joins a game
Failed End Condition	Player does not join the game
Primary, **Secondary Actors**	Player Client, Server, Board, Game, Tile
Trigger	Player presses “Join Game” button
DESCRIPTION	**Action**
1	Player enters their “in game name” in the “input name” text field
2	Player enters a “game code” in the “input code” text field
3	Player clicks “Join Game” button
4	Client checks the “input name” text field is filled
5	Client sends input name of the Player and invite code to server
6	Server checks name and if invite code is valid and sends a copy of the state of the board to the client
7	Client displays board to Player
EXTENSIONS	
4a	Name is not provided: error message appears asking for a name to be provided
6a	Name is not unique: Server makes name unique by appending number to it
VARIATIONS	

Use Case 3	Player makes a move
Goal in Context	Player makes a move
Scope & Level	HTTPS, Primary
Preconditions	Game started
Success End Condition	Player makes a move
Failed End Condition	Player makes invalid move
Primary, **Secondary Actors**	Player Client, Server, Game, Board, Tile
Trigger	Previous player's turn has ended
DESCRIPTION	**Action**
1	Server sends updated board configuration to Client
2	Client displays the board to the Player
3	Player places his tiles/word on the board
4	Player presses the "submit button"
5	Client sends new board configuration to Server
6	Server passes on user board data to Game
7	Game checks board configuration
8	Game updates board and Player data
9	Server sends approved/updated board state and player data to Client
10	Client displays updated board to the Player
11	Player's turn ends
EXTENSIONS	
6a	Board configuration is illegal: Server asks for a board resubmission
VARIATIONS	
1	Player presses "Skip Turn" and their turn ends.
2	Player presses "Exchange Tiles" and their turn ends.

Use Case 4	User Exchanges Tiles
Goal in Context	Player exchanges the tiles in their hand
Scope & Level	HTTPS, Primary
Preconditions	Game started. It is the player's turn.
Success End Condition	Player's requested tiles are exchanged
Failed End Condition	Player's tiles are exchanged incorrectly or not at all.
Primary, **Secondary Actors**	Player Client, Server, Board, Game, Tile
Trigger	Player presses "Exchange Tiles" button
DESCRIPTION	**Action**
1	Player clicks "Exchange Tiles" button
2	Client displays "tile exchange" popup window.
3	Player selects which tiles they wish to exchange
4	Player presses the exchange button.
5	Client sends the tile exchange request to the Server
6	Server passes the exchange request to Game which handles the transaction and update of the player tiles.
7	Server sends updated player tile data back to the Client
8	Client displays the new tiles to the Player
9	Player's turn ends
EXTENSIONS	
4a	Player tried to exchange an amount of tiles greater than the amount currently in the Bag: Server only exchanges an an amount of tiles that is equal to the content of the Bag
VARIATIONS	
1	There are no tiles left in the Board's tilebag so the player is unable to press the exchange tiles button. When the mouse is hovered over the bag it displays a message telling the player that the bag is empty.

Use Case 5	End of Game
Goal in Context	The game ends
Scope & Level	HTTPS, Primary
Preconditions	Player made a turn which exhausted his hand of Tiles. The Bag has no tiles left in it.
Success End Condition	Game ends and a winner is selected
Failed End Condition	Game does not end
Primary, **Secondary Actors**	Player Client, Server, Game, Board, Tile
Trigger	Player just played a turn that exhausted his hand.
DESCRIPTION	**Action**
1	Game calculates final Player points
2	Server sends winning information to Client
3	Client displays that the game is over and the winning information
EXTENSIONS	
	N/A
VARIATIONS	
	No valid move has been made by either player for 6 consecutive turns
	A player disconnects for longer than 1 minute
	Player presses forfeit button

Use Case Diagram



Result of “Structured walk-through”

Step	Player	Use Cases	Use case ID
0	Ren	Create Game	1
0	Stimpy	Join Game	2
1	Ren	Make Move	3
2	Stimpy	Make Move	3
3	Ren	Make Move	3
4	Stimpy	Exchange Tiles	4
5	Ren	Skip Move	4
6	Stimpy	Make Move	3
7	Ren	Exchange Tiles	3
8	Stimpy	Make Move	3
9	Ren	Make Move	3
10	Stimpy	Skip Move	3
11	Ren	Skip Move	3
12	Stimpy	Skip Move	3
13	Ren	Skip Move	3
14	Stimpy	Skip Move	3
15	Ren	Skip Move	3
16	Server	End Game	5

After going through this walk-through we have discovered these missing functionalities:

- We need to add a “Concede” functionality.
- Disconnect Handling

Minutes/Notes of Team Meetings

All members were present during our meetings.

Meeting 01 - 13/10/2020

Sprint #1 Meeting (First Overall Meeting)

- First meeting was an overall collaborative review of the document and we began to work through the key points and items that needed to be addressed in Section 1.0. Each member had already familiarised themselves with the ASD. This was the first stage in the Analysis Phase. We also read through and discuss Section 3.1 of the ASD.

- We then proceeded to discuss and organise how sprints will be done at a high level and 'set up our workflow' as such. Assigning tasks, Sprint planning, outline plan / procedure for daily scrum meetings.

- We assigned each team member to write up their own user stories and try to refine the base specification independently. This was with the aim to come together in the next meeting with a rounded understanding of the overarching system story, as well as a variety of refined requirements from different perspectives. These would all then be discussed by the group in order to be merged.

Meeting 02 - 20/10/2020

Sprint #2 Meeting

- This meeting began with the group sharing their independently written user stories and refinement of the specification.

- The team then came up with primary classes together.

- After merging our user stories, we then split up the subsequent tasks of working on and designing the various use cases and class implementations. With the sprint methodology in mind, we had short daily “stand ups” or text updates to keep on track and iteratively collaborate on the creation of these.

Meeting 03 - 27/10/2020

Sprint #3 Meeting

- Meeting 3 consisted of an overall review of our work on designing use cases and classes, finalizing the specifics of each with the aim of merging them into the structured document at this phase.

- We then went through a structured walk-through of the system as a group, in order to test our use case implementations and overall development thus far. This helped us to identify possible defects, inconsistencies or accidental omissions.

- The primary action items resulting from this meeting were to finalize all remaining aspects of the analysis (committing them to this document) and prepare for submission.