SYSC 3110 Milestone 1 Documentation

Group 17:
Oyindamola Taiwo-Olupeka - 101155729
Oluwatomisin Ajayi - 101189490
Edidiong Okon - 101204818
Ese Iyamu - 101081699

TABLE OF CONTENT

Section	**Content**	**Completed By**
A	Design Decisions	Oyindamola Taiwo-Olupeka
B	UML diagrams	Oluwatomisin Ajayi
C	User Manual	Ese Iyamu
D	ReadMe	Edidiong Okon

A. Design Decisions

For the ScrabbleGame project, we collectively decided on six(6) classes- ScrabbleGame, ScrabbleBoard, Square, Player, Tile and TileRack. This is subject to change as the project progresses.
In a traditional Scrabble game, 2-4 players randomly choose 7 tiles to place on their tile rack. During their turn, they pick a combination of tiles and play on the squares on the board.

Breakdown of classes:

1. **ScrabbleGame:**
   This class initializes the scrabble game. It generally controls the entire flow of the game by distributing 7 random Tiles to each player, verifying the legality of moves and holding all important information. It keeps track of the 2-4 players (specifically, the order they play in, their score, and the start and end of their turns). And lastly, it determines the winner at the end of the game by comparing all the players' scores.
   The play method asks for the number of players and then enters a loop that plays words (horizontally or vertically) on the specified squares. The game loops through the players until a player has no more tiles, and the game is over. ScrabbleGame keeps track of each player's score and determines the winner by whose score is the highest.
   This class also contains the main method to play the game.

2. **ScrabbleBoard:**
   This class initializes a scrabble board which will be made up of a 15x15 array of Square type because a scrabble board is made up of 225 squares. (this is a class we also made to avoid repeating multiple lines of code. Also uses the ScrabbleWords.txt file which serves as a dictionary of legal words in the Scrabble game.

3. **Square:**
   This class will work with the ScrabbleBoard class. It initializes a 2D array to store the position of the square on the board. The position is found by row number and column number, both variables are read as parameters in the constructor method.

4. **Player:**
   This class represents the players in the ScrabbleGame. It holds the name, score, and their Tile racks. It is a dependent class.

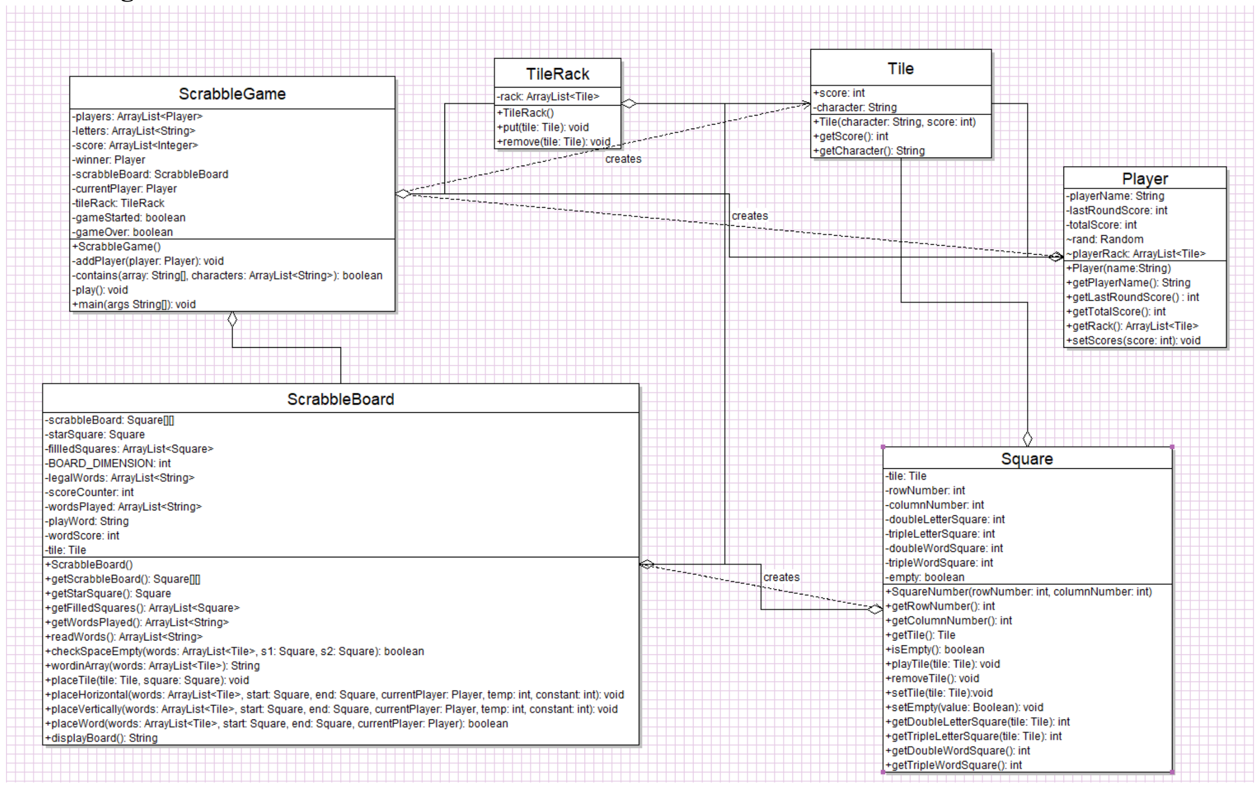
5. **Tile:**
   This class represents the Tiles. Each tile has a character which we declared as a string and a value, which is the score associated with the tile. Both variables are read as parameters in the constructor method. The character range is letters A-Z in the English alphabet and blank tiles which will be implemented in future milestones.

6. **TileRack:**
   This class represents each player's tile rack. It is initialized as an ArrayList of type Tile with a maximum number of 7 tiles at a time.

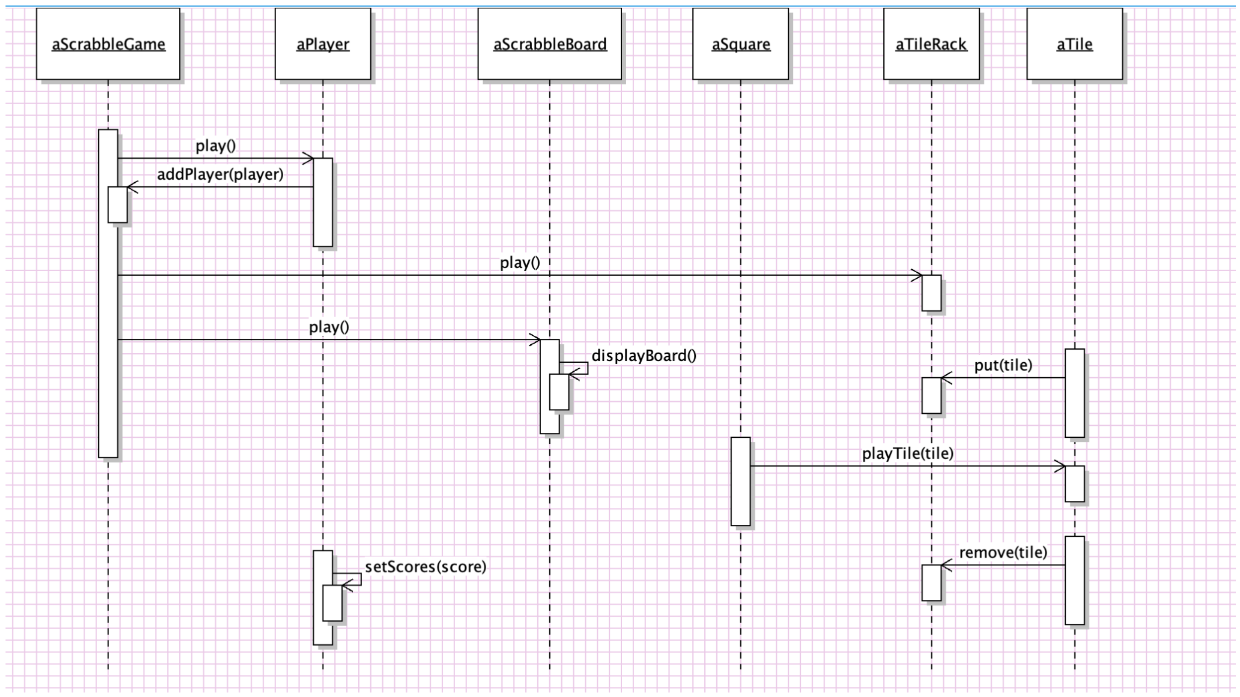
To conclude, the six classes all have individual responsibilities but they work together to make ScrabbleGame functional. All classes are subject to change at each milestone.

B. UML Diagrams

Class Diagram


ScrabbleGame
-players: ArrayList<Player>
-letters: ArrayList<String>
-score: ArrayList<Integer>
-winner: Player
-scrabbleBoard: ScrabbleBoard
-currentPlayer: Player
-tileRack: TileRack
-gameStarted: boolean
-gameOver: boolean
+ScrabbleGame()
-addPlayer(player: Player): void
-contains(array: String[], characters: ArrayList<String>): boolean
-play(): void
+main(args String[]): void
TileRack
-rack: ArrayList<Tile>
+TileRack()
+put(tile: Tile): void
+remove(tile: Tile): void
creates
Tile
+score: int
-character: String
+Tile(character: String, score: int)
+getScore(): int
+getCharacter(): String
creates
Player
-playerName: String
-lastRoundScore: int
-totalScore: int
~rand: Random
~playerRack: ArrayList<Tile>
+Player(name:String)
+getPlayerName(): String
+getLastRoundScore() : int
+getTotalScore(): int
+getRack(): ArrayList<Tile>
+setScores(score: int): void
ScrabbleBoard
-scrabbleBoard: Square[][]
-starSquare: Square
-fillledSquares: ArrayList<Square>
-BOARD_DIMENSION: int
-legalWords: ArrayList<String>
-scoreCounter: int
-wordsPlayed: ArrayList<String>
-playWord: String
-wordScore: int
-tile: Tile
+ScrabbleBoard()
+getScrabbleBoard(): Square[][]
+getStarSquare(): Square
+getFilledSquares(): ArrayList<Square>
+getWordsPlayed(): ArrayList<String>
+readWords(): ArrayList<String>
+checkSpaceEmpty(words: ArrayList<Tile>, s1: Square, s2: Square): boolean
+wordinArray(words: ArrayList<Tile>): String
+placeTile(tile: Tile, square: Square): void
+placeHorizontal(words: ArrayList<Tile>, start: Square, end: Square, currentPlayer: Player, temp: int, constant: int): void
+placeVertically(words: ArrayList<Tile>, start: Square, end: Square, currentPlayer: Player, temp: int, constant: int): void
+placeWord(words: ArrayList<Tile>, start: Square, end: Square, currentPlayer: Player): boolean
+displayBoard(): String
creates
Square
-tile: Tile
-rowNumber: int
-columnNumber: int
-doubleLetterSquare: int
-tripleLetterSquare: int
-doubleWordSquare: int
-tripleWordSquare: int
-empty: boolean
+SquareNumber(rowNumber: int, columnNumber: int)
+getRowNumber(): int
+getColumnNumber(): int
+getTile(): Tile
+isEmpty(): boolean
+playTile(tile: Tile): void
+removeTile(): void
+setTile(tile: Tile):void
+setEmpty(value: Boolean): void
+getDoubleLetterSquare(tile: Tile): int
+getTripleLetterSquare(tile: Tile): int
+getDoubleWordSquare(): int
+getTripleWordSquare(): int

Sequence Diagram



- Games loads.
- Players are added to the game.
- Tile rack is created and a scrabble board is displayed.
- Players start playing. Once a tile is played, it is removed from the rack.
- A winner emerges based on the final score.
- Game ends.

C. User Manual

To Play the board game Scrabble:

- The first player combines two or more letters from their rack to form a word on the board. (player can place the word to read either horizontally or vertically but not diagonally).
- Player's score is calculated and given back as many tiles they used to play to replace their rack.
- Turn is passed to the player on the left. Players can play or pass their turn.
- Players can form new words by adding one or more letters to the words on the board. Note: all new words(horizontally or vertically) formed by the auditions of new tiles must be comprehensible. Tiles can’t be moved around once they’ve been played and scored.
- Up to two blank tiles can be used. Player would have to state what letters the blank represents and it remains that letter till the end of the game.
- Players can use their turn to exchange some or all of their tiles.
- Game ends when all letters have been drawn, players use up all their tiles or all possible moves have been made.

To Play the board game scrabble by keyboard (For Milestone 1):
The game would be played via keyboard and would start from player 1 and go in the order the players were added. When a player gets their turn, they type in the words they want to play and the location on the board.

D. ReadMe

Scrabble
A simplified version of the classic word game, Scrabble, built using Java and java.util packages.

Milestone 1
A text-based version of Scrabble where players use a keyboard to play the game via a console.

Authors
Ese Iyamu
- Created the Player class.
- Worked on the user manual aspect of documentation.
- Added Java Doc comments to classes and methods.

Oyindamola Taiwo-Olupeka
- Worked on the ScrabbleBoard and Square classes.
- Wrote the design decision document.
- Added Java Doc comments to classes and methods.

Oluwatomisin Ajayi
- Worked on the ScrabbleGame class.
- Created UML class and sequence diagrams.
- Added Java Doc comments to classes and methods.

Edidiong Okon
- Worked on the Tile and TileRack classes.
- Wrote the README file.
- Added Java Doc comments to classes and methods.

Known Issues
- The passTurn method in the ScrabbleGame class is not completely implemented.
- The displayBoard in the ScrabbleBoard class is also not fully implemented.

Future Milestones and Deliverables
Milestone 2
- GUI-based version of the Scrabble game.
- Add a GUI to the current text-based version of the game using GameView and GameController classes.
- Create unit tests for the Model.
- Update all UML Diagrams and documentation.

Milestone 3
- Add new features and support for blank tiles, premium squares and AI players.
- Add a method that returns all the possible legal moves and plays the highest scoring move.

- Refine the program to make it robust and smell-free.
- Update UML Diagrams and documentation accordingly.

Milestone 4
- Add a multiple level undo and redo feature.
- Implement Java Serialization and utilize it to add a save and load feature.
- Create custom boards with alternate placement of premium squares using XML or JSON.
- Update documentation.

Change Log
- Not Applicable.

Roadmap
November 11, 2022
- GUI (View and Controller).
- Unit Testing for the Model.

November 21, 2022
- Creation of blank tiles, premium squares and AI players.
- Design of Scrabble move generation algorithm/method.

December 5, 2022
- Addition of multiple game level undo and redo features.
- Save and load features.
- Creation of custom boards.