Team: cse-t047
Jason(Jaeheun) Kim 917-575-9132
Mike 716-901-2460
Tyler D 716-359-3795
Chris Elman 585-455-9219
Shokoor 347-876-2105

CSE 116 Technical Document

Design (for Stage 1 and Stage 2)

The design of our project is as follows: In order to start the game, a class called game is instantiated. This class is responsible for instantiating all other classes. The game class instantiates an instance of the board class, which mainly functions as an array to hold tiles. The game class instantiates turn handler, which is responsible for keeping track of turns, but also taking as input the number of players that will be playing and creating the specified number of player objects. When the player class is instantiated, a corresponding tile rack is also created, which holds the players tiles. In the constructor, of the tile rack, it draws tiles from the already existing inventory. Many of the classes have been given accessor and mutator methods in order to create tests and for better communication between classes. Each class also has its own set of unique methods other than the constructor, accessor, and mutator methods. For example, the TurnHandler class has a method called switchPlayer, and the Board class has addTile, getTile, and removeTile methods.

In order to interact with our program, we implemented a model-view-controller design for our graphic user interface. Editing the graphic representation of our program in turn communicates and executes the corresponding changes on our program's underlying model. Our graphic user interface displays a window for the main playing board and a window for each participating player's tile rack. Tiles are graphically represented by JButtons, as this also allows for a means of interacting with the tiles. By clicking on the JButtons in a player's tile rack followed by a JButton on the board, a player can put tiles on the board and spell words.

Design (for Stage 3)

In order to run the game over RMI, the Server class must be instantiated by the ServerDriver class. Once the Server class is running and operational, Clients can connect to the Server by running their ClientDriver classes. The ClienDriver class instantiates the Game class for each Client, and this creates the GUI for each client to interact with. When a user passes his or her turn, the Client passes a String to the Server

class which enables the Server to update the state of the game to reflect the changes that occurred on a Client's turn.

Also, we have added multipliers at every space in the board. The letter multipliers and the word multipliers are stored in two separate private int arrays in the Board class. At the beginning of the game one letter multiplier and one word multiplier are distributed to each space on the board. This is also represented in the GUI by two numbers on each TileSpace, the one on the left representing the letter multiplier and the one on the right representing the word multiplier.

Testing

Our tests were broken down into classes in the package labeled tests. The classes for the tests correspond to the classes in our package called code.

-The BoardTest class tests the functionality of the board, namely adding and removing objects of type tile.
-The InventoryTest Class verifies that the inventory holds the correct number of tiles from the start.
-The PlayerTest Class verifies that points are correctly added to the player's instance variable.
-The TileRackTest class verifies that a newly instantiated tile rack draws the correct number of tiles from the inventory.
-The TileTest class verifies that a tile has the same value and letter that we assign it in its constructor.
-The TurnHandlers class verifies that the turn changes when switchPlayer is called.
-Tests were added for Inventory
-Tests were added for Pass Button functionality in Extravaganza class
-Tests were added for WordChecker
-Tests were added for placeTile
-Tests were added for ReaderTool
-Tests were added for checking a Player's score

Unimplemented

Unfortunately, we were not able to fully implement all functionality for Stage 2 and Stage 3. Functionality that is missing or broken in our Scrabble program include:

-Othello scoring does not function as intended when a word wraps around the edges of the board. This is because when a word wraps around the edges, it is not recognized as an accurate word (assuming that it is in fact spelled correctly).
-Passing in the text name for the dictionary file does not work.
-You are not able to successfully pass turns over RMI connection