Technical Guide Stage 2

Team 046

**Pre-start**: Before a game can begin a Game object and at least two Player objects must be created and the players must register with the Game. To do this a person would have to instantiate the Driver class with arguments that include the player names, each preceded by the tag “-p” and the exact location of the dictionary file on the computer which is preceded by the tag “-d”. These arguments are passed in as command lines when used in a terminal. The Main method uses these arguments to create Player objects to go with the names and register them with the Game object the method creates. The dictionary file name is used to create the dictionary arraylist in the Board class created upon the calling of the start method on the Game object. The Game object also creates the tile inventory in its constructor, and it has been tested to be the correct size, be in a random order, and hold the correct number of each tile. The game will not start if less than two players have registered, and will not start if the game has been started already, this criteria has been tested and proved to be accurate. All of this is done within the Main method of the Main class.

**Post-start**: Once the game has been started successfully, the Game object will create a new Board object and, for each player, a new GUI(Graphical User Interface) object, both of which hold an association relationship with the Game object that created them. The players are placed into a list and randomized into the order in which they will play for the rest of the game, while acknowledging the first player in the list as the current player. This action unlocks the GUI for that player and allows them to interact with the board through their user interface.

The creation of a new Board object will also accomplish various tasks. Firstly, it reads the dictionary text file within the project and places each word into an arraylist to allow the Bord to easily check if a word formed is valid later on. The dictionary has been tested and the method only returns true if the word is found within it, with no exception. A 20X20 double array of Square objects is created to act as the actual board for gameplay, and the upper right and lower left corners of the array are set to null. The sizes of the null corners are determined by two randomly selected integers between and including 3 to 7 which are also determined within the constructor of the Board object. The board shape has been tested to be the same size every time, and the corners are indeed randomly assigned null upon each creation and the board is still square every time as well. After the array is properly created, a method is called to select a random square in the array to act as the HomeSquare. If the randomly selected square is found in one of the null corners, then the method will repeat until it is a valid square. The location of the HomeSquare can easily identified in the user interface as visually distinct from the rest of the squares at the start of the game. The first tile to be placed must be on this HomeSquare. The formation and location of the HomeSquare has been tested to be accurate and random every time.

The GUI object creates a separate user interface for each player object. Each of them display a representation of the current status of the board, such as the location of the HomeSquare and all of the letter tiles that have been placed on the board. The tile rack of the player is also displayed along the bottom of the window next to a button used to end the player’s turn. A text screen is found next to the board which informs the player when it is their turn to play, and displays any changes to a player’s score that may occur at the end of a turn. A player can only interact with the GUI if it is their turn, otherwise the window is locked and will not react to any action from the mouse in its window. The text screen will also show the winner of the game once it ends.

**Gameplay-Values**: Each Player object holds all of the unique information for that player. Including their name String, their score integer, and their tile racks which draw their tiles from the main inventory from the Game object that they are registered with. Square objects also hold unique information, most importantly of which is the letter tile which they may or not hold. If no tile was placed yet then the reference is null, but placing a tile in a Square will allow the Square to keep track of its Tile. The Square objects also generate letter and word multipliers, which are selected based on the weights provided in the description, with a multiplier of 1 being the most common for both, 2,0, and -1 being options for the letter multiplier and 3,2, and 0 being possible for the word multiplier. These are meant to be taken into account when the word is ultimately scored at the end of a turn. Tile objects are the simplest objects created in the project and just hold references to the letter string that they represent, and the value that is paired with that letter. Once placed, the tile also holds integers which represent the row and column of the double array in which they were placed. The Board object keeps track of the HomeSquare, the row and column of the last placed tile, the valid word dictionary, an arraylist of words formed in the last turn, an arraylist of all words formed in the game, and an arraylist of all tiles placed in the game. The Game object holds references to the remaining tile inventory, the number of turns that have been skipped, what turn number it is, the arraylist of words formed in the last turn, and the list of registered players. All values were tested to be present within the appropriate objects, and checked to see if they can be changed if that functionality is relevant.

**Gameplay**: Players can use the their GUI during their turn to select tiles from their rack and place them on the board, starting with the HomeSquare. All placements must be in the same row or column, and must form a word found within the dictionary by the end of the player’s turn. The action of placing tiles on the board has been tested to only succeed if the desired Square is not in one of the null corners, and does not already contain a tile. Retraction of a tile only works if the attempt is made on the same turn that the tile was placed. Once a player is satisfied with their efforts then they may attempt to end their turn by clicking the button on their GUI. This will only work if the word they formed is valid according to the dictionary, or they placed no tiles at all, which means they decided to skip their turn. The number of skipped turns is recorded by an integer within the Game object, and if all players skip in a row then the game ends. The endTurn criteria has been tested to be accurate, and the game does end if all players skip their turn. All words formed by the player are scored and the subsequent number is added to the player’s total score, and their tile rack is replenished with new tiles from the tile inventory as long as there are tiles remaining in it. The scoring of the words has been tested to be accurate and the tile racks have proven to refresh themselves at the end of their respective player’s turn. It is possible, and has been tested to have the ability to return a list of all words played in the last turn, but this functionality does not seem to be used at any other point in the code. A string representation of the board can also be accessed at any time with a method, but again this only seems to help with testing.

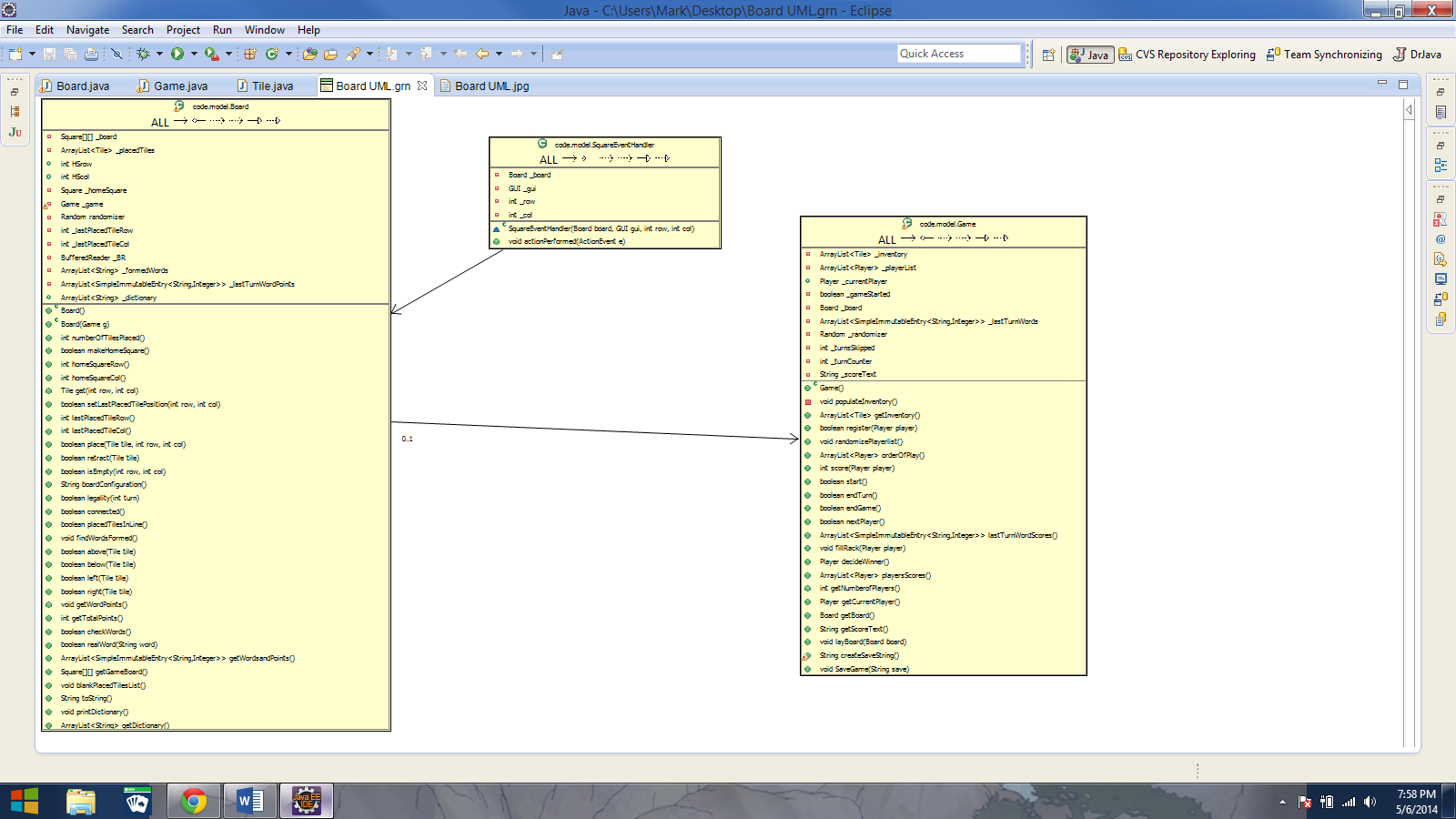
**End Turn Specifics**: The ending of the turn is the most complicated part of the code, but it is actually logically simple. The Board object holds an arraylist of the placed tiles during a turn, and it is there that the Game starts to check for the new words on the board. It goes to each tile placed in that turn and checks the four adjacent tiles. If there is a tile in one of these positions then the program goes all the way down the length of the placed tiles until the end of the word. It then goes back down the length of the tiles while forming a new string from the stored strings in the tiles. Once it reaches the end of the stretch of tiles it adds the word to a list of new words. It then repeats this process for the next placed tile. This lets it track not just the main word formed, but also any words that branch off from the main word. This results in many attempts to try and add the same word to the list, but duplicates are not permitted. Once all of the tiles are checked the word list is looped through and each word is scored according to the point values held by the Tile objects and the multipliers in the Square objects . This value is then added to the appropriate player. The gameplay has been tested, and does result in the correct points, and the dictionary has also proven to be incorporated correctly since the turn will not end if any of the words formed are not in the dictionary. The result of all this work is a turn that will only end is the words formed are valid, or no words were placed, and the correctly multiplied score is given.

**Save Game**: If a player decides to save the game then the Board and Game objects work together to form a large single string which holds the source of the dictionary, the names, scores, and tile racks of the players, and a representation of the board which includes the letters placed on the board, and if the square is empty then it is shown as an underscore character with its multipliers following it. Null squares are shown as a #.

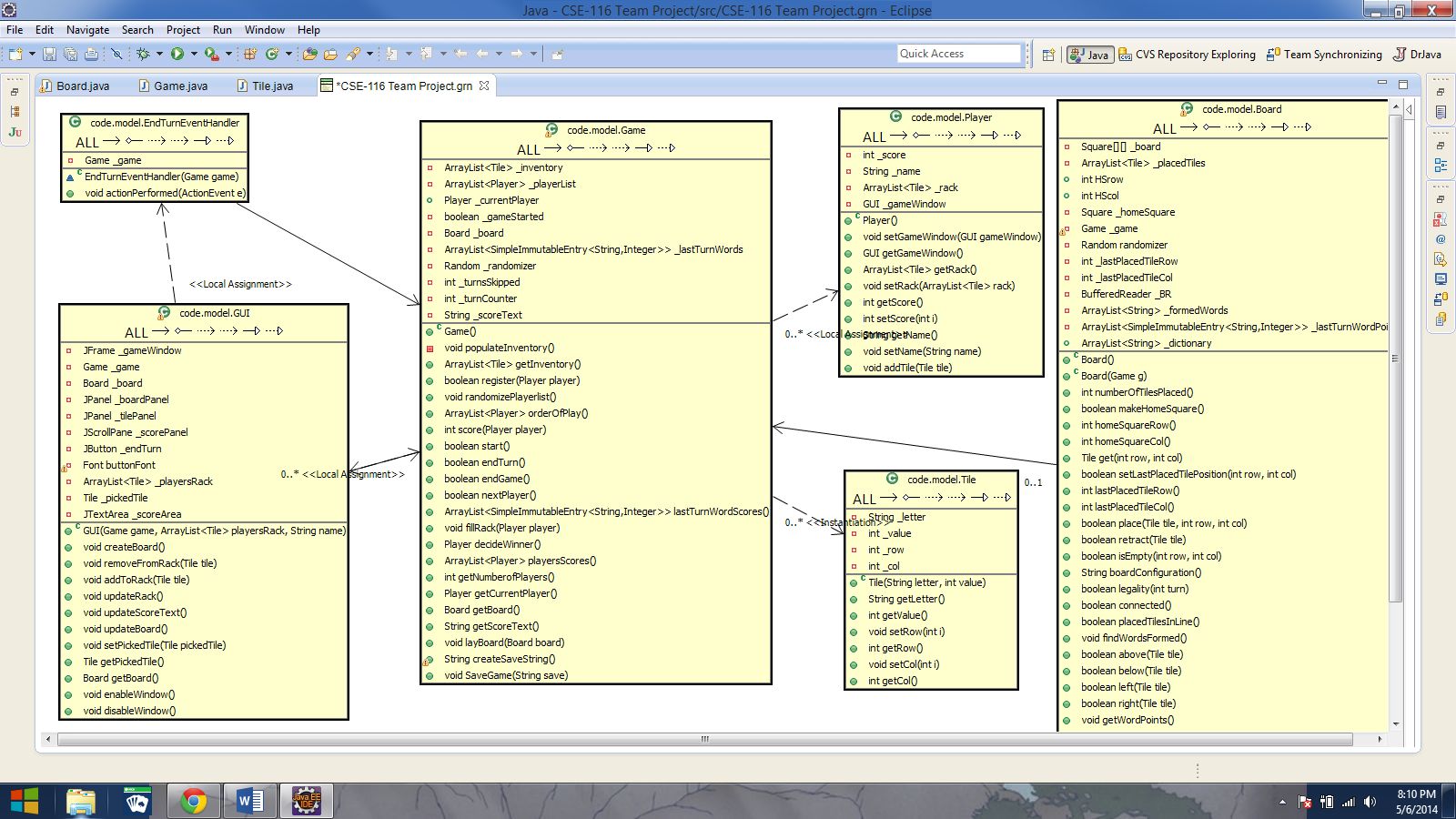
**Test Specifics**: Dozens of tests exist to test various methods and functionality of both stage 1 and stage 2. Each method possesses at least three tests to check a few different scenarios. Some of the functionality causes aspects of the board to be random, such as the tile rack, HomeSquare location, and the null corners of the board. In these cases the tests check to see these methods are truly different upon each attempt. For example, two boards would be created to see if their corners are different sizes, as they should be. This means that some tests will fail since testing randomness will sometimes lead to the random values to be the same. The two boards for example may be the same size on occasion, but all tests pass the majority of the time. System.out.Prinln methods were often utilized to make sure that many of the methods operate correctly. This proved very useful for testing the board configuration and the checking of the board for newly placed words. This testing helped lead to the formation of a “Scrabble”-like game that operates exactly as it should during gameplay.

**UML Diagrams**: Included are a few UML diagrams to help show the interactions between the classes. Each one is based around a major class within the project. They will include diagrams for the Board, Game, GUI, Square, and Tile classes. The files will be found within the eclipse project, but pictures will also be added to this guide in case those files do not work properly.

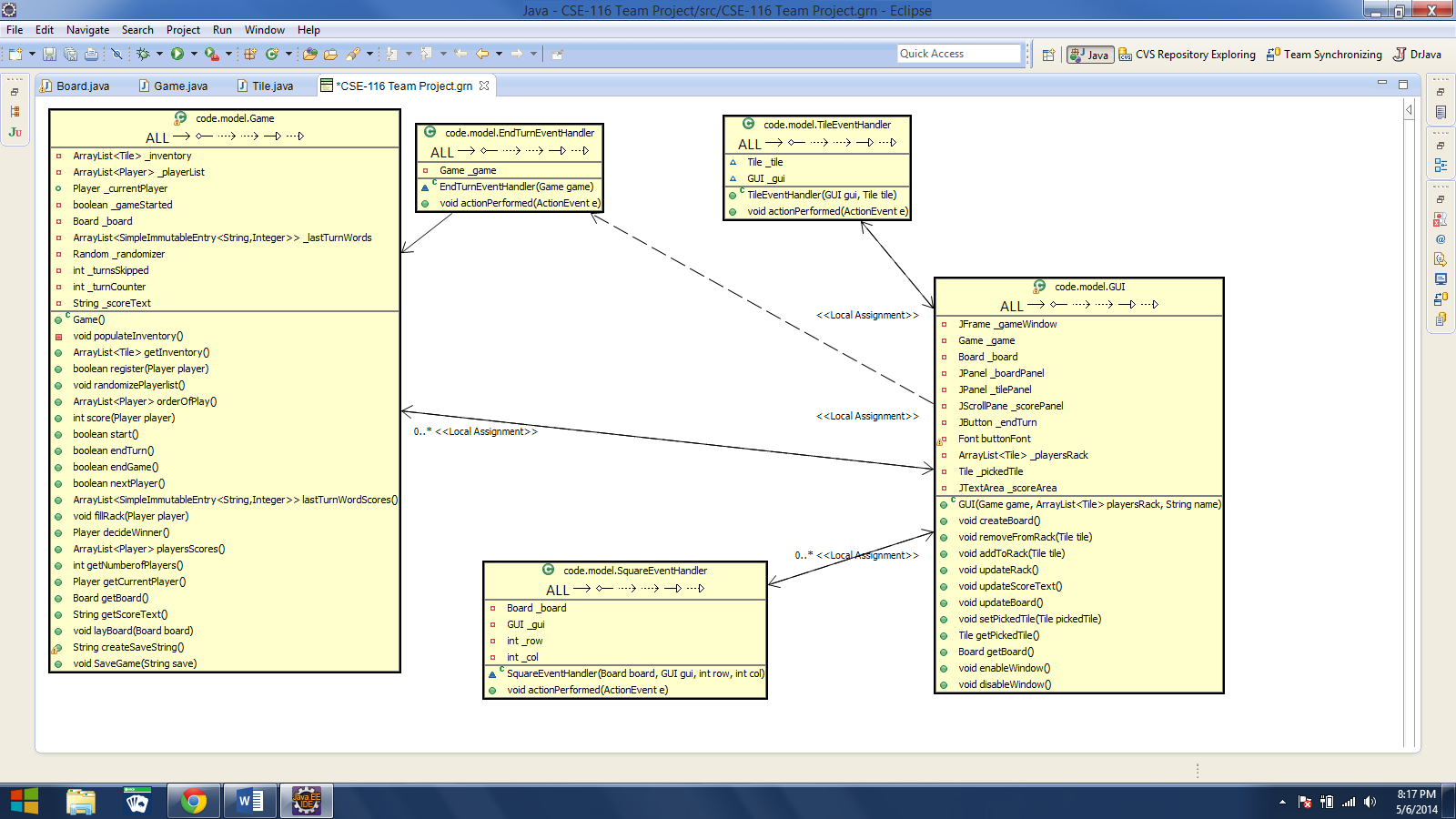
Board UML- The Board class only directly interacts with the Game object that created it and the SquareEventHandler since the Board has an Array composed of Squares and the user interface must interact with them.



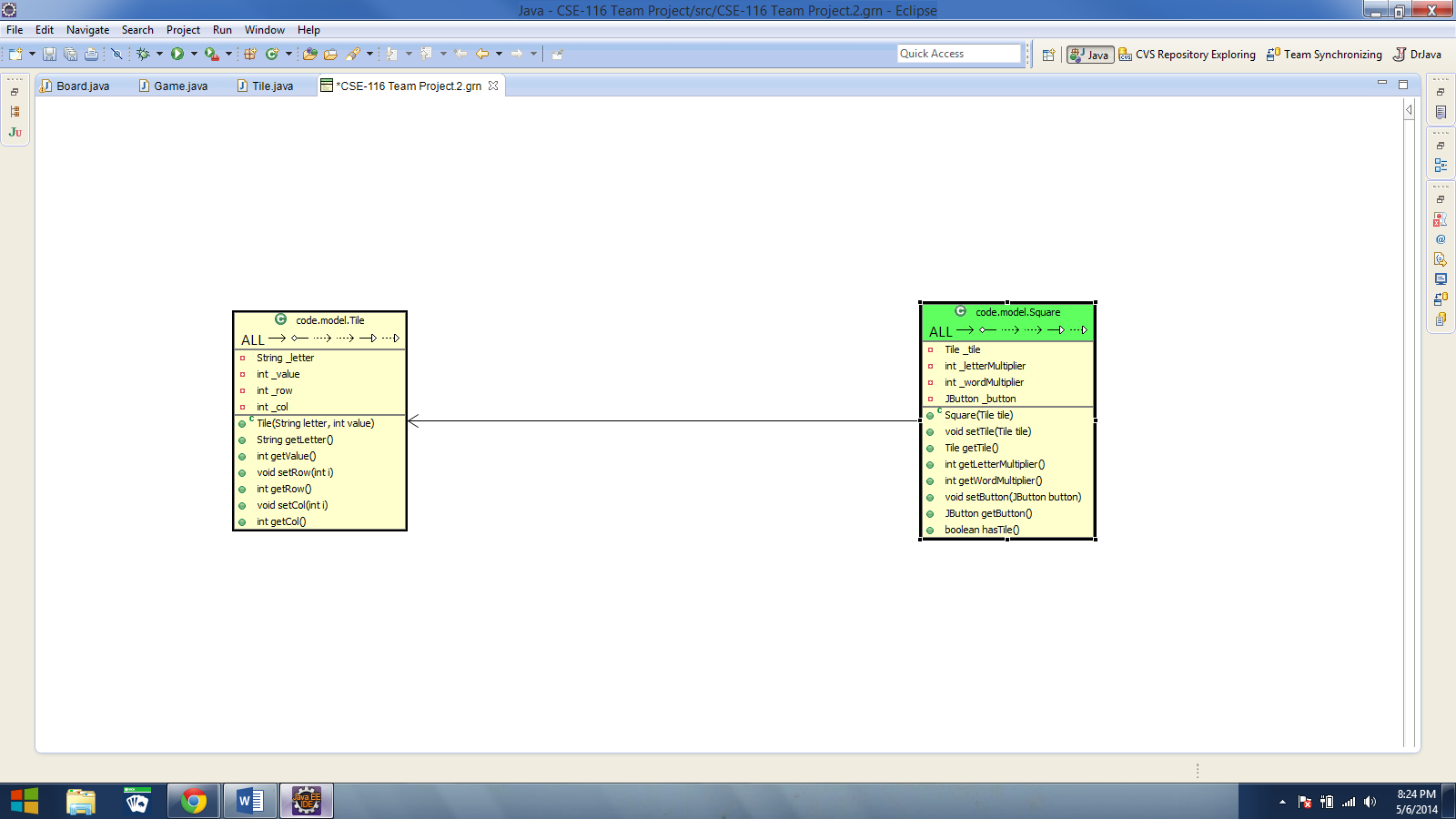
Game UML-A Game object interacts with more objects than any other object in this project. It is connected with every major part of the model with the exception of the Square. This allows it to access the game board, the players, and the user interface with ease so the Game can control the rules, start behavior, and end behavior of the entire game. Since ending a turn is a complicated process, the Game also interacts with an EndTurnEventHandler for the user interface for the button which actually ends the turn. The Game must access many of the values and methods stored within most of the other classes, which is why it holds relationships with so many of the classes in the project.



GUI UML- Despite being such an important part of the project, the GUI object is only connected to the event handlers and the Game object which creates it upon the start pf the game. The event handlers must be connected since they control the behavior of the buttons in the GUI, while the Game object holds references to any and all of the information that needs to go into the user interface since the Game is connected to so much of the project. The result is a GUI that only has one non-event handler, yet can still display all of the information present in the rest of the objects.



Square UML- Square objects are only associated with the Tile objects that the Squares hold. All Squares hold references to a tile upon creation, but at the start of a game this reference would be null which is why the two are not in a composition relationship. This is all that the Square is required to be connected to since the Board just creates an Array of Squares and does not interact directly with the Square class



Tile UML- Tile objects are connected to their event handler, the Square object holding them in the board, and the Game object which is controlling the game. The Board object interacts with the Tiles through the Squares which is why it does not have a relationship with the Tile. Since Tiles hold the references to a letter and a score, they are an essential part of the game which causes the Game object to be dependent on the Tiles in order to create, check, and score words.

