



University of Toronto Mississauga

CSC207 Fall 2022

Design Document

## AUTHORS

Raghav Kanda

Ahmed Mohamed

Justin Tran

Christopher Flores



GITHUB TEAM:  
MAT102Survivors

GITHUB URL:  
<https://github.com/o8RaV/MAT102Survivors.git>

November 12, 2022

# Contents

S.No	Topic	Page Number
1	Title Page	1
2	Project Identification	2
3	User Stories	3
4	Software Design	6
5	Expected Timeline	10

## Project Identification

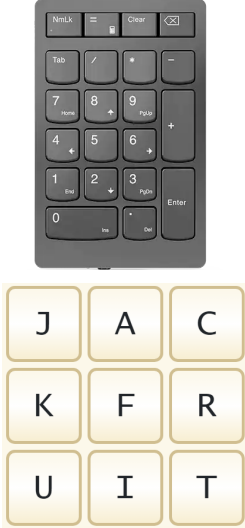
For this project, an implementation for the game of Boggle will be created. This implementation will be built using the code base from Assignment 1. By doing so, various features that are absent in Assignment 1 will be created. These features will give the player a more ideal experience when playing Boggle.

For one, the new Boggle implementation will include an attractive GUI. For the implementation in Assignment 1, the player needs to play using the IDE's console. This is not ideal if they desire a fulfilling experience with the game. After all, most computer games utilize a GUI, which does not require the player to open the code base. Second, the new Boggle implementation will have more features in terms of gameplay. This will make the game more enjoyable for the player since it allows them to customize how they want to play. For example, if the player wants a challenge, then they can select the Hard difficulty before launching the program. Finally, the new implementation will provide additional support for people with disabilities. To contrast, in the current implementation, the visually impaired may not be able to see the letter grid provided in the IDE console that well. So, a GUI and an audio interface will be implemented to exemplify accessibility features, such as assistance for the visually impaired.

The User Stories, Software Design, and Expected Timeline to help achieve these goals will be outlined in this document.

# User Stories

Name	ID	Owner	Description	Implementation	Priority	Effort
Boggle board JavaFX GUI	1.1	Ahmed	As a casual Boggle player, I want to better view my Boggle board. Additionally, I want the ability to interactively select the letters in a boggle board, instead of typing them out.	Using JavaFx, implement a GUI that displays the letters in a boggle board in a grid-like shape. The individual letters will be implemented as JavaFX buttons.	1	3
Saved Boards	1.2	Raghav	As a user, I would like to save a board and start a new one whenever I want. I would also like to come back to that older board and continue playing on that, with my progress saved.	The user will be able to save a board by clicking the button on the UI element then entering the preferred name, this will save the board along with its scores, found words and all_words dictionary and exit the UI. When we start to play the game again, the UI will prompt the user if the player wants to play on a saved board or start a new one. After selecting the option of playing on the saved one, the user will output the saved boards and then the player can choose one of them by typing the name or the number of the board. This will be implemented using the memento design pattern.	1	3
Sizeable Boards	1.3	Raghav	As a user I would also like to play on bigger words, instead of just 4 by 4 and 5 by 5	I will edit the base code from Assignment one to give the user more options to choose the board, also I would change the way the random board is chosen. Instead of choosing from the given pool I will make my own random word generator.	1	2
3x3 Board Text Reader	1.4	Chris	As a user that is visually impaired, I want to be able to utilize a screen reader so that I can play Boggle without having to see the board. With a 3x3 Boggle board, this can be implemented with help from the Numpad. As shown in the pictures on the next page (below the Implementation paragraph), the Numpad of a standard-sized keyboard can represent a 3x3 Boggle board. For instance, with respect to the pictures, the 7 key can represent the letter J, the 8 key can represent the	If the user selects the option to have a 3x3 board text reader, pressing a certain key on the Numpad will trigger an audio message. The program will read out the letter that corresponds to the key pressed. Also, pressing the enter key will trigger an audio message that indicates whether points have been awarded or not. This will be implemented using the MVC design pattern. The pattern will be organized as follows: -The controller will contain the handlers that decide what to do with the Numpad input -The model will interpret what that input means, and the model will send that information to the view. -The view sends that information in the form of audio to the player.	1	3

			<p>letter A, and so on. If the user wants the program to read out a certain letter on the grid, they can press the key on the Numpad that corresponds to it. Then, when the user is ready to type in a word, they will press the spacebar and then proceed to type the word out. They will press enter when they are done. If the user enters a valid word, an audio message will notify them that they have been awarded points. However, if they enter an invalid word, the computer will prompt them to restart.</p>	 <p>(images courtesy of: <a href="#">Lenovo Go Wireless Numeric Keypad   Lenovo CA</a>, <a href="#">Mini Boggle Puzzle II Quiz - By joeydeka (sporcle.com)</a>)</p>		
Tutorial level	1.5	Justin	<p>As a user with difficulties concentrating I may not be able to properly understand the instructions of the game by reading paragraphs of directions. As such, the best way of learning for me would be a hands-on approach. For this, I would like sentences that are not extremely long that help me understand how to play the game with an example boggle board.</p>	<p>Create a method that, when called, begins executing a set of print commands instructing the user how to play. After each one is executed, the user will need to press enter to proceed to the next set of commands. Afterwards, users will interact with a boggle board. The program will only accept specific keyboard inputs which help the user understand how the game works. In doing so the program will be able to continue moving to the next steps until it has finished.</p>	2	2
Letter Highlight	2.1	Ahmed	<p>As a casual Boggle player, I want to clearly see the letters I have selected on the boggle board. Any letter I select should be highlighted, so it's distinguished from the rest of the letters.</p>	<p>Highlight letters with a different color as a user selects them. Using the MVC design pattern, the event handler will directly change the view as the letter selection occurs</p>	2	1
Font Change	2.2	Ahmed	<p>As a user with dyslexia, I want the ability to choose the font of the Boggle board from a range of different fonts.</p>	<p>Present a list of fonts to the user. After the user selects the desired font, every character in the application will change to that font.</p>	2	2
Bigger Font	2.3	Raghav	<p>As a user that uses spectacles, I would like to play the game without using them too. To accomplish that, I would like to be able to change the size of the font on the boggle board.</p>	<p>To implement this, the UI will prompt the user if he/she wants to change the font size, if said yes, the user will be given three size options: <b>default</b> , <b>medium</b> and <b>large</b>. This will change the size on the boggle board UI.</p>	3	2

Difficulty Levels	2.4	Chris	<p>As a user that wants customization in the games I play, I want to be able to change the difficulty of my Boggle game so that I can have the most fulfilling experience possible. The different difficulties in this implementation will be:</p> <p><b>EASY</b> - The game will give hints to the player.</p> <p><b>MEDIUM</b> - The game will not give hints to the player. In addition, there will be a time limit of 3 minutes to find words.</p> <p><b>HARD</b> - The game will not give hints to the player. In addition, there will be a time limit of 2 minutes as well as a deduction of points for each invalid word entered.</p>	This will be implemented using the Strategy design pattern. The game's difficulty can be customized by the user before launching the program. This will be done through a series of aggregational and interface relationships. This is outlined in the UML diagram for the Strategy design pattern (Design Pattern 2).	3	3
Multiplayer	2.5	Justin	As a user, I want to be able to play against a friend. This mode should have both player scores displayed on the screen while the game is active.	Player turns will shift after one player has found a word or presses enter to indicate that they have not found anything. The players will each have their own classes to define them, each of which will differ depending on varying settings applied to each player. As such, their methods will account for the differences applied to them by the settings.	3	3
Red/Green Color Adjustment	3.1	Ahmed	As a user with red/green color blindness, I want the ability to clearly distinguish the letters in my boggle board from the board itself.	Create a version of the GUI that avoids red/green color combinations. Using the MVC design pattern, the event-handler will directly change the view to reflect this version of the GUI.	3	2
Score GUI element	3.2	Ahmed	As a casual Boggle player, I want to keep track of my current score as I find new valid words. In Particular, I want to be able to tell if I have passed my previous high score.	Once a valid word is submitted, the score will be updated using the observer design pattern. The dictionary of valid words will notify the "Score" GUI element to change.	3	1
Memory Saving	3.3	Raghav	As the size of the board increases, it tends to increase the memory required to save the board, no of available words on the board (all word dictionaries) and also ever changing properties like human words.	I will use the Singleton design pattern to save all the above-mentioned variables in one spot and thus will save memory as it ensures a class has only one instance, and provides a global point of access to it.	2	3

# Software Design

## Design Pattern 1: Memento Design Pattern

This design pattern will be used to implement user story 2.1: Saved boards (Figure 1). The attributes and methods that are non-essential to the pattern will be omitted.

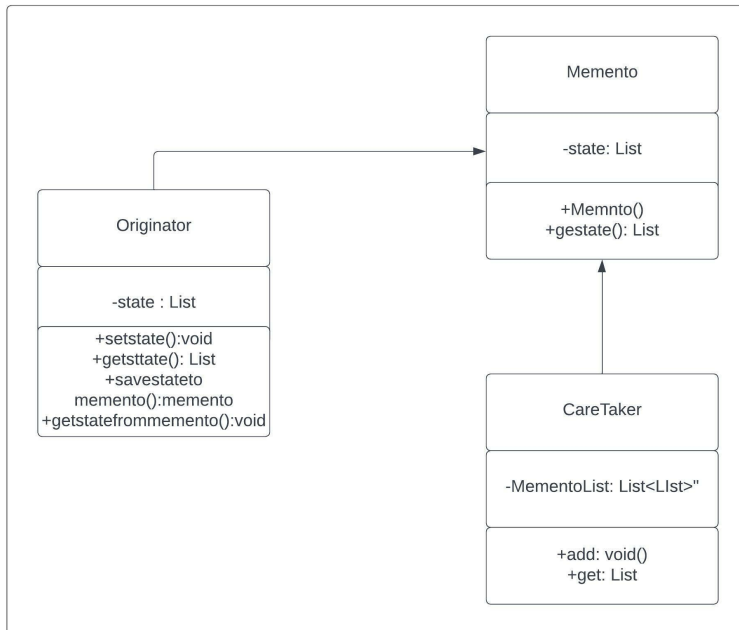


Figure 1

The UML diagram outlines these main components:

- 1) The originator class has 4 methods: *setstate()*, *getstate()*, *savestatetomemento()*, and *getstatefrommemento()*. *setstate()* assigns the current state to the board. *getstate()* gets the specific state from the list of states in the caretaker. *saveststetomemnto()* saves the current state of the board in the caretaker. *getstatefrommememto()* assigns an old state to the board. If needed, more methods could be added later.
- 2) The caretaker class will save the memento files to a list of mentos so that they can be accessed later on when the user starts a new game. It contains a list of mementos, each memento is a list of variables that each boggle board has, for eg the *boggleboard*, *all\_words dictionary*, *human words*, *computer words*, *human score*, *computer score*, *total score*. It has methods to add to the list and get state from the list.
- 3) Memento class has 2 methods: *memento()* and *getstate()*. It is the state in which we are storing the board as, in this case it's a list of variables. *memento()* creates the state and *getstate()* is used to obtain that state.

## Design Pattern 2: Strategy Design Pattern

This design pattern will be used to implement User Story 2.4 (Difficulty Levels). The attributes and methods that are non-essential to the pattern will be omitted. Refer to Figure 2 below:

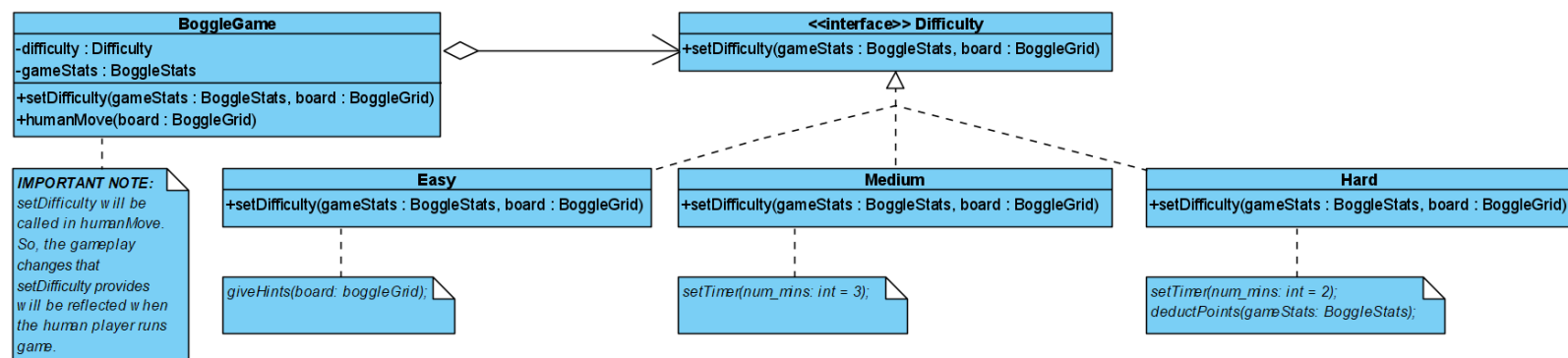


Figure 2

The Strategy design pattern will be outlined in the following:

- The BoggleGame class will be the **context**. It will hold the reference to the difficulty in the *difficulty* attribute. Also, the BoggleGame class will hold the reference to the game stats in *gameStats*. BoggleGame also uses the new method *setDifficulty* along with the existing method *humanMove* to apply the gameplay changes the selected difficulty possesses. In particular, when *humanMove* is called, *setDifficulty* is called within it.
- The **strategy** is the Difficulty interface, which uses the method *setDifficulty* (aggregated from BoggleGame) to execute the various gameplay difficulty changes.
- Easy, Medium, and Hard will be the **concrete strategies** that execute particular gameplay difficulty changes. Each of these difficulties will have *setDifficulty* execute the methods related to that difficulty accordingly. The pseudocode for each *setDifficulty* method is displayed below each difficulty. For instance, Easy will have *setDifficulty* execute *giveHints*, which takes the input *board*. Also, Medium will have *setDifficulty* execute *setTimer* with an input of *num\_mins* = 3. Finally, Hard will have *setDifficulty* execute *setTimer* with an input of *num\_mins* = 2; it will also execute *deductPoints* accordingly, which takes the input *gameStats*. Note that in every difficulty, not all parameters in *setDifficulty* are used.

When the player gets prompted to select their Difficulty of either Easy, Medium, or Hard, the Strategy design pattern outlined above will take care of it.

## Design Pattern 3: Factory Design Pattern

Visual Paradigm Standard (University of Toronto)

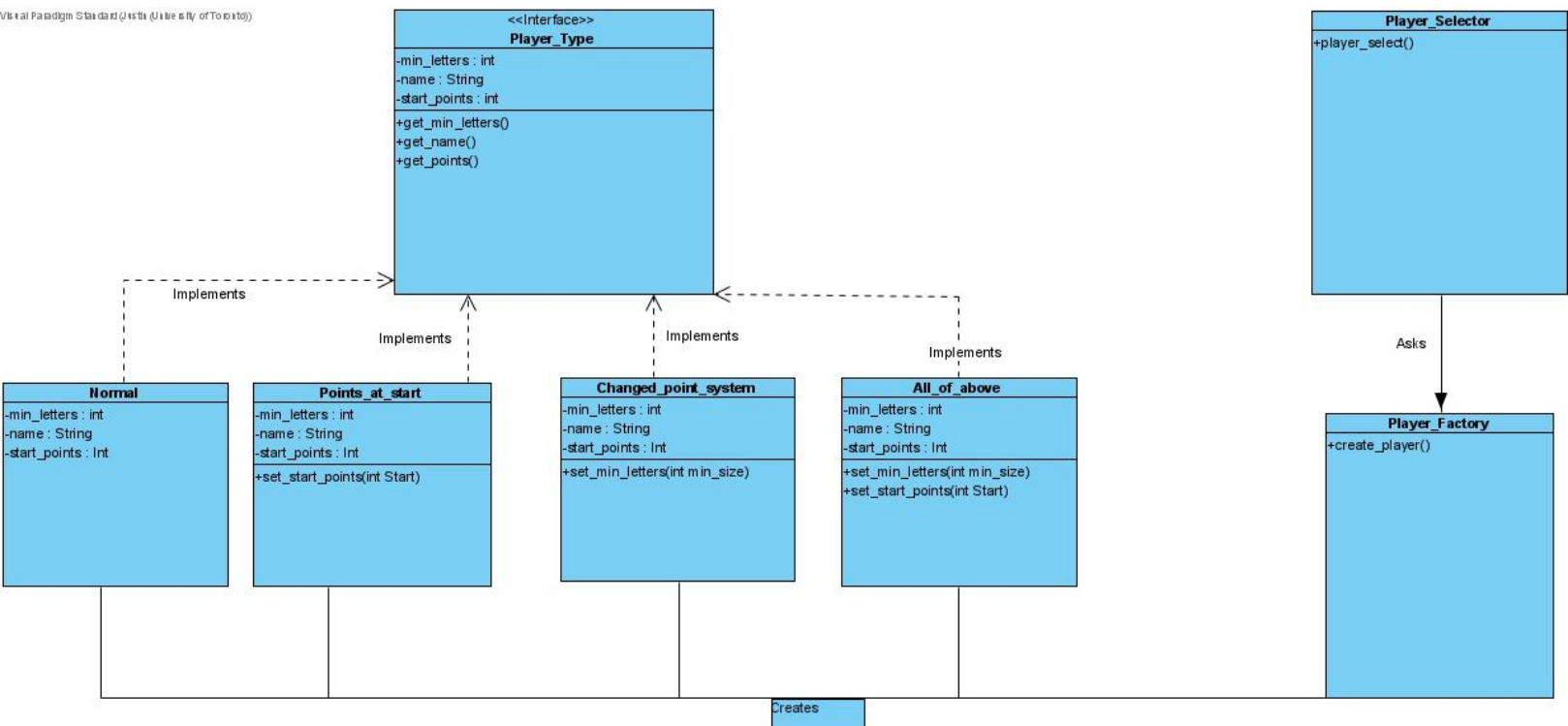


Figure 3

- The Player\_Type interface, which includes attributes and methods designated to set the name of the player, how many points they start with and how many letters are needed at minimum to get a point for a found word. The last two of these attributes can be modified with the selection of the subclasses of Player\_Type.
- Player\_Factory is used as a means of deducing which of the subclasses of Player\_Type is needed for creating a desired type of player.
- Player\_Selector is used to deduce the amount of players needed and calls to its method provides Player\_Factory with the necessary needs when it calls create\_player()

With the use of this design pattern, the user can implement their desired amount of players and can decide on individual rule sets for each of them. The Player\_type interface has the attributes min\_letters to decide the amount of letters needed for a valid word to count for a point. The attribute name will store the name of the player, using it when displaying whose turn it is during the game. The attribute start\_point is the amount of points a player begins with. The subclasses are used as ways to modify these attributes. This is seen with the set\_start\_points and the set\_min\_letters methods. All of the Player\_Type objects are initialized by calls from Player\_Factory's create\_player method, which is tasked with the creation of an amount of these



objects with specific rules. The amount and the specific rules are decided in Player\_Selector's player\_select method.

## Design Pattern 4: MVC and Design Pattern

*Overview:* This pattern will be used to implement User Stories 1.1 (Boggle Board JavaFX GUI) and 3.1 (Score GUI Element)

*UML Diagram:* Refer to Figure 4, below:

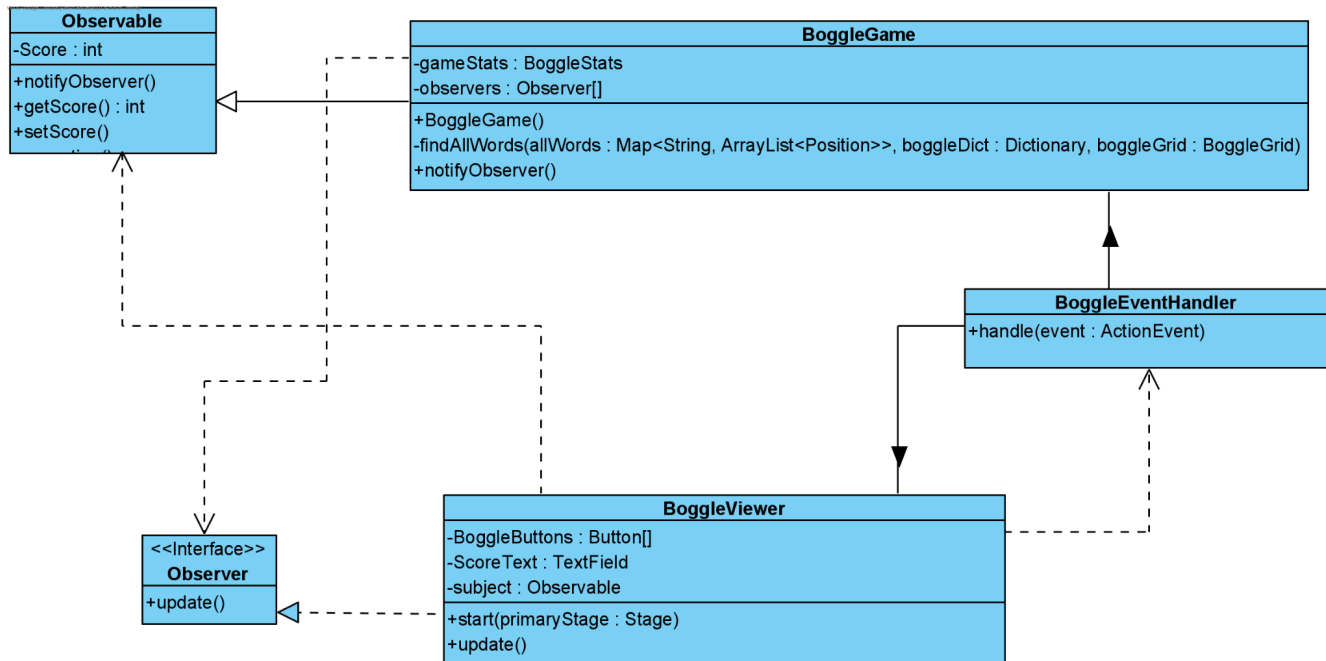


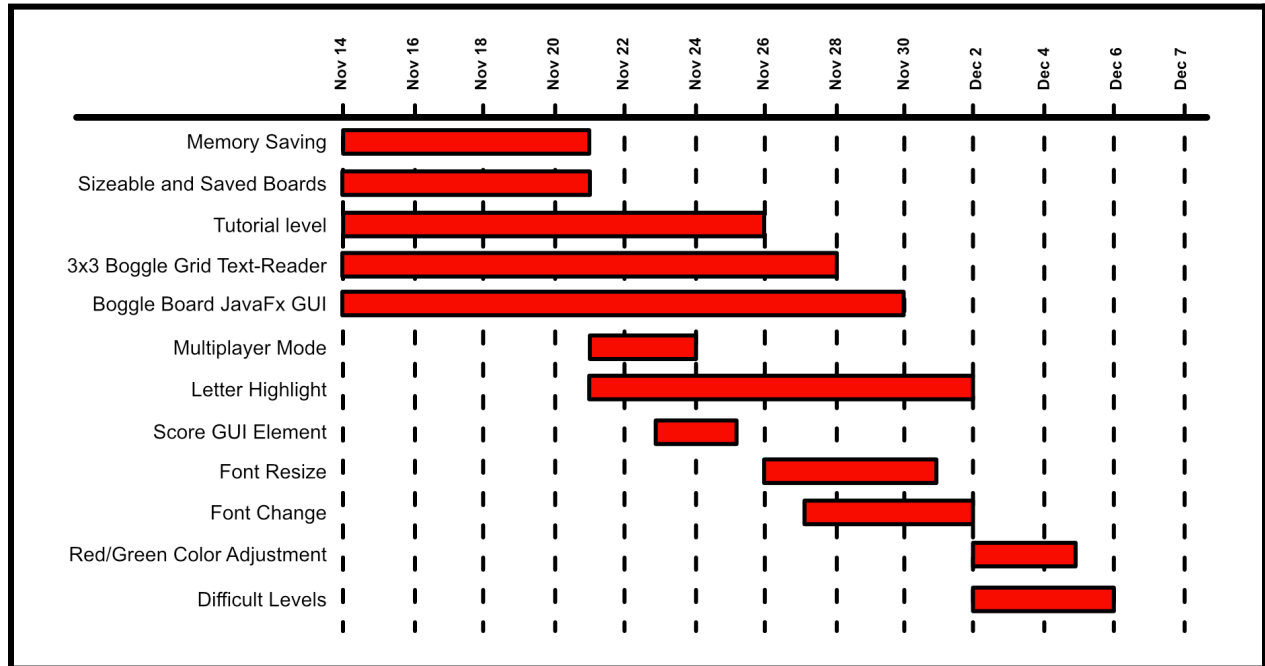
Figure 4

- The BoggleViewer Class, which includes methods *start* and *update*, will be responsible for displaying the JavaFx Boggle application.
- The BoggleEventHandler Class, which includes the method *handle*, will handle all action events (mouse and keyboard events).
- The Observable Superclass will contain the attribute *Score*. This attribute will be used by BoggleViewer to update the *ScoreText* Textfield.
- The Observer interface will be implemented by BoggleViewer. Once notified by BoggleGame, a BoggleViewer object will use its attribute *Subject* to obtain the current *Score (int)*. Once the current *Score* is obtained, the *ScoreText* Textfield will be updated to reflect that change.

To display the Boggle Board JavaFx GUI, the Boggle Game application will use the MVC design pattern, in conjunction with the Observer pattern. The MVC design pattern will be used to control and constantly update the Boggle Board GUI. The Controller (*BoggleEventHandler*) will take in ActionEvents, process them, and then update the Model (*BoggleGame*). Moreover, once BoggleGame has changed, the observer pattern will be used by the Model (*BoggleGame*) to **notify** the view (*BoggleViewer*) to change its GUI.

# Expected Timeline

Here is the team's current assessment of the project timeline:



One of the project's major milestones will be the Boggle Board JavaFx GUI. It will be a collaboration between all team members over a duration of 2 weeks. Moreover, since the Boggle Board GUI has a high priority, the team has allocated it a week of cushioning before the project's deadline. Additionally, another major milestone will be implementing difficulty levels into the game of Boggle. This milestone will signify the project's end, which is currently set on December 7.