University of Toronto Mississauga

CSC207 Fall 2022

Design Document

AUTHORS

Wu Hung Mao
Tianran Hua
Matthew Tay
Zhixing Yu

GITHUB TEAM: **MMTE**
GITHUB URL: **https://github.com/wuhungmao/CSC207-Group-Project**

October 29 2022

1 Project Identification

1. The group MMTE intends to provide a better version of the Boggle game. The current version of Boggle game, with limited functionalities and no visualisation, is not qualified to be an interesting game to play. The reason is that the game is hard to interact with and leaves few options for the user to make choices and decisions.
   We plan to add various new features to the original version of the Boggle game. Most importantly, we shall add a colourful GUI that facilitates interaction and provides accessibility options. Besides, the group shall overcome many limitations of the Boggle game itself. The Boggle game will run on a grid with customisable size (for performance, there is a hard limit of 10x10) and shapes, such as a diamond, a heart, or a cross. Finally, we have decided to introduce a better PvP (Player vs Player) game mode, and add difficulty levels to improve the gaming experiences for PvE games; the group hopes that the diverse game modes fit the habits and abilities of different people.

Name	ID	Owner	Description	Implementation Details	Priority	Effort	Sprint
Game Mode Creation	1.1	Matthew	As a user, I want the game to create different game modes, so I can choose among these modes upon my preference.	Make a GameBuilder class that helps store settings of gamemodes before creating the BoggleModel itself.	2	3	2
Hole Mode	1.2	Matthew	As a user, I want a game mode with holes in the grid so I can have a more characteristic game experience.	Extend the GameBuilder class to make a preset GameMode to modify the createGame() method to help generate a grid filled with holes.	2	3	3
Scroll Bar	1.3	Matthew	As a user, I want to scroll through the application with a scroll bar so I don't have to move the window upwards	Grab all the GUI elements and compose them into a ScrollPane class.	1	1	2
Colourblind Modes	1.4	Wu Hung Mao	As a colourblind user, I want to play in a colourblind mode so I can have an equal game experience to anyone else.	After user select a colorblind mode, the computer will not assign colors that is not distinguisable to the colorblind user.	1	2	1
PvE Difficulties	1.5	Wu Hung Mao	As a user, I want to have different difficulty options so I can defeat the computer in the game.	Easy mode will make computer choose most simple words. Medium mode will be more challenging and computer in hard mode is sometimes invincible.	2	1	2
Grids with Different Sizes	1.6	Zhixing Yu	As a user, I want to have grids of different sizes so I can have flexible playtime.	Modify the constructor of the BoggleGrid class to allow customizable sizes.	1	1	1
Pre-game Settings	1.7	Zhixing Yu	As a user, I want pre-game options so I can configure each game up to my decision.	Make a VBox to contain all GUI components related to pre-game setting and adjust the BoggleModel class to accept customized settings before the gema starts.	1	2	1
PvP Mode	1.8	Zhixing Yu	As a user, I want PvP mode so I can play with others.	Add attributes in the BoggleModel class to allow it to process an extra turn for the PvP mode.	1	3	2
Visualized Gameplay	1.9	Zhixing Yu	As a user, I want a visualization of the game, so I can have a game experience with visual features.	Create a BoggleView class with all GUI components in it.	1	3	2
Customizable Name Field	2	Zhixing Yu	As a user, I want a customizable name fields so I can name my player and identify each player's score.	Add two TextField instances for playeers' names and two Label instances for their scores in GUI.	3	1	2
Customizable Timer	2.1	Zhixing Yu	As a user, I want a a timer which can be paused, so I can set time for each game and leave the game temporarily when I want to.	Add a GameTimer class used by the BoggleGame class.	2	3	3
Editable Word List	2.2	Zhixing Yu	As a user, I want to have editable word lists, so I can see and edit the words I found in the game.	Add two listView instance in GUI.	2	2	2
Hotkeys	2.3	Zhixing Yu	As a user, I want to have a set of hotkeys so I can play the game with either only the keyboard or the mouse for accessibility need.	Make use of the onKeyRelaesed property of javafx.scene.Node.	3	2	3
Colourful Game	2.4	Tianran Hua	As a user, I want to have a colourful interface so the game will be more fun.	Create a method that returns a List of colours with provided settings.	1	2	2
Improved Word Forming	2.5	Tianran Hua	As a player, I want to maximise the number of the words a grid can form instead of not purely random "dices".	Improve the random Grid generation mechanism by adjust frequency of letters to their actual frequency in dictionary words.	2	2	2

2 Software Design

2.1 Design Pattern 1: Template Design Pattern

Overview: This pattern will be used to implement User Story PvE Difficulty.

UML Diagram:

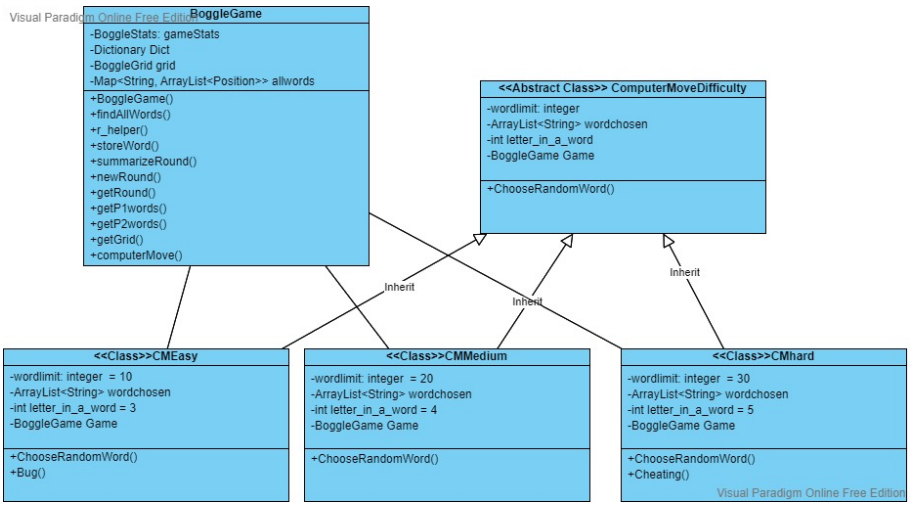

Figure 1: The Template Design Pattern

Implementation Details:
CMEasy, CMMedium, and CMHard are the three concrete implementations of the abstract class ComputerMoveDifficulty.
The template design pattern is for the user story that asks for the different difficulties of Boggle game. Users can choose different difficulties at their own will. This template design pattern contains five classes, one abstract main class is called ComputerMoveDifficuly and three other inheriting classes are called CMEasy, CMMedium and CMHard, respectively.
Each of the three has an association with BoggleGame class. The main class ComputerMoveDifficulty contains 5 attributes: wordlimit, letters_in_a_word, wordChosen, totalScore and allWords, and 3 methods: chooserandomeword, cheating, bug.
- Wordlimit stores the number of total letters in a word a computer can find.
- letter_in_a_word stores the maximum number of a word computer can find.
- Wordchosen is an ArrayList that stores all the words found by the computer.

- AllWords contains all valid words in a grid.
The class CMsimple has wordlimit of 10, and letters_in_a_word is 3.
The class CMmedium is for medium difficulty. It's wordlimit is 30 and letters_in_a_word is 4.
The class CMhard is implemented for hard difficulty. It's word limit is 40 words and maximum number of letters in a word is 5.
chooserandomwords is implemented in the abstract class. Also, CMsimple implements Bug while CMhard implements cheating.

2.2 Design Pattern 2: Observer Design Pattern

Overview: This pattern will be used to implement User Story: Customizable Timer
UML Diagram: Refer to Figure 2 on the second to last page of the document.

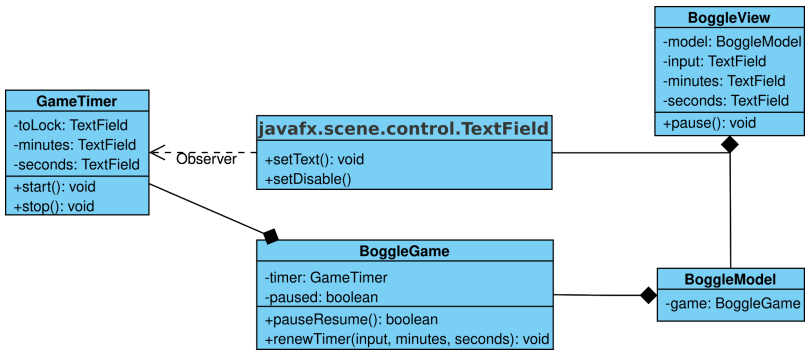

Figure 2: The Observer Design Pattern

Implementation Details: The UML diagram outlines these main components:

1. TextField, a class provided by javafx as the Observable.
2. BoggleModel, with an attribute, game.
3. BoggleGame, with two attribute, timer and paused; and one method, pauseResume.
4. BoggleView, with four attributes, model, input, minutes, and seconds; and one method, pause.

5. GameTimer, the observer with three attributes, toLock, minutes, and seconds; and two methods, start and stop.

The BoggleView class calls renewTimer in BoggleGame which creats a GameTimer and calls its start method to initialize the timer. The start method update the minutes and seconds (GUI components in BoggleView) per second for the user to see the remaining time. This continues untill the remaining time reaches 0, when the toLock (for user input) is disabled and the method stop terminates.
When the user clicks on the minutes or the seconds pause is triggered which calls the pauseResume in BoggleGame which then calls in GameTimer which calls stop or start to pause or resume the GameTimer.

2.3 Design Pattern 3: Builder Design Pattern

Overview: This pattern will be used to implement user story: Game Mode Creation.
UML Diagram: Refer to Figure 3 below.

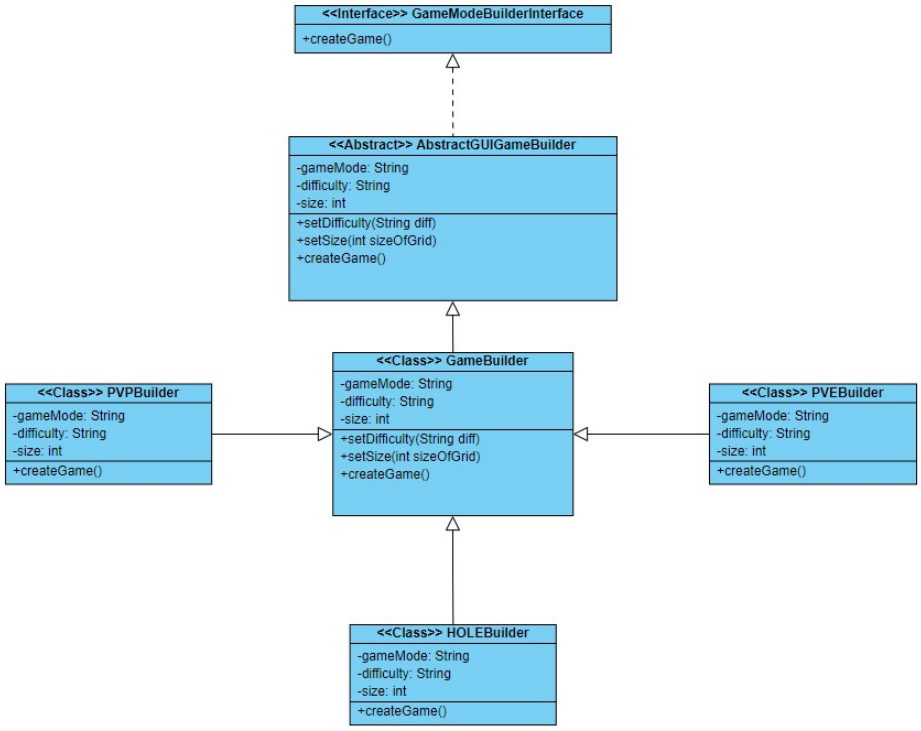

Figure 3: The Builder UML Diagram

Implementation Details: This diagram contains the following components:

1. The **GameModeBuilderInterface** that provides the blueprint for concrete builders to extend and implement.

2. The **GameBuilder** concrete class that can help store settings for a general game mode to build when createGame() is called.

3. The concrete "preset" builders that extend **GameBuilder** have some preset settings that supports making certain game modes much easier.

These different builders help create different variations of the **BoggleModel** class that reflects the game mode depicted by the pre-game settings received from the Boggle application. This design pattern is also made to support the implementation of any other future game modes.

2.4 Design Pattern 4: Iterator Design Pattern

Overview: This pattern will be used to implement User Story 1.4 and 2.4, to colour the game with or without respect to colourblind settings.

UML Diagram: Refer to Figure 4 below

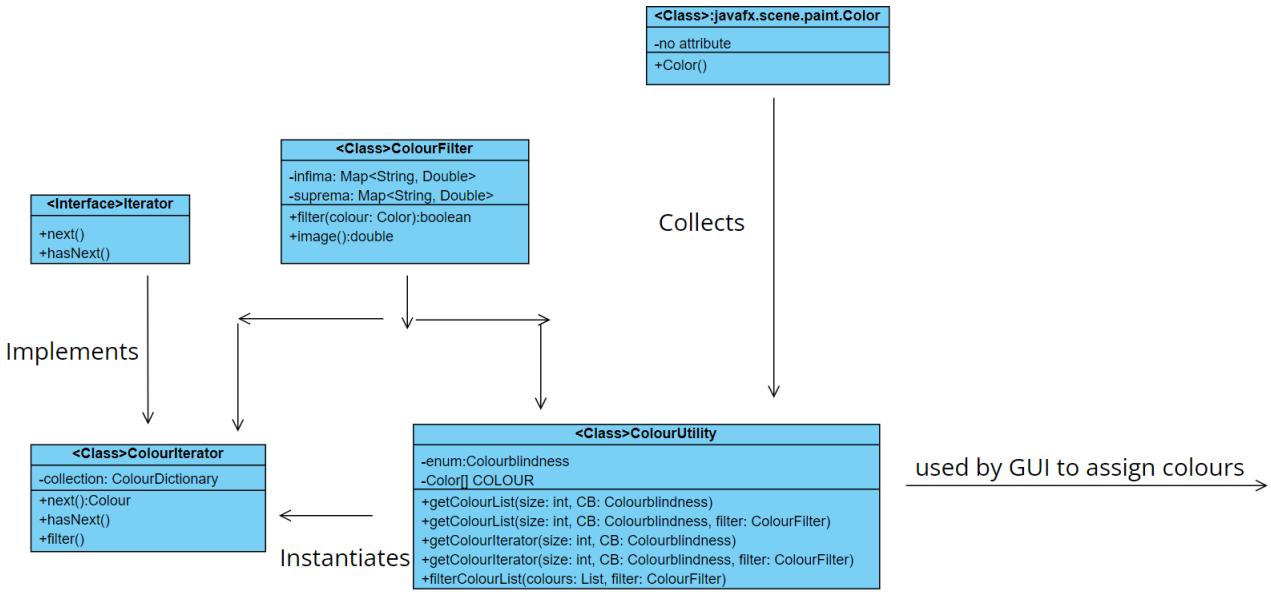

Figure 4: The Iterator UML Diagram

Implementation Details: This diagram contains the following components:

1. The **ColourFilter** class represents a filter to colours contains two **Map<String, Double>** instances that saves the maximally and minimally acceptable red, green, and blue values respectively. The class contains two methods. *filter()* checks if a colour is acceptable; and *image()* checks the percentage of colours that pass through this filter. ‘

2. The **ColourIterator** class represents an **Iterator** for a **List** of **Color** instances. It is special that it can be applied a **ColourFilter** to filter its elements.

3. The **ColourUtility** class contains a few **static** methods that returns a **List** or a **ColourIterator** objects that represents a list of selected JavaFX **Color**'s for uses in the GUI part, such as colouring the **BoggleGrid**. The **enum** *Colourblindness* contains four colourblind settings, *NORMAL, RED_GREEN, BLUE_YELLOW, TOTAL* as the four cases we are considering.

The **ColourUtility** has gathered more than 100 standard JavaFX **Color**'s. The *getColourIterator* and *getColourList* methods are implemented to gather *size* number of **Color**'s that are considered "distinct" enough (in terms of RBG values), controlled by a constant inside this class. Different colourblind settings (passed as a parameter) is used to adjust our criterion on the difference between two colours. The **ColourIterator** class is an implementation of an **Iterator** that supports *next()* and *hasNext()* methods. The other components only need to interact with the **static** methods in the **ColourUtility** class to obtain colours for visualising the game. All other implementations will remain hidden to other parts of the program.

Some specific information on the implementation of the accessible *getColourIterator* methods is below. As accessibility consideration, a normal person is considered trichromatic, i. e., can differentiate red, green, and blue. Therefore, we say the difference between white and black to be 3, thus we expect two colours to differ 3 times the minimal difference constant to be considered distinct. This coefficient is reduced to 2 for dichromatic people, including the "red-green colourblind" and the "yellow-blue colourblind" people, yet our criteria is of course different for them. For a "total colourblind" person, or a monochromatic person, only the greyscale matters, so the difference between white and black will be 1, and two colours are considered different if the difference between their greyness is greater than the pre-established constant.

3 Expected Timeline