(Revised) Design Document

Version 1.0 – 2022.11.13

Created 2022.11.01

Bamboogled

Hassan El-Sheikha

Mustafa Zafar

Sultan Al-Dhalaan

Kevin John Thevara

# section 1: Project identification

Boggle is a game that dates back to the 1970s. Back then, it was much easier to play similar games on a physical board, but nowadays the trend has been to play such games digitally, such as on a computer or smartphone. However, not everyone enjoys a text-based version of a game; it’s too boring, and the black-and-white (or black-and-green) color scheme is irritating to the eye. So, the goal of this project is to create an upgraded GUI version of Boggle, with many upgrades to the initial rules of the game for fun.

# section 2: user stories

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Name | ID | Owner | Description | Implementation Details | Priority | Effort |
| Setup Continuous Integration | 0.0 | Hassan | As a developer doing a code review on another team member’s work, I want to first check that the code compiles, without compiling it on my machine. I also shouldn’t be allowed to merge a branch where the code on it fails any tests. | Use Maven and a CI platform like CircleCi or TravisCi to implement this. | 1 | 1 |
| Model Interface | 1.1.0 | Hassan | As a frontend developer working on the view, I want to be able to reference and call methods of the model in my classes, regardless if it has been implemented or not. | Create an interface IBoggleModel. This will have methods to start a new game, end a game, get and add to the current word, submit the current word, etc. | 3 | 1 |
| Model Implementation | 1.1.1 | Hassan, Sultan (actually implemented by Hassan) | This is the implementation for the interface described above. | -Implement all methods in the IBoggleModel interface in a concrete BoggleModel class.  -Create algorithms for updating the paths leading to a certain word (there could be multiple on the board!).  -Create | 3 | 3 |
| Model Test-Suite | 1.1.2 | Sultan | As a developer working on the BoggleModel, I want to have a high-quality test suite to run tests against my code so that I can make sure it is correct. | Create tests for algorithms and operations in the BoggleModel. | 1 | 1 |
| Text-Based Version | 1.1.5 | Sultan (Actually implemented by Hassan) | As a physical board game “boggler,” I want to be able to play with friends on my computer so that I don’t have to manually check for valid words and memorize scores.  This text-based version of boggle should accept letter-by-letter inputs instead of whole words, so that it simulates the actual GUI version. | Create a TextBased view that accepts character-by-character input instead of whole words. This should update the board to reflect the current path the user has selected after every character input. | 1 | 2 |
| GUI | 1.3 | Mustafa, Kevin, Hassan | As a boggle user, I want the board to be displayed using a GUI so that it can be easier for me to play the game (as opposed to the text-based version). This new GUI should still allow me to play either with a friend or against the computer. This version will be playable with only a keyboard. | Create 5 views:  -The WelcomeView, which is like a home page for the game.  -The  -InstructionsView, which is a page containing instructions on how to play the game  -PlayerInitView, which is a page where the user configures the board size and inputs player names.  -4x4 and 5x5 BoardView. This is the main game page, where the user selects words and increases their score.  -EndGameView, where the winner’s name is displayed and the user can opt to play again. | 3 | 3 |
| Audio-Related Algorithm in Model | 2.2.0 | Sultan (Never Implemented) | As a visually impaired Bamboggled player, I want the screen to read out my next possible moves after I select a letter. | Loop over all possible paths and fetch possible next moves. |  |  |
| Audio-  Only Mode | 2.2.1 | Kevin, Sultan (Actually implemented by Kevin—Incomplete) | As a visually impaired person, I want a UI that allows me to interact with the board without visual elements, but rather only with audio so that I can play the game in the same way that non-visually impaired people play. This new audio-only mode should allow me to play against the computer. | Create button or shortcut that activates visually impaired-mode. If this option is enabled, the computer will read out information on the current view. | 2 | 3 |
| ~~Timed Boggle~~ | ~~3.1~~ | ~~Kevin, Sultan (Never Implemented)~~ | ~~As a competitive boggle player, I want to set a timer so that I can challenge myself.~~ | ~~Backend~~  ~~Timer object~~  ~~Game termination upon timer stop~~  ~~Event handling for timer toggle~~  ~~Frontend~~  ~~Textbox with timer~~  ~~Timer option~~ | ~~2~~ | ~~2~~ |
| Undo | 2.1 | Hassan, Mustafa (actually implemented by Hassan). | As a Boggle player who makes errors when choosing words, I don’t want to have to restart choosing my whole word. Instead, I want to press backspace to delete the most recently pressed letter. | -Memento implementation for model  -Event handling for backspace key | 3 | 2 |
| ~~Powerups~~ | ~~4.1~~ | ~~All (Never Implemented)~~ | ~~As a boggle user, I want to get a little something extra when I find a large word (money would be appreciated).~~  ~~Powerups: Score multiplier, time extend, etc.~~ | ~~Backend~~  ~~-Score multiplier algorithm~~  ~~-Time extension algorithm~~  ~~Backend:~~  ~~-Multiplier redemption toggle~~  ~~-Time extend toggle~~ | ~~1~~ | ~~3~~ |

# section 3: Software design

**Design Pattern #1: Memento**

**Overview: This pattern will be used to implement user story 2.1 (Undo).**

**UML Diagram:**

**Diagram

Description automatically generated**

**Implementation Details:** The UML diagram outlines three main components:

* The BoggleModel class, which is the model.
* The ModelSnapshot class, which saves the state of the BoggleModel at a particular moment in time.
* A stack (ModelHistory). This holds Memento (ModelSnapshot) objects in case the user needs to undo one of their actions, in which case the latest ModelSnapshot is popped off the stack and is restored by the BoggleModel.

Whenever the user selects a new letter for their word, the an event handler will instruct the BoggleModel to add that letter to the current word. The BoggleModel will create a ModelSnapshot (i.e., a backup) and push it onto ModelHistory. When the user would like to undo (by pressing the backspace key), the event handler will instruct the BoggleModel to restore its previous state, and it will do this by popping the most recent ModelSnapshot off of ModelHistory and changing its attributes to match that ModelSnapshot. This will effectively restore the last state of BoggleModel.

**Design Pattern #2: Observer**

**Overview: This pattern will be used to implement user story 1.3 (GUI).**

**UML Diagram:**

Diagram

Description automatically generated

**Implementation Details:** The UML diagram outlines four main components:

* The Display class, which is the GUI which the user interacts with.
* The EventHandler<KeyEvent> interface, which includes the handle method.
* The KeyEventHandlerclass*,* which implements the EventHandler<KeyEvent> interface and handles key events from Display by updating Display class attributes accordingly
* A modified stack (History). This holds Memento (BoardState) objects in case the user needs to undo one of their actions, in which case the latest BoardState is popped off the stack and restored on the Display.

The KeyEventHandlerclass is set as an observer for any KeyEvent changes in the GUI scene in the Display class. When users playing boggle interact with the keyboard to input selected letters, a KeyEvent is sent to KeyEventHandler. KeyEventHandler uses the handle method to appropriately update the letters given by the player into its different attributes (letters, currentWord, currentWordVisual, currentLetter). If the player would like to undo (by pressing the backspace key), KeyEventHandler will pop the most recent BoardState object from the history stack in History and call the restore method of BoardState to restore the last state of the board. Similarly, once the player has chosen their word from the board, they can press the enter key so that the KeyEventHandler can check if the selected word is valid and clear the history stack.

**Design Pattern #3: Singleton**

**Overview:** This pattern will be used to implement user story 1.3 (GUI).

**UML Diagram:**

**Graphical user interface

Description automatically generated**

**Implementation Details:** Each view needs access a BoggleModel. Instead of passing the model through each path (which won’t even work due to a JavaFX limitation), the solution is to get it through the BoggleModel’s static getInstance() method instead.

# section 4: EXPECTED OVERALL PROJECT TIMELINE

In this section, we explain the expected overall project timeline, using the Gantt chart method. We have split up the user stories into 4 main subsections, into its own separate week: Week 1 covers GUI and carrying over the text-based implementations; Week 2 focuses on creating an audio-only mode for visually impaired users; Week 3 is centered on creating timed Boggle; and Week 4 will be spent working on powerups. We have organized the timing of these tasks from those of greatest importance to least, to accommodate for any setbacks, so we can push back or delete tasks from the latter weeks. Each Week has tasks, which consist of several smaller subtasks, to make organizing and scheduling easier for members.

Timeline

Description automatically generated

The current team believes that the tasks and objectives and their respective timelines are fair and achievable. The team believes these deadlines to be necessary in order to create the final product that we desire. The major milestones in this assignment will be creating the general GUI and creating an audio-only mode for visually impaired users. The group has these goals to be the most important due to its importance in making the game entertaining and accessible to more people.