# 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 |
| Multiplayer Text-Based | 1.1 | Sultan | 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. | Implementation in backend:  -Add multiplayers in engine  Implementation in frontend:  -Change console prompts | 3 | 1 |
| Text-Based | 1.2 |  | As a solo player, I want to be able to play a text-based version of boggle against the computer, so that I have “someone” to play my favorite game with. | -Nothing | 0 | 0 |
| GUI | 1.3 | Emzy, Kev, 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. | Backend:  Event handling  Frontend:  Buttons, board, colors, etc. | 3 | 3 |
| Audio-  Only Mode Against Computer | 2.1 | Kevin, Sultan | As a blind 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-blind people play. This new audio-only mode should allow me to play against the computer. | Backend:  Event handling to speak letters  Frontend:  Options to toggle blind-mode on, etc. | 2 | 3 |
| Audio-Only Mode Against a Friend | 2.2 | Mustafa, Hassan | As a blind person using the audio-only mode against the computer, I want to be able to play with a friend (who may or may not be blind). |  | 2 | 1 |
| Powerups | 3.1 | All | 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, hints, etc. |  | 1 | 3 |
| Timed Boggle | 4.1 | Kevin, Sultan | As a competitive boggle player, I want to set a timer so that I can challenge myself. |  | 2 | 2 |
| Undo | 4.2 | Hassan, Moose | As a Boggle player who makes errors when choosing words, I don’t want to have to restart choosing my whole word. |  |  |  |

# section 3: Software design

**Design Pattern #1: Memento**

**Overview: This pattern will be used to implement user story 4.2 (Undo).**

**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 Originator interface, which includes two main methods: saveState and restoreState.
* The Memento interface, which includes one main method: restore.
* 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.

Whenever the user selects a new letter for their word, the Display (via the EventHandler) will create a BoardState (i.e., a backup) and push it onto the stack contained in History. When the user would like to undo (by pressing the backspace key), the EventHandler (see the Observer pattern) will pop the most recent BoardState and call its restore method, which will cause the BoardState call the Display’s restoreState method and pass itself as a parameter, effectively restoring the last state of the board, as required by the user.

**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 1.4 (Undo).

**UML Diagram:**

**Diagram

Description automatically generated**

**Implementation Details:** As mentioned above, a stack of BoardState objects is kept in case the user would like to undo. However, as the display does not have a reference to this History stack, it is unable to pass it to an EventHandler, such as the KeyEventHandler. So, the singleton pattern is used to have only one possible instance of the History class at once, and this is implemented by making the visibility of the History constructor private and creating a static method in the class called getInstance, which will either create a new instance if one does not already exist, or return the current one otherwise.

**Design Pattern #4: Bridge**

**Overview:** This pattern will be used to implement User Story 1.3 (GUI).

**UML Diagram:**

**Diagram

Description automatically generated**

**Implementation Details:** The Display (application class) contains references to a collection of LetterElement objects. These visual objects contain a letter and a color, and are rendered on the screen. However, there are many colors which these LetterElements can take on, and so we split the color class from the LetterElement class into two separate hierarchies to avoid creating many different classes. This is done by making the LetterElement class hold a reference to a color.