# Über Boggle

Software Design Document

Team Name
Charles Kaup
Logan McCarthy
Taylor Schrader

## Introduction

### Purpose

The purpose of the system is to provide users with a application in which they are able to play the popular game Boggle on their computer.

### Design Goals

Have the Boggle game run on any computer with Java installed. Build the game so that it is cleanly coded and maintainable while designed such that it can be updated at any point.

### Design Trade-Offs

At the current stage of development, there have been no significant trade offs made worth mentioning.

### Definitions, Acronyms, and Abbreviations

Gameboard: A 4x4 grid of letter dice that the use will be able to select words from.

Dice: Each tile on the gameboard that will each have 6 options of letters that will be decided randomly at the beginning of every game.

Word Checking API: API used to validate whether the word entered by the user is a legal english language word.

Timer: A 3-minute countdown that designates the start and end of each game played.

Player: A user that is interacting with the application

End of Game: When the timer reaches 0 the player will be unable to enter any more words until the game is restarted

GUI: Graphical User Interface

Boggle: a word game designed by Bill Cooke, invented by Allan Turoff and originally distributed by Parker Brothers. The game is played using a plastic grid of lettered dice, in which players attempt to find words in sequences of adjacent letters.

Rules: Can be found HERE (https://en.wikipedia.org/wiki/Boggle#Rules)

### References


I. https://www.hasbro.com/common/instruct/boggle.pdf

II. https://en.wikipedia.org/wiki/Boggle#Rules


### Overview

The game will be designed to allow users to play the game of Boggle on their computer. The game will be single player and each game will last 3-minutes. Words will be checked to make sure they are legal english words and that they do properly appear on the gameboard. The game will keep score of how many words the user finds during the time and presents the score after the timer has expired.

## Current Software Architecture

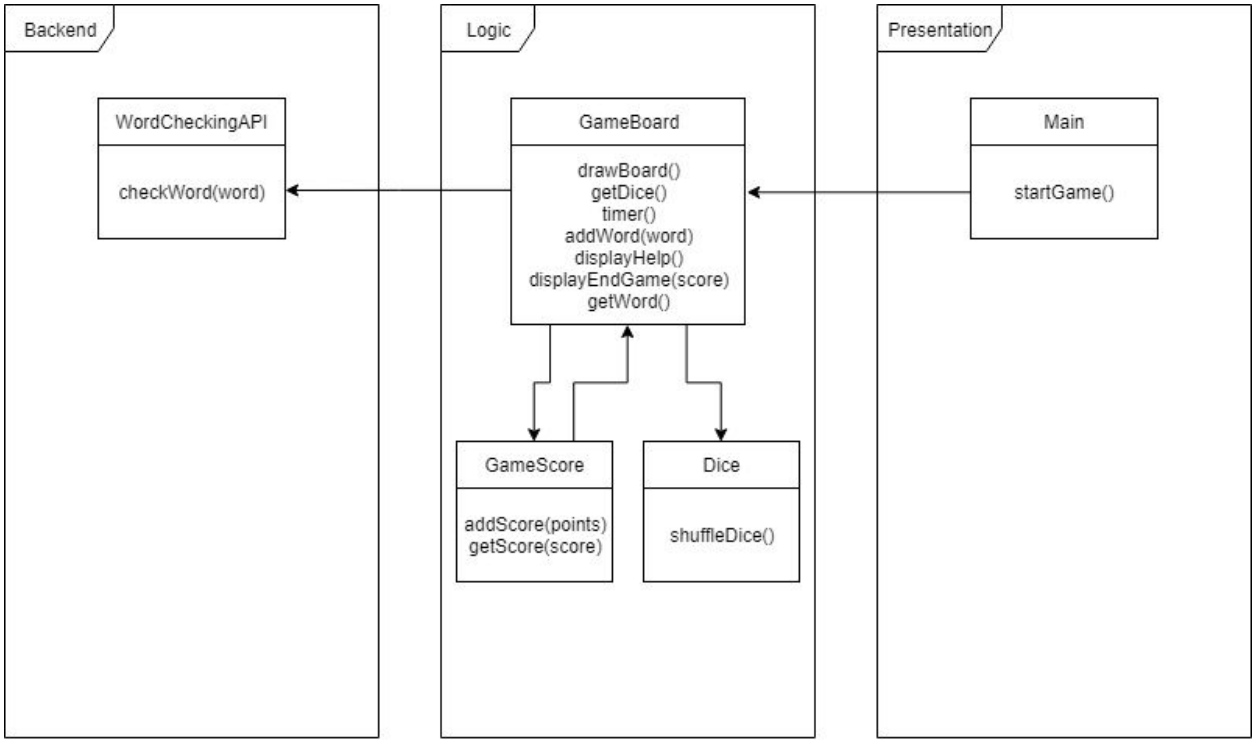
### Overview

Uber Boggle will be a Java application using the JavaFX library for graphics and a word-checking API. The program will replicate the physical game of Boggle.

### Subsystem decomposition

The system will contain a variety of subsystems, induding: Main, GameBoard, Dice, WordCheckingAPI, and GameScore subsystems. These will use a peer-to-peer model to communicate between each other. The game will be initiated within the Main class which will then call the GameBoard class. The GameBoard class will draw the board, get the dice from the Dice class (which will randomize which side of each die is “up”), and start the timer. When a word is added the word will be sent to the WordCheckingAPI to check whether it is a valid word and then the GameBoard will keep track of all the words entered. At the end of the game the words are sent to the GameScore class to get the points for each word and then add them to the total score. This is then sent back to the GameBoard to display during the end game sequence. If at any time the user selects the “?” on the screen a help

screen in overlayed to show the rules of the game.




## Hardware/software mapping

This is not applicable to the Über Boggle application, as it is purely software based and does not rely on external hardware.

## Persistent data management

Persistent data management is outside the scope of this project. As it is a simple Boggle game, there is no need for a database or other types of persistent data management.

## Access control and security

This game does not contain any private information or anything that would need to be secured or protected, the only bit would be the possible Oxford Dictionary API keys and we will deal with those when we implement them.

## Global software control

We will implement global software control with splitting up the tasks of the game to seperate classes that call on each other for specific functionalities. As well this will help to fix synchronization and concurrency issues as it will only allow one class to call the function and will have the others wait till the class returns the desired info to go further.

### Boundary conditions

Start-up: Player launches the game and presses newGame which follows the Start Game use case in our use case model.

Shutdown: The player closes the application and everything is simply shut down. All data is not saved (we may add this functionality in the future for scoreboard or something but not as of now)

Error Behavior: System should detect the error and attempt to move past it without the Player experiencing it but otherwise if critical enough it should close out with an error that doesn't effect the system.

## Subsystem Services

Main: Only starts the game by calling the GameBoard's start method and only consumes services.

GameBoard: Only consumes services

WordChecker:

1. findWords(): Is called at by the gameOver method in the GameBoard class and finds all valid words on the board.
2. searchWords(): is called by findWords() to search for valid words on the board by connecting dice
3. isSafe(): Checks if a die is valid: within bounds of matrix, not a repeat

Die:

1. createDice(): Is called by the GameBoard to shuffle and create the dice on the board

## Proposed Software Architecture

No Change

## Class Interfaces

Main: This class starts the game and calls on the GameBoard for all it functionality with communicating with the other classes and their functions.

GameBoard: The GameBoard is the main logic class where many of the games functions are stored and calculated

Dependencies:

1) WordChecker for checking legality of words entered and scoring them.
2) Die class for creating, shuffling, and animating the dice on the board.

Operations:

1) Draws the board with the JavaFX start method and helper "create..." methods.
2) Gets the dice from Die class with createDice method

3) addWord gets words from the user input text box and adds them to the enteredWordsList
4) displayHelp displays the help menu when user presses “?” button for instructions
5) gameOver displays the end game screen with score after the timer runs out

Exceptions: Word checker not finding the scrabbleWords.txt file.

Public field and Methods: enteredWordsList,, start, fillPane.

WordChecker: The WordChecker will operate in the backend of the system and verify that words entered by the user are valid words in the Scrabble dictionary and on the board. Also totals the score and sends it back to the GameBoard when the game is over.

Dependencies: Trie and TrieNode classes.

Operations:

1) findWords: find all words on the board
2) searchWords: iterate through all dice on the board to construct words for the findWords method
3) isSafe: checks if a die is valid when constructing a word with searchWords: is in array bounds, is not a repeat.

Exceptions:

1) File not found if the scrabbleWords.txt file is removed from the resources folder.

Public field and Methods: findWords()

Die: The Die class will take the array of all possible letters for the dice and randomize which set of letters will go to which Die, and which letter will be shown.

Dependencies: N/A

Operations:

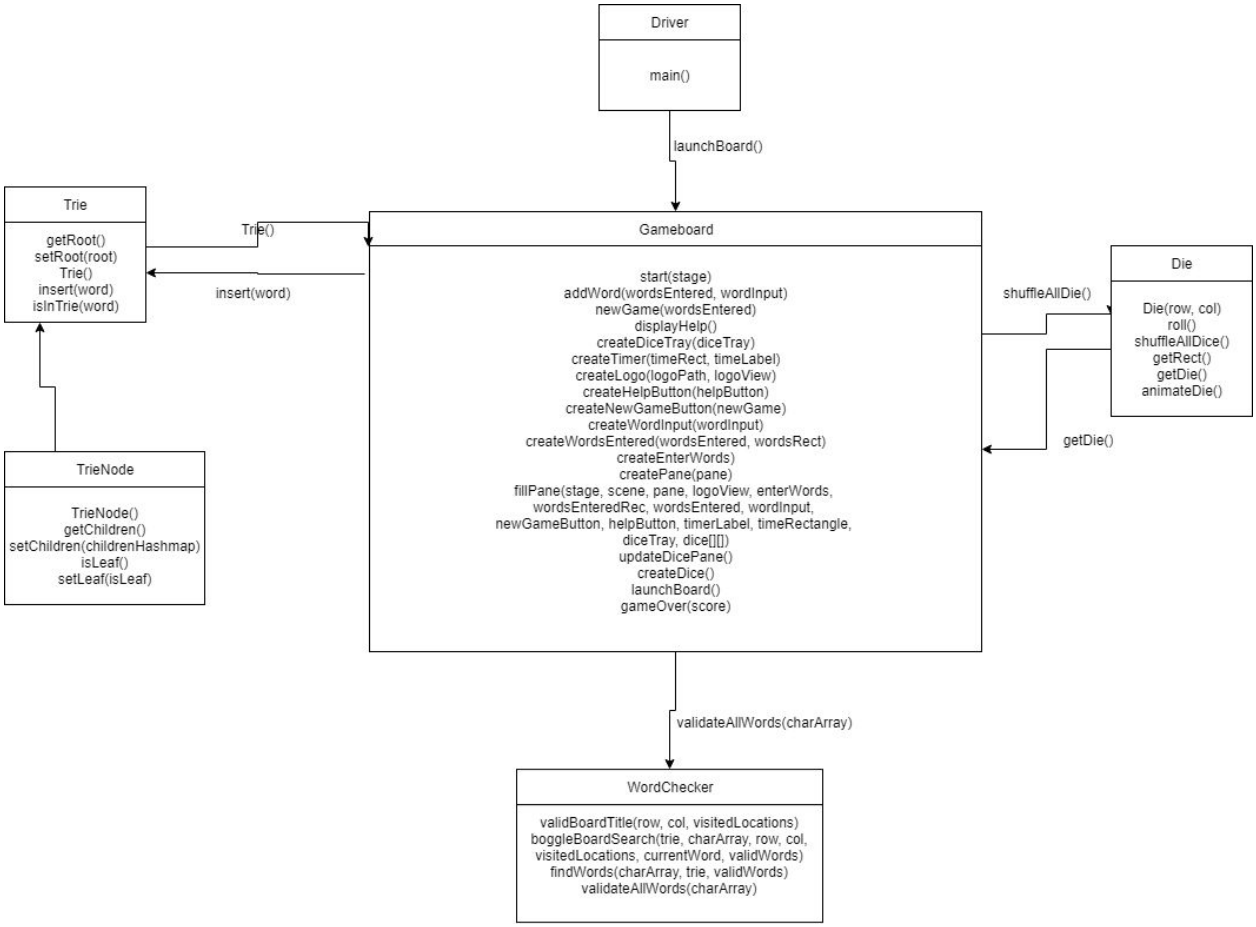
1) Shuffles the dice and returns them to the GameBoard to be displayed

Exceptions: None that we can think of as of now

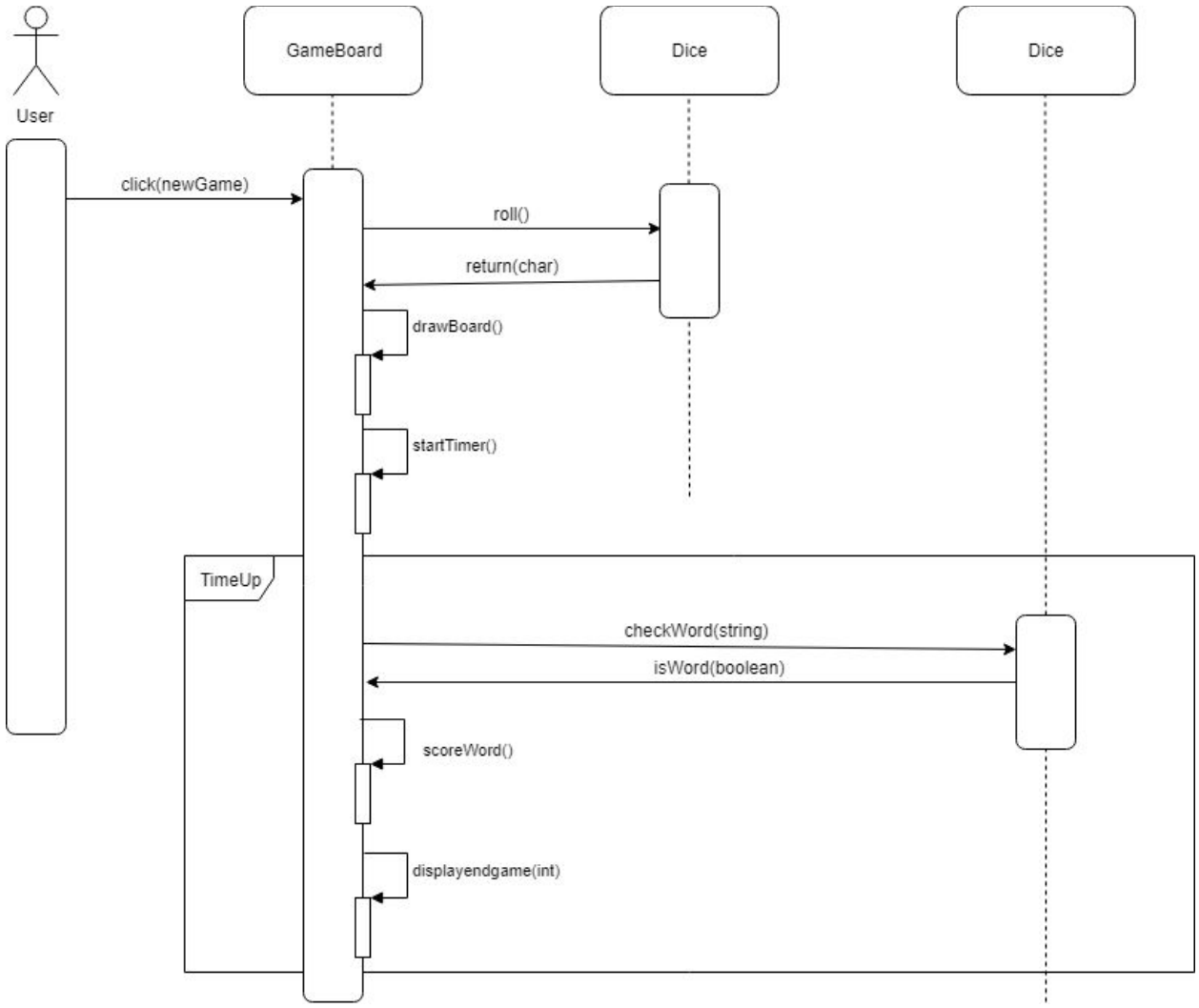
Public field and Methods: shuffleAllDice(), roll(), getRect(), getDieText().

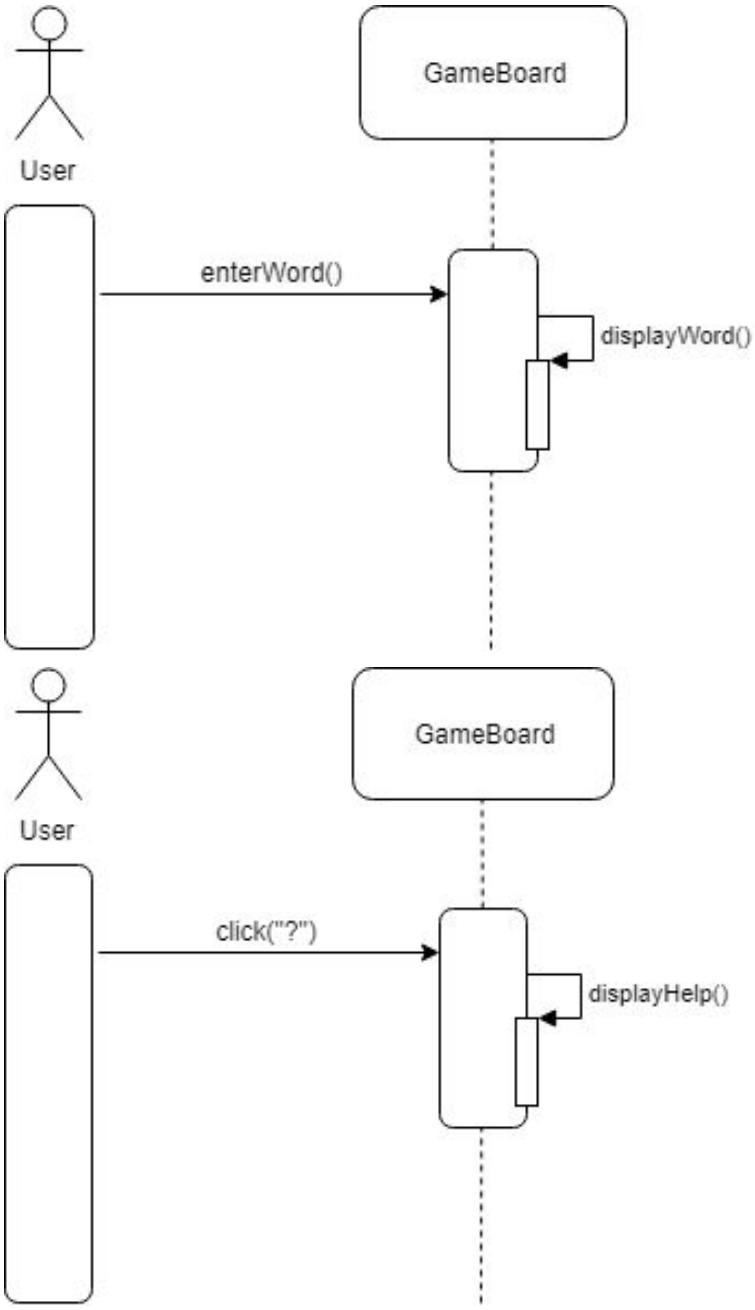
## Detailed Design

Class diagram:




Sequence Diagrams:


User
GameBoard
Dice
Dice
click(newGame)
roll()
return(char)
drawBoard()
startTimer()
TimeUp
checkWord(string)
isWord(boolean)
scoreWord()
displayendgame(int)


GameBoard
User
enterWord()
displayWord()
GameBoard
User
click("?")
displayHelp()

## Glossary

N/A