Boggle by Team 12

Software Design Document

Team 12
Quinn Lennemann
Kolby Johnson
Nischal Neupane

Introduction

Purpose

Our purpose is to create a digital single player Boggle game that is readily available to users to experience fun by themselves if they are lonely, such as during Valentine's Day.

Design Goals

- Reliability
- Modifiability
- Maintainability
- Understandability
- Efficiency
- Portability
- Robustness
- High-performance
- Good documentation
- User-friendliness
- Readability
- Flexibility

Design Trade-Offs

- Functionality vs. Usability
- Efficiency vs. Portability
- Robustness vs. Functionality

Definitions, Acronyms, and Abbreviations

1. RAD : Requirements Analysis Document

2. UI: User Interface
3. SDD: Software Design Document
4. TBD : To be determined
5. Timer : an object enforcing the time limit of the game

References

1. Increment 3.pdf
2. Boggle Game Rules

Overview

This document consists of the discussions about the designs for the project with visual representation and a prototype UI software design. The Proposed Software Architecture section consists of the basic overview of what the architecture may look like when its implemented along with the details about the environment and the packages that may be used in it. the document also consists of and explains the subsystem components and their uses in the software.

Current Software Architecture

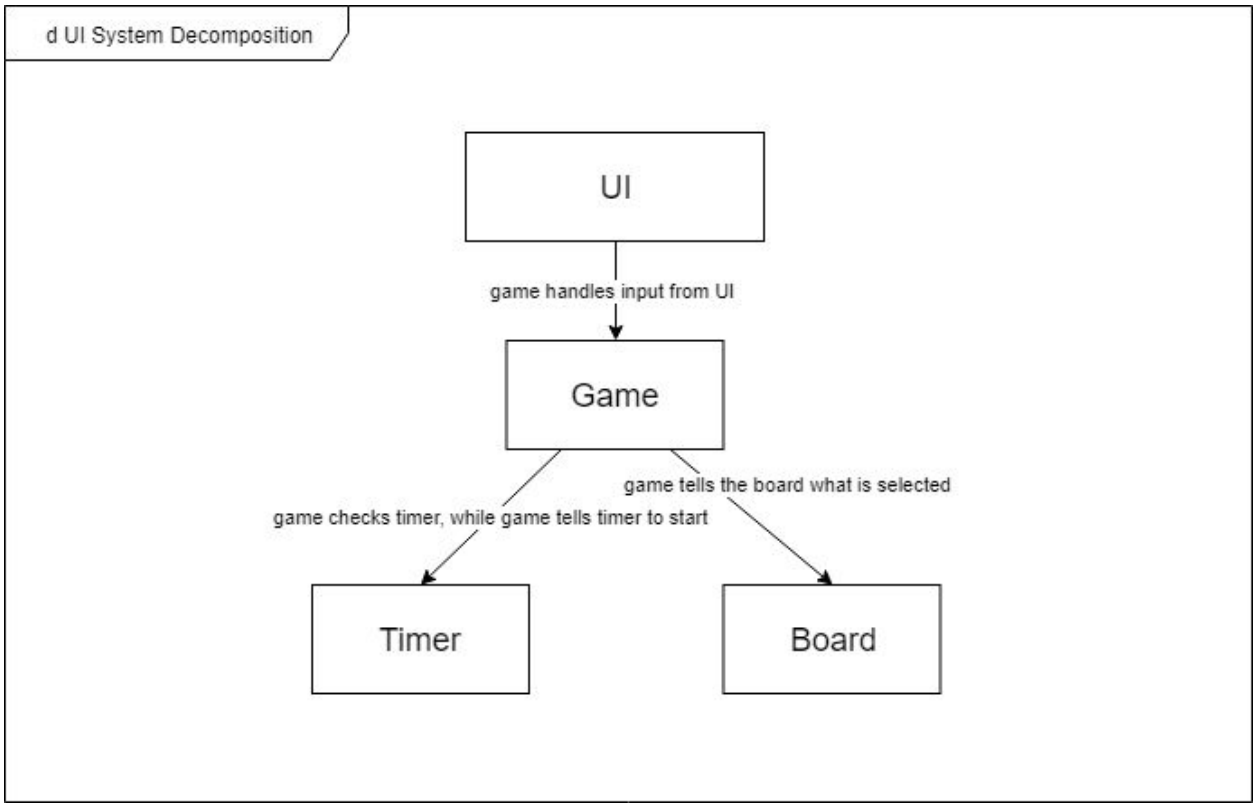
TBD

Proposed Software Architecture

Overview

The architecture that we decided to use is the three layer architecture which has UI as the presentation layer, game as the logic layer, timer and the board as the data layer. Each layer has a depends on relationship with the layer above it.

Subsystem decomposition



Our architecture follows a 3-layer architectural style. The UI represents the presentation layer, since this is what the player will be interacting with. Game represents the application layer, because it is where all of the logic of the game will occur. Timer and Board represent the data layer, as these subsystems will keep track of important information related to the game.

UI: The User Interface is responsible for handling user input and providing a visual representation of the game.

Game: The Game subsystem is responsible for the logic of the game. It handles input from the UI in order to read and update data about the Timer and Board.

Timer: This is the most simple subsystem in our architecture. It is responsible for keeping track of the time that has passed since the game started.

Board: This subsystem is responsible for keeping track of the board's status. This includes which sides of the dice are currently shown, where the dice are located on the board, and which dice have been selected by the user.

Hardware/software mapping

N/A, because our subsystems are not assigned to hardware

Persistent data management

N/A, because we do not have a database to store the data outside of the scope of the application.

Access control and security

N/A, since everyone is a player there doesn't need to be a distinction between the users.

Global software control

N/A, because it is a single player game and there is only one object being used.

Boundary conditions

Upon start-up, the program reads the dictionary file and the User Interface will be initialized. If the player enters a non-valid word it will completely be ignored. Upon shutdown, the program will simply quit.

Subsystem Services

UI:

The UI will only consume services. It will visually display certain services by getting data through the Game subsystem.

Game:

- startGame() - Allows the player to start the game through the UI
- endGame() - Process that occurs when checkTimer() from the Timer is 0
- inputLetter(letter) - Allows the player to input a letter through the UI
- getScore() - Provides the current score of the game to the UI
- submitWord() - Submits the current selection of letters and updates the score and UI if necessary

Board:

- randomize() - Randomizes the position and orientation of the dice when the game is started
- clearHighlights() - Removes all highlighting on the board. This is called by the Game subsystem after a word is submitted or when an input is invalid
- animate() - This is called by Game during the startGame() process, which displays an animation of the dice on the board being randomized

Timer:

- startTimer() - Called when the player starts the game
- isRunning() - Checks if the game is running

Class Interfaces

UI:

This class has little functionality on its own. It is simply to contain the necessary Swing elements needed to run the game

GameTimer:

starts and stops time while checking in between

Timer depends on Game to start the time

Board:

Board randomizes the letters and the dice, also highlights letters that are being used in the current word while checking if that word is valid

Board depends on Game to know when to randomize and when to manage highlighting

Die:

This class contains the specific letters for each side of the 16 different dice. It can randomize sides, and it is used by the Board class.

DieLabelFactory:

This class simply makes JLabels used on the board to represent the dice visually. It is called by Board and the Labels are added to the UI.

Game:

logical part of the system that interacts with all the other subsystems

Game's attributes are timer, board, dictionary, score, and word

Game depends on Timer to check the time, on Board to check if the word is valid, and on dictionary to check if word is valid.

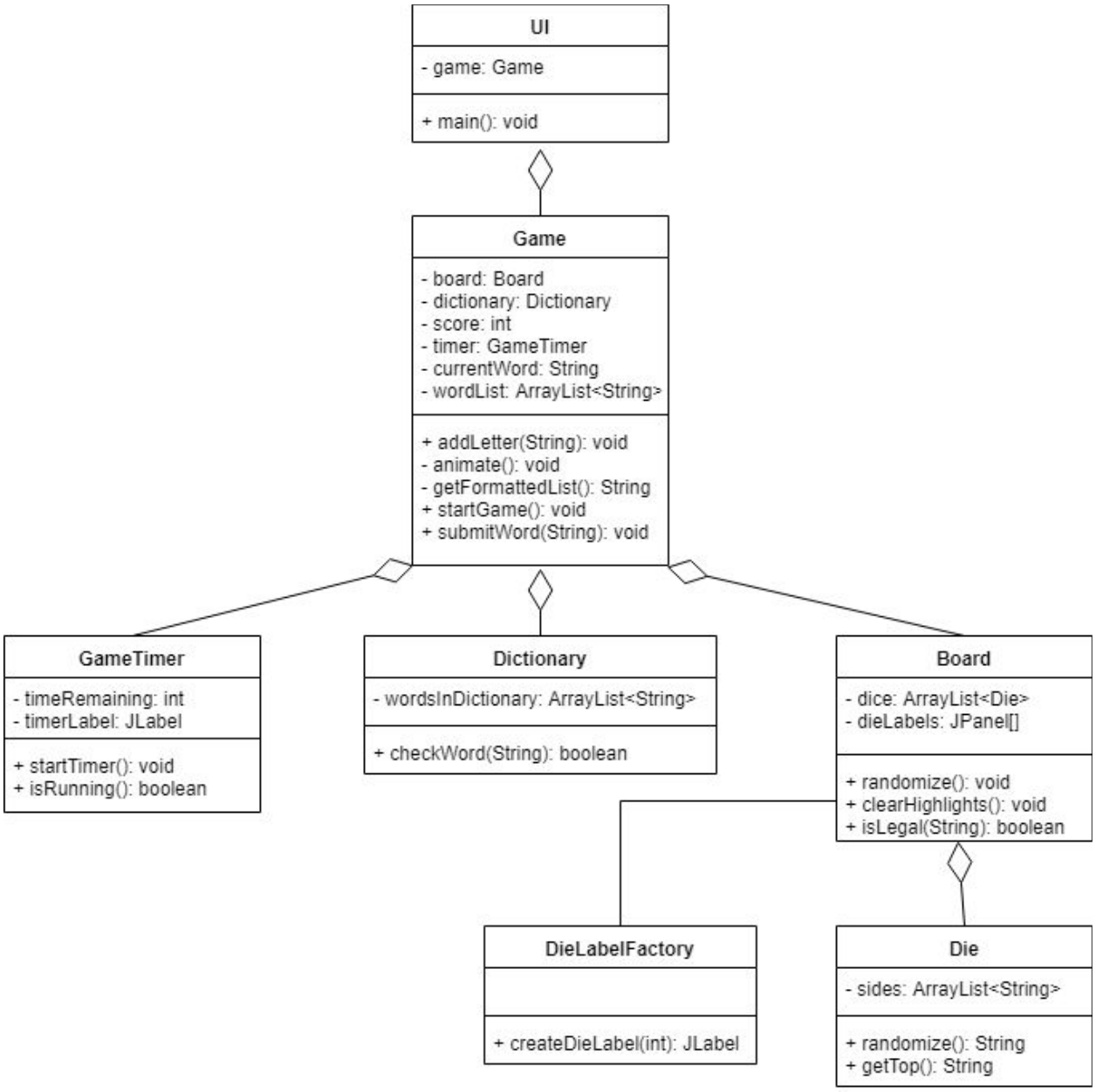
Dictionary:

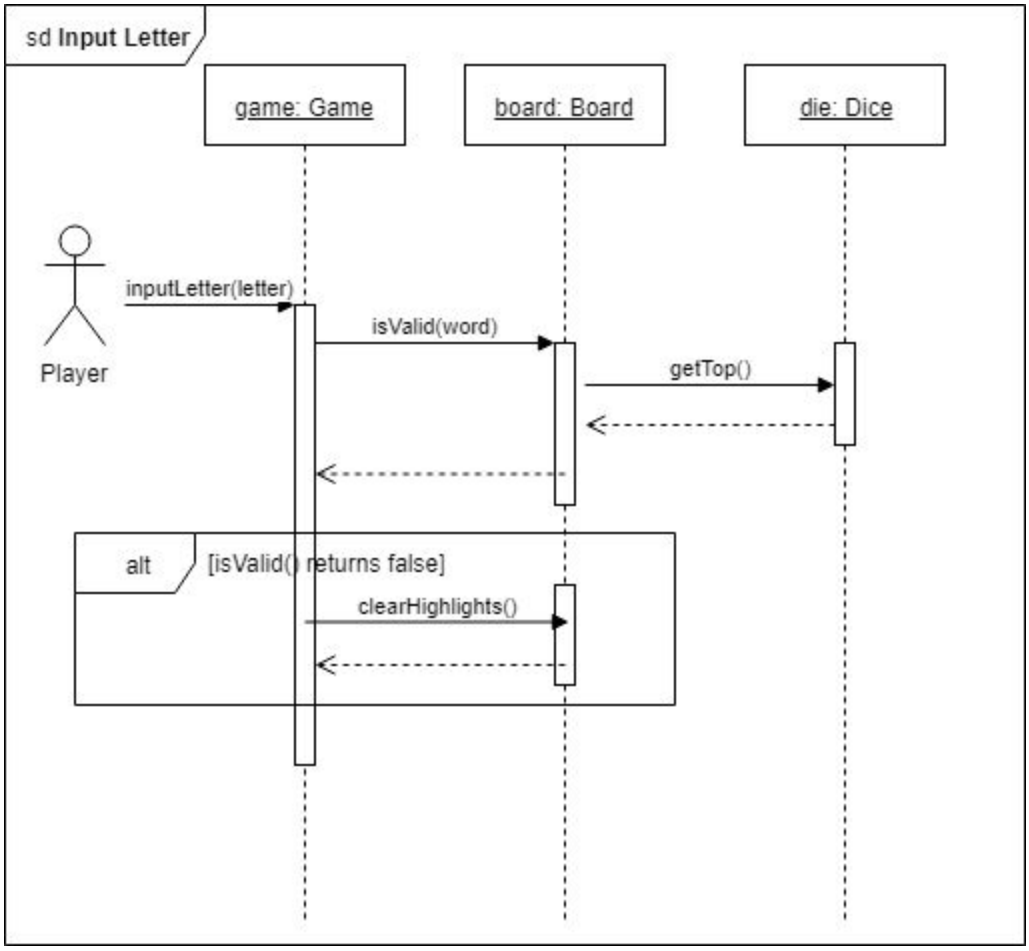
checks if submitted word is valid

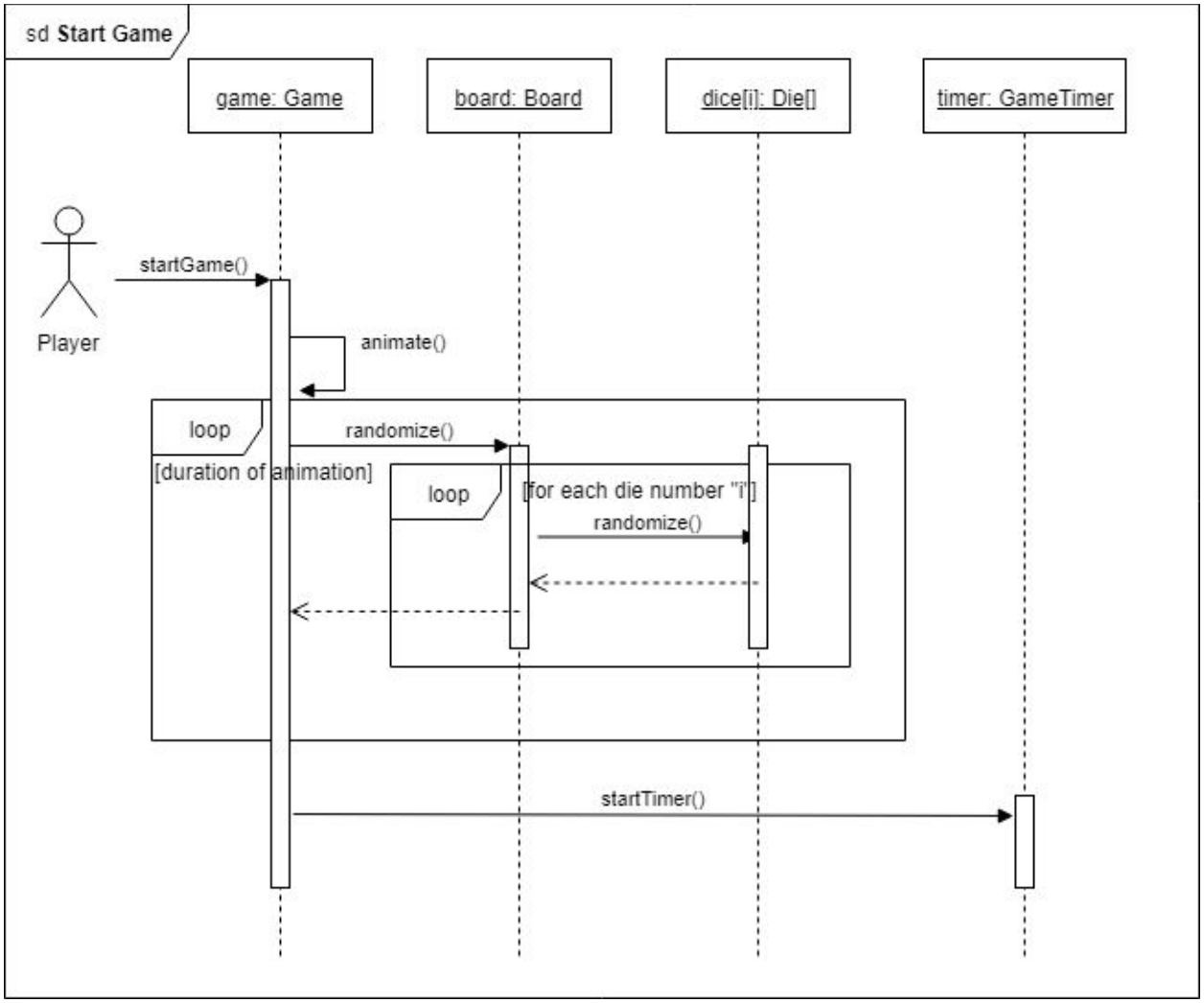
Dictionary's only attribute is the file with the words on it

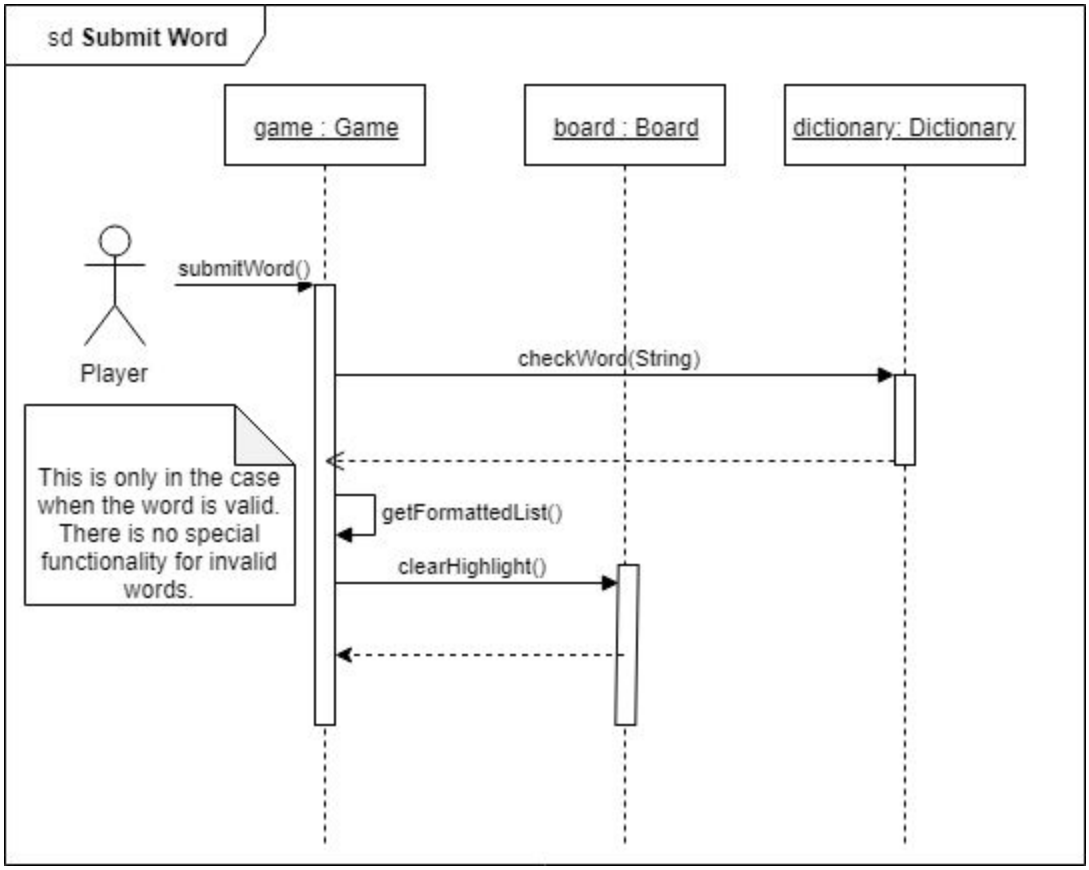
Dictionary depends on Game for the words that it needs to check

Detailed Design


UI
- game: Game
+ main(): void
Game
- board: Board
- dictionary: Dictionary
- score: int
- timer: GameTimer
- currentWord: String
- wordList: ArrayList<String>
+ addLetter(String): void
- animate(): void
- getFormattedList(): String
+ startGame(): void
+ submitWord(String): void
GameTimer
- timeRemaining: int
- timerLabel: JLabel
+ startTimer(): void
+ isRunning(): boolean
Dictionary
- wordsInDictionary: ArrayList<String>
+ checkWord(String): boolean
Board
- dice: ArrayList<Die>
- dieLabels: JPanel[]
+ randomize(): void
+ clearHighlights(): void
+ isLegal(String): boolean
DieLabelFactory
+ createDieLabel(int): JLabel
Die
- sides: ArrayList<String>
+ randomize(): String
+ getTop(): String


sd Input Letter
game: Game
board: Board
die: Dice
Player
inputLetter(letter)
isValid(word)
getTop()
alt
[isValid() returns false]
clearHighlights()


sd Start Game
game: Game
board: Board
dice[i]: Die[]
timer: GameTimer
Player
startGame()
animate()
loop
[duration of animation]
randomize()
loop
[for each die number "i"]
randomize()
startTimer()



Glossary

1. Boggle- Simple word game created by Hasbro