Boggle
CSCE361

Dylan Bonner, Ivy Chobanyan, Matthew Geary, Nga Pham

Architecture

MVC

We opted for an MVC architecture as opposed to iDesign due to the limited amount of data transfer making the setup of data contracts unnecessary.

Structure

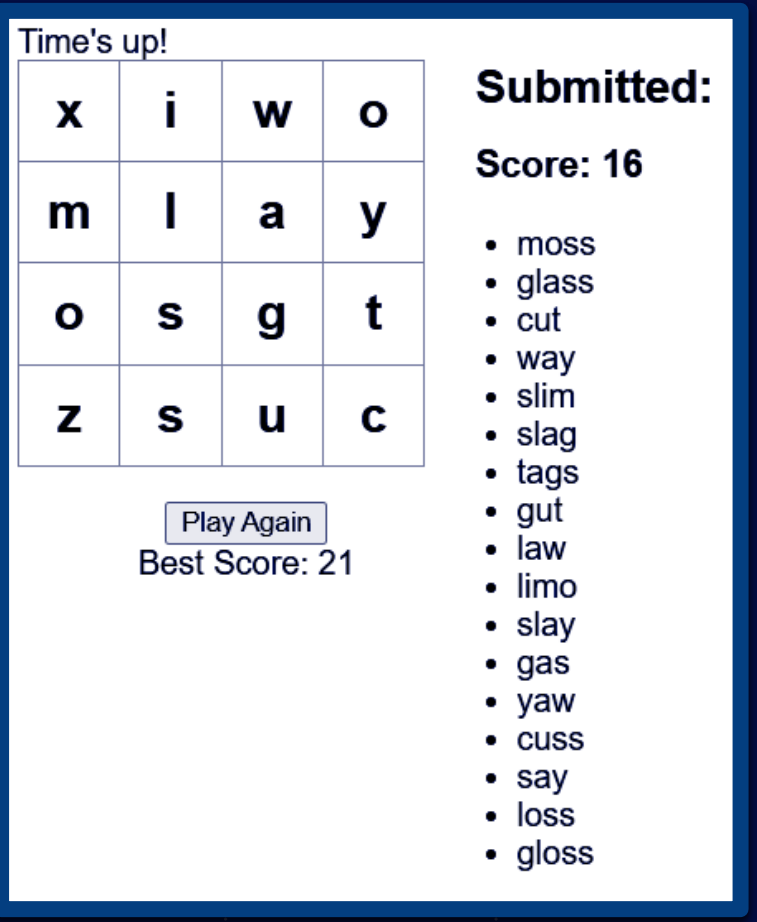
Our central Board Controller manipulates a number of models to contact the frontend view, which collects and constraints user input before sending it back to the controllers for validation.

Front End

Structure

Frontend created in React.js

Features a main Board class for computation and connection with backend, and several subclasses to handle interactive elements



Features

Base rules of boggle, words from adjacent tiles and 3 minute timer

Intuitive minimalist input method

Alert when low on remaining time

High-score tracking between playthroughs

Backend

Program.cs

This class contains the main method which opens the application in the browser, with the url http://localhost:3000

Die.cs

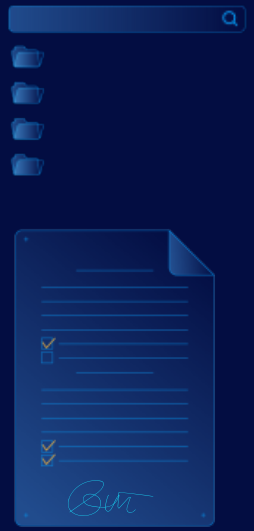
A Die is made up of a string, representing the 6 letters of the die, and an int which represents the index of the side of the Die which was “rolled”.

Board.cs

A Board is made up of a list of strings, representing the 16 Die in the board.

BoardController.cs

A BoardController communicates with the frontend and is a 2D array of Die. It has functions for creating a random board, checking if a given word exists in the dictionary files, and scoring a word.



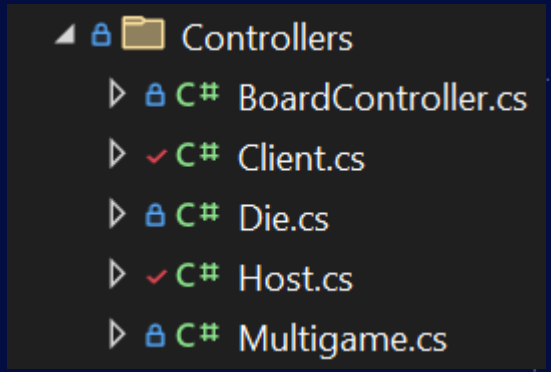
Backend

Client.cs

A model under Multigame which would track a client user and allow them to connect and interact with a host player.

Host.cs

A model under Multigame which would track a host user and allow them to coordinate a multiplayer session.



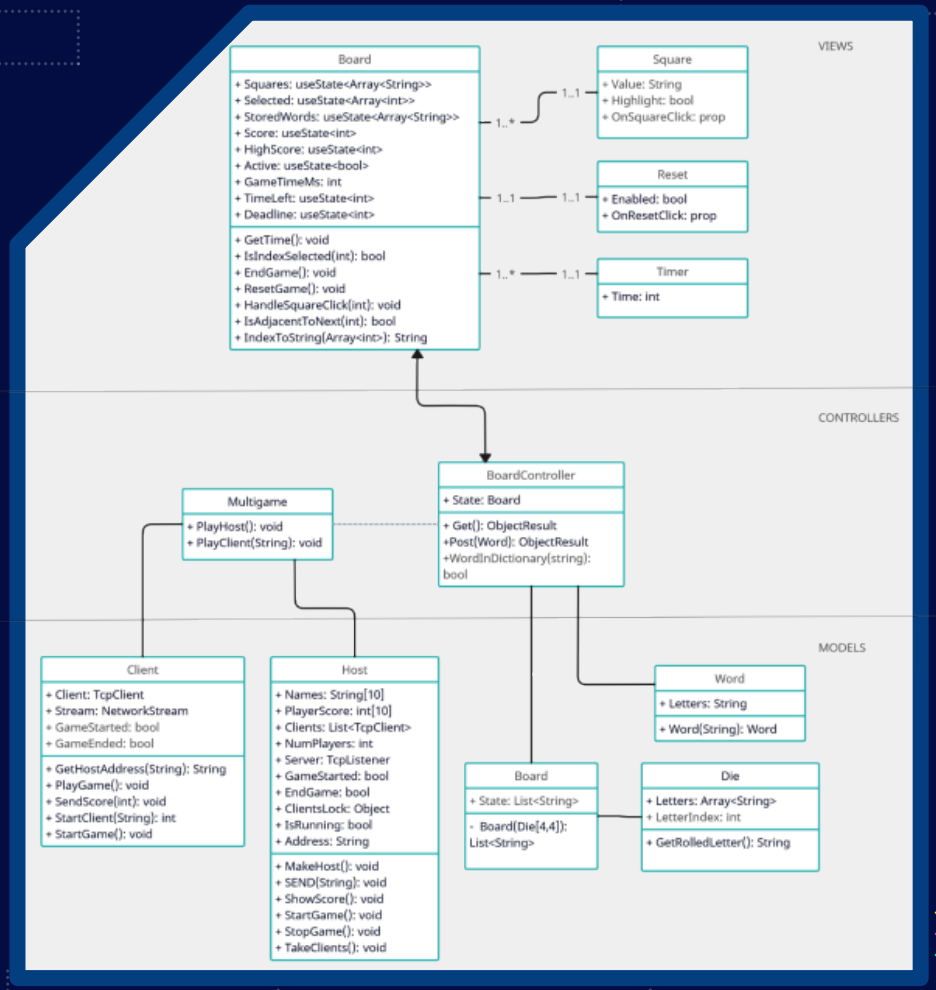
Multigame.cs

The controller managing multiplayer gameplay, which organizes host and client players to run a multiplayer session.

Word.cs

Contains a string that stores and manages the information associated with a word found by the board.

Backend


VIEWS
Board
+ Squares: useState<Array<String>>
+ Selected: useState<Array<int>>
+ StoredWords: useState<Array<String>>
+ Score: useState<int>
+ HighScore: useState<int>
+ Active: useState<bool>
+ GameTimeMs: int
+ TimeLeft: useState<int>
+ Deadline: useState<int>
+ GetTime(): void
+ IsIndexSelected(int): bool
+ EndGame(): void
+ ResetGame(): void
+ HandleSquareClick(int): void
+ IsAdjacentToNext(int): bool
+ IndexToString(Array<int>): String
1..*
1..1
Square
+ Value: String
+ Highlight: bool
+ OnSquareClick: prop
1..1
1..1
Reset
+ Enabled: bool
+ OnResetClick: prop
1..*
1..1
Timer
+ Time: int
CONTROLLERS
BoardController
+ State: Board
+ Get(): ObjectResult
+Post(Word): ObjectResult
+WordInDictionary(string): bool
Multigame
+ PlayHost(): void
+ PlayClient(String): void
MODELS
Client
+ Client: TcpClient
+ Stream: NetworkStream
+ GameStarted: bool
+ GameEnded: bool
+ GetHostAddress(String): String
+ PlayGame(): void
+ SendScore(int): void
+ StartClient(String): int
+ StartGame(): void
Host
+ Names: String[10]
+ PlayerScore: int[10]
+ Clients: List<TcpClient>
+ NumPlayers: int
+ Server: TcpListener
+ GameStarted: bool
+ EndGame: bool
+ ClientsLock: Object
+ IsRunning: bool
+ Address: String
+ MakeHost(): void
+ SEND(String): void
+ ShowScore(): void
+ StartGame(): void
+ StopGame(): void
+ TakeClients(): void
Word
+ Letters: String
+ Word(String): Word
Board
+ State: List<String>
- Board(Die[4,4]): List<String>
Die
+ Letters: Array<String>
+ LetterIndex: int
+ GetRolledLetter(): String

Database

SQL/NoSQL

We decided against a dedicated database due to the project's lack of persistent data requirements, as no user data needs to be stored between sessions.

Dictionary Files

Instead, we implemented a series of local word lists to verify words against, since these dictionary files are the only persistent data we need. These CSV files were made publicly available by Manas Sharma of bragitoff.com

https://github.com/manassharma07/English-Dictionary-CSV

Unit Testing

Testing BoardController, Client, Die, Host, and Multigame class

- Structure test cases using Arrange-Act-Assert
- Using MS Test frameworks
- All test cases passed

```
68      [TestMethod]
        | 0 references
69      public void wordInDictionaryTest()
70      {
71          // Arrange
72          string testWord = "word";
73          BoardController boardController = new BoardController();
74
75          // Act
76          var result = boardController.WordInDictionary(testWord);
77
78          // Assert
79          Assert.IsTrue(result);
80      }
81
82      [TestMethod]
        | 0 references
83      public void wordNotInDictionaryTest()
84      {
85          // Arrange
86          string testWord = "wodr";
87          BoardController boardController = new BoardController();
88
89          // Act
90          var result = boardController.WordInDictionary(testWord);
91
92          // Assert
93          Assert.IsFalse(result);
94      }
```

Diversity

The different abilities of everyone in our team largely determined how work was distributed.

- Ivy: frontend - prior experience with Javascript and UI design
- Dylan: middleware/backend
- Nga: unit testing
- Matthew: middleware - multiplayer functionality and unit testing

Everyone made contributions with design decisions and determining how to move the project forward.

Thanks!

GitHub repository:
https://github.com/DylanBonner/CSCE361-Boggle