# Wordle Clone

Team 10

Joe Hummer, jmhummer@ncsu.edu

Ben Morris, bcmorri4@ncsu.edu

Nick Sanford, nksanfor@ncsu.edu

Nick Schauer, njschaue@ncsu.edu

**NOTE**: Your final documents should **NOT** contain any << >>'s or any comments or suggestions given for each section.

# Introduction

Wordle is a simple five-letter word guessing game that rose to popularity in 2021. During its viral rise it even caught The New York Times Company eye who acquired it to challenge its millions of subscribers, and the general public, to guess the five-letter 'Wordle' of the day. Today, it is a daily routine for some to guess the word and see how long of a streak they can hold. The goal of this comprehensive exercise is to create a similar clone to the popular word guessing game.

The program will select a random word from a list of five-letter words for the player to guess. The player will then be given six tries to guess the word. After every guess, each letter is marked by blue, orange or black: orange indicating that the letter is correct and in the correct position, blue indicating the letter is in the word but in the wrong position, and black indicating that the letter is not in the word at all. Each five-letter guess must be a word otherwise the program will prompt the player to try again before accepting the guess and marking any letters.

This clone will be made for two players. Each player will have 5 rounds to try and guess a random five-letter word. If a player successfully guesses the word they will be awarded points based on how many guesses it took (more points for less guesses). The player with the most points at the end of the 5 rounds wins the game.

# Software Requirements

## Program Start

For normal game play (randomly selected wordles) the following command will execute the program:

- `$java -cp bin WordleGUI`

For the testing mode (predefined set of wordles) us the following command line argument '-1':

- `$java -cp bin WordleGUI -1`

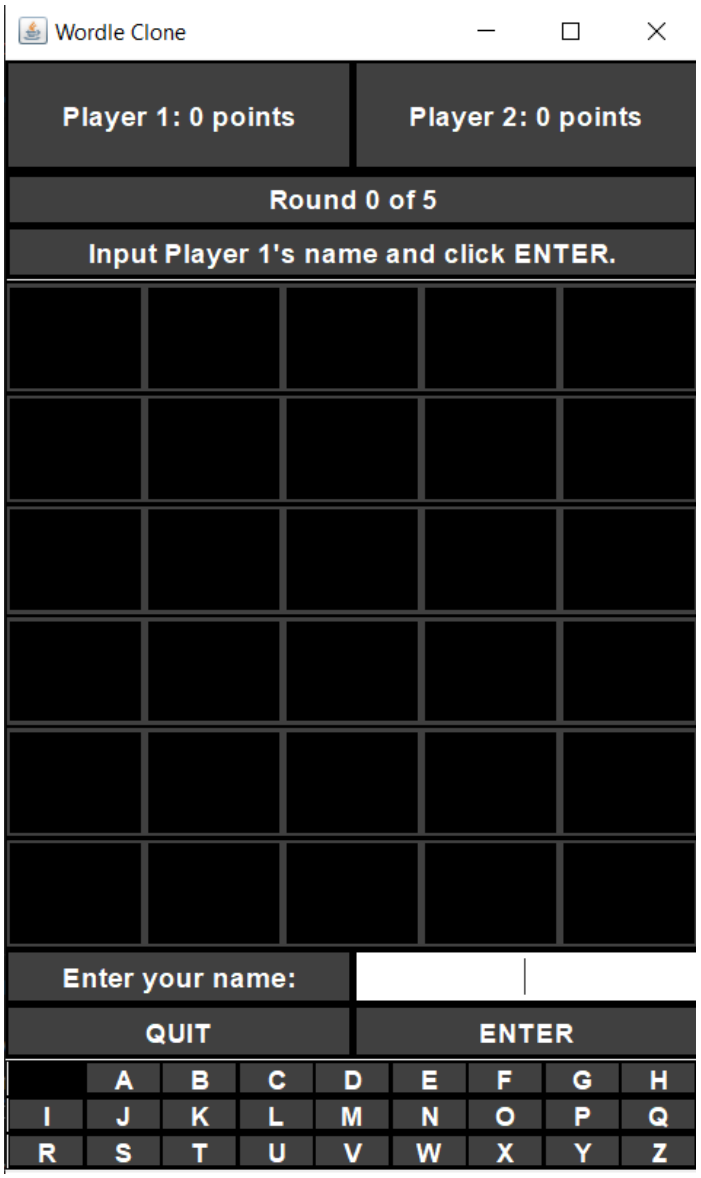
The following input files will need to be located in the 'words' folder. The location and names are hard coded into the Wordle object to be accessed.

- **WordleList.txt** - The list of five-letter words to randomly select the Wordle. This list has inappropriate words removed from it.
- **GuessList.txt** - The list of five-letter words to confirm a correctly spelled English language word.
- **TestList.txt** - A predefined list of Wordles used for testing proposes.

# GUI

Below is the general layout composed of 4 sections:

1. **Top Grid**: Scores, Rounds & Message Area
2. **Middle Grid**: Board
3. **Bottom Grid**: Inputs & Quit
4. **Alphabet:** Letters guessed and remaining




# Gameplay

## Start Menu:

The game will launch to the GUI shown above. The first player will be prompted to input their name.

- Top Grid:
  - **Scores**:
    - Player 1: 0 points
    - Player 2: 0 points
  - **Rounds**: Round 0 of 5
  - **Message Area**: Input Player 1's Name and Click ENTER.
- Middle Grid:
  - **Board**: 'blank' with all the tiles black and no letters.
- Bottom Grid:
  - **Input Prompt**: Enter your name:

    - **User Input**: ‘blank’ ready for user input
    - **Quit** and **Enter** buttons to enter input or quit the game.
- Alphabet:
    - **Alphabet:** in three rows of nine with the first box of the first row blank and black. The letters all have a DARK GRAY background.

## Player Name Input:

When the game starts each player will need to input their name before the game begins. The program will request Player 1’s name first, followed by Player 2’s name.

- Top Grid:
    - **Scores**:
        - [Player 1 or PlayerName]: 0 points
        - [Player 2 or PlayerName]: 0 points
    - **Rounds**: Round 0 of 5
    - **Message Area**: Input Player [player #]’s Name and Click ENTER.
- Middle Grid:
    - **Board**: ‘blank’ with all the tiles black and no letters.
- Bottom Grid:
    - **Input Prompt**: Enter your name:
    - **User Input**: ‘blank’ ready for user input
    - **Quit** and **Enter** buttons to enter input or quit the game.
- Alphabet:
    - **Alphabet:** in three rows of nine with the first box of the first row blank and black. The letters all have a DARK GRAY background.

## Players, Rounds and Guesses:

When the game commences (after player name inputs) the round will start at 1. After each player will have a chance to guess at a Wordle. After each player’s guess, the score will update, and after each round, the round will update. The message area will provide the player a prompt on how to continue the game. After each guess, the guess will show up with the letters indicating if the letter guessed matches the target word in the correct place (orange), if the letter guessed is in the target word but in the wrong position (blue) or if the letter guessed isn’t in the word at all (black). The alphabet at the bottom matches the cumulative guesses.

- Top Grid:
    - **Scores**:
        - [PlayerName]: [##] points
        - [PlayerName]: [##] points
    - **Rounds**: Round [##] of 5
    - **Message Area**: As the game progresses the message area will have the possible prompts below.
        - [PlayerName]: Guess the Wordle.
            - Start of turn or guess again.
        - [PlayerName]: Must be a 5 letter word, try again.
            - Incorrect input, must be a 5 letter word.
        - [PlayerName]: Not a word, try again.
            - Incorrect input, not an english word.
        - [PlayerName]: Out of guesses. Click ENTER to Continue.
            - Out of guesses to guess Wordle, advance to the next player.
        - [PlayerName]: Correct! Click ENTER to Continue.

            - Correct guess, add score, and advance to the next player.
        - Game Over. It is a TIE!
            - Game resulted in a tie.
        - Game Over. [PlayerName] WINS!
            - Game resulted with [PlayerName] winning.
- Middle Grid:
    - **Board**: if the guess is a 5-letter word the board will display each letter of the word in its own box and the corresponding background color indicator (right sport, right letter, right sport and letter).
- Bottom Grid:
    - **Input Prompt**: Enter your guess:
    - **User Input**: 'blank' ready for user input
    - **Quit** and **Enter** buttons to enter input or quit the game.
- Alphabet:
    - **Alphabet:** After each guess, the cumulative guesses will show up with the letters indicating if the letter guessed matches the target word in the correct place (orange), if the letter guessed is in the target word but in the wrong position (blue) or if the letter guessed isn't in the word at all (black).

# Rules

- This is a two player game.
- The game has five rounds.
- Each round, each player will guess at a Wordle.
- Each round, each player will score points based on the scoring system below.
- Each Wordle will be unique from the WordleList (no Wordle can be randomly selected more than once).
- Each player gets six guesses at their Wordle.
- Each guess must be a valid five-letter word (check against the GuessList).
- After each guess, if it is not the Wordle, the color of the tiles/letters will change to show how close your guess was to the Wordle.
    - **Orange** indicates the letter is in the word and in the correct location.
    - **Blue** indicates the letter is in the word but in the wrong location.
    - **Black** indicates the letter is not in the word in any location.

# Score

If the player correctly guesses the Wordle, they will receive points based on the number of guesses needed (listed below). Zero points are awarded if the player is unable to guess the word in six tries. The cumulative score will be computed after each round for each player. The player at the end with the most points wins.

1. Correct guess on **first** guess: 60 pts
2. Correct guess on **second** guess: 50 pts
3. Correct guess on **third** guess: 40 pts
4. Correct guess on **fourth** guess: 30 pts
5. Correct guess on **fifth** guess: 20 pts
6. Correct guess on **sixth** guess: 10 pts

# Testing Mode

Test mode can be activated by entering “-1” as a command line argument directly after the java file name (e.g. “`$java WordleGUI -1`”). The test mode differs from the standard mode in that it reads a separately prepared list of 10 words (titled TestWords.txt) in order instead of reading GuessWords.txt randomly. This allows predictable system testing of an entire Wordle game, including 5 words for each player. TestWords.txt will contain the following words in this order:

shape, store, round, storm, boxed, fruit, grape, sweet, berry, grant

The Wordle GUI and program will be identical in all other aspects, such that the target words for each player in each round are as shown below:

| **Round** | **Player 1 Target Word** | **Player 2 Target Word** |
|---|---|---|
| 1 | shape | store |
| 2 | round | storm |
| 3 | boxed | fruit |
| 4 | grape | sweet |
| 5 | berry | grant |

# Software Design

## WordleGUI

### Instance Variables

**private JLabel playerLabel[]:** Label for players' names and scores

**private JLabel roundLabel:** Label for listing the current round

**private JLabel messageLabel:** Label for the message panel

**private JLabel letterLabel[][]:** Displays the letters of the guesses

**private JLabel directionsLabel:** Directions for the next input

**private JLabel alphabetLabel[][]**: Labels for the alphabet panel

**private JPanel mainPanel**: Panel within which to combine all other panels

**private JPanel panelOne**: Panel that includes the two Player labels

**private JPanel panelTwo**: Panel that includes the round and message labels

**private JPanel guessPanel[]**: Panel that shows all of the guesses

**private JPanel bottomPanel**: Panel that shows directions, blank text box, quit button, and enter button

**private JPanel alphabetRowPanel**: Panel containing one row of the alphabet section

**private JPanel combinedAlphabetPanel**: Panel combining the rows of the alphabet section

**private JTextField guessTextField**: Text field for entering player names and guesses

**private JButton enterButton**: ENTER button

**private JButton quitButton**: Quit button

### Constructors

**public WordleGUI(int testFlag)**: Builds Wordle GUI in 'start game' state. Labels at top to provide player, game, and instruction information. A board in the middle to display guesses and indicators per each letter, an input area, with an enter and quit button at the bottom, and an alphabet at the bottom to show letters guessed and remaining. The testFlag is passed to the Wordle class. This flag is used to designate if the program should load/run in 'test mode' but the value of -1.

### Instance Methods

**public void updateGUI():** Uses WordleGame.next(input, message) in order to progress the game. Also updates all of the labels in the GUI.

**public void actionPerformed(ActionEvent e):** When the enter button is pressed, updateGUI will be called. When the quit button is pressed the program exits immediately.

**public void keyPressed(KeyEvent e):** When enter on the keyboard is pressed, updateGUI will be called.

**public static void main(String[] args):** Starts the Wordle game by referencing the constructor to build the GUI and passing testFlag if ‘test mode’ is applicable. A try/catch is used to validate any command arguments that may be misused.

**public static void main(String[] args):** Starts the Wordle game

### Static Method Headers

**public static void main(String[] args):** Starts the Wordle game by referencing the constructor to build the GUI and passing testFlag if ‘test mode’ is applicable. A try/catch is used to validate any command arguments that may be misused.

## WordleGame

### Public Class Constants

**public static final int NUMBER_OF_PLAYERS = 2**: Number of players in game.

**public static final int NUMBER_OF_ROUNDS = 5**: Number of rounds in game.

**public static final int NUMBER_OF_GUESSES = 6**: Number of guesses per player in a round.

**public static final int NUMBER_OF_LETTERS = 5**: Number of letters in a wordle.

### Instance Variables

**private int currentPlayer**: Keeps track of current player in game.

**private int currentRound**: Keeps track of current round in game.

**private int currentGuess**: Keeps track of current guess for player in game.

**private Player[] player**: Creates player objects depending on how many players.

**private Wordle wordle**: Creates wordle object to check user inputs against.

**private Board board**: Creates board object to update GUI for letters and indicators.

### Constructors

**public WordleGame(int testFlag)**: Starts the game and initializes all the necessary objects. Creates a Wordle object and passes the testFlag depending on if the ‘test mode’ is wanted. Sets the first answer to be used as the ‘Wordle’. Creates a Player[] array of Players to store/access the different player’s information in the game. Each player is initialized in the player array. Creates the Board object to house the board letters/colors during gameplay. And starts the instance variables currentPlayer = 1 (player 1 starts first), currentRound = 0 (round zero collects player info before beginning round 1) and currentGuess = 1 (first guess starts at 1).

### Instance Methods

**public void next(String input, String oldMessage)**: Holds the logic to the Wordle game. Based on the user input and oldMessage being displayed the next phase of the game will advance. This method will keep track of the current player, round and guess as the game continues. A return message will be provided to prompt the player to the appropriate next step in the game.

**public int getCurrentRound()**: Returns the current round of the game.

**public Player[] getPlayer()**: Returns the players object array to get player specific information.

**public Board getBoard()**: Returns the board object to access all the board information.

**public int getCurrentGuess()**: Returns the current guess in the game.

## Wordle

### Instance Variables

**private int test**: Indicates if program is in test mode or standard mode

**private int index**: Index of GuessList array from which to pull the target word

**private String wordle**: Target word for user to guess

**private String[] wordleList**: Stores all of the possible words that could be selected as the Wordle (the target word to be guessed), populated from a text file.

**private String[] guessList**: Stores all of the words that are valid guesses, populated from a text file.

### Constructors

**public Wordle(int test) {}**: Reads the .txt files to create arrays of words. Three total .txt files can be read (GuessList.txt, WordlesList.txt, and TestList.txt) but only two will be read upon execution of the method. If the program is in test mode, TestList.txt is read as the list of target words, otherwise WordleList.txt is read as the list of target words.

**public void newAnswer() { }** : Randomly assigns a target word for a Wordle game. The target word is picked randomly from the wordleList array (constructed from the WordleList.txt file) and assigned to the **wordle** instance variable. The given **index** on the array is then replaced with a “null” value. If the **index**

value randomly chosen corresponds to a “null” value on the array, the **index** value will increase by 1 until a non-”null” value is found before assigning this value as the **wordle** instance variable.

If the **test** instance variable is set to -1 the program will instead assign the **index** instance variable to 0 and **wordle** will be assigned to the first value on the wordleList array (constructed from the TestList.txt file). The **index** will then increase by one each time the method is called. Throws an **IllegalArgumentException** with the message “TestList array length exceeded” if the **index** value in test mode is greater than the wordleList array length.

### Instance methods

**public String getAnswer():** Returns the target word produced by newAnswer()

**public boolean isWord():** Searches the GuessList array of possible words produced by Wordle and indicates if a guess is a word

**public boolean isAnswer():** Determines if the input word is the target word

**public void createWordleListArray(String wordleFile)**: Initializes the **wordleList** array with Strings read in from either the WordleList.txt file or the TestList.txt file if testing mode is on.

**public void createGuessListArray(String guessFile)**: Initializes the **guessList** array with Strings read in from the GuessList.txt file.

## Player

### Public Class Constants

**public static final int STARTING_POINTS = 60**: The amount of points each player receives in a new round

**public static final int POINT_COST_PER_GUESS = 10**: The amount of points subtracted from the players score per guess

**public static final int MINIMUM_GUESSES = 1**: The minimum number of guesses a player can make to win a round

**public static final int MAXIMUM_GUESSES = 6**: The maximum number of guesses a player can make to win a round

### Instance Variables

**private int score:** The current player’s score

**private String name**: The player’s name

### Constructors

**public Player():** Sets the player name to null and score to zero

### Instance Methods

**public int getScore()**: Returns the player's current score

**public void addScore(int currentGuess)**: Adds the appropriate amount of points to the player's score depending on the value of **currentGuess** (60 for 1 guess, 50 for 2 guesses, etc.). **Throws** an IllegalArgumentException if the number of guesses (**currentGuess**) is greater than 6 or less than 1.

**public String getName()**: Returns the Player's name

**public void addName()**: Assigns or replaces the player's name

## Board

### Public Class Constants

**public static final int NUM_LETTERS_IN_ALPHABET = 26**: 26 letters in the alphabet

**public static final int CAPITAL_A_ASCII_VALUE = 65**: The ASCII value of the character 'A'

### Instance Variables

**private char[][] letters**: A 2D char array representing all the guesses made so far. Each row represents the word guessed, and each column within the row is a char representing the letter in that position in the word. Note: the default initialization of a char is '\\u0000'.

**private String[][] colors**: A 2D String array representing all the colors associated with each guess made so far. Each row contains a String array where each element is the String of the color associated with that particular letter. These color Strings can be "orange" to indicate the letter is in the correct position, "blue" to indicate that the letter is in the word, but is in the incorrect position, or "black" to indicate that the letter does not exist in the word.

**private String[] alphabetColors**: A String array to hold the colors of the alphabet letters for the purpose of displaying those colors on the corresponding keys on the GUI. The String "black" indicates that the letter does not appear in the target word, the String "blue" indicates that the letter does appear in the target word, but was guessed in the incorrect position, and the String "orange" indicates that the letter was guessed in the correct position. All values in this array will be initialized to "dark gray" to indicate that the letter has not yet been guessed.

**private int currentGuesses**: The number of guesses made so far and currently stored in the **Board** object. This is also used as the index to store the next guess into the letters and colors arrays.

**private String correctPositionColor**: The color "orange" which will be stored in the **colors** array. This color represents that the guessed letter is in the same position as the letter in the target word.

**private String incorrectPositionColor**: The color "blue" which will be stored in the **colors** array. This color represents that the guessed letter exists in the target word, but is not in the correct position.

**private String notInWordColor**: The color “black” which will be stored in the **colors** array. This color represents that the guessed letter does not exist in the target word. This is also the value to which all elements in **colors** are initialized.

**private String notGuessedColor**: The color “dark gray” which represents that a letter has not been guessed yet.

**private char[] targetWordArray**: The char array representation of the word that the user is trying to guess, the “wordle”.

## Constructors

**public Board (int numTotalGuesses, int wordLength, String targetWord)**: Creates a new **Board** object by initializing the **letters** and **colors** arrays to both have a number of rows equal to the total number of guesses that a player can make, and a number of columns equal to the length of each word. Initializes the **targetWordArray** using the **targetWord** argument.

## Instance Methods

**public void addGuess(String guessWord)**: Adds a char array of the **guessWord** to the next row in the **letters** array. Adds a String array of the colors of each letter in the guessed word to the **colors** array. Sets each letter’s color to “orange” if it is in the correct position, sets each letter’s color to blue if it is in the word but in the incorrect position, and keeps the letter’s color black otherwise. Throws an **IllegalArgumentException** if the length of the referenced arrays is greater than the length of the 2D arrays’ row length.

**public char[] getGuessLetters(int index)**: Returns a reference to the char array at the specified index within the **letters** array.

**public String[] getGuessColors(int index):** Returns a reference to the String array at the specified index within the **colors** array.

**public String toString()**: Returns a String representation of the **Board** object. For each guess stored in the **letters** array/**colors** array, two lines are added to the String. The first line contains the word stored in the letters array for that row, while the second line contains the colors for each letter, separated by a space. For instance, if the **letters** array and **colors** array only have one guess stored, the output might be: happy\ngreen yellow green gray gray \n

**public int getCurrentGuess()**: Returns the number of guesses made so far.

**public char[][] getGuessLettersArray()**: Returns the array of all guesses made.

**public String[][] getGuessColorsArray()**: Returns the array of the background colors of letters in all of the guessed words so far.

**Public String[] getAlphabetColorsArray()**: Returns the array of the background colors for the letters in the GUI keyboard.

# Implementation

Object-Oriented Programming will be used to divide the Wordle Clone into different objects encapsulating data and performing tasks. The WordleGame (client) is used to execute the overall logic of gameplay while calling on the Wordle, Player, and Board classes. Wordle is used to store, change and test if a player has guessed the Wordle. The Player class is used to store the player's name and score, while the Board class is used to store the letter and indicator based on the 'taken' player guesses. The WordleGUI is used to provide the interactive user interface during the gameplay calling on the WordleGame to update and display the current game results and prompts.

Other concepts being used:

- WordleGUI
    - The program must exit (stop) as soon as the user clicks the Quit button, System.exit(1).
    - Use GridLayout to layout the GUI.
    - Use keyListener with keyPressed and actionListener with actionPerformed in order to detect ENTER and QUIT
    - Use @override to override base methods and run child methods
    - Use switch statements to translate Strings of colors into background colors for guess letters and the alphabet letters guessed and remaining
- WordleGame
    - Object-Oriented Programming, WordleGame being the client using the Player, Wordle, Board objects to get/set data to keep the logic clean, organized and easy to follow.
- Wordle
    - Use the Random() class to randomly select a new wordle in the **newAnswer()** method.
    - Use a Scanner() to input the WordleList.txt, GuessList.txt, and TestWords.txt files into individual arrays in their individual **Create()** methods. Scan each word using line-based processing, Scanner lineScanner = new Scanner(line);.
    - java.io.* used to create FileInputStream() and FileNotFoundException when using a Scanner() to read words into an array.
    - Use IllegalArgumentException when an array that is being accessed sequentially (i.e. the array created from TestList.txt) has an index value that exceeds the array length
- Player
    - Use IllegalArgumentException when the amount of guesses is out of bounds
    - Use a For loop to determine the number of points a player receives in a round
    - Use Getters and Setters for the player name and score
- Board
    - Throws an IllegalArgumentException when invalid arguments are passed to the constructor method or to the addGuess method.
    - The addGuess() method contains logic to determine, for each letter of the guessed word, whether that letter is in the correct position, in the incorrect position in the word, or is not in the word at all. This implementation involves keeping track of how many instances there are of each letter in the target word ("wordle"), and decrementing when those letters are found in the guessed word. The first pass determines whether the letter is in the correct position, then another pass is made to determine if another instance of the letter exists in the word in an incorrect position.

# Testing

**System Testing**

- The system testing of the Wordle Clone program should be performed according to the SystemTestPlan_CE.pdf document located in the project-docs folder.

**Unit Testing**

- No unit test files or documentation are required this semester, so this can be omitted.
- BoardTest.java and WordleTest.java were added as just a trial on some unit testing.

# Team Reflection

Overall the project went relatively smooth. We would tackle and try to divide up the portions equally for each checkpoint. Obviously, nothing is perfect and some portions took longer than others as we faced challenges or people wanted to take on more challenging tasks in general.

Probably one of the more difficult parts was trying to balance the communication and our workload along with each team member's busy life schedules. We used our team meetings to walk through the current task and discuss any differences or talking points we needed to decide on. Google Docs was an excellent way for us to post comments and have each person weigh in on a decision to be made. We regularly kept communication through a group email chain updating the team on any changes or talking points.

The process of developing our software design and implementation plan prior to coding allowed us to have a clear work stream once it was time to create .java programs. Because of this, we were able to prioritize which classes needed to be completed first in order to allow testing/functionality for other classes. Minor issues were seen with a branch divergence when using github but for the most part we worked in separate .java files which allowed everyone to make progress without any conflicts.

In implementing the WordleGame class, certain unexpected issues regarding class interaction became apparent. This is to be expected when trying to merge work from different team members on any project. Nick Schauer put a lot of time and effort into smoothing over these issues and creating a working program for the GUI to reference. His involvement in creating this class meant that he had the most extensive knowledge of the overall program and was able to assist the most with questions and problems from other group members.

The GUI implementation was the most burdensome portion of the project. Ben took this challenge head on, and did an excellent job, really tying everything together. Our individual portions, due to the software design planning phase, were pretty much complete but during the GUI implementation we found additional portions needed to be added or updated so everything worked. The team would update their individual portions as needed, or Ben would make some changes on the fly for the sake of time.

The CE assignment was a true teamwork effort/challenge. The time provided to complete the assignment felt right, we had vacations/moving/weddings we all were able to balance while stepping through each checkpoint. Luckily our group was very open, sharp and willing to meet and step up to do their portion.

We all appreciated that we had a variety of topics/games to choose from to suit our interests at the beginning of the project and were welcome to present a unique project topic. Giving us the option to do a game that has become so popular also allowed us to share it with friends and family and motivated us to create a polished final product.