Yahtzee

Final Report

Full House

Suyash Kushwaha, Connor Whyte, Talia Frausto, Nathan Bashant-Coon

Course: CPSC 224 - Software Development

Table of Contents

I. Introduction and Project Description **3**

II. Team Members - Bios and Project Roles **4**

III. Project Requirements **5**

IV. Solution Approach **10**

V. Test Plan **16**

VI. Project Implementation Description **19**

VI. Future Work **24**

VII. Glossary **24**

VIII. References **24**

I. Introduction and Project Description

Our final project is a playable GUI form of the well-known board game Yahtzee. This form of the game Yahtzee has been coded using Java and other tools to create the GUI interface. The final Yahtzee GUI displays, but is not limited to, the current 5 6-sided dice hand as images in a frame. The user can click a button to display the current state of a scorecard including bonuses and totals. Additionally, the user can click on a control widget, a checkbox, or an image of a die, to select the dice and click a button to initiate a roll or reroll. The game then displays the possible scores of a hand in a frame and is able to click on a control to select a line to place the score. The possible scores additionally only include unused scorecard lines. Our final version continues the Lizard Spock Yahtzee rules by allowing the user to choose the number of dice. We couldn't make the number of die sides customisable due to some technical reasons.

Selections are made by clicking on the category. In order to end a game, and in turn win, the player must fill in all 13 category boxes. Now, the player must add up his or her score as follows. The upper section total is recorded, and a 35-point bonus is added if you scored 63 points or more. The lower section total is recorded in the total box. The grand total is then calculated by adding the upper and lower section totals together. The player can win the game by having the highest Grand Total score.

The purpose of this document was to understand how we completed this as a team but also understand how breaking down a project like this helps with errors and helps the project management understand our thought process. Throughout the process of this project we worked on reports to break down what we were doing and in the end gave us a good outline and order of how we should approach this project. It breaks down the train of thought and why we thought this would work. It was a slow but rigorous process with merge errors and communication issues but helped us in the end to be better software developers.

II. Team Members - Bios and Project Roles

Connor Whyte is a computer science student who knows C++, java, and python, and is interested in learning JS and HTML/CSS. He hasn't really done many personal projects yet in his computer science career but is interested in coding his own discord bot in an effort to learn JavaScript. For this project, he looked into how to create animations in java swing, something few groups tried to do, and eventually settled for creating a clean dice roll animation for the gameplay window that tied into Talia's theme well.

Suyash Kushwaha is a computer science student interested in web development and anything computer-related. His personal projects involve various node.js projects, some full-stack projects and an android application that scrapes media files from various websites. Suyash is very proficient with JavaScript. But in the last few years, he has gotten somewhat used to Java and C++. In this project, Suyash's code was used as the starting point. Also, he was responsible for implementing the CPU players, merging pull requests and resolving merge conflicts. He also fixed any errors that arose from merging the pull requests.

Talia Frausto is a computer science student interested in web development, app development and graphic design. Her personal projects involve various little websites and graphics. Talia is proficient in HTML and somewhat in C++ and Java. In this project, her responsibilities were to update the GUI and make it more user friendly. She created sketches and outlines to make it easier for the teammates to visualize the game design.

III. Project Requirements

Part	Settings screen
Priority	HIGH
Purpose	The settings screen have customizable field to enter preferred die count, player count, and reroll count. The player will change this as they see fit and then start the game.
Inputs / Needs	This will take die sides, a number of dies, and player count as the input.
Operators / Actors	Handanalyzer, scoresheet, settingsWindow, roundedLabel, roundedPanel
Outputs	All the inputs will be passed on to other objects to configure the game appropriately.

Part	Home screen
Priority	HIGH
Purpose	The home screen will allow to play or change the settings.
Inputs / Needs	No inputs
Operators / Actors	homeWindow, roundedLabel, roundedPanel
Outputs	No outputs for this one, just links to the settings menu that will then link to the gameplay.

Part	Game Screen
Priority	HIGH
Purpose	This will have the main gameplay section and will display all the dice roles and will allow players to decide how many times they want to reroll.
Inputs / Needs	Takes the number of players, CPU players, and DICE as inputs in order to generate the game environment.
Operators / Actors	Handanalyzer, scorecard, leaderboard
Outputs	Outputs the results of each dice roll so handanalyzer can compute the possible scores for the scorecard.

Part	DICE
Priority	HIGH
Purpose	A variable number of dice objects will be printed onto the game screen depending on how many dice were selected in settings before playing.
Inputs / Needs	Number of dice from settings
Operators / Actors	Handanalyzer, scorecard
Outputs	Outputs the side that each die landed on so handanalyzer can calculate totals.

Part	Scoresheet
Priority	HIGH
Purpose	This will be an in-game viewable scoresheet to keep track of the current player’s points.
Inputs / Needs	Takes in scores from handanalyzer and stores them permanently. Also keeps track of
Operators / Actors	customTableModel, roundedButton, roundedLabel, player, handAnalyzer, Scoresheet
Outputs	Total score after each turn to the leaderboard.

Part	Scoreboard
Priority	HIGH
Purpose	This will be an in-game viewable scoreboard to keep track of all players points. Will update after each round.
Inputs / Needs	Takes in total score for each player and CPU player from the scoreboard and displays them in descending order.
Operators / Actors	customTableModel, roundedLabel, player, Scoresheet

Outputs	Outputs the winning player’s name to be displayed on the victory screen.

Part	GameOverScreen
Priority	MEDIUM
Purpose	This will have a celebration for the winner and a scoreboard to display the total points of all players.
Inputs / Needs	This will simply regurgitate the scoreboard data.
Operators / Actors	customTableModel, roundedButton, roundedLabel, handAnalyzer, scoreSheet
Outputs	null

Part	CPU player
Priority	MEDIUM
Purpose	For beginner players/testing.
Inputs / Needs	none
Operators / Actors	Every object in the game is involved, as the CPU players run through the entire game as a normal player would.
Outputs	Outputs scores to the scorecard and eventually leaderboard to compare to the real players.

Part	GUI classes for all functional objects
Priority	HIGH
Purpose	To bring the functional game to life and allow easier play.
Inputs / Needs	Java swing

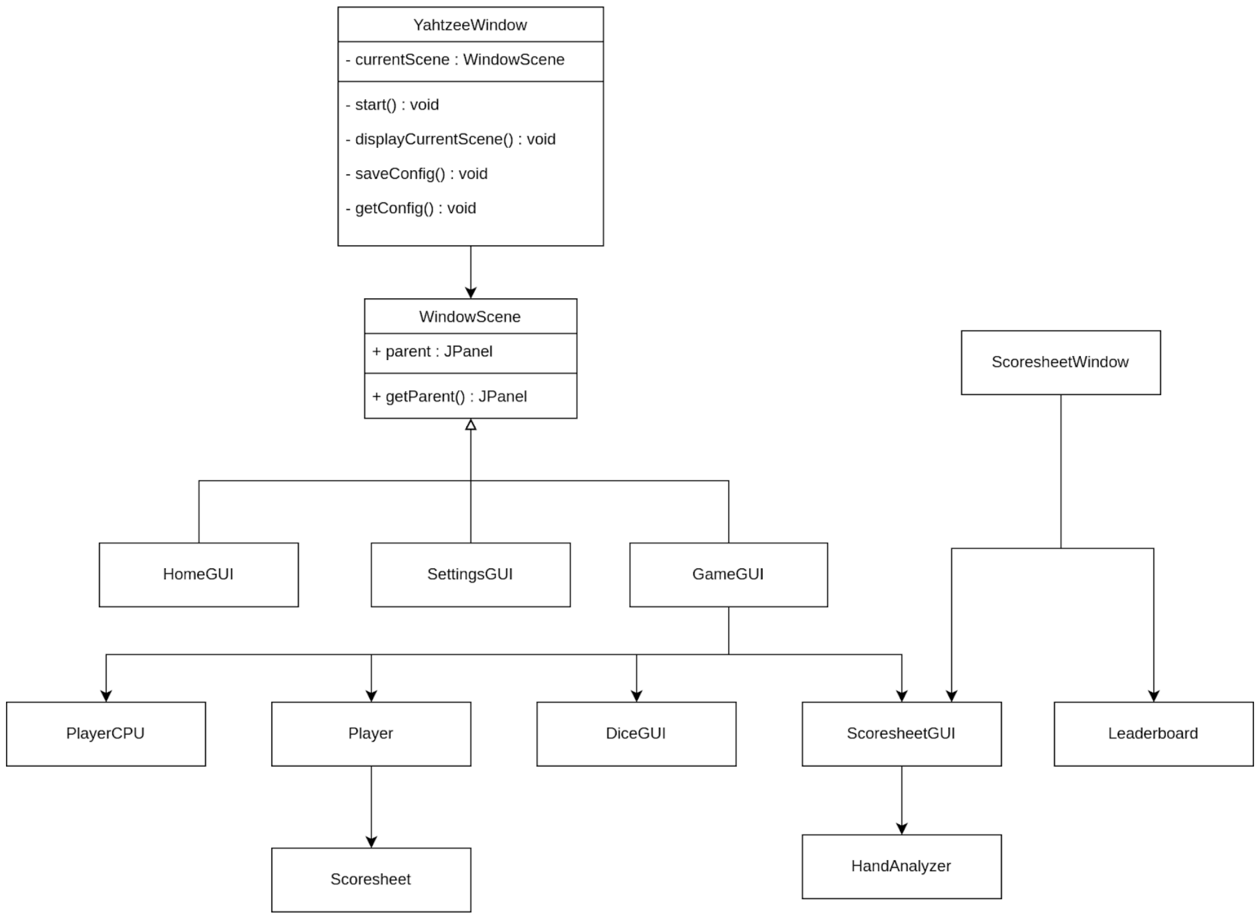
Operators / Actors	All classes
Outputs	null

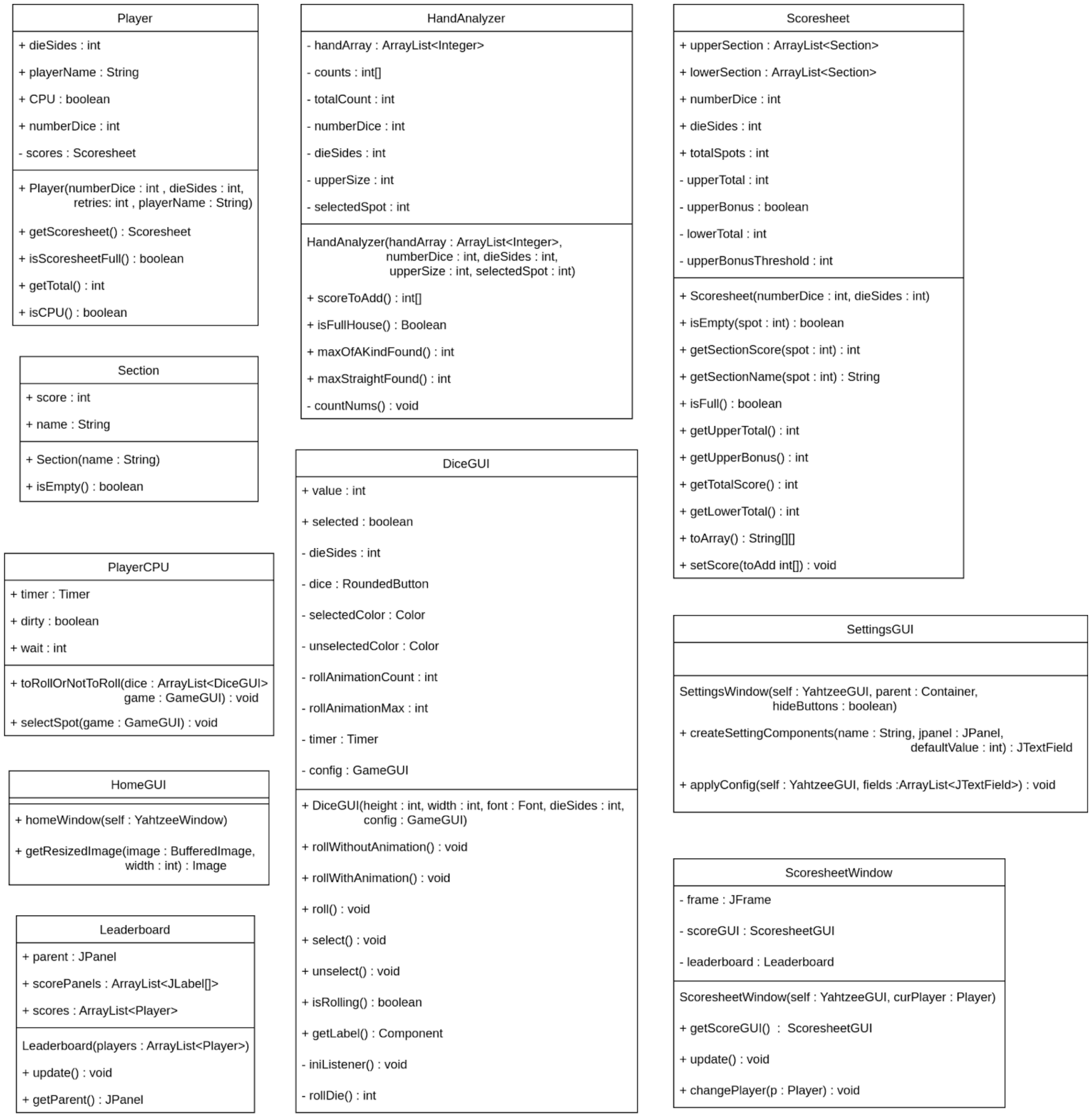
Part	Animations
Priority	HIGH
Purpose	To bring the functional game to life and create a cleaner, more professional looking project.
Inputs / Needs	Java swing, gif images, png images.
Operators / Actors	diceGUI, gameGUI, roundButton, roundpPanel
Outputs	null

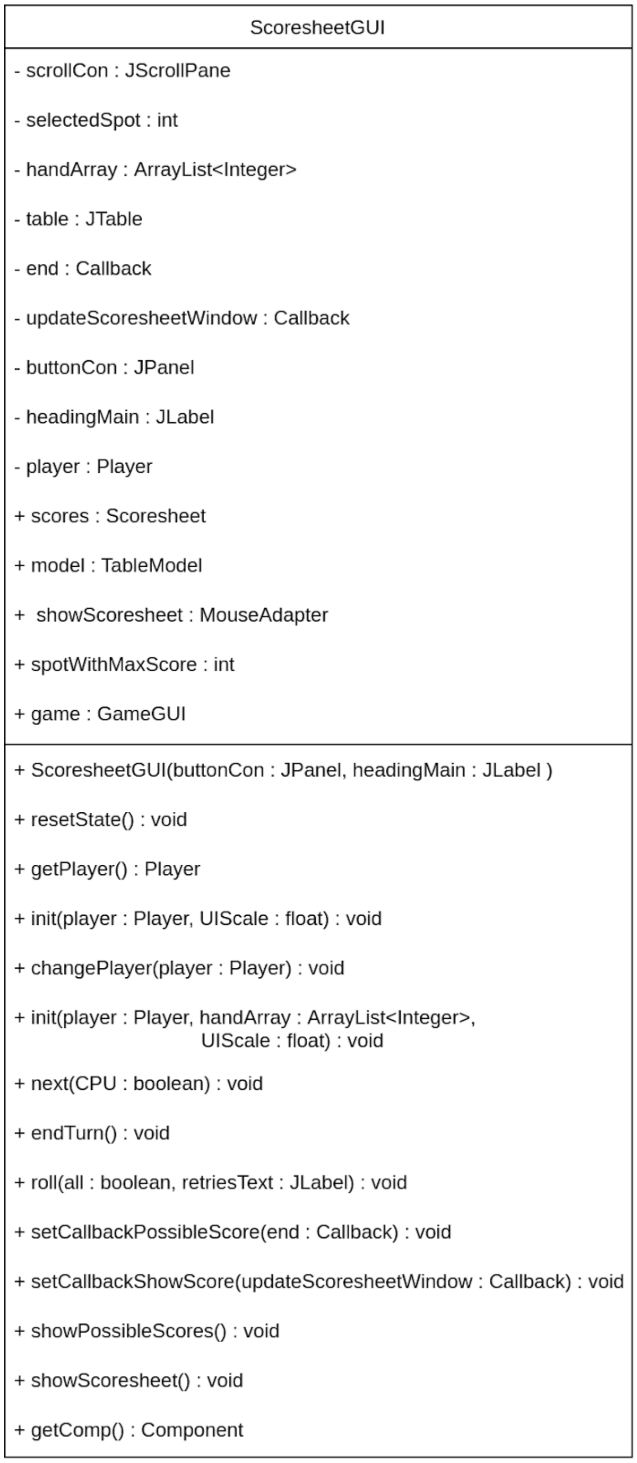
IV. Solution Approach

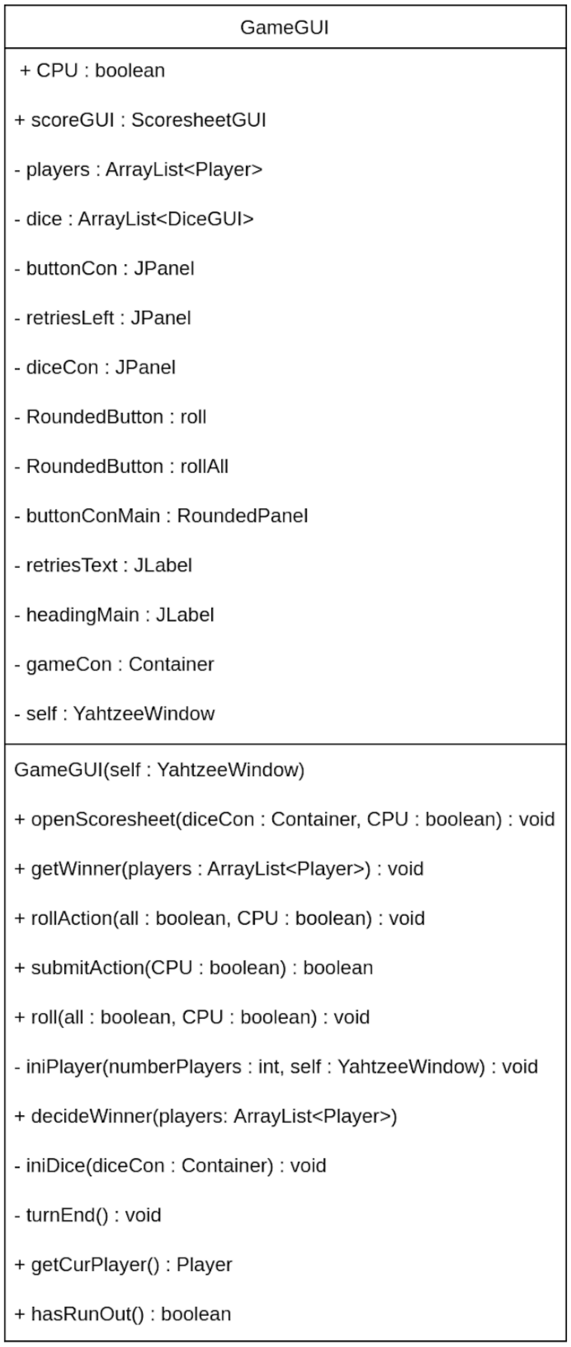
We used an object oriented design that separated GUI and functional aspects of the program into their own classes/objects that accessed each other. Since we used Suyash’s code as the base for our project, this organized approach was essential for the rest of the group members to be able to easily read, traverse, and modify the code.

Our goal was to create a polished game with more rich features. We wanted the game to have a Yahtzee theme throughout the GUI, so we revamped it quite a bit, while also adding animations on the home screen and play screen to give the application more life. Additionally, we added a functional non-AI computer player option if the user would like to play an opponent and does not have anyone in real life to play with. Finally, we limited the number of die sides and players possible in order to reduce the buggy and laggy outcome of having too much going on at once.

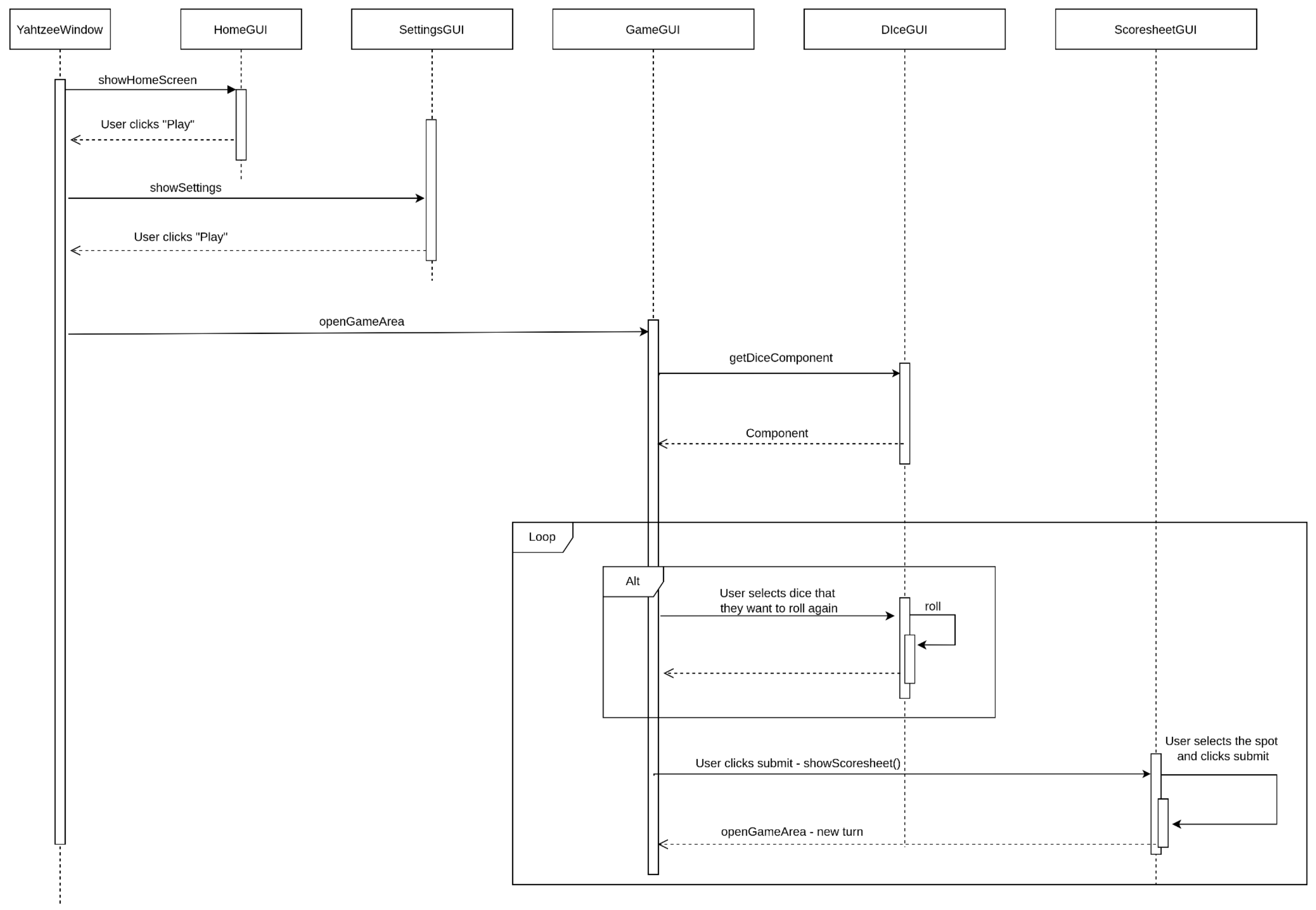

YahtzeeWindow
- currentScene : WindowScene
- start() : void
- displayCurrentScene() : void
- saveConfig() : void
- getConfig() : void
WindowScene
+ parent : JPanel
+ getParent() : JPanel
ScoresheetWindow
HomeGUI
SettingsGUI
GameGUI
PlayerCPU
Player
DiceGUI
ScoresheetGUI
Leaderboard
Scoresheet
HandAnalyzer


Player
+ dieSides : int
+ playerName : String
+ CPU : boolean
+ numberDice : int
- scores : Scoresheet
+ Player(numberDice : int , dieSides : int, retries: int , playerName : String)
+ getScoresheet() : Scoresheet
+ isScoresheetFull() : boolean
+ getTotal() : int
+ isCPU() : boolean
Section
+ score : int
+ name : String
+ Section(name : String)
+ isEmpty() : boolean
PlayerCPU
+ timer : Timer
+ dirty : boolean
+ wait : int
+ toRollOrNotToRoll(dice : ArrayList<DiceGUI> game : GameGUI) : void
+ selectSpot(game : GameGUI) : void
HomeGUI
+ homeWindow(self : YahtzeeWindow)
+ getResizedImage(image : BufferedImage, width : int) : Image
Leaderboard
+ parent : JPanel
+ scorePanels : ArrayList<JLabel[]>
+ scores : ArrayList<Player>
Leaderboard(players : ArrayList<Player>)
+ update() : void
+ getParent() : JPanel
HandAnalyzer
- handArray : ArrayList<Integer>
- counts : int[]
- totalCount : int
- numberDice : int
- dieSides : int
- upperSize : int
- selectedSpot : int
HandAnalyzer(handArray : ArrayList<Integer>, numberDice : int, dieSides : int, upperSize : int, selectedSpot : int)
+ scoreToAdd() : int[]
+ isFullHouse() : Boolean
+ maxOfAKindFound() : int
+ maxStraightFound() : int
- countNums() : void
DiceGUI
+ value : int
+ selected : boolean
- dieSides : int
- dice : RoundedButton
- selectedColor : Color
- unselectedColor : Color
- rollAnimationCount : int
- rollAnimationMax : int
- timer : Timer
- config : GameGUI
+ DiceGUI(height : int, width : int, font : Font, dieSides : int, config : GameGUI)
+ rollWithoutAnimation() : void
+ rollWithAnimation() : void
+ roll() : void
+ select() : void
+ unselect() : void
+ isRolling() : boolean
+ getLabel() : Component
- iniListener() : void
- rollDie() : int
Scoresheet
+ upperSection : ArrayList<Section>
+ lowerSection : ArrayList<Section>
+ numberDice : int
+ dieSides : int
+ totalSpots : int
- upperTotal : int
- upperBonus : boolean
- lowerTotal : int
- upperBonusThreshold : int
+ Scoresheet(numberDice : int, dieSides : int)
+ isEmpty(spot : int) : boolean
+ getSectionScore(spot : int) : int
+ getSectionName(spot : int) : String
+ isFull() : boolean
+ getUpperTotal() : int
+ getUpperBonus() : int
+ getTotalScore() : int
+ getLowerTotal() : int
+ toArray() : String[][]
+ setScore(toAdd int[]) : void
SettingsGUI
SettingsWindow(self : YahtzeeGUI, parent : Container, hideButtons : boolean)
+ createSettingComponents(name : String, jpanel : JPanel, defaultValue : int) : JTextField
+ applyConfig(self : YahtzeeGUI, fields :ArrayList<JTextField>) : void
ScoresheetWindow
- frame : JFrame
- scoreGUI : ScoresheetGUI
- leaderboard : Leaderboard
ScoresheetWindow(self : YahtzeeGUI, curPlayer : Player)
+ getScoreGUI() : ScoresheetGUI
+ update() : void
+ changePlayer(p : Player) : void

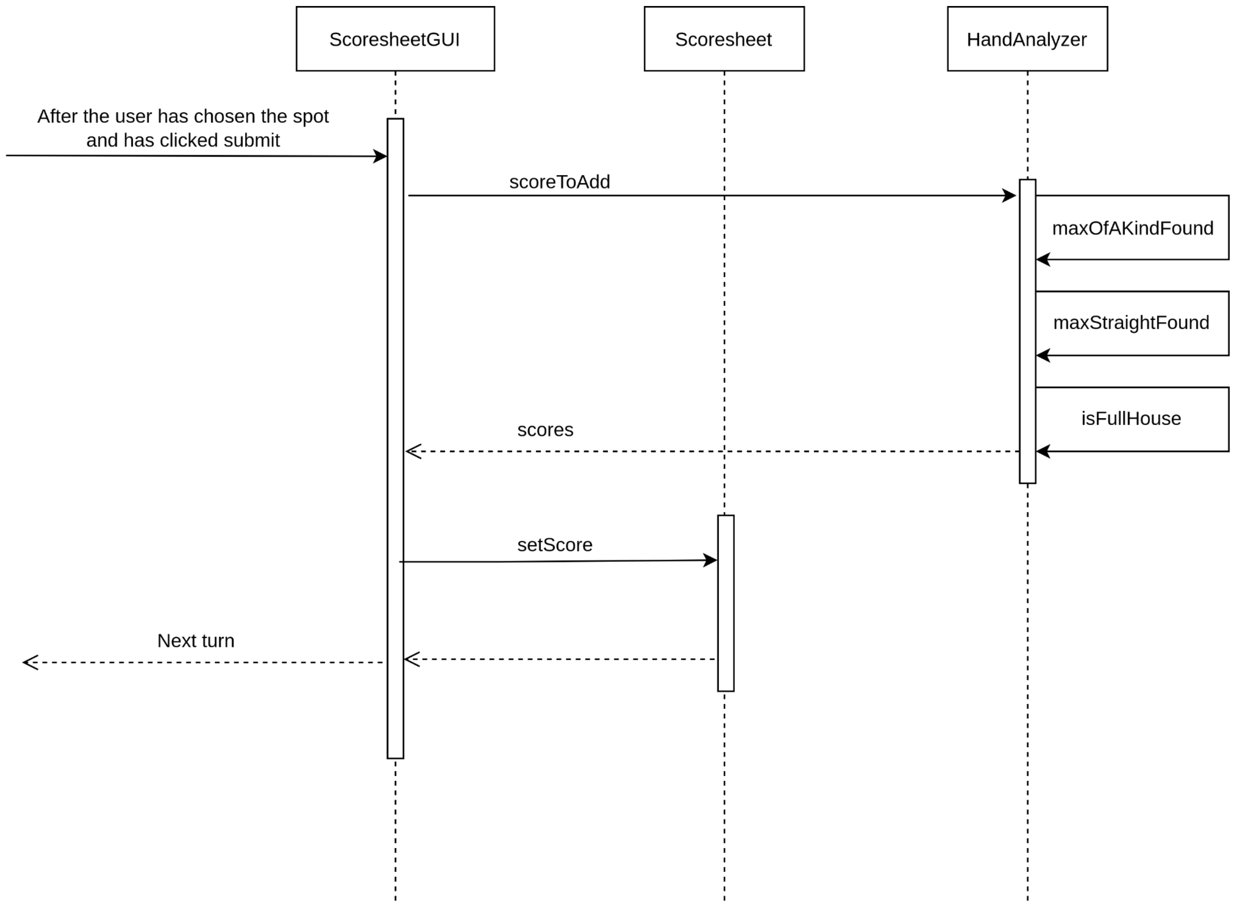

ScoresheetGUI
- scrollCon : JScrollPane
- selectedSpot : int
- handArray : ArrayList<Integer>
- table : JTable
- end : Callback
- updateScoresheetWindow : Callback
- buttonCon : JPanel
- headingMain : JLabel
- player : Player
+ scores : Scoresheet
+ model : TableModel
+ showScoresheet : MouseAdapter
+ spotWithMaxScore : int
+ game : GameGUI
+ ScoresheetGUI(buttonCon : JPanel, headingMain : JLabel)
+ resetState() : void
+ getPlayer() : Player
+ init(player : Player, UIScale : float) : void
+ changePlayer(player : Player) : void
+ init(player : Player, handArray : ArrayList<Integer>, UIScale : float) : void
+ next(CPU : boolean) : void
+ endTurn() : void
+ roll(all : boolean, retriesText : JLabel) : void
+ setCallbackPossibleScore(end : Callback) : void
+ setCallbackShowScore(updateScoresheetWindow : Callback) : void
+ showPossibleScores() : void
+ showScoresheet() : void
+ getComp() : Component


GameGUI
+ CPU : boolean
+ scoreGUI : ScoresheetGUI
- players : ArrayList<Player>
- dice : ArrayList<DiceGUI>
- buttonCon : JPanel
- retriesLeft : JPanel
- diceCon : JPanel
- RoundedButton : roll
- RoundedButton : rollAll
- buttonConMain : RoundedPanel
- retriesText : JLabel
- headingMain : JLabel
- gameCon : Container
- self : YahtzeeWindow
GameGUI(self : YahtzeeWindow)
+ openScoresheet(diceCon : Container, CPU : boolean) : void
+ getWinner(players : ArrayList<Player>) : void
+ rollAction(all : boolean, CPU : boolean) : void
+ submitAction(CPU : boolean) : boolean
+ roll(all : boolean, CPU : boolean) : void
- iniPlayer(numberPlayers : int, self : YahtzeeWindow) : void
+ decideWinner(players: ArrayList<Player>)
- iniDice(diceCon : Container) : void
- turnEnd() : void
+ getCurPlayer() : Player
+ hasRunOut() : boolean

How scores are calculated and added:



How different GUI elements are managed:



V. Test Plan

1. Unit Testing

Unit testing will be the smallest and most important level of tests, as it will test each individual function of each class. Not only that, but each function would more than likely have multiple unit tests. When testing, we really want to ensure the program has 100% functionality from the ground up; each function will have anywhere from 1-3 small unit tests that test the functionality of a small function or subfunction. Each person will complete and document the unit tests for their own code that they end up writing.
Unit testing is the smallest and most important level of testing, as it tests each individual function of each class. Each function had multiple unit tests and when testing we wanted to make sure that it was 100% functional. For the core classes we used integration testing.

The documenting process will be as follows:

- When finishing a function, the author will write the sufficient number of unit tests (decided by them) on that function on a separate testing file or the main file.
- These tests will aim to test specific functionalities within the scope of these functions. NOT to test the functionality of functions in relation to other functions.
- After the tests have been completed and the author of the function is satisfied with the test results, they will note down in a shared document that the tests for their specific function have been completed.
- They will also write the purpose of the test and the function it is testing (with a description of that function), to make sure that any other coder in the group is able to easily test other parts of the code that they didn't write with ease.
    - This will allow us to have a comprehensive database of all functions that have been tested, and will tell us when all our bases are covered and we are ready to start part 2: integration testing.

2. Integration Testing

When performing integration testing, we are looking at the interaction of two or more classes. For example, we tested how the Player class has a Scoresheet. We tested how the player would have access to this scoresheet and if the dice roll is being recorded properly onto the scoresheet.

This test was carried out by allowing the user access to open multiple scoresheet windows and the only scoresheet available to them is their own. The other player's scoresheets will be kept in a private class and only accessible when it's a certain player's turn. We also tested to make sure that when a player rolls their dice and they choose which ones to keep and then keep rolling, the score they choose on the score sheet is the correct one and is then recorded on their scoresheet permanently until the game ends and restarts.

3. System Testing

3.1. Functional Testing

Function testing in our program entailed testing our functional requirements by testing if all of the JButtons interact with the page and guide the user to the correct page or allow the user to roll die.

When the game runs we want to make sure that all the users who play have access to the current player's scoresheet. Checking that the user has access to their scoresheet only to keep scores private from other players. It was ideal for us that the user would have access to their sheet and the leadership board throughout the game.

Another function that tested would be the tracking of the score. Making sure the user's score is correct and the computer is keeping the score from the correct dice roll.

3.2. Performance Testing

There were a few ways in which the performance of the program was tested. The number of dice can be increased to a limited number (10) and we checked how long it takes the JButtons to render in the scenario.

We also checked to see how the CPU characters would affect the game itself and if the dice would still roll our how the game would react to multiple CPU players.

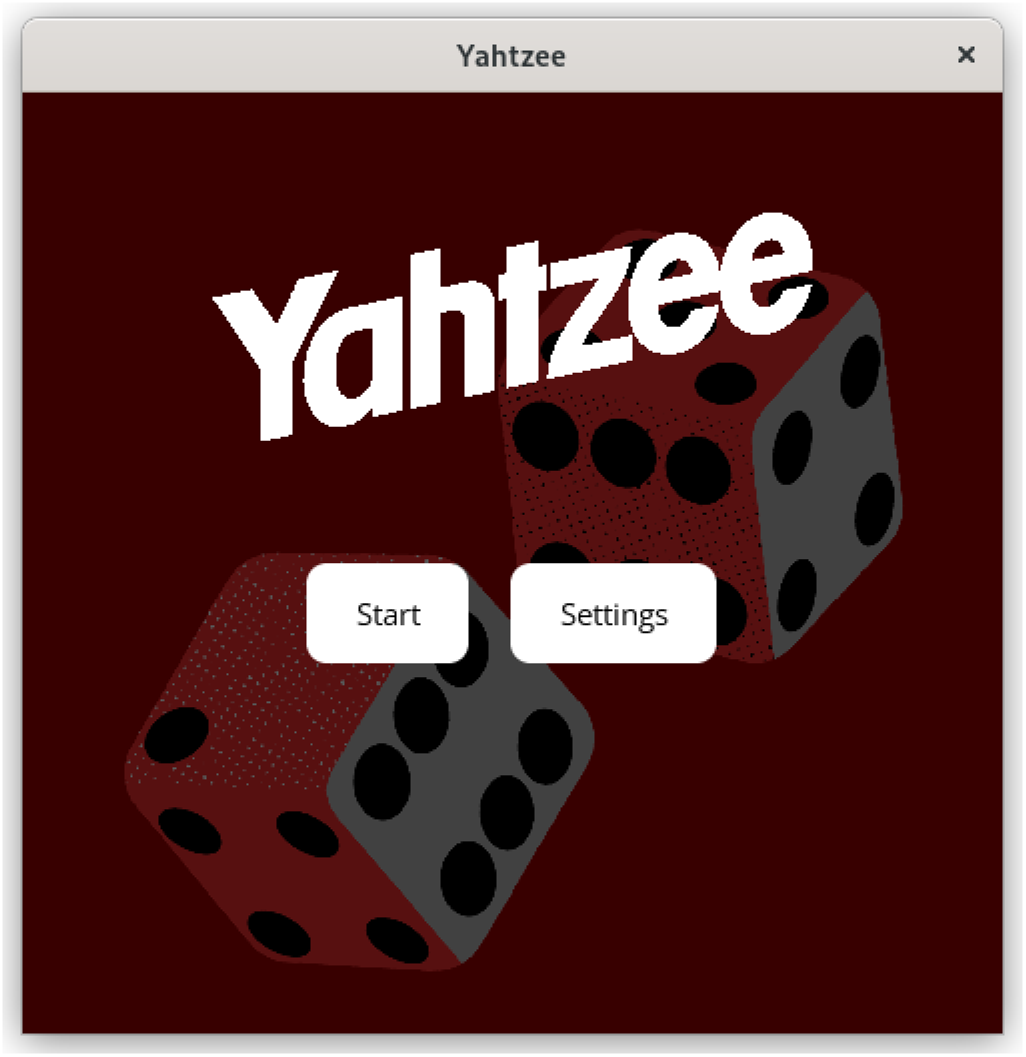
3.3. User Acceptance Testing

All the team members were required to make other people, preferably non-programmers, use the program. They cannot intervene when the users are testing the app, as that would defeat the purpose of this test. This test will focus on two main things: the UI and the functionality.

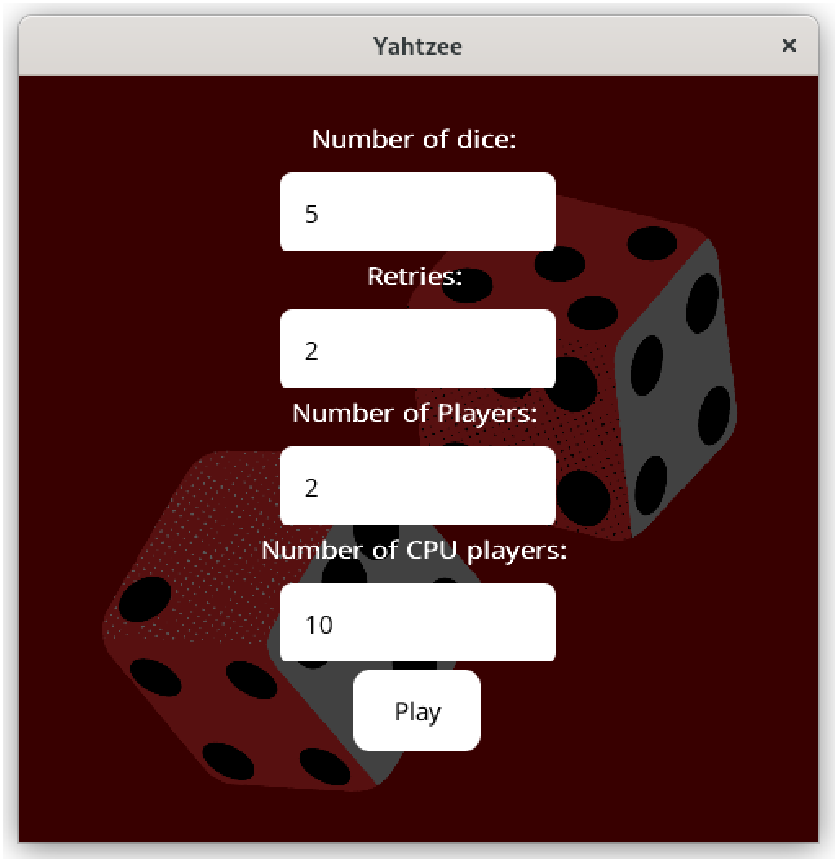
If any UI is deemed unintuitive, or if the UI breaks at any point, then an issue must be opened so it can be fixed. If, however, a critical bug that breaks the program itself is discovered, then the team member should save the stack trace and include it in the bug report.

VI. Project Implementation Description

- We wanted the homescreen to have three options: “Start”, “Settings” and “About”. We, however, weren’t able to implement the “About” section.



- Clicking on “Start” would give users the option to configure things. It would have been ideal to let users change the number of sides a die has, but this interfered with the animations, so we disabled this setting.


Yahtzee

Number of dice:

5

Retries:

2

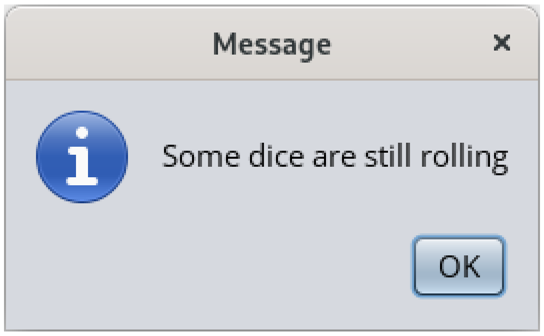
Number of Players:

2

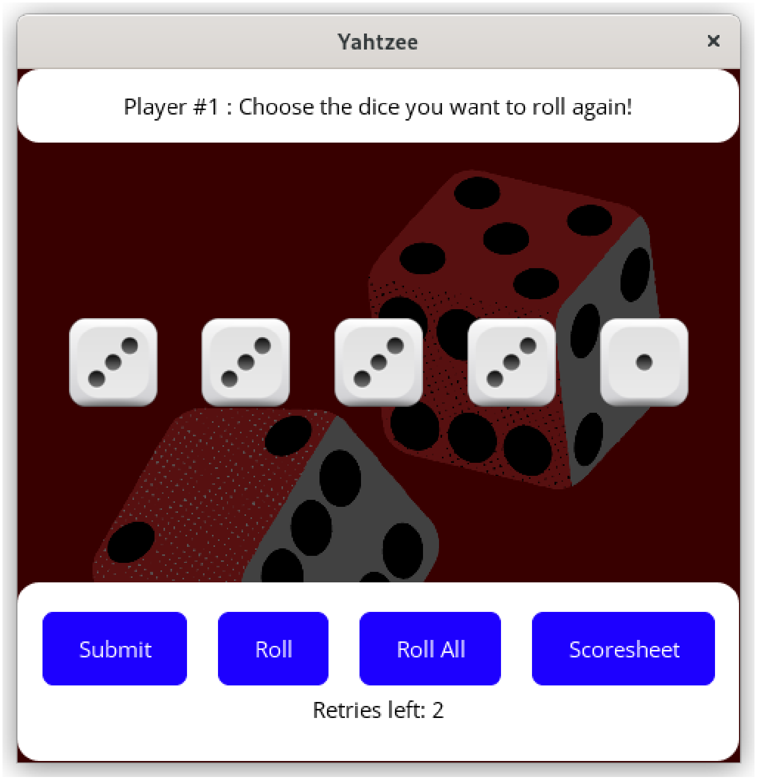
Number of CPU players:

10

- Clicking on “Play” would start the game. Rolling the dice would play an animation. We made it so the users couldn’t submit their hand while the dice were rolling:


Message

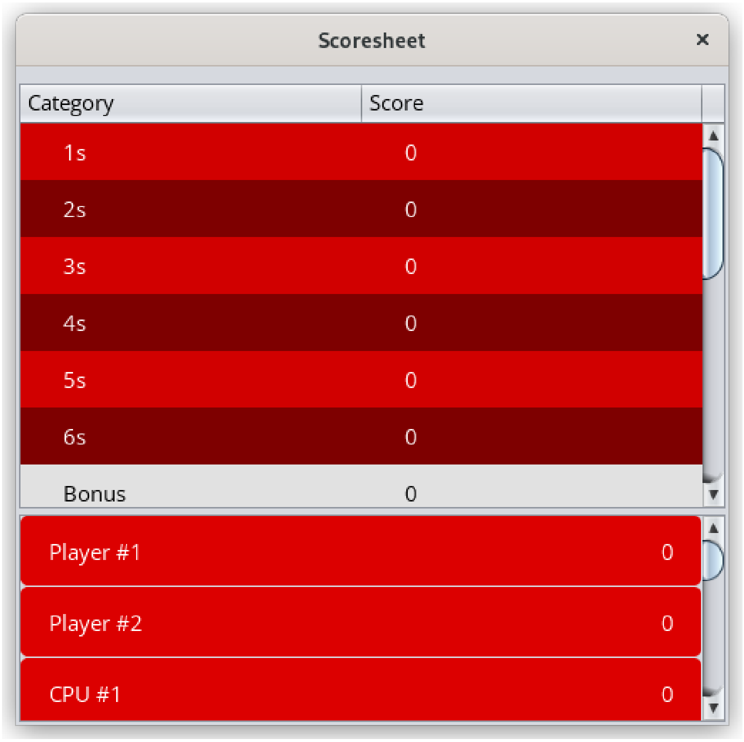
Some dice are still rolling



Player #1 : Choose the dice you want to roll again!

Retries left: 2

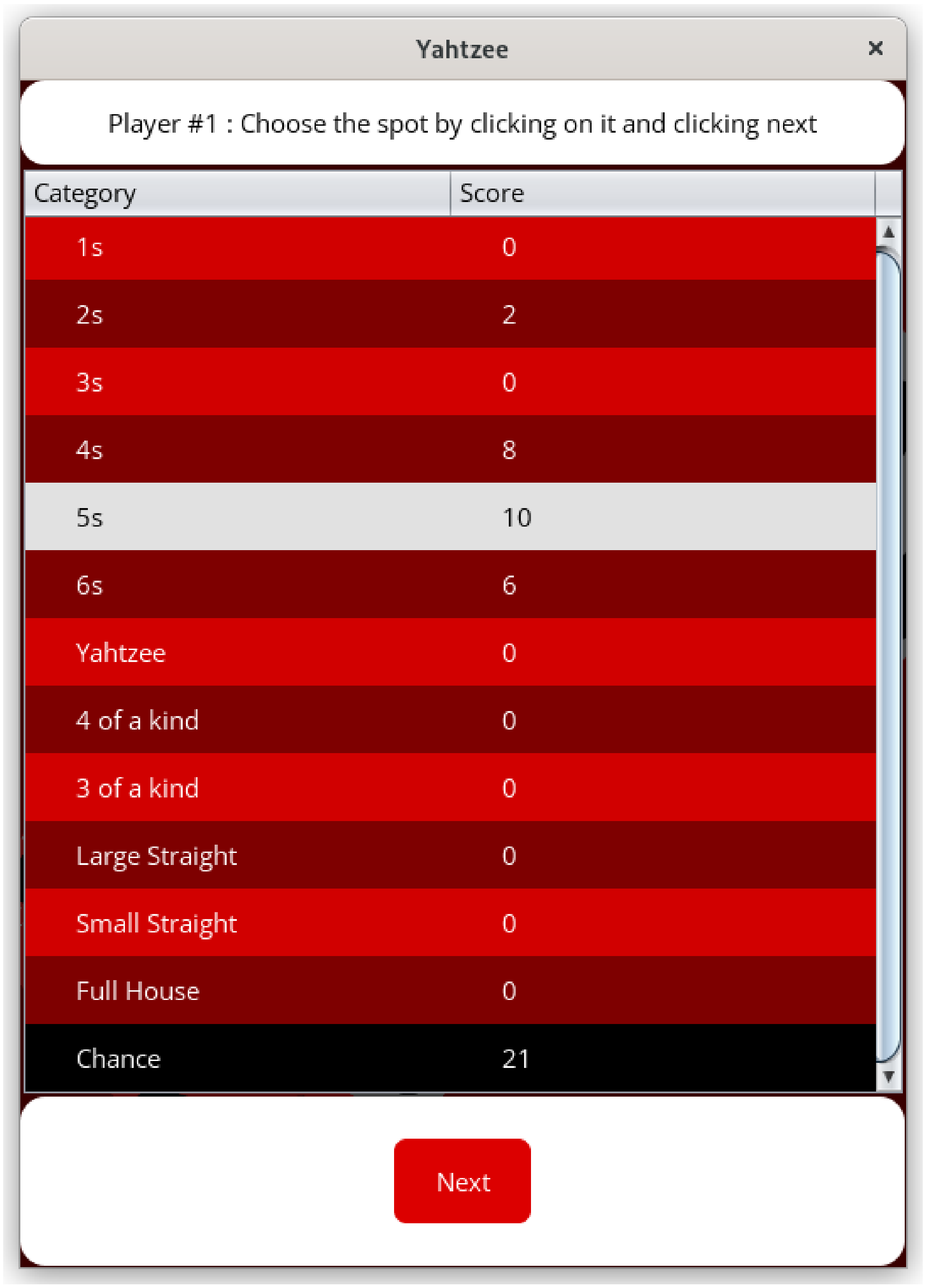
- Clicking on "Scoresheet" opened a window that showed the scoresheet of the current player and also a leaderboard:



Category	Score
1s	0
2s	0
3s	0
4s	0
5s	0
6s	0
Bonus	0

Player #1	0
Player #2	0
CPU #1	0

- Then the users are allowed to select which spot they want to put the hand in. Non-empty spots are colored white. The spot that has currently been selected is colored black. The rest of the rows are colored different shades of red.

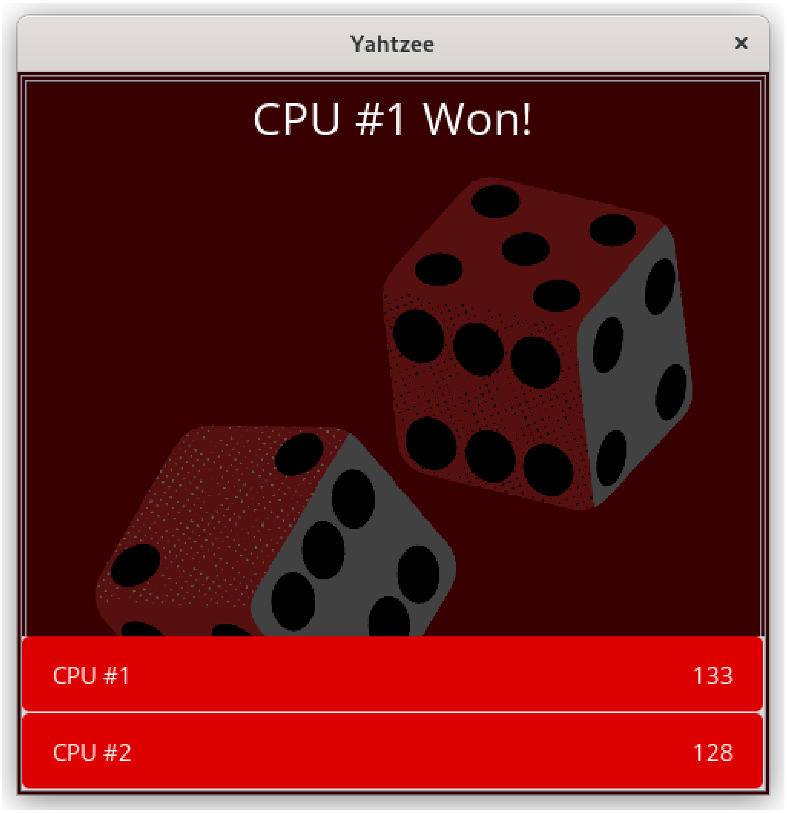


Yahtzee

Player #1 : Choose the spot by clicking on it and clicking next

Category	Score
1s	0
2s	2
3s	0
4s	8
5s	10
6s	6
Yahtzee	0
4 of a kind	0
3 of a kind	0
Large Straight	0
Small Straight	0
Full House	0
Chance	21

- We also implemented CPU players that made random moves automatically. A game could have an unlimited number of CPU players.
- We also planned on letting users enter custom usernames, but we ran out of time before we could implement that.
- We had also planned on making the program work on different systems, rather than a single system. This was a huge undertaking, so we dropped this feature.
- The end screen also contained a leaderboard:



CPU #1 Won!

CPU #1	133
CPU #2	128

- Github Link: https://github.com/Gonzaga-CPSC-224/cpsc224_fp_f22-full-house

VI. Future Work

In the future, the most logical features for us to improve in our project are the GUI and the CPU players. The GUI is cool because it has just about unlimited potential for improvement. For example, we could try and add a custom yahtzee font to the play screen, or add more custom animations.

For the CPU, it would be fun to try and create an AI CPU player. The current CPU player is mostly used for testing purposes and just picks the highest score each turn based on random rolls/hands. This is good for beginner players as well as testing, but in order to generate a tough opponent it would be best to use artificial intelligence.

VII. Glossary

GUI	Graphical User Interface
JButtons	Buttons generated by Java Swing

VIII. References

No references were used