# Point and Click Yahtzee

## Final Report

**Java the Hutt**

Cameron Williamson, Ryan Gavino, Dominic Orsi, Connor Deide

Course: CPSC 224 - Software Development

# I. Introduction and Project Description

This final group project's assignment was to simulate a full game of multiplayer Yahtzee according to the original rule set provided for the first homework assignment. In addition, once the program is functionally complete, the team must present their project in a manner similar to how an industry software development team would pitch their project to an audience. The purpose for this all-encompassing document is to provide both the project members, the supervisor and the reader an accurate insight into several important factors of a group project that would possibly have impacted both the environment the members worked in as well as how the final product would turn out. Moreover, the provided sections within this document will hopefully cover the most significant factors in how successful/detrimental a project was and at what stages were fruitful or areas of improvement.

The program can handle performing turns with up to 4 players at once. The game will begin with the default ruleset where each player holds five six-sided dice in one hand and can roll them up to three times in a given round. At the start of each round, the first player will perform their full turn until after the player has chosen a row to fill with a score. Following this, the game will switch to the next player in order and continue to finish turns until every user's turn has completed. This will complete one round and begin another until the final player's scorecard has been completely filled. Once play has stopped, the program will declare the winner as the player with the greatest score among the participating users. The key feature that makes this final project feel as significant as it is is going from the singleplayer variation of Yahtzee that all members had previously implemented to a multiplayer version over the course of one month.

# II. Team Members - Bios and Project Roles

The following group of paragraphs are brief descriptions into each participating group member that includes their current area of study, general interests, technical skills, and their established role throughout the project's lifetime.

Ryan Gavino is a Computer Science student interested in machine learning and artificial intelligence. Ryan's technical skills include Python, C++, and Java. For this project, their role was the scrum master coordinating meeting times, communication methods, project pacing, and scribing the project plans and results as they go. Additionally, they helped integrate the multiplayer functional requirements of this project.

Cameron Williamson is a Computer Science Student interested in DevOps, and backend development. Cameron's skills include C++, Go, SQL, and Java. For this project, their responsibility included everything relating to a Yahtzee scorecard and helping with project design and planning.

Dominic Orsi is a Computer Science student interested in cyber security and cloud systems. Dominic's skills include C++, Python, and Java. For this project, their role was creating new dice images, creating the game over screen, and keeping things light hearted.

Connor Deide is a Computer Science student interested in artificial intelligence, mobile development and game development. Connor's technical skills include C++, Java, and SQL. For

this project, their responsibilities included creating the beginning splash screen, implementing the ‘quit’ button, and writing the Player classes base functionalities.

## III. Project Requirements

Before beginning to put code to screen, our team needed to lay out the functional requirements that would deliver the most accurate experience of Yahtzee to the user(s). The functional requirements of the program can be categorized into two primary objectives: ensuring the graphical user interface behaves the user should expect to (under the assumption they know how Yahtzee is played), and translating a single player’s turn loop into going around every player participating and performing a separate turn for each. A more detailed description of each functional requirement can be found at this **link**.

For the following requirements, they serve the purpose of meeting every functional requirement that would be similarly detailed for a single player’s entire turn, from rolling the dice to selecting a score:

- **For a player’s turn to being, the user will click the roll button to begin their turn.**
- **Before each additional roll, the player can select the dice that they wish to keep rather than reroll with ther rest in their hand.**
- **The program must limit the player on how many rolls they can use before selecting a score.**
- **The scorecard should have selectable rows that allow the player to choose which possible score they want to fill in for their turn.**

During development of the program, our first milestone was aimed at implementing a variation of our singleplayer code to meet all of these requirements of playing and completing a turn of play. Concurrently while we were implementing the base classes, we took into account what decisions in our base class implementation could afford us an effective yet simple alteration to our game loop to integrate multiplayer as a feature. In order to keep track of how much progress we’ve made going from singleplayer to local multiplayer, we listed the following functional requirements:

- **The app will allow the user to specify how many players will be participating, and allow them to enter names for each player.**
- **Once the current player has selected a scorecard row to fill (i.e. they’ve completed their turn), the program should swap to the next player in line and begin a new turn for them, updating the user interface correspondingly.**
- **After the last player in order has ended their turn, the program will check if the game is over.**
- **Once the game is over, the game will declare the winner as a game over message.**

## IV. Solution Approach

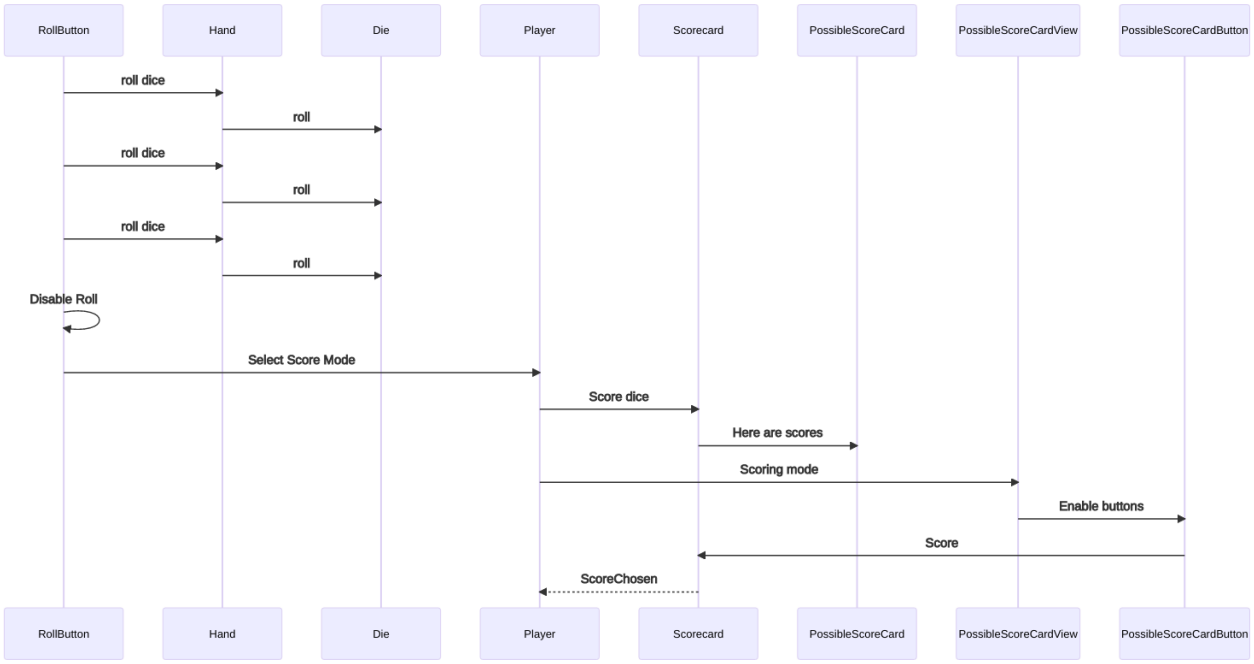
The key feature of our final project was making a seamless transition from playing with just one player to allowing multiple players to play the game in the same session. One of the assumptions we made was it would've been a straightforward process of just implementing our single-player base classes, adding the GUI elements inspired from Cameron's design, and then just tie it all together into each individual player. As we got further into finishing the lower level API, it became clearer that the switch to multiplayer was as vague as the "just tie it all together" idea we had.

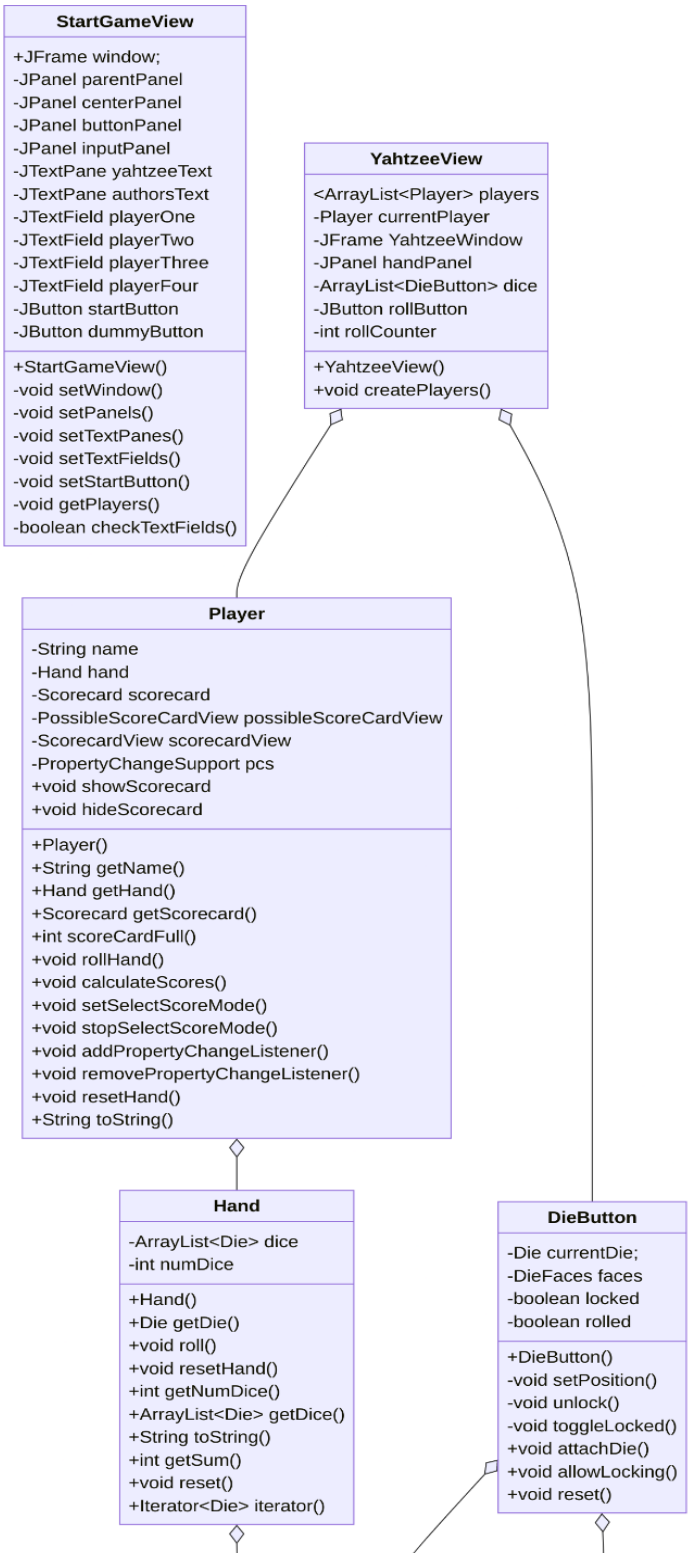
The approach we ended up using for the final product was still built off of our initial idea of tying together both the models a Player object needs to hold data and the views that the current user can interact with to play the game. The Player object would be both responsible for creating the backend data structures needed for the game, such as the Hand and Scorecard, in addition to containing the operations that represent different parts of a player's turn, such as rolling their hand of dice or calculating/selecting the score. This encapsulation of both holding data and the game's functions is essential to our next step, where we set the status of the user currently playing the game as currentPlayer. By labeling the player that's currently in control of their turn, we were able to hold the rest of the created users in an ArrayList and simply iterate through the group by shifting who was currentPlayer to every following player. In our code, this takes the form of:

1. Create players and set the first created player to currentPlayer
2. Assign the GUI elements in the frame, which were created in YahtzeeView, to the data contained within currentPlayer.
    a. i.e. Each DieButton was linked to every one of currentPlayer's dice in their Hand, the possible scorecard on the right would change their selectable row's values based off currentPlayer's hand, and their actual scorecard would store the selected values from currentPlayer's PossibleScoreCardView
3. Once a score has been selected and a property change has been fired back to YahtzeeView, shift currentPlayer to the next one in line and reassign the GUI elements to the currentPlayer's data.
4. Repeat until the last player in order has completed their turn. Then, check whether the game is over; display a game over message if it is over, otherwise shift currentPlayer back to the front and repeat the round as normal.

Functionally, this method of shifting the current player pseudo pointer to a different player each time worked well in our system tests. However, another non-functional requirement was conspicuous as we performed our system tests, which was how to the regular no prior knowledge user it was unclear which player was the current player. The only thing differentiating one player's view from the following one was the differences in the regular scorecard. As a result, another secondary feature we wanted to implement was tying a specific color to each

created player. This color would be helpful in making every player visibly unique in how their scoreboard number would appear in the top left as well as changing the background color of the currently visible scorecards to the player's color.




StartGameView
+JFrame window;
-JPanel parentPanel
-JPanel centerPanel
-JPanel buttonPanel
-JPanel inputPanel
-JTextPane yahtzeeText
-JTextPane authorsText
-JTextField playerOne
-JTextField playerTwo
-JTextField playerThree
-JTextField playerFour
-JButton startButton
-JButton dummyButton
+StartGameView()
-void setWindow()
-void setPanels()
-void setTextPanes()
-void setTextFields()
-void setStartButton()
-void getPlayers()
-boolean checkTextFields()
YahtzeeView
<ArrayList<Player> players
-Player currentPlayer
-JFrame YahtzeeWindow
-JPanel handPanel
-ArrayList<DieButton> dice
-JButton rollButton
-int rollCounter
+YahtzeeView()
+void createPlayers()
Player
-String name
-Hand hand
-Scorecard scorecard
-PossibleScoreCardView possibleScoreCardView
-ScorecardView scorecardView
-PropertyChangeSupport pcs
+void showScorecard
+void hideScorecard
+Player()
+String getName()
+Hand getHand()
+Scorecard getScorecard()
+int scoreCardFull()
+void rollHand()
+void calculateScores()
+void setSelectScoreMode()
+void stopSelectScoreMode()
+void addPropertyChangeListener()
+void removePropertyChangeListener()
+void resetHand()
+String toString()
Hand
-ArrayList<Die> dice
-int numDice
+Hand()
+Die getDie()
+void roll()
+void resetHand()
+int getNumDice()
+ArrayList<Die> getDice()
+String toString()
+int getSum()
+void reset()
+Iterator<Die> iterator()
DieButton
-Die currentDie;
-DieFaces faces
-boolean locked
-boolean rolled
+DieButton()
-void setPosition()
-void unlock()
-void toggleLocked()
+void attachDie()
+void allowLocking()
+void reset()

## V. Test Plan

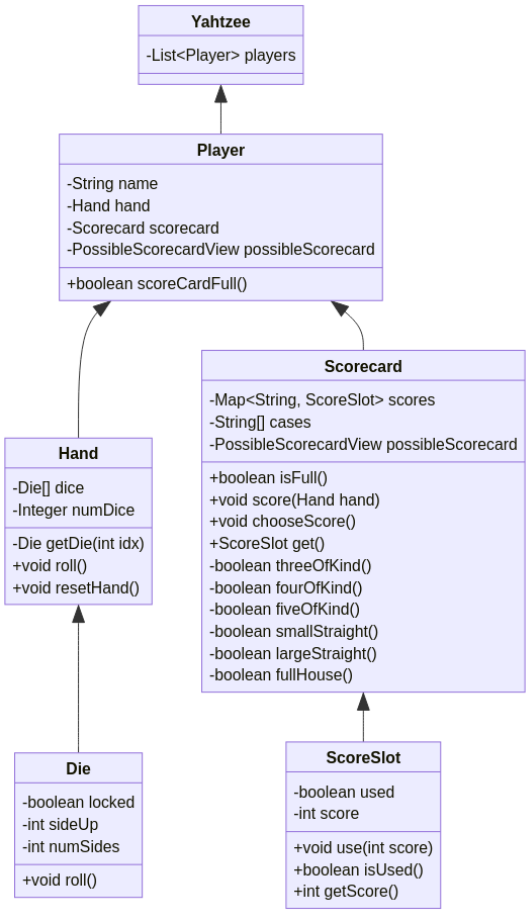
Our original testing plans were to perform unit tests on our code that made up the model, create integration tests when combining these classes to check that they work together as expected, and finally perform system tests on our game as soon as we got parts of our GUI functional.

When carrying out these testing plans we were pretty successful in our execution and stayed true to most of our original plans. Each person who wrote source code for our base model classes wrote unit tests for them which could be verified by outputting their results to the system terminal. The most helpful type of testing during this process was definitely our system testing, as it exposed the most bugs and helped us figure out what issues need to be resolved. The one thing we could have done better to stay closer to our original test plans would have been to integrate more unit tests. We never explicitly wrote any integration tests for our program, instead we checked that our code was what we expected through pull requests and good communication.

https://github.com/GU-2021-Fall-CPSC224/finalyahtzee-java-the-hutt.git

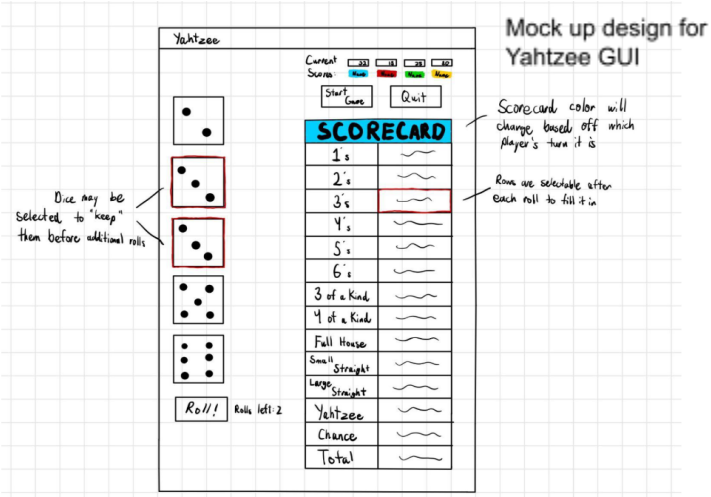
## VI. Project Implementation Description

In the beginning, besides making our project plans, our first coding milestone we wanted to accomplish was implementing the lower level API. This meant simply implementing the base classes used in all of our versions of single player Yahtzee, such as Die, Hand, Scorecard, and Player. Progress was gradual but steady in terms of implementing these classes, however the one hiccup we had with the scorecard was future-thinking about how we want to contain both possible scores a player can choose and the selected scores the player has already chosen to be added to the total. For this project, Cameron was our core designer and planner for how both the data structures and visuals should be implemented inside a scorecard. After discussion, we decided that the planned implementation was separating both the possible scores a player can choose from and the selected scores a player has chosen into two different objects, or at the very least one inheriting from the other. The UML diagram to the left was the initial plan for how our lower level classes were going to work.



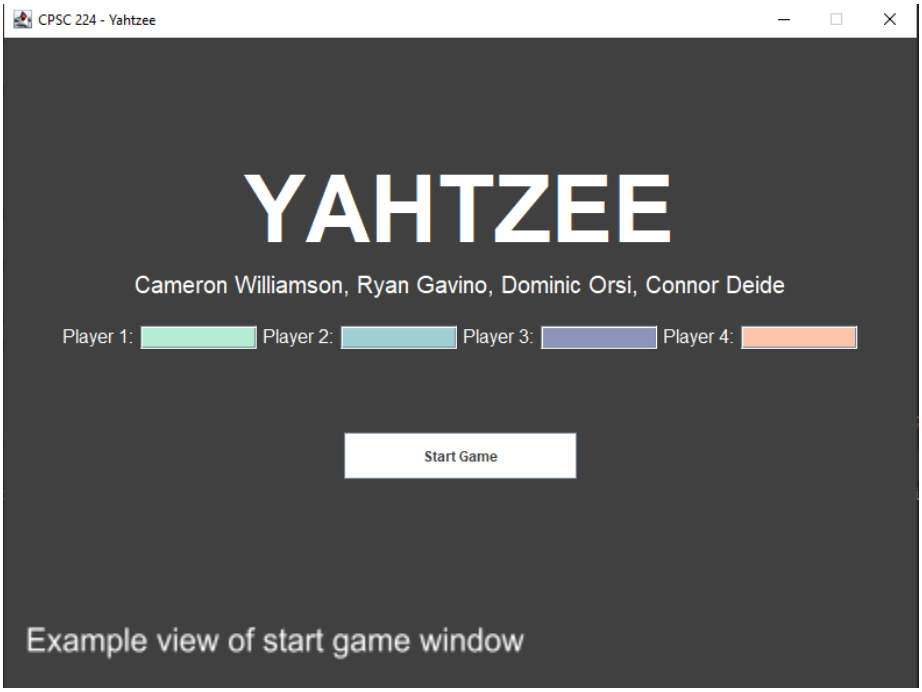
Initial UML class diagram from project plans

After reaching the milestone of finishing the lower level classes, it was time to start designing and creating the GUI for Yahtzee. When we were discussing our project plans in the first week, the initial mock-up design for our GUI contained similar elements to the final design with the exception of two elements. First, the scorecard user interface would only contain the possible scores a player can choose from, and the idea would be just refreshing the possible scorecard to represent moving on to the next player, while also changing the background color of the scorecard to further reinforce the continuous loop of player rotation. Second, the start game button was initially planned to be available upon initialization of the frame to allow the starting user to personally set how many players would be participating in the current session.



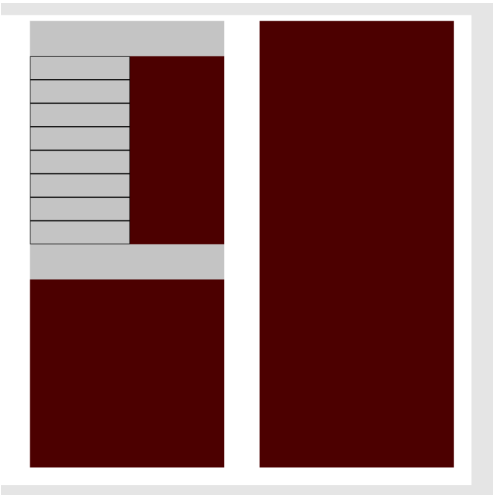

Shortly into the start of development, both Connor and Cameron had come up with ways to expand upon editing the start of the game and showing off the scorecards. Connor, who was in charge of implementing how to get the game started, took the idea of a start game button and

expanded it into a separate new frame that would prompt the user to enter names into corresponding text fields to add extra customization to the player's creation. Furthermore, the usage of text fields to enter player's names allowed us to create players based off of whether the current user had entered anything into the resulting text. After the user would enter some amount of names (up to 4 players), the start game window would close and the YahtzeeView frame would be created and the game would officially start. In addition, by saving the name to each Player object upon creation, this allowed Dominic to easily implement the game over message for the end of the session, creating a pop up window that would announce "Winner: _____ with a score of _____!"

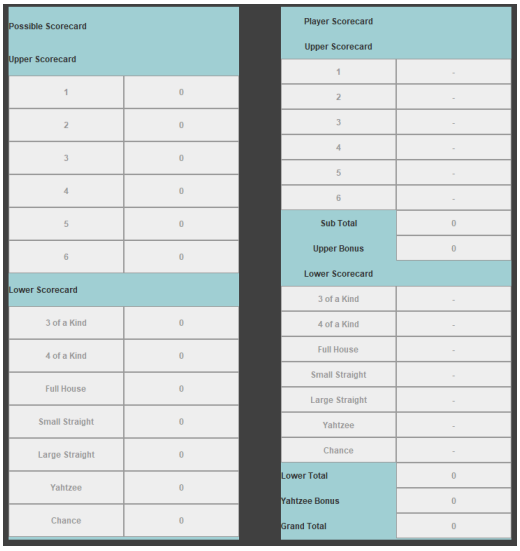


Example view of start game window

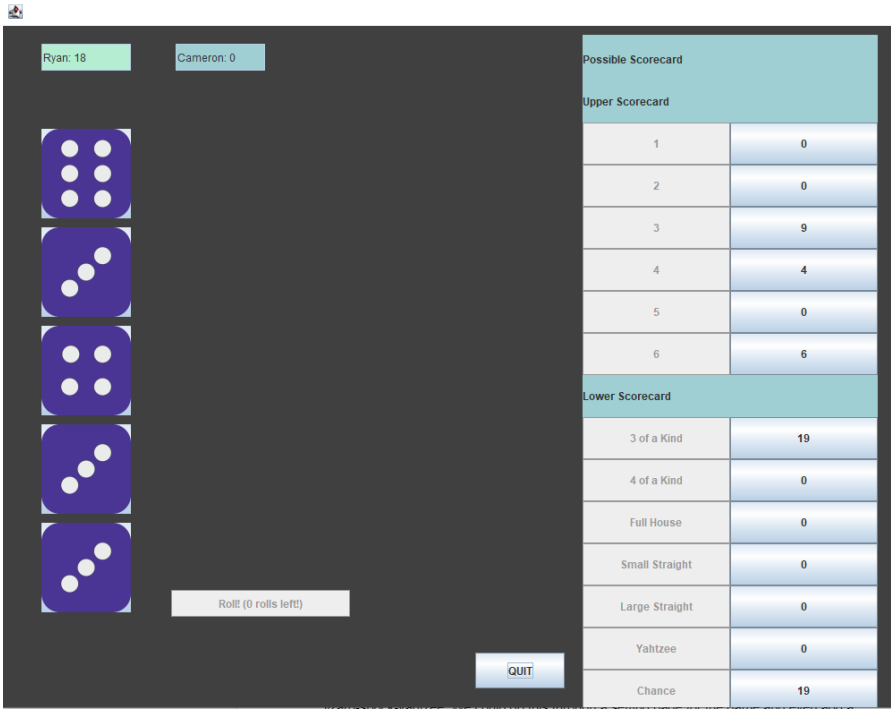
Next, Cameron wanted to redo both the class structure and the player's view of a scorecard in a way that took more advantage of utilizing the method of storing both possible and selected scores in two different objects. The new and improved look of representing the scorecard had stretched the frame lengthwise to add a second scorecard interface that would serve the sole purpose of showing the current player's selected scores alongside the hand's currently possible scores. The purpose for doing so would be to assist the player in strategizing how they might want to try and score for the current turn, given that the view of the selected scores would allow the current player to see which categories have already been filled in rather than guessing based off of the total score at the top left.



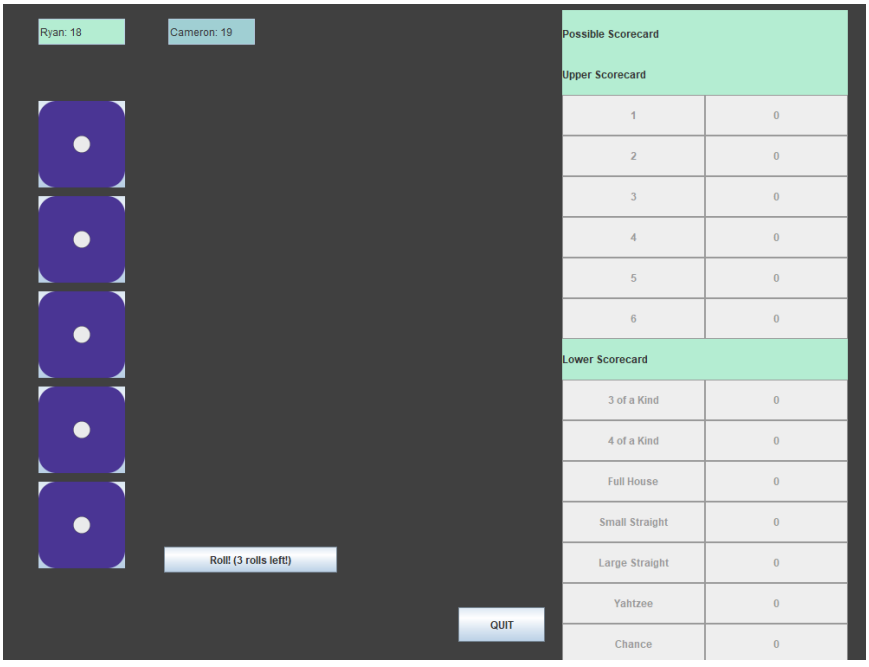
Left: Prototype design for possible + selected scorecards
Right: Final design



Last but not least, Ryan's final implementation of enabling multiplayer would utilize the synchronous behavior of the currentPlayer assignment being shifted alongside updating the GUI components to work with the new currentPlayer's data. The explanation of how the variable currentPlayer and the GUI components are updated were explained in the solution approach section above, but the core idea remained about how once the currentPlayer's turn was over, the GUI would refresh itself just like in the single player variation, but two major visual updates occur: the scorecard's view would change color to indicate the new currentPlayer's scorecard is now in use, and the score tracker in the top left is updated with the old player's new total.




Cameron is ready to score in 3 of a kind for 19 points…




The dice are reset, the scorecard is changed to be Ryan's, and Cameron's score has increased!

## VII. Future Work

The first thing we could do to create a more complete product would be to clean up the GUI further. The functionality of our code met all of the requirements we established in the beginning so the first improvement would be a visual one. Our GUI actually looked quite good in the end and included changing colors given the current player, but there is always more that can be done. Another feature that we could implement given more time would be the dynamic rules of lizard-spock-yahtzee. We could do this through a setting page for the game and even add a how-to-play page like other teams did.

## VIII. Glossary

## IX. References

Cite your references here if you've got any. At the very least you can cite this:

https://en.wikipedia.org/wiki/Yahtzee#:~:text=Yahtzee%20is%20a%20dice%20game,Edwin%20S.%20Lowe%20in%201956.

https://www.ultraboardgames.com/yahtzee/history.php

## X. Appendices