Multiplayer Yahtzee

Final Report

Zags Help Zags
Jackie Ramsey, Katie Imhof, Jesse Adams, John Stirrat

Course: CPSC 224 - Software Development

I. Introduction and Project Description

The purpose of this document is to review the project process implemented by each group member. This also summarizes the approach we took and how we succeeded in completing this project. Included are also the technical details used to engineer the final product of our code.

For this project, we built a game called Yahtzee. The version that we created is the original, multiplayer game where players are competing to get the highest total score and win. The way the game works is by each player taking turns rolling dice and adding up the total for specified score combinations. These score categories can be found on the provided score card that contains different types of dice combinations with an associated score. On each turn, the player will roll dice up to 3 times in order to get the highest scoring combination for one of the 13 categories on the scorecard. There are a total of 5 dice for each turn. On the first roll, the player must roll all the dice. However, for the second and third roll, the player can decide which dice they would like to keep or reroll by either inputting “y” or “n” for each die. The purpose of this is so that the player can strategize which score category they are going to try and get the most points for.

Once the player finishes rolling, they must place the score or zero in one of the available categories by inputting the keycode associated with the score category. Every turn, it is required that a score category must be filled. Additionally, each category can only be used one, so players must strategize where to put their points in an attempt to have the best score. In addition to the 13 categories on the scorecard, there is a bonus that will be added to the final total of the scorecard if the requirements are met. One significant score category to be mindful of when playing this game is the Yahtzee category. If a player gets 5 of the same number, the Yahtzee category can be filled with 50 points which is a huge advantage.

When the player is finished with their turn, the next player plays their turn and so on. This process is repeated for each player until all players have filled the 13 categories in their score card. Keeping track of progress is important, so at any time, players can view the current scorecards by inputting the associated command. Once the score cards of all the players are filled, the game is over. The total score is calculated for each player’s score card, and the player with the highest total score will be determined as the winner.

II. Team Members - Bios and Project Roles

Jackie Ramsey is a computer science student interested in software development and other related fields. Her prior projects have come from other classes she has taken and encompass topics such as SQL website design, data visualization, and others. Jackie’s skills included common languages such as C++, Java, Python, SQL, assembly language, and more. For this project her responsibilities involved group coordination, checking off issues, group coding, and testing.

Katie Imhof is a computer science student and is interested in software application development. Her prior projects have come from other classes, which includes topics such as data visualization, APIs, and others. Katie’s skills include programming in C++, Java, Python and some Processing. For this project, her responsibilities involved group coordination, checking off issues, group coding and testing.

Jesse Adams is a computer science student and is interested in operating system design and implementation, software development, and autonomous vehicle software design. His prior projects include building a working enigma machine in python, and building a simple game of uno in C++. Jesse's skills include programming in python, C/C++, Java, and Javascript. For this project his responsibilities involved group coordination, checking off issues, group coding, testing and creating the HandPanel.

John Stirrat is a computer science student interested in software engineering, data science, and embedded systems, as well as other similar topics in computer science. His prior projects include mostly class projects from other classes, including things like working with algorithms, APIs, and some minor machine learning. John's skills include C++, Python, Java and Java Swing, and some Processing. For this project his responsibilities involved group coordination, checking off issues, group coding, and testing.

III. Project Requirements

This section contains functional and nonfunctional requirements that our group wanted involved in our project when designing the final product of Yahtzee. These requirementents were scaled by priority so that we could assess what ideas were most important and needed to be addressed first. For example, since functional requirements are essential to the code's ability to complete tasks, they were prioritized before the nonfunctional requirements. The nonfunctional requirements were then addressed when we all completed what was necessary. In this table, we also included the outputs so that we had a better sense of what the result would be and if this is what we wanted out of the requirement. This allowed us to visualize the process of our game and how the user would interact with it.

Priority	Purpose	Inputs/Needs	Operators/Actors	Outputs
High	The user shall be able to choose the number of players for the current game of Yahtzee	Number of players, need at least one, likely cap at some reasonable value to avoid excessively long games	The user and the main game loop	Number of players in the main game loop shall be set to the users value
High	The user shall be able to view the current scorecard at any point during their turn	User must have an instantiated player object	Scorecard object	The GUI must change to display the scorecard with accurate score values for the current game
High	As a part of the user's turn, they shall be able to roll the current hand	User must press a button on the GUI, up to 3 times on a turn, only the "unkept" dice should	Player/Hand/Dice objects and the user	The current hand of dice for the player must be updated to new values, and the

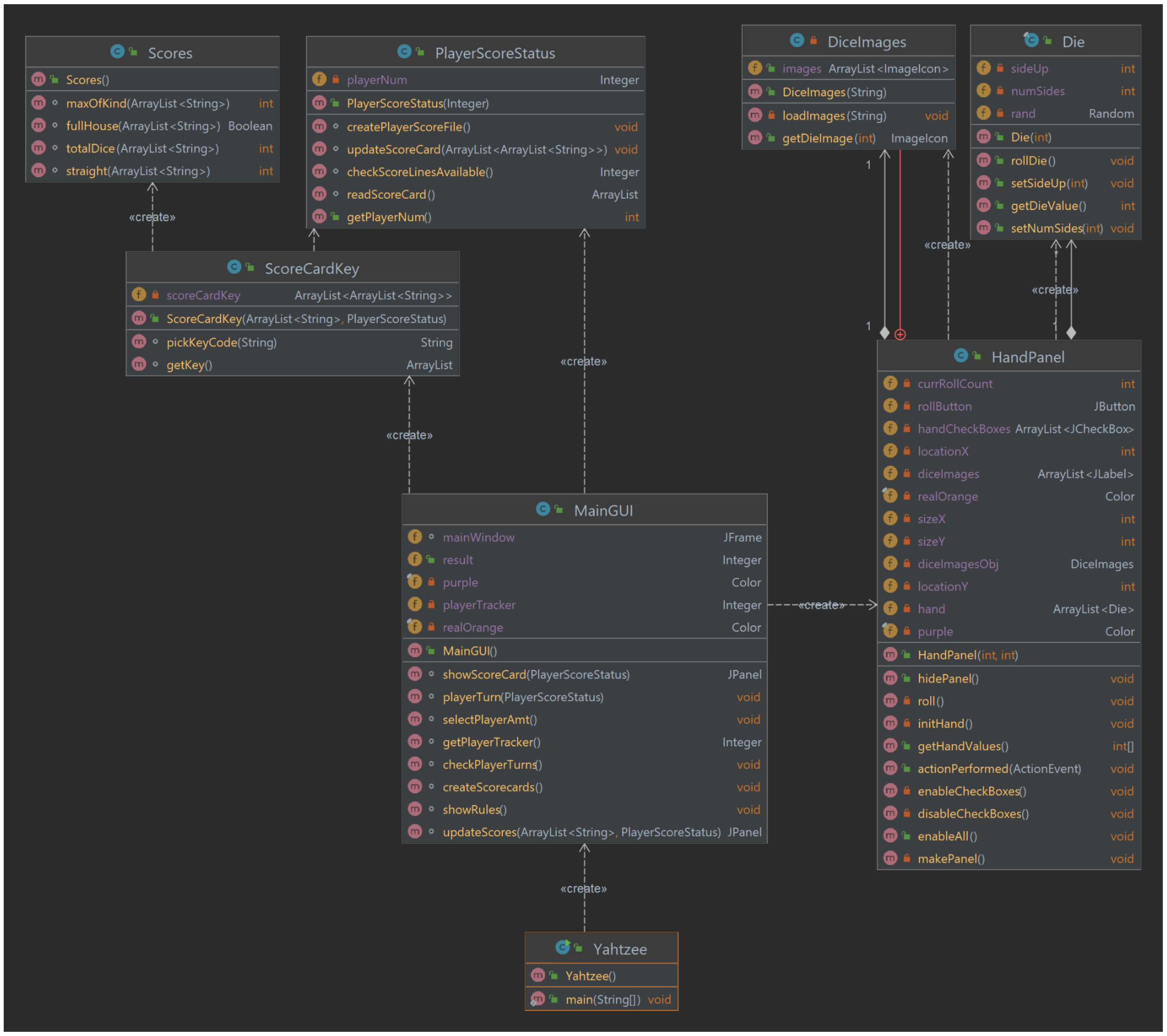
	of dice	be affected		GUI must change to reflect this
High	User should be able to choose dice to not be rolled on their turn, before rolling the dice	User must click on the corresponding widget on the GUI for the dice they would like to keep. Keeping all or none should be valid options	The user and the Player/Hand/Dice objects	The hand object should be updated with the dice that are not going to be rolled. Other objects that rely on these values should be updated/should reflect this change
High	At any point during their turn, the player should select a score line from the remaining, unused scores. This score represents the score from the current hand of dice	User must be able to enter a value/click a corresponding widget on the GUI that correlates to a scoreline. Must be scoreline that has not already been selected	The scorecard/hand/dice objects are all involved as well as the user	The corresponding scoreline should be marked as used, and should now have a value equal to a score. Whatever objects use this value should also reflect this change
High	After a players turn has come to completion, the game should switch to the next player's turn	Player's turn should inform the main game loop that a turn has ended/A player should be prompted to end turn and should click a button. If only one player is in the game, they should just take another turn. If the last player out of how many are participating ends turn, it should return to the first player's turn	The game loop and player objects and the user	The current object that is being used to play the game must make sure it is using the correct player's information
High	After all players have taken 13 turns, the amount of rounds in the game, the final scores should be determined	Needs to keep track of current turn of game/remaining scores tro be taken	Main game loop and the player objects	Final scores should be updated for all players, and should be displayed/reflected somewhere on the GUI. Should also reflect any bonuses added

High	After the final scores have been calculated, a winner should be announced	The final scores must all be compared for each player	Player objects and the main game loop	The player that has the highest score should be stored somewhere, and the GUI should change and reflect the winners
Low	When the game is first launched, a window should be launched that shows the Yahtzee logo and a button to begin the game	On launch, the window should be opened for the user. To move on from the screen, the user must click a button on the GUI	User and the main game loop	The game should progress to the setup for the game when the button is pressed
Non-Functional	Limit number of players to prevent game from being too long, 4 for simplicity	Player amount selection process will be capped at 4	main game loop	The game should take a normal amount of time
Non-Functional	The player's scorecard should be easy to access/see	Need to have the updated scorecard be shown on the main screen at all times for the current player	main game loop, player object/scorecard	The scorecard should be easily accessible
Non-Functional	Add a fun theme to make the game stand out	Need to find a cohesive theme and make it consistent throughout the entire program	All classes	The game should look like the theme we chose

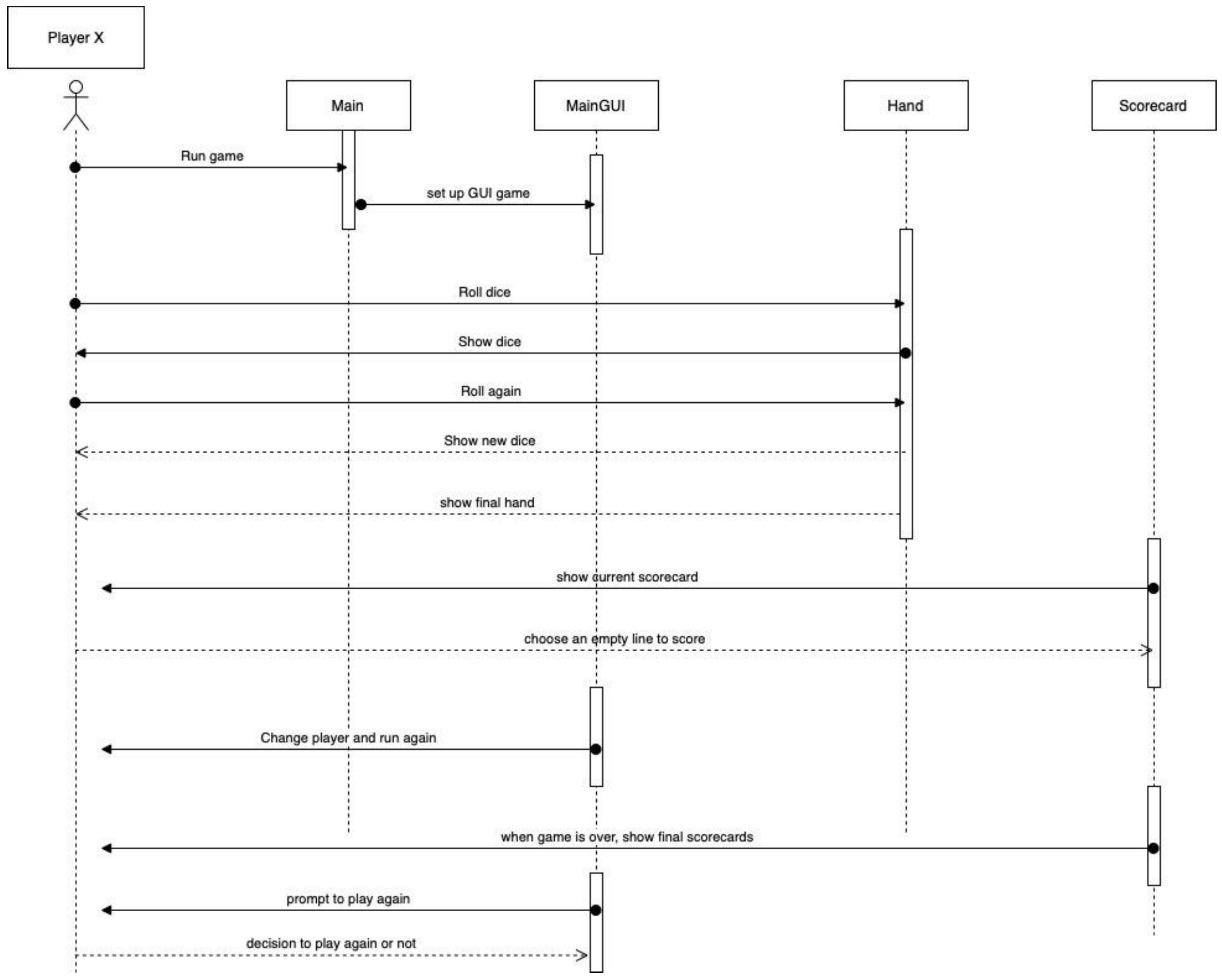
IV. Solution Approach

This section includes a breakdown of how our code was sectioned off into classes and distributed to work on among members. Our approach to this project is represented through the UML and sequence diagrams by showing event interaction and object oriented programming. The sequence diagram is a helpful interpretation of how the user interacts with the GUI interface. The UML diagram is a helpful representation of how different objects are used to store and call information.

UML Class Diagram:



UML Sequence Diagram:



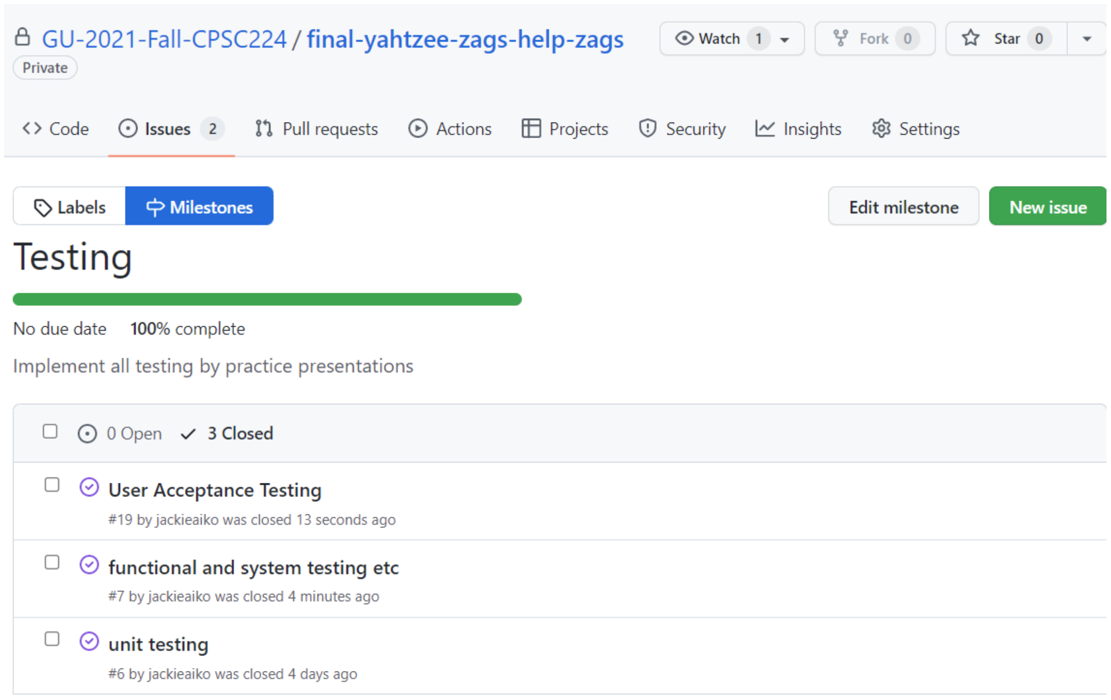
V. Test Plan

Plan:

1. Identify the requirements and goals for the software's testing.
2. Identify the necessary tests to be implemented for each module.
3. Create issues and milestones for types of testing and testing process.
4. Perform unit tests on single player configuration with 3 rolls, 5 dice, 6 side assumption
5. Document the findings throughout this testing process.
6. If bugs are found in testing, the issue is addressed and solved as a group during the hacking session.
7. After unit testing is complete for 1 player configuration, unit testing occurs for all configurations.
8. Identify the necessary tests to be implemented for integration and user acceptance testing.
9. Perform integration testing and fix any bugs that may arise as a result.

10. After review by the group, user acceptance testing by group members is performed and reported.

Results:



GU-2021-Fall-CPSC224 / final-yahtzee-zags-help-zags

Private

Watch 1 | Fork 0 | Star 0

Code | Issues 2 | Pull requests | Actions | Projects | Security | Insights | Settings

Labels | Milestones | Edit milestone | New issue

Testing

No due date 100% complete

Implement all testing by practice presentations

0 Open ✓ 3 Closed

User Acceptance Testing
#19 by jackieaiko was closed 13 seconds ago

functional and system testing etc
#7 by jackieaiko was closed 4 minutes ago

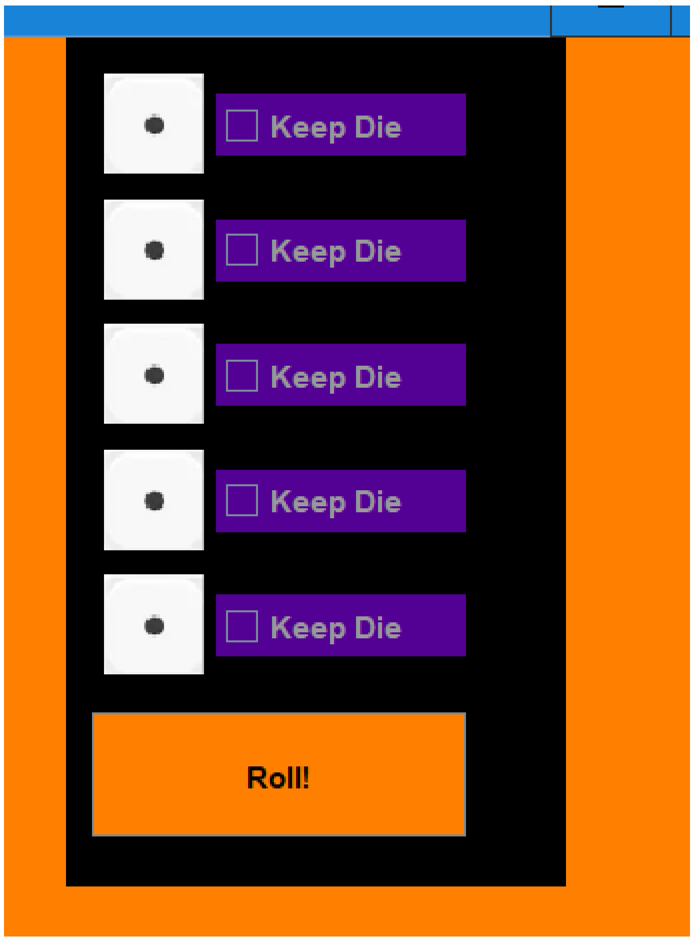
unit testing
#6 by jackieaiko was closed 4 days ago

VI. Project Implementation Description

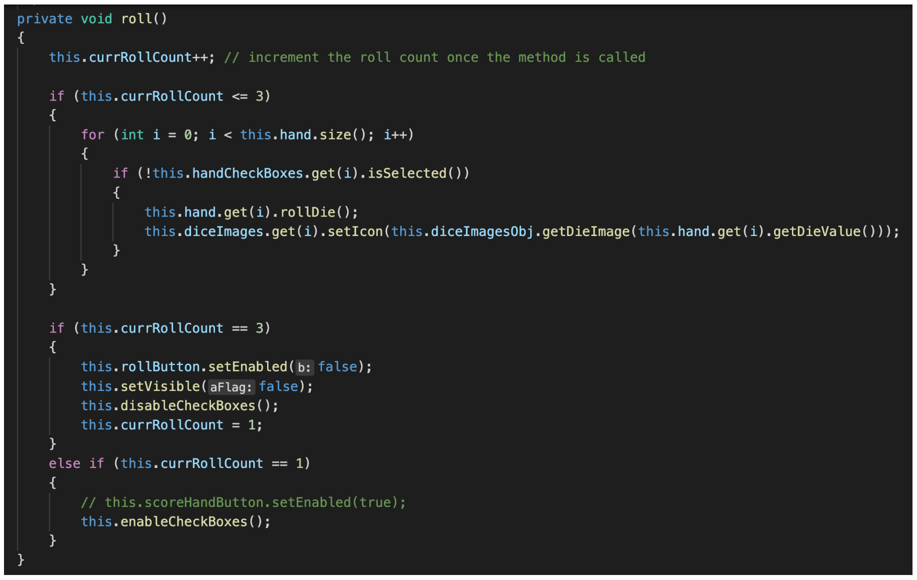
General Architecture:

- Hand and Rolls
- Scorecard and Scoring Key
- Player Tracking
- Overall GUI

Hand and Rolls:



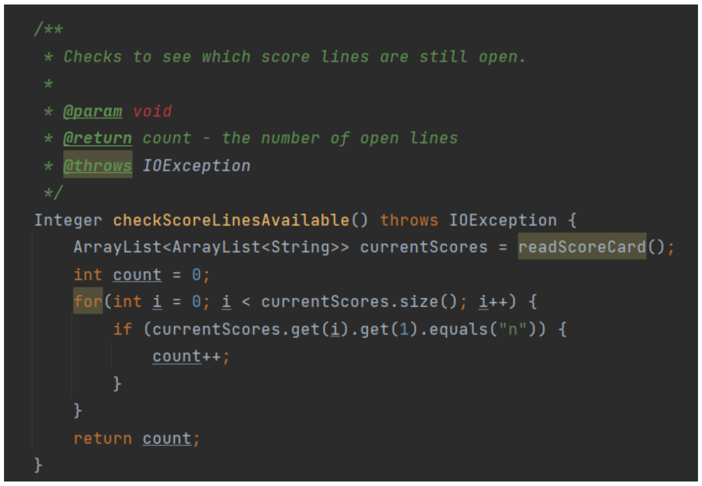
Pictured above is the GUI representation of our HandPanel object, which handles all rolling functionality for our program. Once instantiated, it is always used by each of the players, and the dice values are returned to other objects when needed.



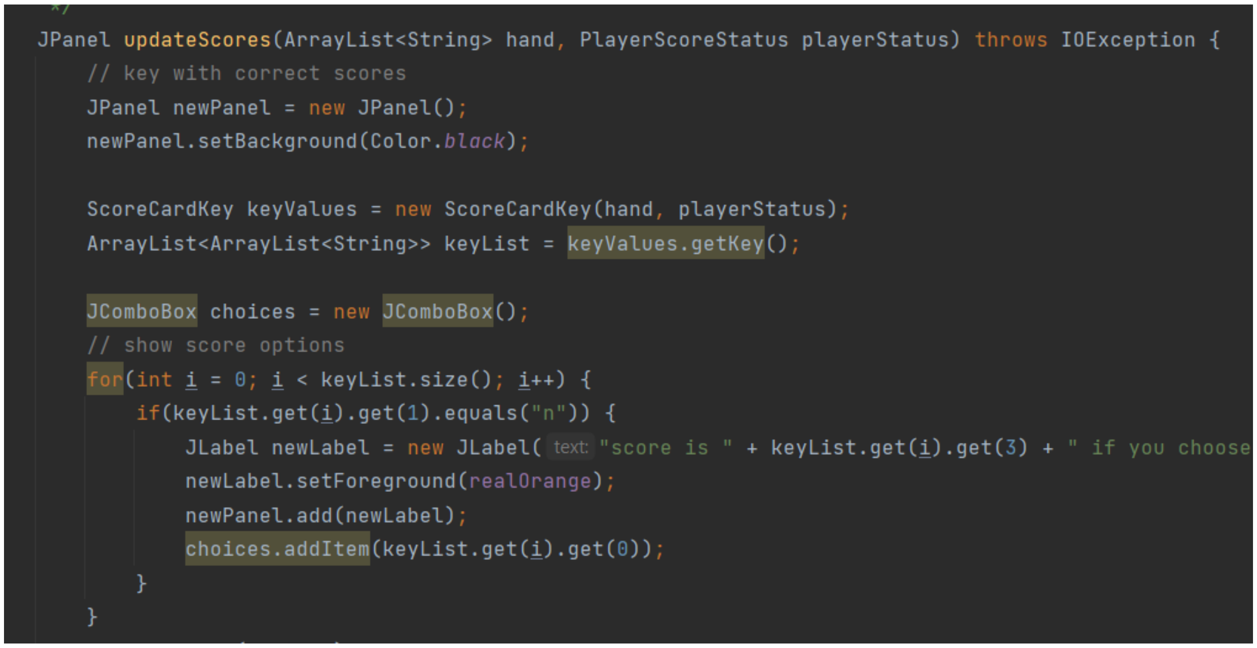
```java
private void roll()
{
    this.currRollCount++; // increment the roll count once the method is called

    if (this.currRollCount <= 3)
    {
        for (int i = 0; i < this.hand.size(); i++)
        {
            if (!this.handCheckBoxes.get(i).isSelected())
            {
                this.hand.get(i).rollDie();
                this.diceImages.get(i).setIcon(this.diceImagesObj.getDieImage(this.hand.get(i).getDieValue()));
            }
        }
    }

    if (this.currRollCount == 3)
    {
        this.rollButton.setEnabled( b: false);
        this.setVisible( aFlag: false);
        this.disableCheckBoxes();
        this.currRollCount = 1;
    }
    else if (this.currRollCount == 1)
    {
        // this.scoreHandButton.setEnabled(true);
        this.enableCheckBoxes();
    }
}
```

The above screenshot shows the code used to roll the dice. When the rull button is pressed, this method is run. It rolls each die in the hand, 5 in total, and updates the image associated with each die value so that the player always has an up to date view of their hand.

Scorecard and Score Key:



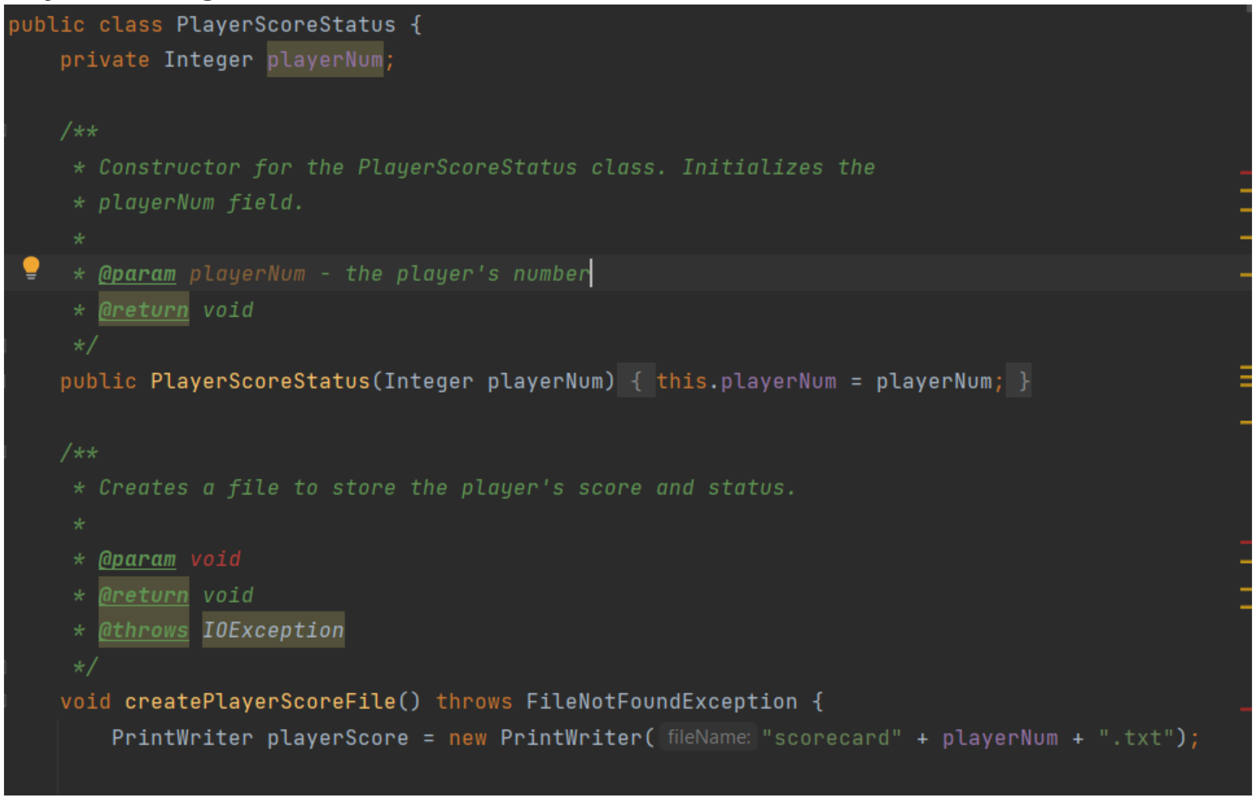
```java
/**
 * Checks to see which score lines are still open.
 *
 * @param void
 * @return count - the number of open lines
 * @throws IOException
 */
Integer checkScoreLinesAvailable() throws IOException {
    ArrayList<ArrayList<String>> currentScores = readScoreCard();
    int count = 0;
    for(int i = 0; i < currentScores.size(); i++) {
        if (currentScores.get(i).get(1).equals("n")) {
            count++;
        }
    }
    return count;
}
```



```java
JPanel updateScores(ArrayList<String> hand, PlayerScoreStatus playerStatus) throws IOException {
    // key with correct scores
    JPanel newPanel = new JPanel();
    newPanel.setBackground(Color.black);

    ScoreCardKey keyValues = new ScoreCardKey(hand, playerStatus);
    ArrayList<ArrayList<String>> keyList = keyValues.getKey();

    JComboBox choices = new JComboBox();
    // show score options
    for(int i = 0; i < keyList.size(); i++) {
        if(keyList.get(i).get(1).equals("n")) {
            JLabel newLabel = new JLabel( text: "score is " + keyList.get(i).get(3) + " if you choose
            newLabel.setForeground(realOrange);
            newPanel.add(newLabel);
            choices.addItem(keyList.get(i).get(0));
        }
    }
```

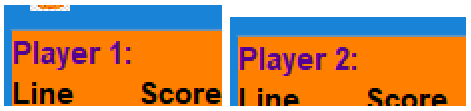
We wanted the scorecard to show the player’s current score and update accordingly. They must be able to pick their score and only have the available options for a score they have not picked. By checking which score lines are available, the panel to update the score only gives the user the score lines they are allowed to pick. This prevents errors and cheating when calculating and totaling the scores for each player.

Player Tracking:



```
public class PlayerScoreStatus {
    private Integer playerNum;

    /**
     * Constructor for the PlayerScoreStatus class. Initializes the
     * playerNum field.
     *
     * @param playerNum - the player's number
     * @return void
     */
    public PlayerScoreStatus(Integer playerNum) { this.playerNum = playerNum; }

    /**
     * Creates a file to store the player's score and status.
     *
     * @param void
     * @return void
     * @throws IOException
     */
    void createPlayerScoreFile() throws FileNotFoundException {
        PrintWriter playerScore = new PrintWriter( fileName: "scorecard" + playerNum + ".txt");
```

In order to make this a multiplayer game, we created a scorecard file for each player. We successfully completed this functional requirement by creating a constructor with a parameter that took in the player's number. By doing this, each player was assigned an object that could track and update their scorecard. In the GUI, this can be seen when the player's turn is changed.



Overall GUI:

The panels and frame derived from our code can be found in the Appendix. Each step in the player's interaction with the game contains key functional requirements to the tasks that allow the user to successfully play a full game of Yahtzee.

Project Github Repo:

[GU-2021-Fall-CPSC224/final-yahtzee-zags-help-zags: final-yahtzee-zags-help-zags created by GitHub Classroom]()

VII. Future Work

For future work on our project, we would like to add more features that we talked about, but did not end up implementing. These include things like inputted player names for each player and displaying the winner on the final screen separately instead of just displaying all player's scores among other similar features that do not impact how the game is played. We also would like to create a way to delete the text files that are created, as they may cause issues if they are a part of the github repo if we decide to actually release this. After we reach a point where we feel like we have a complete game that meets all of our requirements, we will work on packaging the game and releasing it for download.

VIII. Glossary

GUI - Graphical User Interface

Unit testing - process in which a small segment of code is tested for proper operation

User acceptance testing - a process in which the code is tested in the real world by a user

UML diagram - Unified Modeling Language diagram, visually represents the overall structure of our classes to help create a better understanding of the design of the project

Sequence diagram - Another UML diagram that represents how the main process of a program passes messages between each object over time

Functional Requirement - a feature that must be occur in order for the tasks to be accomplished

Nonfunctional Requirement- attributes of the system that influence the performance, reliability, and usability of the tasks

IX. References

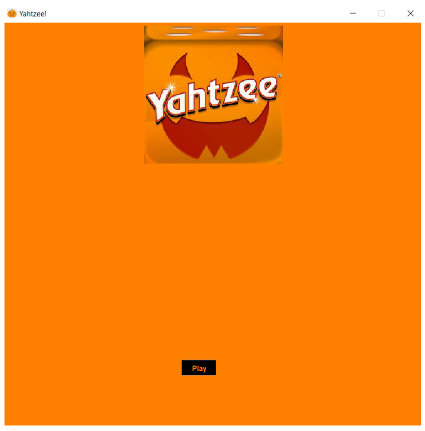
Cite your references here if you've got any. At the very least you can cite this:

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

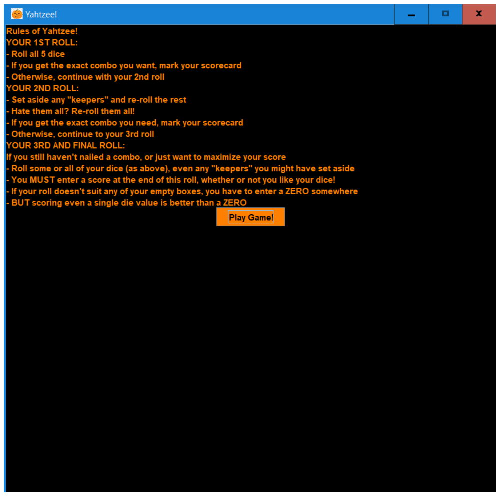
https://www.ultraboardgames.com/yahtzee/history.php

X. Appendix

A. Start Screen

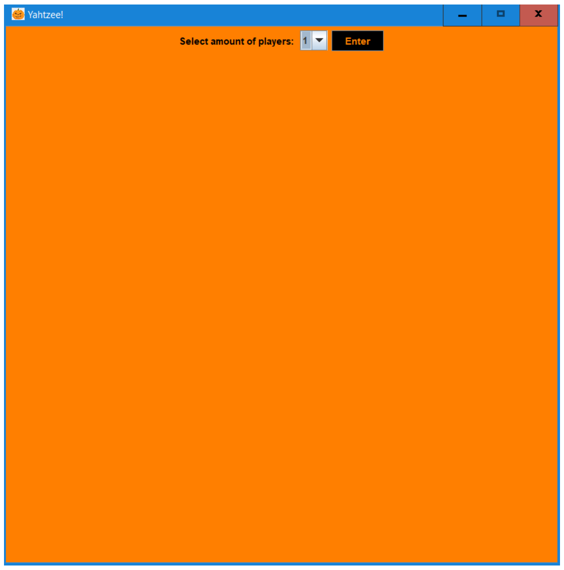


B. Rules Screen



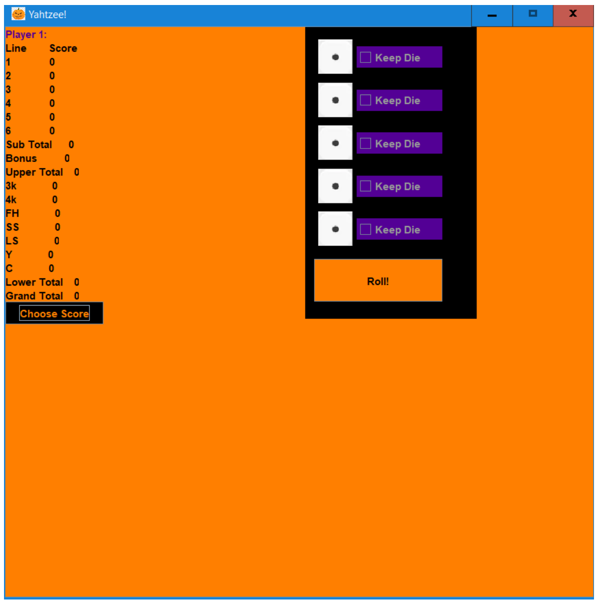
```
Rules of Yahtzee!
YOUR 1ST ROLL:
- Roll all 5 dice
- If you get the exact combo you want, mark your scorecard
- Otherwise, continue with your 2nd roll
YOUR 2ND ROLL:
- Set aside any "keepers" and re-roll the rest
- Hate them all? Re-roll them all!
- If you get the exact combo you need, mark your scorecard
- Otherwise, continue to your 3rd roll
YOUR 3RD AND FINAL ROLL:
If you still haven't nailed a combo, or just want to maximize your score
- Roll some or all of your dice (as above), even any "keepers" you might have set aside
- You MUST enter a score at the end of this roll, whether or not you like your dice!
- If your roll doesn't suit any of your empty boxes, you have to enter a ZERO somewhere
- BUT scoring even a single die value is better than a ZERO
```

C. Player Select Screen



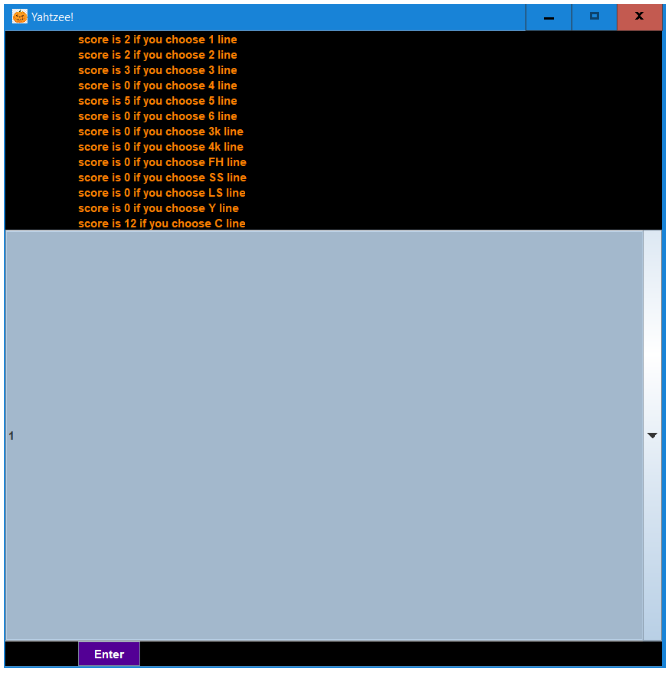
Select amount of players: 1

D. Main Game Screen



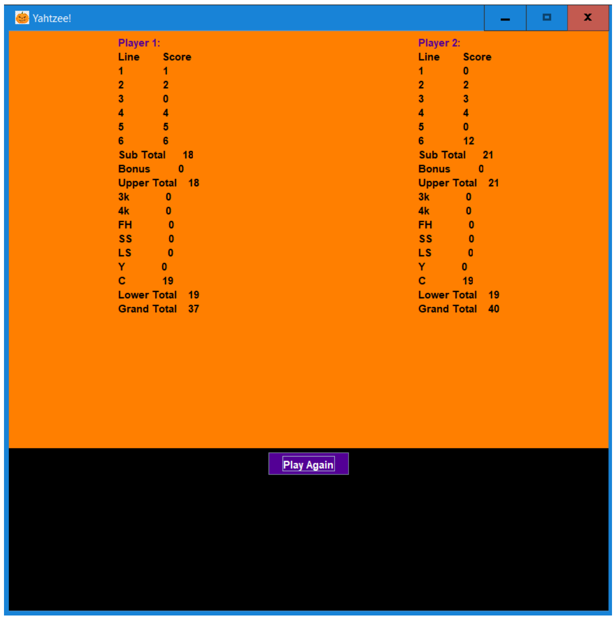
```
Player 1:
Line      Score
1         0
2         0
3         0
4         0
5         0
6         0
Sub Total    0
Bonus        0
Upper Total  0
3k         0
4k         0
FH          0
SS          0
LS          0
Y         0
C         0
Lower Total  0
Grand Total  0
```

E. Score Select Screen



```
score is 2 if you choose 1 line
score is 2 if you choose 2 line
score is 3 if you choose 3 line
score is 0 if you choose 4 line
score is 5 if you choose 5 line
score is 0 if you choose 6 line
score is 0 if you choose 3k line
score is 0 if you choose 4k line
score is 0 if you choose FH line
score is 0 if you choose SS line
score is 0 if you choose LS line
score is 0 if you choose Y line
score is 12 if you choose C line
```

F. Final Screen



```
Player 1:                      Player 2:
Line      Score                Line      Score
1         1                    1         0
2         2                    2         2
3         0                    3         3
4         4                    4         4
5         5                    5         0
6         6                    6         12
Sub Total    18                Sub Total    21
Bonus        0                 Bonus        0
Upper Total  18                Upper Total  21
3k        0                    3k        0
4k        0                    4k        0
FH        0                    FH        0
SS        0                    SS        0
LS        0                    LS        0
Y         0                    Y         0
C         19                   C         19
Lower Total  19                Lower Total  19
Grand Total  37                Grand Total  40
```