Yahtzee in Android

Organization of Programming Language

Samman Bhetwal

Ramapo College of New Jersey

1. **Bug Report**
2. Unfixed Bugs:

* None.

1. Fixed Bugs:

* The Help message was not logged during the demonstration which is now fixed.
* When the user got dice values which could fill in the Five Straight Category, but the Five Straight category was already filled and the Four Straight category was empty. Help suggested to Fill Five Straight category instead of suggesting to fill the Four Straight category.

1. **Features:**
2. Missing Features:

* None

1. Extra features:

* The game shows the updated total score every time a player scores points.
* There is a back button which can lead the user to the main menu of the game.

1. **Classes and Data Structures**
2. **Game Class**

* This class represents the main game logic for the Yahtzee game. It manages the scorecard, current round, players, dice, and player queue.

Data Structure:

* ScoreCard: An instance of the ScoreCard class that holds the current scores for each player, tracking their progress throughout the game.
* current\_round: An integer representing the current round number in the game.
* players: A list of Player objects representing all participants in the game.
* List<Die>: A list of Die objects representing the dice used in the game.

1. **Category Class**

* This class represents the different scoring categories in the Yahtzee game. It provides mappings between category enums and their display names, as well as utility methods to work with these categories.

Data Structure:

* Map<Category, String>: A map that associates each category enum with its display name.
* Map<String, Category>: A map that associates each display name with its corresponding category enum.
* List<Category>: A list of all category enums for easy iteration.

**c. Player Class**

* This class represents a player in the Yahtzee game. It manages player-specific information such as the player's name, score, and actions.

Data Structure:

* String: Used to store the player's name.
* int: Used to store the player's score.
* List<Integer>: Used to store the player's dice values.

**d. Human Class**

* This class represents a human player in the Yahtzee game. It extends the Player class to provide specific functionality for a human player.

Data Structure:

* Inherits all data structures from the Player class.

**e. Computer Class**

* This class represents a computer player in the Yahtzee game. It extends the Player class and implements specific strategies for the computer player to make decisions during the game.

Data Structure:

* Inherits all data structures from the Player class.

**f. Dice Class**

* This class provides methods to simulate rolling dice in the Yahtzee game. It includes methods to roll a single die and multiple dice.

Data Structure:

* List<Integer>: Used to store the values of the rolled dice.

**g. DiceAnalysis Class**

* This class represents the analysis of dice rolls in the Yahtzee game. It provides information about the remaining dice, the category being analyzed, the potential score, and the dice that can be kept to maximize the score.

Data Structure:

* List<Integer>: Used to store the values of the remaining dice and the dice that can be kept.
* Category: An enum representing the different scoring categories in Yahtzee.
* int: Used to store the potential score.

**h. Die Class**

* This class represents a single die used in the Yahtzee game. It provides methods to roll the die, lock it, mark it for help, and get its string representation.

Data Structure:

* int: Used to store the value of the die.
* boolean: Used to store the states of the die (locked, marked for lock, marked for help).

**i. DieAnalysis Class**

* This class analyzes the final roll of dice in a Yahtzee game to determine the best possible scores and categories. It provides information about the final roll, kept dice, rolled dice, and the scorecard.

Data Structure:

* List<Integer>: Used to store the values of the final roll, kept dice, rolled dice, and dice in hand for the final roll.
* -ScoreCard: An object representing the scorecard used to retrieve open categories and calculate scores.
* -List<Category>: Used to store the categories that yield the maximum score.
* -int: Used to store the maximum score and combined score.

**j. Help Class**

* This class provides assistance to the computer player in the Yahtzee game by determining the best move, including which category to choose and whether to keep, roll, or stand with the current dice.

Data Structure:

* Category: An enum representing the different scoring categories in Yahtzee.
* List<Integer>: Used to store the dice values that should be kept.
* boolean: Used to indicate whether the computer should stand

**k. Log Class**

* This class provides logging functionality for the Yahtzee game. It follows the Singleton pattern to ensure only one instance of the Log class exists and manages a list of log messages.

Data Structure:

* List<String>: Used to store the log messages.

**l. Scorecard Class**

* This class manages the scorecard for the Yahtzee game. It keeps track of the scores for each category, round, winner, and points, and provides methods to manipulate and retrieve scorecard information.

Data Structure:

* Map<Category, Optional<ScoreCardEntry>>: A map that associates each category with an optional scorecard entry.
* StringBuilder: Used to construct the formatted string representation of the scorecard.

**m. SingletonGame Class**

* This class manages a single instance of the Game object using the Singleton pattern. It provides methods to set and retrieve the current game instance, as well as to manipulate the game's state.

Data Structure:

* Game: An object representing the current game instance.
* ScoreCard: An object representing the scorecard of the game.

**n. App Class**

* Since I first wrote a working java Yahtzee code, the App class served as the entry point for the Yahtzee game application in CLI. It is designed to initialize the game, check if the user wants to load a saved game, and run the game loop until the game is over. The class also handles saving the game state after each round and displaying the final scores.

Data Structure:

* Game: An object representing the current game instance.
* ScoreCard: An object representing the scorecard of the game.
* List<Player>: A list to store the human and computer players participating in the game.
* Game: An instance of the Game class that manages the game state, including the scorecard, rounds, and players.
* Optional<Player>: Used to retrieve and display the winner of the game if there is one.

**o. Round Class**

* In CLI, The Round class is responsible for managing the rounds in the Yahtzee game. It includes methods to handle player order, resolve tie-breakers, and sort players based on their scores.

Data Structure:

* Queue<Player>: Used to manage the order of players for each round, including resolving tie-breakers and sorting players by their scores.
* Map<Player, Integer>: A map that associates players with their respective scores, used for sorting players based on their performance.

**p. Turn Class**

* In CLI, the Turn class is responsible for managing a single turn in the Yahtzee game. It handles the logic for rolling dice, keeping dice, and determining the player's strategy and target for the turn.

Data Structure:

* List<Integer>: Used to store the dice that the player decides to keep during their turn.
* Optional<Map<Category, Reason>>: Represents the player's pursuit strategy, indicating which category they are aiming for and the reason behind it.
* Optional<Map. Entry<Category, List<Integer>>>: Represents the specific dice the player is aiming to roll for their target category.

**q. DiewView Class**

* This class represents a custom view for displaying and interacting with a die in the Yahtzee game. It handles user touch events and notifies listeners of die state changes.

Data Structure:

* Die: An object representing the die whose state is being displayed and interacted with.
* MotionEvent: An object representing touch events.

**r. FirstPlayerDeterminationActivity Class**

* This class manages the activity for determining the first player in the Yahtzee game. It handles the logic for rolling dice, setting dice values, and determining the winner of the tiebreaker.

Data Structure:

* ImageView: Used to display the die images.
* RotateAnimation: Used to animate the rotation of the die images.
* Player: Represents a participant in the game (Human or Computer).
* Log: Used to log messages related to the game state.

**s. GameActivity Class**

* This class manages the main game activity for the Yahtzee game. It handles the user interface, game logic, and interactions during the game.

Data Structure:

* List: Used to store collections of objects.
* Map: Used to store key-value pairs.
* Optional: A container object which may or may not contain a non-null value.

**t. LogActivity Class**

* This class manages the activity for displaying the game log in the Yahtzee game. It initializes the user interface, displays the log content, and handles user interactions.

Data Structure:

* Bundle: Used to pass data between activities and save the state of the activity.
* TextView: Represents a text widget for displaying the log content.
* Button: Represents a push-button widget for user interactions.

**u. MainActivity Class**

This class serves as the main entry point for the Yahtzee game's user interface. It initializes the main activity's layout, configures UI behavior, and handles user interactions.

1. **Log**

**Oct 30 – Nov 1**

* Integrated the Game classes to manage the flow of rounds and player turns more efficiently. Ensured that the game progresses from one round to the next, handling player interactions and score updates automatically. (5 hours)
* Improved turn-based interactions, adding prompts for the human player to manage dice rolls and re-roll decisions. (2 hours)  
  **Total Hours: 7 hours**

**Nov 2 – Nov 4**

* Refined the game structure by adding better error handling for player actions such as invalid dice selections and score entries. Updated the user interface to provide clearer instructions and feedback. (6 hours)
* Began research on Java for Android to port the current Yahtzee game to an Android platform using Android Studio. Focused on understanding how to integrate the current Java code into an Android app framework. (5 hours)  
  **Total Hours: 11 hours**

**Nov 5 – Nov 7**

* Finalized the turn-handling system with a more fluid player transition between the human and computer player turns, ensuring that the game state properly updates after each player's action. (2 hours)
* Worked on adjusting the Game class to better manage the switching between turns and score updates. (1.5 hours)  
  **Total Hours: 3.5 hours**

**Nov 8 – Nov 10**

* Created the SingletonGame class for centralized management of the game's state and to ensure consistency throughout the game session. This class tracks game state across multiple screens or activities. (4 hours)
* Implemented basic game-saving functionality by enabling the serialization of game data, allowing for saving and loading the game state to/from a file. (4 hours)  
  **Total Hours: 8 hours**

**Nov 12 – Nov 14**

* Worked on the Android version of the Yahtzee game, implementing basic view layouts and controllers for player interactions. Adapted core gameplay logic for mobile touch interfaces. (6 hours)
* Conducted usability testing of the Android app's initial layout, focusing on user experience and game flow. Fixed issues such as unresponsive buttons and unclear instructions. (5 hours)  
  **Total Hours: 11 hours**

**Nov 16 – Nov 18**

* Focused on improving the AI (computer player) logic, enhancing decision-making to prioritize high-value categories based on dice rolls. Integrated functions like get\_sequence and longest\_sequence\_length into the decision-making process. (4 hours)
* Conducted extensive testing to ensure the computer player follows the optimal strategy, refining its actions based on real-time feedback. (6 hours)  
  **Total Hours: 10 hours**

**Nov 19 – Nov 21**

* Finalized the Android version of the Yahtzee game, ensuring smooth transitions between activities, proper game state management, and an intuitive user interface. (8 hours)
* Completed debugging and tested the Android app across multiple devices to ensure compatibility and stability. (4 hours)  
  **Total Hours: 12 hours**

**5. Generative AI Assistance**

Location in Code:

Class: Computer

Function: getHelp ()

Functionality: The computer logic to keep dice values and for human to provide help.

Nature of Assistance:

Help in debug. The program crashed during demo, and I checked the logcat which said there was a null pointer exception.

Description of Help:

The AI provided guidance on using Log cat and helped me find out the bug and the nature of the bug easily.

When I figured out the I was getting a null pointer exception I immediately figured out that we did not get any category that the computer was trying to fill because the dice values did not fill that category. Therefore, the category we received was null and adding an if statement helped solve the problem.

* 1. **Screenshots**

A screenshot of a video game

Description automatically generated A screenshot of a video game

Description automatically generated

Fig: Main menu Fig: First player determine

A screenshot of a game

Description automatically generated A screenshot of a game

Description automatically generated

Fig: Winner of the first dice roll Fig: The Main game screen layout

A screenshot of a game

Description automatically generated A screenshot of a game

Description automatically generated

Fig: Human setting aside Die Fig: Human asking for help(above help button)

A screenshot of a computer game

Description automatically generated A screenshot of a game

Description automatically generated

Fig: Computer scoring Full house Fig: Computer Keeping [2,2] and program

In Round 7 and displaying scores displaying potential categories

A screenshot of a game

Description automatically generated A screenshot of a computer

Description automatically generated

Fig: Computer scoring 3 of a kind Fig: Program displaying log of computer

Scoring 3 of a kind

A screenshot of a computer game

Description automatically generated A screenshot of a phone

Description automatically generated

Fig: Display Winner in GUI Fig: Display Winner in Logs

A screenshot of a game

Description automatically generated A screenshot of a computer

Description automatically generated

Fig: Saving a game and loading a saved game

A screenshot of a cell phone

Description automatically generated

Fig: Loading the saved game.