**Rubric for Assessing Yahtzee Game in Android**

Name: Samman Bhetwal

Carefully **highlight** **all** the items that **work correctly**. Incorrect entries may be penalized. Not all the entries may be used for grading.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Setup** | | | | | |
| **Players** | One player is Human | One player is computer |  | Players alternate |  |
| **ScoreCard** | Has 12 categories | Categories are named | Winner of each category is shown | Points earned for the category is shown | Round when the category was filled is shown |
| **First player** | Each player tosses a die | Player with the larger die plays the first turn | The other player plays the second turn |  | If a tie, dice are tossed repeatedly till a first player is determined |
|  | On second and subsequent rounds, player with the lower score plays first | |  | If a tie in the scores, dice are tossed to determine the first player | |
| **Computer player** | | | | | |
| **First roll** | Rolls 5 dice | Announces which dice will be set aside and not re-rolled |  | Lists all the scorecard categories applicable to the dice set aside | Does not list any category already filled |
|  | Announces the category or categories it wants to pursue | Explains why it wants to pursue that category or categories | The explanation is context-specific | Lists the range of points it can earn with the selected category or categories | States whether it wants to stand or roll again |
| **Second roll** | Rolls up to 5 dice | Does not roll dice set aside from the first roll |  | Lists all the scorecard categories applicable to the dice set aside | Does not list any category already filled |
|  | Announces the category or categories it wants to pursue | Explains why it wants to pursue that category or categories | The explanation is context-specific | Lists the range of points it can earn with the selected category or categories | States whether it wants to stand or roll again |
| **Third roll** | Rolls up to 5 dice |  | Does not roll dice set aside in the first roll | Does not roll dice set aside in the second roll |  |
| **End of turn** | Announces the category that will be filled in that turn | Announces the points earned in that category |  | Updates the Scorecard with its name, points and round number |  |
| **Computer’s Strategy** | | | | | |
| **Strategy** | Has strategy for which dice to set aside after each roll | Has strategy for which category to pursue after each roll | Has strategy for whether to roll again or stand |  | Has strategy for which category to fill at the end of the turn |
| **Describe the strategy used to determine the dice to set aside after each roll** | | 1. Generate all possible final rolls based on current kept dice.  2. Calculate scores for all possible final rolls.  3. Find the best roll with the highest score and minimal difference.  4. Determine which dice to keep from the final roll and kept dice. | | | |
| **Describe the strategy used to decide which category to prioritize after each roll** | | Get the best possible final roll based on the strategy above.  2. Pursue the strategy of the best roll. | | | |
| **Describe the strategy used to decide whether to roll again or stand** | | 1. Get best possible final roll  2. Calculate the difference between best possible final roll and current roll.  3. If the current roll is equal to the difference stand else roll again. | | | |
| **Describe the strategy used to decide which category to fill at the end of the turn** | | 1. Get all unfulfilled categories and applicable categories.  2. Calculate the intersection of the two.  3. Calculate score for each intersecting category.  4. Select category with highest score | | | |
| **Human player** | | | | | |
| **First roll** | Must roll 5 dice | Will announce which dice will be set aside and not re-rolled | Will list all the scorecard categories applicable to the dice set aside | Cannot list any category already filled | Can list only correct categories |
|  | Will announce the category or categories to pursue | Category to pursue is correct |  |  | Can stand or roll again |
| **Second roll** | Can roll up to 5 dice | Cannot roll dice set aside in the first roll | Will list all the scorecard categories applicable to the dice set aside | Cannot list any category already filled | Can list only correct categories |
|  | Will announce the category or categories to pursue | Category to pursue is correct |  |  | Can stand or roll again |
| **Third roll** | Can roll up to 5 dice |  | Cannot roll dice set aside in the first roll | Cannot roll dice set aside in the second roll |  |
| **End of turn** | Will identify the category that will be filled in that turn | The category to be filled is correct | Will identify the number of points earned for the category | The number of points is correct | Will update the Scorecard with name, points and round number |
| **Playing the Game** | | | | | |
| **End of Turn** | Turn ends after the third roll |  | Turn ends after the player stands |  |  |
| **End of Round** | Round ends when each player has taken one turn | Scorecard correctly updated for both players | The total score of each player is announced | The total score announced is correct | Round can end after only one player has taken turn |
| **End of Tournament** | Tournament ends when all the Scorecard categories have been filled |  | The completed Scorecard is displayed | The total score of each player is announced | The total score announced is correct |
|  | Player with the greatest score is announced as the winner of the tournament | Tournament is a draw if both players have the same number of wins |  |  | Program exits after announcing winner of the tournament |
| **Implementation Features** | | | | | |
| **Serialization** | Provides option to save the game after each round | Game is saved into text file | Correct format used for text file | Game state correctly saved | Game quits upon serialization |
|  | Provides option to resume the game from the text file | Prompts for the name of the text file |  | Resumes loaded game |  |
| Correctly restores round number | Correctly restores the Scorecard |  |  |  |
| **Dice** | For each roll, asks whether to generate or manually input dice | If generate, generates dice randomly |  | If manual input, reads the required number of dice from the user | Validates dice input |
| **Help Mode** | User has the option to ask for help after each roll |  | Help advises on the dice to set aside from that roll | Help lists all the scorecard categories applicable to the dice set aside so far |  |
|  | Help recommends the category or categories that should be pursued | Help explains why it recommends the category or categories | The explanation is context-specific | Help lists the range of points that can be earned for each category being pursued |  |
|  | Help recommends whether to roll again or stand | At the end of the turn, help recommends the category that should be filled |  |  | Computer uses its own strategies to provide help |
| **Graphical User Interface** | | | | | |
| **GUI for input/Input validation:** | To start a game using a text file | The selection of the text file from which to resume a game |  | To suspend a game after a turn | The selection of the file in which to save the game |
|  | Manual input of dice |  | Asking for help from the computer |  |  |
|  | Input of dice to set aside/re-roll | Input of categories applicable to the dice set aside so far | Input of category or categories to be pursued |  | Input of whether to roll or stand |
|  | Input of the Scorecard category to be filled at the end of the turn | Input of points earned in the turn to be entered into the Scorecard | Input of round number to be entered into the Scorecard |  |  |
| **GUI for Output:** | Scorecard displayed in tabular format | Scorecard correctly updated after each turn |  | Dice |  |
|  |  |  | Help provided graphically |  | Computer’s move displayed graphically |
| **GUI for Game State:** | Total score displayed at the end of each round | Total score displayed at the end of the tournament | Next player displayed |  | Winner/loser of the tournament |
| **GUI Quality** | Minimizes text input | Precludes invalid inputs |  |  |  |
|  | Interface is functional – only necessary widgets displayed | Interface is logical – widgets displayed only at the correct time | Interface is intuitive – user always knows what options are available next |  | Interface is aesthetically designed |
| **Design** | | | | | |
| **MVC design** | Model and view are separate classes | All game logic handled by the model | All game state saved in the model only |  |  |
| Model has no reference to the view, but view has reference to the model | View updates itself based on the model |  | Controller updates the model only, not the view | Controller refreshes the view when necessary |
| **Object-oriented design** | At least 6 classes are included (as listed below) | Each class is complete – self-contains all the necessary functionality |  | Inheritance is used for player classes: computer and human inherit from a base class |  |
| **Code Design – Data flow** | Data: Only independent variables saved, dependent variables saved sparingly, only for efficiency | Data is *not* saved redundantly, no potential fidelity problems in data storage | Data is encapsulated – access to data is controlled | Changes to data always validated |  |
| **Code Design – Control flow** | Overall design is hierarchical, and evident in main() | Code for repeated execution separated from  code for single execution (e.g., of round) | | Display issues separated from problem logic (Model Vs View) | |
| **Code Reuse** | Code properly factored out of if-else, loops | Functions defined for any code executed more than once | Each function in charge of only one logical task |  |  |
| **Implementation** | | | | | |
| ***Scorecard***  **Class** | All data members are private | Constructor initializes *all* data members | Selectors are const, don’t break encapsulation | Mutators validate input, don’t break encapsulation |  |
| ***Player***  **Class** | All data members are private | Constructor initializes *all* data members | Selectors are const, don’t break encapsulation | Mutators validate input, don’t break encapsulation |  |
| ***Human***  **Class** | All data members are private | Constructor initializes *all* data members | Selectors are const, don’t break encapsulation | Mutators validate input, don’t break encapsulation |  |
| ***Computer***  **Class** | All data members are private | Constructor initializes *all* data members | Selectors are const, don’t break encapsulation | Mutators validate input, don’t break encapsulation |  |
| ***Round***  **Class** | All data members are private | Constructor initializes *all* data members | Selectors are const, don’t break encapsulation | Mutators validate input, don’t break encapsulation |  |
| ***Tournament***  **Class** | All data members are private | Constructor initializes *all* data members | Selectors are const, don’t break encapsulation | Mutators validate input, don’t break encapsulation |  |
| **Identifiers** | All classes have names corresponding to nouns in the problem description | All client functions have names corresponding to verbs in the problem description | Any abbreviations in the names are readable |  |  |
| **Coding style** | No global variables used | Symbolic constants are used whenever possible | All literal constants are explained at *each* occurrence | Principle of least privilege used for parameter passing |  |
| **Courtesy Programming** | | | | | |
| **Listing** | Code is indented properly |  |  | Classes are listed from basic to composite and derived | Each class listed in the following order: public, protected and private |
| **Documentation** | Every function has a complete header | Within each function, code is properly commented – steps in the algorithm are listed | Comments in the code describe semantics, not syntax | Comments in the code do not have spelling/ grammatical errors. |  |
| **Submission - Manual** | | | | | |
| **Screen shots of:** | First player of the round being determined | Computer’s move being explained | Computer providing help |  | Winner of the tournament being announced |
| **Includes:** | Bug report | Missing features report |  | Project log |  |
|  | Description of classes | Description of data structures |  | Source and documentation are placed in a directory and the directory is zipped | |
| **Milestones uploaded?** | First: | | Second: | | |

***Do not delete the blank pages below***