Yahtzee

Coons, John A.

CIS-5

Fall 2017

48978

|  |  |
| --- | --- |
| **TABLE OF CONTENTS** |  |
| **INTRODUCTION** | **1** |
| **THE BASICS** | **1-4** |
| Game Objectives  Rules | **1**  **1-2** |
| Scoring | **2-3** |
| **DESIGN PROCESS** | **3-9** |
| Function Concepts | **4** |
| User Accessibility/Info | **4-9** |
| **IMPLEMENTATION** | **10** |
| Pseudocode | **5** |
| Flowchart | **5** |
| **CONSTRUCTS & CONCEPTS UTILIZED** | **9** |
| Project Checklist | **9** |
| **SCREEN SHOTS** | **11** |
| **REFERENCES** | **11** |
| **SOURCE CODE** | **15** |

**INTRODUCTION**

The following pages will cover the design process and development of a text based version of the classic American game Yahtzee. Yahtzee is a dice game that was made by Milton Bradley in the early 1940’s. Since its inception it’s been a staple in family games. In real life the game consists of five dice, a cup, and something to keep score with. In this version of the game no additional materials will be required other than the necessary equipment to run the program.

**THE BASICS**

**Game Objectives**

Have the most points at the conclusion of the thirteenth round.

**Rules**

The rules are simple. Up to four people may play at one time. Single player is also available. Prior to beginning multiplayer games, players take turns rolling all five dice. The player with the largest sum starts the game and the second largest goes second, etc. This order is maintained throughout the entirety of the game. Each game of Yahtzee consists of 13 rounds. Each player begins their turn by rolling all 5 dice. Next the player decides which dice to keep and which to roll again. A player can choose to roll all the dice again or none, concluding their turn. Players may roll up to three times per round. After the third roll players must enter a score *(see the Scoring section pgs. 2-4)*. Players can decide to stop rolling at any point and record the current value of their dice. However, once the player has recorded a score for that round their turn is over. The player with the largest sum of points at the end of the 13th round is deemed the winner!

**Scoring**

The scoring card is separated into a top half and a bottom half totaling 13 categories; 1-6 on top and 7-13 on the bottom:

\*\*\* Upper Section \*\*\*

1)'Aces' - Add all ones in hand, and enter the sum to score.

2)'Twos' - Add all twos in hand, and enter the sum to score.

3)'Threes' - Add all threes in hand, and enter the sum to score.

4)'Fours' - Add all fours in hand, and enter the sum to score.

5)'Fives' - Add all fives in hand, and enter the sum to score.

6)'Sixes' - Add all sixes in hand, and enter the sum to score.

\*\*\* Lower Section \*\*\*

7)'3 of a Kind' - Add all 5 dice, and enter sum to score.

8)'4 of a Kind' - Add all 5 dice, and enter sum to score.

9)'Full House' - 3 of a kind, and a separate pair. Worth 25 points.

10)'Small Straight' - Roll 4 sequential numbers. Worth 30 points.

11)'Large Straight' - Roll 5 sequential numbers. Worth 40 points.

12)'YAHTZEE!' - Roll 5 of the same number. Worth 50 points.

13)'Chance' - Add the total of all five dice, and enter sum to score.

* *Categories 1-6 are easily explained in the above list.*
* *To score categories 7 & 8 you must have three or four dice, respectively, of the same value. Then the player takes the sum of all the dice (not just the ones involved in the “Three/Four of a Kind”) and scores it in the appropriate category.*
* *Categories 9-12 are easily explained in the above list.*
* *Category 13 is used whenever the player chooses, but usually once all available options have been exhausted.*

A player must enter a value into a category each round. Once a category has been scored it is out of play for the rest of the game. **For example**: A player rolls 4 number 5 dice and 1 number 2 die on their first roll and scores it as a “Four of a Kind” using their 5’s. Two turns later the same player rolls 4 number 3 dice and 1 number 6 die on their first roll and wishes to score it as a “Four of a Kind” with 3’s. However, since they have already entered a value into that category two turns earlier, they must choose another scoring option. Instead of scoring a “Four of a Kind” with 3’s the player could score 12 points under the “Fours” category using the sum of their number 3 dice.

**DESIGN PROCESS**

**Function Concepts**

For this program to be successful I needed five functions that essentially accomplished the following tasks:

1. Find the sum of the initial roll of all five dice to find out player order in multiplayer mode.
2. Generate five random dice rolls for normal play (both single player and multiplayer mode).
3. Import title.
4. Import instructions.
5. Import scoring info.

**User Accessibility/Info**

# Starting a multiplayer game.

|  |
| --- |
| **Fig. 1**   * Initial Execution   + Displays rules and asks for number of players. |
| **Fig. 2**   * Enter desired number of players.   + ‘2’ was entered for this example. * Displays instructions for deciding who begins the game and each subsequent round.   + ‘If ‘1’ was selected for the number of players then this process is skipped (*see Gameplay section to see where the program continues after entering ‘1’*). | | |

|  |  |
| --- | --- |
| **Fig. 3**   * The player with the largest sum goes first at the beginning of the game and each round.   + In this example player 1 wins. * Enter player names.   + I chose player 1 and my brother, Matthew, was assigned player 2 **(*prior to rolling*)**. | |
| **Fig. 4**   * After entering the last players name the game continues on as normal (*see Gameplay section*), rotating between players in the assigned order. | |

**Gameplay**

After deciding who goes first, the only difference between single player and multiplayer is the alternating between player turns. Below is an example of a round of single player. For multiplayer the process would be the same with additional players completing their rounds as well. Since the initial execution of single player was also shown in the above example of multiplayer it has been left out *(see* ***Fig. 1 & 2****)*.

\*\*\*\*\*\*\*\*\*\*CONTINUED ON NEXT PAGE\*\*\*\*\*\*\*\*\*\*

|  |
| --- |
| **Fig. 5**   * Displays result of first roll. * Choose which dice to keep (enter ‘1’) and which to reroll (enter ‘0’).   + The order in which you enter your five numbers for your next roll dictate which dice get rolled and which ones do not. **(*Just look at the picture…)***   + I chose to reroll the 6, 2 and 1 in hopes of getting more 4’s. |

|  |
| --- |
| **Fig. 6**   * Repeat the process from the first roll.   + I changed my strategy because my second roll gave me a small straight (2, 3, 4, and 5). So, I chose to reroll my second 4 in hopes of getting a large straight. |

|  |
| --- |
| **Fig. 7**   * Final roll is displayed along with scoring options and instructions.   + I ended up rolling a 6 which gives me a large straight. |

\*\*\*\*\*\*\*\*\*\*CONTINUED ON NEXT PAGE\*\*\*\*\*\*\*\*\*\*

|  |
| --- |
| **Fig. 8**   * Enter the number corresponding to the category in which you wish to score.   + In my case it was 11 for a large straight * Enter the instructed value to confirm your score.   + I entered 40; the prompted score and point value of a large straight. |

**Termination**

|  |
| --- |
| * After the last person has completed the 13th round the score is shown in descending order.   + This is the same display for both single player mode and multiplayer mode. |

**IMPLEMENTATION**

**Pseudocode**

*Bring in system libraries*

*Function prototypes*

*Enter main, create RNG seeder*

*Create vector that stores integer and*

*String for a list of the final scores*

*Call title() and inst() functions to*

*display the title and instructions*

*Initialize variable pAgain to “Y”. Used*

*to either stop or restart the game loop.*

*Begin while loop while pAgain == “Y”.*

*This is the game loop.*

*Prompt user for number of players.*

*stored as int variable “numP”.*

*Receive input for number of players.*

*Nested while loop to confirm valid input.*

*Create a vector to store the names of the players.*

*Nested if statement used to get the name of a*

*player during a solo game.*

*Begin else if loop for multiplayer.*

*Declare string variable named message.*

*Used once all players are ready and enter “ok”.*

*Output instructions on determining order.*

*While loop that accepts “ok” from the user*

*As long as message=false.*

*Create vector initR to store and sort each player's initial role.*

*Vector is destroyed upon exit of if statement.*

*Create a for loops that calls the sum() function*

*displays the return and stores the roll in the vector.*

*Sort the initial rolls and get the two largest values alone.*

*Create a while loop that compares the two largest sums*

*and allows the two users to reroll upon a tie.*

*Create a for loop to store names in vector “names”*

*for multiplayer mode.*

*Create two arrays (4x13) of type bool (sComb) and*

*int (combPoints) to store player scores and track*

*used categories.*

*Use for loop and nested for loop to initialize arrays to “false”*

*and “0” respectively.*

*Create Yahtzee bonus array and for loop to add the Yahtzee*

*bonus to applicable scores.*

*Use a cons int of “13” to define the number of rounds.*

*Initialize variable “round” to “1” to set the beginning*

*round at one.*

*Start while loop that ensures the game ends after 13*

*rounds.*

*Start a for loop that cycles through every player each round.*

*Declare string variable “keepers”.*

*Create a 6 element array name “die”.*

*Output telling user whose turn it is.*

*Output telling user the result of their first roll.*

*Uses a for statement that calls the rollDie(t2) function*

*6 times and stores the 6 random results in array “die”.*

*Prompt user to decide which dice to keep.*

*User decides which dice to keep.*

*Stored under string variable named “keepers”.*

*Use for loop to step through the “keepers” string and dictate which*

*dice get rolled again.*

*Repeat the above steps for the second and third roll.*

*Call info() function to display the scoring instructions.*

*Define goto position as tryAgn.*

*Declare int variable “option” for storing user’s scoring option*

*and prompt user for selection.*

*Assign user selection to variable “option”.*

*Create a switch using int variable “option” as the control statement.*

*Each switch case is essentially the same using if else statements.*

*If the elements of array sComb that corresponds to the selected*

*value is equal to true than program goes back to tryAgn.*

*else the user can enter the value of their score and store it in*

*array sComb element corresponding to the selected category.*

*Increment counter for while loop (game loop).*

*Create int array named fScores to store the final scores of all the players.*

*Create two arrays uTotals[4] and lTotals[4].*

*Use a for loop to store the point values from the upper half and lower half*

*of the scoring card into the arrays made in the last step.*

*Create for loop to store player’s final scores.*

*Initialize variable “tempScr to “0”.*

*Use if statement to add 35 points to users score in the upper total if greater than 63.*

*Also adds 100 in the player has Yahtzee bonus.*

*tempScr array saved to fScores array.*

*fScores and names arrays are paired together and saved to pScores array.*

*Output score header.*

*Sort player scored from high to low and output the list.*

*Prompt players to play again or quit.*

*Obtain use input on whether or not to play again.*

*Use while loop to cause the game loop to reiterate as long as a user input “y” to play again.*

*Clear name and fScores arrays.*

*Create function for rolling 6 random dice and obtaining the sum of them for all users.*

*Create a function for a single dice roll used in all games, but not the placement roll.*

*Create three functions that import the title, game rules and scoring information.*

*Terminate program.*

**CONSTRUCTS & CONCEPTS UTILIZED**

**Libraries**

<iostream> Library

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Frequency | Description | Location |
| cout |  |  | 33, 38, 39, 50, 60, 63, 64, 66, 68, 73, 82, 92, 98, 104, 109, 143, 145, 152, 154, 155, 167, 168, 170, 183, 184, 186, 193, 203, 208, 218, 223, 233, 238, 248, 253, 263, 268, 277, 282, 292, 297, 307, 312, 322, 327, 337, 342, 352, 357, 371, 377, 379, 381, 383, 386, 388, 395, 405, 410, 418, 462, 465, 472, 475, 476, 481. |
| cin |  |  | 34, 41, 51, 69, 74, 110, 157, 173, 195, 210, 225, 240, 255, 269, 284, 299, 314, 329, 344, 359, 397, 412, 477, 483 |
| getline() | 3 | Inputs string data. | 521, 539, 558 |

<string> Library

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Frequency | Description | Location |
| string |  |  | 25, 29, 44, 48, 58, 108, 141, 469, 520, 538, 557. |
| getline() | 3 | Inputs string data. | 521, 539, 558 |
|  |  |  |  |

<vector> Library

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Frequency | Description | Location |
| vector |  |  | 25, 77, 429, 469 |
| getline() | 3 | Inputs string data. | 521, 539, 558 |
| begin() | 1 | Returns the iterator to the first element in the container. | 469 |
| end() | 1 | Returns an iterator pointing to the end of the container | 470 |
| rbegin() | 2 | Returns a reverse\_iterator pointing to the last element in the container. | 87, 468 |
| rend() | 2 | Returns a reverse\_iterator pointing to the first element in the container. | 87, 468 |

<algorithm> Library

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Frequency | Description | Location |
| sort |  | Sort elements in range. | 87, 468 |

<ctime> Library

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Frequency | Description | Location |
| time\_t | 1 | Set reference time.  Used to seed RNG to time of execution. | 20 |

<cstdlib> Library

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Frequency | Description | Location |
| rand  srand | 6  1 | Generate random number.  Seed RNG. | 496-500,  23 |

<fstream> Library

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Frequency | Description | Location |
| fstream | 3 | Import data text. | 515, 533, 552. |

**\*\*\*\*\*CONTINUED ON NEXT PAGE\*\*\*\*\***

**Data Types**

|  |  |  |
| --- | --- | --- |
| Name | Frequency | Location |
| int  unsigned int  string  long  bool  const int | 37  6  11  5  2  1 | 11, 17, 25, 32, 77, 79, 81, 97, 106, 116, 117, 119, 127, 132, 133, 138, 142, 147, 159, 175, 192, 429, 431, 432, 433, 443, 445, 469, 494, 496-500, 502, 507, 509  117, 119, 127, 159, 175, 433  25, 29, 44, 48, 58, 108, 141, 469, 520, 538, 557  11, 12, 22, 494, 507  115, 126  132 |

**Conditional Statements**

|  |  |  |
| --- | --- | --- |
| Conditional Statement | Frequency | Starting Location |
| if  if/else  string  long  bool  const int | 37  6  11  5  2  1 | 11, 17, 25, 32, 77, 79, 81, 97, 106, 116, 117, 119, 127, 132, 133, 138, 142, 147, 159, 175, 192, 429, 431, 432, 433, 443, 445, 469, 494, 496-500, 502, 507, 509  117, 119, 127, 159, 175, 433  25, 29, 44, 48, 58, 108, 141, 469, 520, 538, 557  11, 12, 22, 494, 507  115, 126  132 |