Project 1

Yahtzee

CIS 17C
Juan Castellon
10/30/18

Introduction

Title: Yahtzee

Yahtzee is point-based game played with 6 six-sided dice. The game is played by the players initially rolling the dice to see who goes first based on the highest rolling player. Players then take turns rolling the dice to see who can get the highest rolls and then choosing how to score themselves based on the rolls.

The point system is as follows:

Score 1s: Add up all 1s and add them to your total points.
Score 2s: Add up all 2s and add them to your total points.
Score 3s: Add up all 3s and add them to your total points.
Score 4s: Add up all 4s and add them to your total points.
Score 5s: Add up all 5s and add them to your total points.
Score 6s: Add up all 6s and add them to your total points.
Score 3 of a Kind: Multiply 3 of the same roll together.
Score 4 of a Kind: Multiply 4 of the same roll together.
Score Full House: If 3 dice are of one value and 2 are of another value, gain 25 points.
Score Short Straight: Gain 30 points if you roll 3 dice sequentially.
Score Long Straight: Gain 40 points if you roll 4 dice sequentially.
Score Yahtzee: Gain 100 points if all dice are same value. 50 points for each additional Yahtzee.
Score Chance: Add up all dice together.

The game lasts 10 turns and the player with the highest number of points is the winner.

Summary:

Project Size: ~770 lines of code

I did Yahtzee as my project because it's been an extension of my previous versions of Yahtzee ever since CSC-5. This time around, I went and implemented some concepts from the STL library which shrunk my line count considerably, around 200 lines or something from the last iteration. The containers I used were stacks,maps, STL arrays, and vectors. Of those containers, I used iterators to sort my vectors and used a swap function to sort my vectors in another fashion. The (string) stacks were used to announced the winners, the maps were used to order the players by their initial rolls (highest to lowest), the STL array was used to keep track of the dice, and the vector was used to hold player class objects depending on how many players were playing the game.

Inclusion of STL:

As stated above, I implemented various elements of the STL into my game, and I will list where they were implemented in my program here :

Vector
Lines: 22-24, 40, 98.
Purpose: I used vectors, to keep track of my objects, namely the Dice object and Player objects.

STL Array
Line: 188.
Purpose: Used an STL array to copy values from the Dice object and sort them.

Map
Lines: 22, 36, 137.
Purpose: I used maps to organize players into descending order. Their names were the keys and the associated data was their initial roll.

Stack
Line: 576.
Purpose: I used a stack to read off the names of the winners in descending order.

Iterators
Line: 460, 477, 512, 535.

Purpose: Iterators associated with the STL array, used to sort through the contents of the STL Array which has dice values. Sorted for scoring.

Algorithms (Sort and Swap)

Line for Swap: 170.

Line for Sort: 460, 477, 512, 535.

Purposes: The swap was used in conjunction with 2 for loops to sort through the vector of players based on their initial rolls to see who goes first. The sorts were used in conjunction with iterators and the STL array to sort the values of the dice rolls to score them.

Pseudo Code:

--main.cpp--

--Function Prototypes--

Prototypes for starting the game:
First player function for seeing which player goes first

Essentials for playing the game:
Turn function for playing a turn

Ending the game:
Winner function for outputting the final scores and placements

--Function Prototypes End--

--Enter Main--

Set random number seed

--Declare Variables--

Map of types string and int
Vector of type Player

Game related variables:
Decision to play the game or not
Game lasts 10 turns, starts at 1
Amount of players

Member Function Variable :
Name, score, yahtzee counter

--End of Variable Declaring--

Initialize some variables that need to be initialized:

Introduction to game

Input decision to play or not
Input validation regarded decision

Exit if no, continue if yes

Use cin.ignore to skip to next line

Input user names

Roll to determine who goes first

Determine which person is player 1,player 2,etc

Turn do-while loop

Organize vector of objects for the first person to be on top

Output winner's stats

Exit main

--First Player Function--
Declare some variables for totals
Roll for all players

Add up their roles to their respective index variables within the structure
Sort the player structures based on the one with the highest amount of points
--End of First Player Function--

--Turn Function--
Declare and initialize some variables
Ask the user if they want to reroll
If they answer yes, while loop until conditions are not met
Output categories for scoring
Prompt user to choose
Validate input
Use scoring function
Output score and return it
Switch statement for all of the scoring possibilities:
Score 1s: Add up all 1s and add them to your total points.
Score 2s: Add up all 2s and add them to your total points.
Score 3s: Add up all 3s and add them to your total points.
Score 4s: Add up all 4s and add them to your total points.
Score 5s: Add up all 5s and add them to your total points.
Score 6s: Add up all 6s and add them to your total points.
Score 3 of a Kind: Multiply 3 of the same roll together.
Score 4 of a Kind: Multiply 4 of the same roll together.
Score Full House: If 3 dice are of one value and 2 are of another value, gain 25 points.
Score Short Straight: Gain 30 points if you roll 3 dice sequentially.
Score Long Straight: Gain 40 points if you roll 4 dice sequentially.
Score Yahtzee: Gain 100 points if all dice are same value. 50 points for each additional Yahtzee.
Score Chance: Add up all dice together.
Output score and return it
--End of Turn Function--

--End of main.cpp--

--Player.h--

Public :

Declare default constructor
Declare Constructor
Declare Destructor

Declare Setter Functions
Declare Getter Functions

Increment Yahtzee Function
Increment Chance Function
Increment Turn

Overloaded ++ Operator Function
Overloaded -- Operator Function

Private :
Declare a bunch of game variables (name, score, rolls, etc)

--End of Player.h--

--Player.cpp--

Constructor Function

Setter Functions
Getter Functions

Game Functions (AKA Increment Functions)

Operator Overload Functions for ++ and --

--End of Player.cpp--

--Dice.h--
Public :
Default Constructor Function

Setter Functions (for rerolls primarily)

Getter Functions (for displaying these rolls)

Game Functions :
Roll, Display, Sum, Sort

Private :
Dice 1-5 as ints

--End of Dice.h--

--Dice.cpp--

Default Constructor Function (sets all dice to 0)

Setter Functions :
One setter functions for each die, sets them equal to a random value from 1-6)

Getter Functions

Game Functions :
Roll Function - sets the all the dice equal to a random number from 1-6
Display Function - displays the rolls
Sum Function - Sums all the dice

--End of Dice.cpp--