**Project 2**

**<Yahtzee>**

**CIS-17A**

**Name: Choy, Arthur**

**Date: 12/18/20**

**Introduction**

Title: Yahtzee

Yahtzee is a game played with five dice and a scorecard for as many players that are playing. It is usually played between 2 players, however more can play in a singular game. When the game begins, one player will roll the five dice. They can reroll the dice two extra times, being allowed to choose dice to not roll. After the rolling phase, the rolling player can put their dice for that turn into one of thirteen categories.

The thirteen categories are split into two sections, upper and lower categories. In the upper categories, there are six categories: Ones, Twos, Threes, Fours, Fives, and Sixes. In the lower categories there are Three-of-a-Kinds, Four-of-a-Kinds, Full Houses, Small Straights, Large Straights, Yahtzees, and Chances. Each category will grant a certain amount of points depending on the dice you have. Once a category has been filled, it cannot be used again.

For the upper categories, the scoring is dependent on the amount of dice you have of the same value as the category. The amount of dice you have for that category will add their values to your score. For example, you have 3 dice with four dots; choosing the “Fours” category will give you 12 points; 4 points for every dice you have with four dots. If you had 4 dice with one dot, choosing the “Ones” category will give you 4 points; 1 point for every dice you have with one dot.

For the lower categories, scoring is unique for each category.

Three-of-a-Kinds require you have three dice with the same number; and if so, all numbers on all dice you have at the time will be added to your score (for example, 1-4-5-5-5 will give you 20 points).

Four-of-a-Kinds are similar to Three-of-a-Kinds, except for the fact that you need four dice of the same number in order to add to your score.

Full Houses require you have a Three-of-a-Kind, with the two remaining dice having the same number (for example, 4-4-5-5-5 is eligible for a Full House, but 1-4-5-5-5 or 4-4-5-5-6 is not); if so, you will get 25 points.

Small Straights require you have 4 dice with their numbers in sequence (for example, 1-3-4-5-6 is eligible for a small straight, as 3-4-5-6 are all in a row); if so, you will get 30 points.

Large straights require toy have all 5 dice with their numbers in sequence (for example, 1-2-3-4-5 is eligible for a large straight); if so, you will get 40 points.

Yahtzees require you have a Five-of-a-Kind (all dice have the same number); if so, you will get 50 points and be eligible for Yahtzee bonuses (will be covered later).

Chances have no requirements to grant points, the score for this category is the sum of all the numbers on all your dice.

One thing to note about the lower categories is that if your dice do not fill the requirements for the category, it will grant zero points. Filling in zero points for Yahtzee will also remove the ability to get Yahtzee bonuses.

Yahtzee bonuses: if the Yahtzee category is filled, any other Yahtzees afterwards can be placed on any upper or lower category, granting the normal amount of points (does not have to fulfill the requirement) with an 100 points bonus.

Upper Category bonus: Scoring a total of 63 points or more in the Upper Categories will give 35 points.

After every player has all categories filled, the winner is determined by which player has the most points

**Summary**

Project size: 1000 lines

The number of variables: around or over 25

The number of classes: 4

The number of functions: 18

The biggest struggle in programming this project was the classes, obviously. Because I had so many interlacing classes, it was difficult to keep track of them sometimes, particularly in terms of what is and isn’t available at any given moment while programming object member functions.

The finalized program took around 3-4 days.

I remember one issue in particular I ran into was that most of the arrays in my classes used to be dynamically created and destroyed; however that brought up issues in the fact that most of those arrays were made of other objects. This caused my program to fail at certain points, and it was incredibly difficult to understand why, as it compiled successfully and wouldn’t give me any errors beforehand. Eventually, after a bit of research on the internet (thank you, StackOverflow), I found that my issue came in the fact that I was attempting to access data that didn’t exist via pointers. Eventually, I found that I could simply statically create most of the arrays, as they did not have any variable size. The only array of objects I had was my player array, but that was significantly easier to work with as I didn’t have other layers of object arrays.

**Description**

The main point of this program is to utilize as many of the concepts form Chapter 13 through 16 as possible.

**Pseudo Code**

***Classes***

*Dice: holds all the dice and calculates the scores the dice would give for each category*

*Category: contains the category score value and whether or not it was filled*

*Player: holds the player’s name, all 13 categories, whether or not the player got an upper category bonus, and the player’s total score*

*Game: Internalize all game processes into a class, including players, turns, rounds, dice rolling and score calculations*

***Functions***

*dspMenu(): display the start menu*

*gtPlyrs(): get the amount of and names of players*

*start(): driver function for the game class*

*Declare and initialize variables and pointers*

*Call gtPlyrs() function*

***Call initial() function***

*Initialize all players; set all categories to unfilled state*

*Bonus scores have not been gotten*

*Total score is set to zero*

*Have one turn for every player; after all player turns have passed, one round has passed; there are 13 rounds, one for each category*

***Start the turn of a player***

*Roll 5 dice*

*Repeat up to two times*

*If player wants to stop rolling, have them select all dice*

*Else, keep rolling*

*Have user select which category they want to fill*

*Sort the dice array*

*Call freq() function*

*Call seq() function*

*Categories:*

*Ones-Sixes: Find all dice of matching value to category, add the amounts of those dice to total score (ie. Having 3 fives dice and selecting the fives category will gain you 15 points)*

*Three-of-a-Kind: If there are three dice of the same value in the dice array, total all dice values and add to score*

*Four-of-a-Kind: If there are four dice of the same value in the dice array, total all dice values and add to score*

*Full House: If there is a three-of-a-kind and the remaining two dice are of the same value, add 25 points to score*

*Small Straight: If there are four dice values in a row (ie. 1-2-3-4; 3-4-5-6), add 30 points to score*

*Large Straight: If there are five dice values in a row, add 40 points to score*

*Yahtzee: If there is a five-of-a-kind, add 50 points to score, and allows the player to get Yahtzee bonuses; if the player gets another five-of-a-kind, they will gain a 100 point bonus to the category of their choosing; however, if this category is filled without a five-of-a-kind, the Yahtzee bonus is not available for the player*

*Chance: Add all dice values currently present in array, no requirements needed*

*After 13 rounds have passed, calculate the scores, determine the winner and display*

**Major Variables**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Variable Name** | **Description** | **Location** |
| Integer | amount | Amount of players in a game | gtPlyrs() |
|  | rlCount | Roll count during turn | Game.h |
|  | rCount | Round count | Game.h |
|  | tCount | Turn count | Game.h |
|  | pCount | Amount of players in a game | Game.h |
|  | diceArr | The dice array | Dice.h |
| Player | \*plyrs | Pointer that will hold dynamically allocated player array | Game.h |
| Game | \*g | Pointer that held dynamically constructed game | main() |
| bool | keep | Array for which dice to keep | Dice.h |

**Checkoff Sheet**

|  |  |  |
| --- | --- | --- |
| **Chapter** | **Topic** | **Location (Line #)** |
| 13 | Instance of a Class | main.cpp (149) |
|  | Private Data Members | Game.h (14) |
|  | Specification vs. Implementation | Game.cpp |
|  | Inline | Game.h (42) |
|  | Constructors | Game.cpp (17) |
|  | Destructors | Game.cpp (53) |
|  | Arrays of Objects | Game.cpp (31) |
|  | UML | Documentation .html’s |
| 14 | Static | Game.h (14) |
|  | Copy Constructors | Game.cpp (43) |
|  | Operator Overloading | Player.h (56) |
|  | Aggregation | Player.h (21) |
| 15 | Protected Members | Dice.h (15) |
|  | Base Class to Derived | Player.h (14) |
|  | Polymorphic Associations | Player.h (49) |
| 16 | Exceptions | Game.cpp (147) |
|  | Templates | main.cpp (210) |
|  | STL | Dice.cpp (54) |

**References**

1. textbook
2. powerpoint presentations
3. lectures
4. Various online coding forums

**Program**

**main.cpp**

/\*

\* File: main.cpp

\* Author: Arthur Choy

\* Created on December 17, 2020, 6:52 PM

\* Purpose: Create a board/card game that utilizes many of the concepts learned in class

\* Changelog: -Got program functional

\* -Added static member sin the form of constant integers

\* -Removed unnecessary dynamically allocated arrays

\* -Added scorecard display

\* -Some of the overloaded operators had to be removed as they didn't

\* work properly (ie. reset some of the members in an object)

\*/

//System Libraries

#include <cstdlib>

#include <iostream>

#include <cstring>

#include <string>

#include <iomanip>

#include "Game.h"

#include "Dice.h"

using namespace std;

//Global Constants

bool qToMenu; //Whether or not the user is mid-game

//Function Prototypes

void dspMenu(); //Display menu

Game \*gtPlyrs(int &); //Get the amount and names of players

void start(Game \*); //Start the game

template <class T>

T grandTotal(T, const Game \*); //Total score of all players in game

template <class T>

T mean(T, const Game \*); //Average score between all players

//Execution Begins Here

int main(int argc, char\*\* argv) {

int input; //Input character

int amount; //Amount of players in game

Game \*g = nullptr; //Game pointer

qToMenu = false; //Initialize quit to menu

string msg = "That input is not valid";

//Main menu

do

{

dspMenu();

cout << "Please enter the number corresponding to the option you would like: ";

try

{

if(!(cin >> input))

{

cin.clear();

cin.ignore();

throw msg;

}

try

{

switch(input)

{

case 1: //Start a new game

if(qToMenu) //If they were in the middle of a game, ask them if they want to overwrite it

{

cout << "This will overwrite the current game, " <<

"are you sure you want to continue? (Y/N): ";

char confirm;

cin >> confirm;

if(confirm == 'N' || confirm == 'n') break;

}

//Get players and initialize

g = gtPlyrs(amount);

qToMenu = false; //Signify user started a new game

//Begin the game

start(g); //Start the game

break;

case 2: //Continue game

if(qToMenu) //Only allow option if user was in the middle of the game

{

qToMenu = false;

start(g);

}

//Else, tell them they are not in the middle of a game

else cout << "There is currently no game loaded into the program." << endl;

break;

case 3: //Quit the program

cout << "Quitting Program..." << endl;

break;

default: //Inform the user of an invalid input

cout << "That is not a valid input" << endl;

break;

}

}

catch (Game::qToMenu) //Catch whether or not player quit to menu

{

cout << "Quitting to menu..." << endl << endl;

qToMenu = true;

}

}

catch (string msg)

{

cout << "That is not a valid input" << endl << endl;

}

}while(input != 3); //Keep asking the user for a main menu prompt while they don't want the game to close

//Delete pointer

delete g;

//Exit Program

return 0;

}

//Function to display the menu

void dspMenu()

{

//Display menu options

cout << "1. Start a new game" << endl <<

"2. Continue the current game" << endl <<

"3. Quit Program" << endl;

//Return to main

return;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Game functions \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Get the names of the players in a game

Game \*gtPlyrs(int &amount)

{

bool badInput;

//Prompt for amount of players

do

{

badInput = false;

cout << "Please enter the amount of people that will be playing: ";

if(!(cin >> amount))

{

cin.clear();

cin.ignore();

cout << "That is not a valid input" << endl << endl;

badInput = true;

}

if(amount < 1 && !badInput) cout << "That is not a valid amount of players" << endl;

}while(amount < 1);

//Dynamically create a new game

Game \*g = new Game(amount);

cin.ignore(); //Clear input

//Ask for names of players

for(int i = 0; i < amount; i++)

{

cout << "Please enter the name for player " << i + 1 << ": ";

string name;

getline(cin, name);

g->plyrs[i].stName(name);

}

//Return game to main function

return g;

}

//Start a game

void start(Game \*g)

{

//Try code so exceptions can be thrown

try

{

//Continue until error is throw

while(true)

{

//Print current player's name

cout << endl;

cout << g->plyrs[g->gtTCount()].gtName() << "'s Turn" << endl;

//Start player's turn

cout << endl;

g->turn(g->plyrs[g->gtTCount()]);

g->nextT(); //Increment to next turn

}

}

catch (Game::end) //Catch whether or not the game has ended

{

//Determine the winner

g->detWin();

//Determine game statistics

int num;

num = grandTotal(num, g);

cout << "====== Game Statistics ======" << endl;

cout << "Total of all players: " << num << endl;

num = mean(num, g);

cout << "Average Score: " << num << endl;

cout << endl;

}

//Return from function

return;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Game Statistic functions \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Template total function

template <class T>

T grandTotal(T num, const Game \*g)

{

num = 0; //Initialize number to 0

//Add all players score totals to number

for(int i = 0; i < g->gtPCount(); i++)

{

num += g->plyrs[i].gtTotal();

}

//Return number

return num;

}

//Template average function

template <class T>

T mean(T num, const Game \*g)

{

//Divide overall total by player count and return

return num/g->gtPCount();

}

**Game.h**

/\*

\* File: Game.h

\* Author: Arthur Choy

\* Created on December 14, 2020, 12:40 AM

\*/

#ifndef GAME\_H

#define GAME\_H

#include "Player.h"

//Game class

class Game {

private:

static const int DICE\_SIZE = 5; //Amount of time

int rlCount; //Roll count (max. 3 per player)

int tCount; //Turn count (max: amount of players)

int rCount; //Round count (max 13)

int pCount; //Player count

bool quit; //Whether or not player quit to menu

public:

Player \*plyrs; //Players pointer

//Default Constructor

Game();

//Constructor

Game(int);

//Copy Constructor

Game(const Game &obj);

//Destructor

virtual ~Game();

//Exception Classes

class invalidOption { }; //Exception class for when the player inputs an invalid value

class qToMenu { }; //Exception class for when the player quits to the menu

class end { }; //Exception class for when the game ends

//Mutator functions

void stRLCount(int rl)

{ rlCount = rl; }

void stTCount(int t)

{ tCount = t; }

void stRCount(int r)

{ rCount = r; }

void stPCount(int p)

{ pCount = p; }

//Accessor functions

int gtRLCount() const

{ return rlCount; }

int gtTCount() const

{ return tCount; }

int gtRCount() const

{ return rCount; }

int gtPCount() const

{ return pCount; }

bool getQtoM() const

{ return quit; }

//Functions

bool reroll(Player &); //Rerolls dice array

void turn(Player &); //A single turn in the game

void nextT(); //Move to next turn

void dspCard(Player); //Display player's scorecard

void detWin(); //Determine which player won the game

};

#endif /\* GAME\_H \*/

**Game.cpp**

/\*

\* File: Game.cpp

\* Author: Arthur Choy

\* Created on December 14, 2020, 12:40 AM

\*/

#include "Game.h"

#include <iostream>

#include <cstdlib>

using namespace std;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Constructors/Desctructors \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Default Constructor

Game::Game() {

plyrs = new Player[0]; //No players in array

//Initialize game variables to 0

tCount = 0;

rCount = 0;

pCount = 0;

//Player had not quit to menu yet

quit = false;

}

//Constructor

Game::Game(int p) {

plyrs = new Player[p]; //Create dynamic array depending on amount of players

//Initialize game variables to 0

tCount = 0;

rCount = 0;

pCount = p;

//Player had not quit to menu yet

quit = false;

}

//Copy Constructor

Game::Game(const Game& obj) {

tCount = obj.tCount;

rCount = obj.rCount;

pCount = obj.pCount;

quit = obj.quit;

plyrs = new Player[obj.pCount];

for(int i = 0; i < obj.pCount; i++) plyrs[i] = obj.plyrs[i];

}

//Destructor

Game::~Game() {

delete []plyrs;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Game Functions \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Reroll function; reroll dice depending on which \*

// the dice the player didn't want to keep \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool Game::reroll(Player &p) {

bool dont = true; //Whether or not to reroll again

bool \*keep = p.keep(); //Copy keep dice boolean array

//If player doesn't want to keep a dice, then reroll

for(int i = 0; i < DICE\_SIZE; i++) if(keep[i] == false) dont = false;

//Return whether or not to reroll

return dont;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Singular game turn (one per player in round); player \*

// rolls three times and sets their dice to a category. \*

// Also allows for showing of scorecard and quitting to \*

// menu. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Game::turn(Player &p) {

//Options menu constant/array

const int SIZE = 16; //Size of options array

std::string options[SIZE] = { "Ones", "Twos", "Threes", "Fours", "Fives", "Sixes", //String array holding all different options after dice rolling

"Three of a Kind", "Four of a Kind", "Full House", "Small Straight",

"Large Straight", "Yahtzee", "Chance", "Show Scorecard", "See All Players' Cards",

"Exit to Menu" };

//Holding variable for player input

int option;

//Dice copy array

int \*copyDice = new int[DICE\_SIZE];

//If the game hadn't been paused

if(!quit)

{

//Player had not rolled yet

rlCount = 0;

//Roll once

p.roll(); //Roll dice

p.rstKeep(); //Reset dice to keep

rlCount++; //Player rolled once

bool reroll = false; //Initialize whether or not to reroll

while(rlCount < 3 && !reroll) { //No more than 3 rolls, 2 rerolls

//Copy the dice the player has to print

for(int i = 0; i < DICE\_SIZE; i++) copyDice[i] = p.gtDice(i);

cout << "Dice: ";

for(int i = 0; i < 5; i++) cout << copyDice[i] << " ";

cout << endl;

//Start reroll function

reroll = this->reroll(p);

p.roll();

p.rstKeep();

rlCount++; //Increase roll count

}

}

quit = false; //Reset whether or not the game was paused

bool proceed = false; //Whether or not to end the turn

while(!proceed) {

//Redisplay dice for convenience

cout << "Dice: ";

for(int i = 0; i < 5; i++) cout << copyDice[i] << " ";

cout << endl << endl;

//Present options

for(int i = 0; i < SIZE; i++) std::cout << i+1 << ". " << options[i] << std::endl;

//Prompt for option

cout << "Please enter the option you would like to choose: ";

//Get option, if player inputs something that isn't an integer, an option,

//or an unfilled category, throw exception

try

{

if(!(cin >> option))

{

cin.clear();

cin.ignore();

throw invalidOption();

}

if(option > 16 || option < 1 || p.card[option-1].gtFill()) throw invalidOption();

proceed = true;

}

catch (Game::invalidOption) //Exception catch for invalid options

{

proceed = false;

cout << "That is not a valid option." << endl << endl;

}

if(option = 15)

{

cout << "Which player would you like to see the scorecard of?" << endl;

for(int i = 0; i < this->pCount; i++)

{

cout << i+1 << ". " << this->plyrs[i].gtName() << endl;

}

int input;

try

{

if(!(cin >> input))

{

cin.clear();

cin.ignore();

throw invalidOption;

}

else if(input < 1 || input+1 > this->pCount) throw invalidOption();

dspCard(this->plyrs[input-1]);

}

catch (Game::invalidOption)

{

cout << "That is not a valid option" << endl << endl;

}

proceed = false;

}

//If option is to display the player's scorecard, display the scorecard

//and set to repeat options menu

if(option == 14)

{

dspCard(p);

proceed = false;

}

}

//Delete copy

delete []copyDice;

//If option involves a score, set it

if(option <= 13)

{

p.stCat(option-1);

if(option > 6 && option < 13 && p[13] == 50 && p.freq() == 5) p.addTotal(100);

}

//If option is to quit to the menu, throw exception

else if(option == 16)

{

quit = true; //Set quit to menu flag to true

throw qToMenu();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Move on to next turn, move on to next \*

// round if all turns in round are over. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Game::nextT() {

//Increase turn count

tCount++;

//If all turns have passed in the last round, throw an end of game exception

if(tCount >= pCount && rCount >= 12) {

throw end();

}

//If all player turns have passed in a round, move to next round

if(tCount >= pCount) {

rCount++;

tCount = 0;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Display player's scorecard, along with progress \*

// to upper category bonus and score total. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Game::dspCard(Player p) {

//String array holding name of all categories

string cats[13] = { "Ones", "Twos", "Threes", "Fours",

"Fives", "Sixes", "3-of-a-Kind", "4-of-a-Kind",

"Full House", "Small Straight", "Large Straight",

"Yahtzee", "Chance" };

cout << endl;

cout << p.gtName() << "'s Scorecard" << endl;

//Display scores of upper category

cout << "====== Upper Categories ======" << endl;

for(int i = 0; i < 6; i++) cout << cats[i] << ": " << p[i] << endl;

cout << endl;

//Calculate player's upper category bonus

int upper = 0;

for(int i = 0; i < 6; i++) upper += p[i];

if(upper > 63) upper = 63;

cout << "Upper Category Bonus: " << upper << "/63" << endl << endl;

//Display scores of lower category

cout << "====== Lower Categories ======" << endl;

for(int i = 6; i < 13; i++) cout << cats[i] << ": " << p[i] << endl;

cout << endl;

//Display total

cout << "Total Score: " << p.gtTotal() << endl << endl;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Determine which player won the game \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Game::detWin() {

int winner = 0; //Initialize which player won the game

//Compare total scores

for(int i = 0; i < pCount-1; i++) {

if(plyrs[i] > plyrs[winner]) winner = i;

}

//Display player that won game

cout << endl;

cout << plyrs[winner].gtName() << " wins the game!" << endl;

cout << endl;

}

**Player.h**

/\*

\* File: Player.h

\* Author: Arthur Choy

\* Created on December 13, 2020, 2:32 AM

\*/

#ifndef PLAYER\_H

#define PLAYER\_H

#include "Dice.h"

#include "Category.h"

#include <string>

//Player class, inherits dice class

class Player : public Dice {

private:

std::string name; //Player name

bool uBonus; //Upper cate. bonus flag

protected:

int total; //Total Score

public:

Category card[SCORES\_SIZE]; //Scorecard

//Default Constructor

Player();

//Constructor

Player(std::string);

//Copy Constructor

Player(const Player &obj);

//Destructor

virtual ~Player();

//Mutator functions

void stName(std::string n)

{ name = n; }

void stCat(int);

void stTotal(int t)

{ total = t; }

void addTotal(int a)

{ total += a; }

void subTotal(int s)

{ total -= s; }

//Accessor functions

std::string gtName() const

{ return name; }

virtual int gtScore(const int i) const

{ return card[i].gtScore(); }

int gtTotal() const

{ return total; }

//Overloaded operator

//Compares total scores

bool operator >(const Player &);

bool operator <(const Player &);

//Get category specified in sub, as if accessing scorecard as an array

int operator [](const int &);

};

#endif /\* PLAYER\_H \*/

**Player.cpp**

/\*

\* File: Player.cpp

\* Author: Arthur Choy

\* Created on December 13, 2020, 2:32 AM

\*/

#include "Player.h"

#include <iostream>

#include <cstdlib>

using namespace std;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Constructors/Destructors \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Default Constructor

Player::Player() {

name = " ";

total = 0;

uBonus = false;

for(int i = 0; i < SCORES\_SIZE; i++) card[i].stScore(0);

}

//Constructor

Player::Player(std::string n) {

name = n;

total = 0;

uBonus = false;

for(int i = 0; i < SCORES\_SIZE; i++) card[i].stScore(0);

}

//Copy Constructor

Player::Player(const Player& obj) {

name = obj.name;

uBonus = obj.uBonus;

for(int i = 0; i < SCORES\_SIZE; i++) card[i] = obj.card[i];

}

//Destructor

Player::~Player() {

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Set score player wants to set, see if they earn upper \*

// category bonus \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Player::stCat(int i) {

//Set score of wanted category, set category as filled

card[i].stScore(scores[i]);

card[i].fill();

total += scores[i]; //Add score to total

//Determine whether or not player earns upper category bonus

int upper = 0;

for(int i = 0; i < 6; i++) upper += card[i].gtScore();

if(upper >= 63 && !uBonus) {

uBonus = true;

total += 35;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Overloaded Operators \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// < Operator; sees if right element is greater than the left \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool Player::operator <(const Player &right) {

bool status = false;

if(total < right.total)

status = true;

return status;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// > Operator; sees if left element is greater than the right \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool Player::operator >(const Player &right) {

bool status = false;

if(total > right.total)

status = true;

return status;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// [] Operator; access player scorecard as if player was \*

// score array \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Player::operator [](const int &sub) {

return card[sub].gtScore();

}

**Dice.h**

/\*

\* File: DICE.h

\* Author: Arthur Choy

\* Created on December 13, 2020, 1:06 AM

\*/

#ifndef NEWFILE\_H

#define NEWFILE\_H

class Dice {

private:

static const int DICE\_SIZE = 5; //Dice and score array sizes

int diceArr[DICE\_SIZE]; //Dice array

protected:

static const int SCORES\_SIZE = 13; //Size of arrays involving categories

int scores[SCORES\_SIZE]; //Score array

bool keepArr[DICE\_SIZE]; //Keep dice

public:

//Default Constructor

Dice();

//Destructor

virtual ~Dice();

//Accessor functions

int gtDice(int i) const

{ return diceArr[i]; }

virtual int gtScore(int i) const

{ return scores[i]; }

//Functions

void roll(); //Roll all die

void rstKeep(); //Reset which dice are kept

void detScre(); //Determine scores available from current dice

void rstScre(); //Reset scores available from dice

int rollDie(int); //Roll one die

int freq(); //Find dice mode for determining scores

int seq(); //Find longest sequence for determining scores

bool \*keep(); //Determine which dice to keep

};

#endif /\* NEWFILE\_H \*/

**Dice.cpp**

/\*

\* File: Dice.cpp

\* Author: Arthur Choy

\* Created on December 13, 2020, 1:20 AM

\*/

#include "Dice.h"

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <algorithm>

using namespace std;

//Default Constructor

Dice::Dice() {

rstScre();

rstKeep();

srand(static\_cast<unsigned int>(time(0)));

for(int i = 0; i < DICE\_SIZE; i++) keepArr[i] = false;

}

//Destructor

Dice::~Dice() {

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Dice functions \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Roll all die depending on which the player doesn't want to keep \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Dice::roll() {

for(int i = 0; i < DICE\_SIZE; i++)

if(!keepArr[i])

diceArr[i] = rollDie(6);

detScre();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Reset which dice to keep; set all elements to false \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Dice::rstKeep() {

for(int i = 0; i < DICE\_SIZE; i++)

keepArr[i] = false;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Determine scores from dice available; store in potential score array \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Dice::detScre() {

rstScre(); //Reset scores in array to be 0

//Score determining functions

sort(diceArr, diceArr + DICE\_SIZE);

int freq = this->freq(); //Highest mode in dice array

int seq = this->seq(); //Longest sequence in dice array

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Upper Categories \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

for(int i = 0; i < 6; i++) //Ones thru Sixes

{

for(int j = 0; j < DICE\_SIZE; j++)

if(diceArr[j] == i+1)

scores[i] += diceArr[j];

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Lower Categories \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if(freq >= 3 || freq == 5) //Three-of-a-Kind

{

for(int i = 0; i < DICE\_SIZE; i++) scores[6] += diceArr[i];

}

if(freq >= 4 || freq == 5) //Four-of-a-Kind

{

for(int i = 0; i < DICE\_SIZE; i++) scores[7] += diceArr[i];

}

if(freq == 3 || freq == 5) //Full House

{

if(diceArr[0] == diceArr[1] || diceArr[3] == diceArr[4] || freq == 5)

{

scores[8] = 25;

}

}

if(seq >= 4 || freq == 5) //Small Straight

{

scores[9] = 30;

}

if(seq == 5 || freq == 5) //Large Straight

{

scores[10] = 40;

}

if(freq == 5) //Yahtzee

{

scores[11] = 50;

}

for(int i = 0; i < DICE\_SIZE; i++) scores[12] += diceArr[i]; //Chance

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Reset calculated scores \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Dice::rstScre() {

for(int i = 0; i < SCORES\_SIZE; i++)

scores[i] = 0;

}

// Roll a dice

int Dice::rollDie(int sides) {

int dice = rand() % sides + 1; //Choose a number between 1 and the number of sides

return dice; //Return the dice value

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Determine which dice to keep; ask player for each individual dice \*

// whether or not they want to keep it \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool \*Dice::keep() {

cout << "Please enter Y/N depending whether or not you want to keep a dice" << endl;

for(int i = 0; i < DICE\_SIZE; i++) {

char input;

cout << "Dice " << i+1 << ": ";

cin >> input;

if(input == 'Y' || input == 'y') keepArr[i] = true;

}

return keepArr;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Score Determining Functions \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Determine the highest dice frequency in array \*

// (ie. 1, 1, 1, 1, 1 = freq of 5; \*

// ie. 1, 1, 1, 2, 3 = freq of 3) \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Dice::freq() {

int maxFreq = 0; //Variable holding maximum frequency

int pos = 0; //Variable holding position in array

do

{

int freq = 0; //Initial frequency is 0 (for numbers that aren't present)

int nfocus = diceArr[pos]; //Sets focus to current position in array

for(int i = 0; i < DICE\_SIZE; i++) //Goes through entire array, and sees if any values match the focus

{

if(nfocus == diceArr[i])freq++;

}

pos += freq; //Shift the position to next number focus (works on assumption that array has been sorted

if(freq>maxFreq)maxFreq=freq; //Updates or keeps max frequency

}while(pos != DICE\_SIZE); //While the position has not hit the end of the array

//Return max frequency and exit function

return maxFreq;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Determine longest sequence in the array \*

// (ie. 1, 2, 3, 4, 5 = seq of 5; \*

// ie. 1, 2, 3, 5, 6 = seq of 3) \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Dice::seq() {

//Array assumed to be sorted from least to greatest

int x = diceArr[0]; //Declare and initialize check value

int seq = 1, maxSeq = 1; //Maximum sequence and sequence count (minimum of one)

for(int i = 0; i < DICE\_SIZE - 1; i++) //Check the whole array

{

//If the next element in the array follows the sequence, increase the sequence count

//If the next element in the array doesn't follow the sequence, reset the count,

//Unless it is a repeat digit, in which it will not affect the sequence count

if(diceArr[i + 1] == diceArr[i] + 1) seq++;

else if(diceArr[i + 1] == diceArr[i]) seq += 0;

else seq = 1;

x = diceArr[i+1];

if(seq > maxSeq) maxSeq = seq; //Update or keep the maximum sequence count

}

//Return sequence value and exit function

return seq;

}

**Category.h**

/\*

\* File: Category.h

\* Author: Arthur Choy

\* Created on December 13, 2020, 3:20 AM

\*/

#ifndef CATEGORY\_H

#define CATEGORY\_H

class Category {

private:

int score; //Category score

bool filled; //Whether or not the category was filled

public:

//Default Constructor

Category();

//Copy Constructor

Category(const Category &obj);

//Destructor

virtual ~Category();

//Mutator functions

void stScore(int);

void fill();

void unfill();

//Accessor functions

virtual int gtScore() const

{ return score; }

bool gtFill() const

{ return filled; }

};

#endif /\* CATEGORY\_H \*/

**Category.cpp**

/\*

\* File: Category.cpp

\* Author: Arthur Choy

\* Created on December 13, 2020, 3:20 AM

\*/

#include "Category.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Constructors \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Default Constructor

Category::Category() {

score = 0;

filled = false;

}

//Copy Constructor

Category::Category(const Category& obj) {

score = obj.score;

filled = obj.filled;

}

//Destructor

Category::~Category() {

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Accessor Functions \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Category::stScore(int s) {

score = s;

}

void Category::fill() {

filled = true;

}

void Category::unfill() {

filled = false;

}