Project 2 - Yahtzee

Michael Cooper

17A - 48591

Dr. Lehr

December 18, 2021

https://github.com/SouL909/Cooper_Michael_CSC17A_48591/tree/main/Project/Project%202%20-%2017A

Introduction

Title: Yahtzee

The classic table top game which involves rolling up to 3 times with 5 dice. The namesake of which is rolling the same of all 5 dice among various other scoring possibilities.

Game Summary:
On each turn, roll the dice to get the highest scoring combination for one of the 13 categories. Keep any dice you do not not wish to reroll. You can roll up to 3 times, but once you fill a category with a score it cannot be replaced. After you finish rolling your score (or zero) will be logged. The game ends when all 13 boxes are filled by all players, scores are then totaled, including any bonus points and the player with the highest total wins.

The Thirteen Categories:
There are two sections in which to place scores: an upper area and a lower area. The upper section is aces-sixes (63+ scores a 35 bonus).

- For example: Total of aces(ones) only, total of twos only, and so on...

The lower section consists of the following:

- 3 of a kind = total of all 5 dice
- 4 of a kind = total of all 5 dice
- Full house (3 of a kind and a pair) = 25 points
- Small straight (any sequence of 4 numbers) = 30 points
- Large straight (any sequence of 5 numbers) = 40 points
- Yahtzee (five of a kind) = 50 points
- Chance: score the total of any 5 dice, this comes in handy when you can't or don't want to score in another category and don’t want to enter a zero.

Project Summary
Line size: 1011
Number of variables: 30+

When converting this project from cis-5, the addition of structs, pointers and arrays, allowed the main to be extremely simplified. In combination with the functions, each process easier to follow and identify. While initially focusing on making sure the prompts from the program were clear and easy to understand, I also tried to lay out the game just like an arcade version. Four separate players should be able to easily navigate through the 3 possible rolls and log their scores throughout, while having the program automatically detect special rolls such as Three of A Kind, Full House, Straight, etc. The program will be able to loop for another round or save each players data for later use.

Layout:
This version includes the use of 2 header files and 2 source files, so that the main is extremely simplified and the data is passed through the various files. The player information (player.h/.cpp) has been stored in public and private classes and allows that information to be completely separate from the actual turn processes (Yahtzee.h/.cpp).

A Look at the Main:

```
13  int main(int, char**) {
14      Yahtzee yhtz;
15      //Creating an instance of yahtzee class from the .h file
16      yhtz.play();
17      //Starts the game
18      return 0;
19  }
```

The use of header and source files resulted in the main becoming extremely brief. An instance of a class is created from the respective .h file and then passes that data into yhtz.play() to start the actual game processes.

Header Files:

Player.h:

The player files are designed to prompt user input data, and account for each players turn. In addition, roll data is stored and equated to each player's turn and score. To fulfill requirements, the addition of a computer player was implemented and had to be accounted for in addition to each players name and score. Human players are the default and COM players must be manually selected. The player header file also sets the max number of rounds and players.

```
class Player
{
protected:
    std::array<int, 5> rolls;
    std::string name;
public:
    Player();
                // Default Constructor
    Player(std::string name);

    Player(Player& player);     // Copy Constructor

    void setName(std::string name);     // Setters

    void setRoll(int nth, int roll);    // --------

    std::string getName() const;        // Getters

    int getRoll(int nth) const;         // --------

    virtual char get_input() = 0;

    friend std::ostream& operator<<(std::ostream& os, const Player& player);
    int operator[](int index);
    void operator=( const Player& player);
};
```

```
class ComputerPlayer : public Player
{
public:
    ComputerPlayer();

    ComputerPlayer(std::string name);

    ComputerPlayer(ComputerPlayer& player);

    char get_input();
};

// HumanPlayers are the default
// Player when creating a player
class HumanPlayer : public Player
{
public:
    HumanPlayer();

    HumanPlayer(std::string name);

    HumanPlayer(ComputerPlayer& player);

    char get_input();
};
```

Yahtzee.h:

The use of public and private class was implemented to hold public or open information that would be accessible to all players using the actual boardgame, as well as information that accounts for individual player data. Many of the private classes were previous functions implemented that accounted for the state of the game

```
public:
    // Default Constructor
    Yahtzee();
    // Deconstructor
    ~Yahtzee();

    void play();
private:
    void print_rules();

    bool load();

    void setup_players();

    int player_turn(int index);

    void readyup();

    void save_game();

    void print_scores();

    bool two_of_a_kind(int rolCts[], int Size, int &index);
    //turn processes and player info
};

#endif
```

Source Files:

Player.cpp:

The player file keeps track of player status in relation to the game for scoring, including of prompting the user to select which dice to keep and which to preroll. The program must be able to account and store each players info after each round. The source file also accounts for a series of validations for entry mistakes and possible issues with dice rolls

yahtzee.cpp:

This source file is essentially the entire game of Yahtzee as it relates to actual turn processes as if played in an “arcade” style.

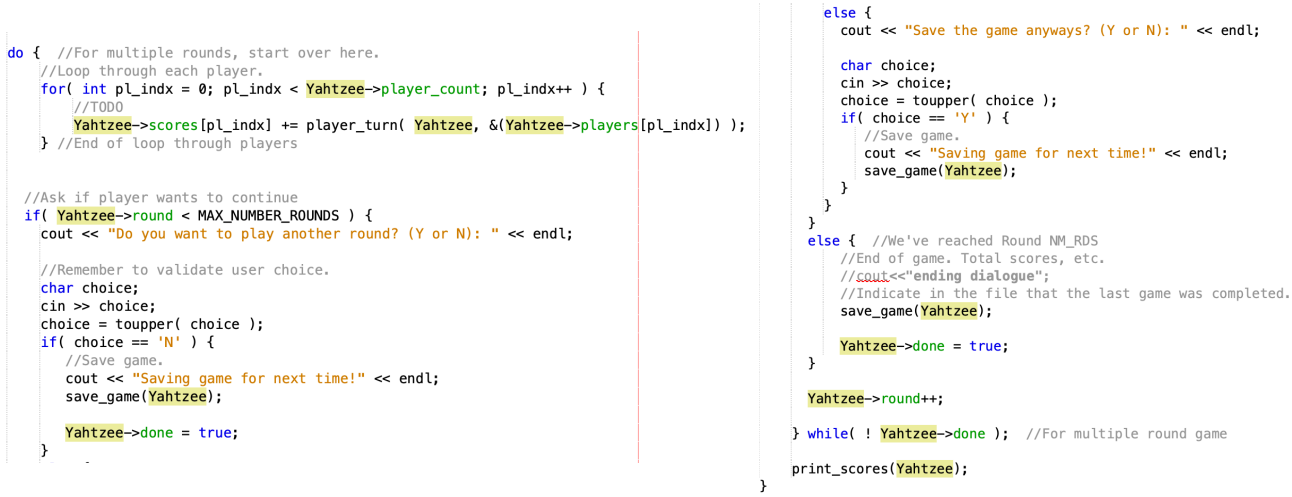
The player is shown the tutorial and then prompted with an option to exit, or start the game. Then, the program checks if there is a save data file ready to load and add to the number of rounds, importing player names and scores as well.

If there is no save data, the program asks and validates how many players there will be and prompts the user to input names. If the player is ready, the game begins.

```
bool load(Yahtzee* Yahtzee) {
    ifstream In( FILE_DIR, ios::binary  );

    //NOTE: Probably should check for file-open errors here.
    if( ! In.is_open() ) return false;  //Can't open file

    In >> Yahtzee->round >> Yahtzee->player_count;

    //Read game data in as vectors.
    vector<string> N;
    vector<int> S;
    for( int i = 0; i < Yahtzee->player_count; i++ ) {
    string name;
    int    score;
    In >> name >> score;
    N.push_back( name );
    S.push_back( score );
    }
    In.close();

    Yahtzee->players = new Player[Yahtzee->player_count];
    for( int i = 0; i < N.size(); i++ ) {
        Yahtzee->players[i]=Player();
    }
    //Copy vectors to arrays.
    for( int i = 0; i < N.size(); i++ ) {
        Yahtzee->players[i].name = N[i];
        Yahtzee->scores[i] = S[i];
    }

    return true;
}
```

This game is essentially a long do while loop that is designed to allow players the chance to immediately begin the next round or exit the program. The loop utilizes functions and arrays by indexing players, executing their respective turns, and then checking if the number of rounds is less than the maximum. They are then asked if they would like to continue, save or if they

have reached the end (13 rounds), the game ends and scores/averages are printed.



```
do {  //For multiple rounds, start over here.
    //Loop through each player.
    for( int pl_indx = 0; pl_indx < Yahtzee->player_count; pl_indx++ ) {
        //TODO
        Yahtzee->scores[pl_indx] += player_turn( Yahtzee, &(Yahtzee->players[pl_indx]) );
    } //End of loop through players


  //Ask if player wants to continue
  if( Yahtzee->round < MAX_NUMBER_ROUNDS ) {
    cout << "Do you want to play another round? (Y or N): " << endl;

    //Remember to validate user choice.
    char choice;
    cin >> choice;
    choice = toupper( choice );
    if( choice == 'N' ) {
       //Save game.
       cout << "Saving game for next time!" << endl;
       save_game(Yahtzee);

       Yahtzee->done = true;
    }
      else {
        cout << "Save the game anyways? (Y or N): " << endl;

        char choice;
        cin >> choice;
        choice = toupper( choice );
        if( choice == 'Y' ) {
          //Save game.
          cout << "Saving game for next time!" << endl;
          save_game(Yahtzee);
        }
      }
    }
    else {  //We've reached Round NM_RDS
        //End of game. Total scores, etc.
        //cout<<"ending dialogue";
        //Indicate in the file that the last game was completed.
        save_game(Yahtzee);

        Yahtzee->done = true;
    }

    Yahtzee->round++;

  } while( ! Yahtzee->done );  //For multiple round game

  print_scores(Yahtzee);
}
```

The load function implements a binary file and inputs the stored data through vectors and then copies the vectors into the appropriate arrays The program uses a series of if statements and arguments to give prompts and outcomes for making sure the players are ready, setting them to the appropriate scores, and then printing the intro and rules before the game starts:

The primary function:

- Int **PlyTurn** (PlayerTurn)

- Essentially the entire process for rolling the dice:

Starts by creating 5 dice (int roll[5]) and keeping tracking of the number of each number rolled with the variable int rolCts[6]. Then booleans are used to keep track of which dice the user would like to keep and if they would like to roll for a second or third time. The name of the player and clear instructions are given for each roll/ turn and a series of for loops keep track of which dice to display by determining which dice to keep and which dice to preroll. The process repeats for 3 rolls and then proceeds to a scoring sequence which utilizes rolCts[6] and a series of if else statements to determine scoring for each turn and round. This function ends with showing the user their name and score and returns the score value.

The function then goes through a lengthy process to determine if any special roll such as three of a kind, straight, or Yahtzee has occurred and record the appropriate score to the correct player.

```
int player_turn(Yahtzee* Yahtzee, Player* player) {
    //The roll of dices for turn 1 of up to 3
    int  rolCts[6];
    bool keep[5];
    //Dice to keep for 1st,2nd, and 3rd rolls.
    bool r2ndT, r3rdT;                      //Rolled 2nd Time? Rolled 3rd time?
    //Will player roll a 2nd or 3rd time?
    char ans;
    string s;
    int scr3k;              //Score with 3 of a Kind

    cout << "\n================================================\n"
         << "It's " << player->name << "'s turn:" << endl;

    //Turn begins //First roll
    cout << "roll 1:\n(Dice are labeled a-e)\n";
    for( int i = 0; i < 5; i++ )
      player->rolls[i] = rand() % 6 + 1;

    //Show dice
    for( int i = 0; i < 5; i++ )
```

```
    //Scoring sequence beings
    //Score UPPER section.
    //Clear out count
    cout << "Scoring..." << endl;
    for( int i = 0; i < 6; i++ )
      rolCts[i] = 0;

    //Look at each die in a roll.
    //Add the number to the appropriate entry in Counts
    for( int r = 0; r < 5; r++ ) {  //Loop through rolls
        int index = player->rolls[r] - 1;
        rolCts[ index ]++;
    }

    //Debugging: printing out roll counts.
    cout << "Roll counts:" << endl;
    for( int i = 0; i < 6; i++ ) //Loop through rolls
        cout << "# of " << (i+1) << "'s: " << rolCts[i] << endl;
    cout << endl;

    int score = 0;

    bool gt_yhtz = false;
    for( int i = 0; i < 6; i++ ) {
        if( rolCts[i] == 5 ) {
          cout << "Yahtzee!" << endl;
          score += 50;
          gt_yhtz = true;
        }
    }
```

```
    //Check for straights
    if( (rolCts[0] == 0 && rolCts[1] == 1 && rolCts[2] == 1 &&
         rolCts[3] == 1 && rolCts[4] == 1 && rolCts[5] == 1) ||
        (rolCts[0] == 1 && rolCts[1] == 1 && rolCts[2] == 1 &&
         rolCts[3] == 1 && rolCts[4] == 1 && rolCts[5] == 0) ) {
      cout << "Large straight." << endl;
      score += 40;
    }
    else {
      if( (rolCts[0] == 1 && rolCts[1] == 1 && rolCts[2] == 1 &&
           rolCts[3] == 1) ||                                          //1,2,3,4
          (rolCts[1] == 1 && rolCts[2] == 1 && rolCts[3] == 1 &&
           rolCts[4] == 1) ||                                          //2,3,4,5
          (rolCts[2] == 1 && rolCts[3] == 1 && rolCts[4] == 1 &&
           rolCts[5] == 1) ) {                                         //3,4,5,6
        cout << "Small straight." << endl;
        score += 30;
      }
    }
  }  //end of else

  if( score == 0 ) {  //Chance
      for( int j = 0; j < 6; j++ )
        score += rolCts[j] * (j+1);
  }

  //Scores[RoundNum-1] = score;
  cout << "Congratulations " << player->name << "!\nYour Score: " << score << endl;
  return score;
```

- The program determines the winner and the average score

```
void print_scores(Yahtzee* Yahtzee) {
    SortScr( Yahtzee->players, Yahtzee->scores, Yahtzee->plyr_cnt );
    cout << "Congratulations, " << Yahtzee->players[Yahtzee->plyr_cnt-1].name << "!" << endl;
    cout << "You are the winner!!!" << endl;
    cout << "\n\n";
    cout << setw(20) << "NAME" << setw(10) << "SCORE" << "\n"
         << "-------------------------------------------------" << endl;
    for( int i = Yahtzee->plyr_cnt - 1; i >= 0; i-- ) {
        cout << "|" << setw( 20 ) << Yahtzee->players[i].name << setw(10) << Yahtzee->scores[i]
             << " | " << endl;
    }

    //Print average.
    if( true ) {
        float avg = CalcAvg( Yahtzee->scores, Yahtzee->plyr_cnt );
        cout << "\n"
             << "Average score: " << avg << endl;
    }
}
```

```
float  CalcAvg( int Scores[], int NumPlrs ) {
   int sum = 0;
   for( int i = 0; i < NumPlrs; i++ )
      sum += Scores[i];

   float avg = (float)(sum) / NumPlrs;

   return avg;
}
```

- void **SortScr** (Sort Score)

- This function utilizes a bubble sort to sort player names and scores

```
void SortScr( Player* players, int Scores[], int NumPlrs ) {
    for( int i = 0; i < NumPlrs - 1; i++ ) {
        for( int j = i + 1; j < NumPlrs; j++ ) {
            if( Scores[i] > Scores[j] ) {
                //Swap scores & player's names
                int temp_score = Scores[i];
                Scores[i] = Scores[j];
                Scores[j] = temp_score;
                Player temp_name = players[i];
                players[i] = players[j];
                players[j] = temp_name;
            }
        }
    }
}
```

- void **SaveGme** (Save Game)
- The final function saves the round number, number of players, names, and scores into a binary file and utilizes a counter of times saved for utilization of a static variable.

```
void save_game(Yahtzee* Yahtzee) {
  static int sve_cnt = 0;

  cout << "The game has been saved " << sve_cnt << " times" << endl;

  ofstream Out( FILE_DIR,ios::binary );

  //NOTE: Probably should check for file-open errors here.

  Out << Yahtzee->round << " " << Yahtzee->plyr_cnt << endl;
  for( int i = 0; i < Yahtzee->plyr_cnt; i++ ) {
    Out << Yahtzee->players[i].name << " " << Yahtzee->scores[i] << endl;
  }
  Out.close();

  sve_cnt++;
}
```