Project 1 - Yahtzee

Michael Cooper

17A - 48591

Dr. Lehr

November 14, 2021

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: 538
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.

Constants and Structs:

```
23
24  using namespace std;            //Library Name-space
25  //Global/Universal Constants -- No Global Variables
26  const int MAX_PLAYERS = 4;
27  const int MAX_NUMBER_ROUNDS = 13;
28  const char FILE_DIR[] = "yahtzee.txt";
29  //User Libraries
30  struct Player {
        int* rolls = new int[5];
32      string name;
33  };
34
35  enum DiceFace { ONE = 1, TWO, THREE, FOUR, FIVE, SIX, };
36
37  struct Yahtzee {        //Keeps track of the state of the game
        int current_round = 0;
39      int scores[MAX_NUMBER_ROUNDS];
        bool done = false;
41      int player_count;
42      int round;
43      Player* players;
44      Yahtzee();
45  };
```

Constants are used to set a limit on the number of players and rounds that will be kept track of and saved to the .txt file. A struct is used to implement the dice and names, along with an enumerator to set the dice accordingly.
The Yahtzee struct is meant to keep track of the game state at all times including the current round, score, and number of players and will connect to processes in the main.

Function Prototypes:

A look at the various prototypes that will be implements for the tutorial, turn processes, dialogue, and scores. The prototypes point to the Yahtzee struct and include a save and load function. Other functions serve to keep track of special rolls and sort through scores and averages.

```
//Function Prototypes
void print_rules(Yahtzee* Yahtzee);
void print_intro(Yahtzee* Yahtzee);
void play(Yahtzee* Yahtzee);
bool load(Yahtzee* Yahtzee);
void setup_players(Yahtzee* Yahtzee);
void readyup(Yahtzee* Yahtzee);
int player_turn(Yahtzee* Yahtzee, Player* player);
void print_scores(Yahtzee* Yahtzee);
void save_game(Yahtzee* Yahtzee);

bool two_of_a_kind(int rolCts[], int Size, int &index);
void SortScr( Player* players, int Scores[], int player_count );
float CalcAvg( int Scores[], int NumPlrs );
```

Return Parameters:

This parameter is meant to create a default game state to be passed into a function to start the game, combining the setup sequence with the scoring sequence.

```
63
64  Yahtzee* makeGame() {       //ties to struct to start game (creating the round)
65      Yahtzee* yahtzee = new Yahtzee;   //implemented for function return requirement
66      yahtzee->current_round = 0;
67      yahtzee->done = false;
68      yahtzee->player_count = 0;
69      yahtzee->round = 1;
70      return yahtzee;
71  }
```

A Look at the Main:

```
//Execution Begins Here
int main(int argc, char** argv) {
    //Set the Random number seed
    srand( time(0) );

    //Declare variables
    Yahtzee* game = makeGame();
    play(game);
    return 0;
}
```

The main is extremely brief. It sets the random number seed and ties the makeGame parameter to the Yahtzee struct. Passing game into Play starts the game sequence.

Implementation of Functions:

The game begins with a set up sequence:

- void **play**(Yahtzee* Yahtzee)

```
void play(Yahtzee* Yahtzee) {
    print_rules(Yahtzee);

    string user_input = "";
    cout<<"Press 'e' to exit, or any other key to start the game!"<<endl;
    cin>>user_input;

    if (toupper(user_input.c_str()[0]) == 'E') {
        return;
    }

    bool game_load = load(Yahtzee);

    if(!game_load || Yahtzee->round > MAX_NUMBER_ROUNDS) {
        setup_players(Yahtzee);
    }
```

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.

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);
}
```

- bool **load**(Yahtzee* Yahtzee)

The load function implements a binary file and inputs the stored data through vectors and then copies the vectors into the appropriate arrays

```
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;
}
```

The following functions all use 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:

- void readyup(Yahtzee* Yahtzee)
- void setup_players(Yahtzee* Yahtzee)
- void print_intro(Yahtzee* Yahtzee)
- void print_rules(Yahtzee* Yahtzee)

<u>The primary function:</u>

- 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.

```
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++ )
       cout << (char)('a' + i) << ") Roll: " << player->rolls[i] << endl;
       cout << endl;
        r2ndT = r3rdT = false;
        //setting to false so user can decide if they want to set to true.
        //after first roll dialogue

        cout << "Do you want to (S)top, or choose dice to hold and (R)oll again? [S or R]:";
        cin >> ans;
        ans = toupper( ans ); //will make sure entry is uppercase

        if( ans == 'R' ) {  //User wants to reroll
            r2ndT = true;

            cin.clear();
            cin.ignore();
            cout << "\nType letters of dice (a - e) you would like to keep:"<<endl;
            cout << "(please type a space between letters)"<<endl;
            std::getline( cin, s );  //Get whatever user types ('a' - 'e)
```

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.

```
//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;
```

- Void **PrntScr** (Print Score)/float **CalcAvg**(int Scores[], int NumPlrs)
- This function determines the winner and the average score along with the Calc Avg Function.

```
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++;
}
```