SRI LANKA INSTITUTE OF INFORMATION TECHNOLOGY

BSc (Hons) in Computer Science

Year 1 Semester 1

Programming Methodology - SE1012

Implementing a Two-Player Yahtzee Game in C

Registration Number	Name
IT23231528	R. M. C. S. Rathnayaka

List of Figures

Introduction

The implementation of a two-player Yahtzee game in C, where a human player competes against an optimally playing computer player, presents several challenges. The game consists of 13 rounds, with each player rolling five dice and having the option to roll up to three times per turn. The computer player must make optimal decisions to maximize its score while adhering to the game's rules and mechanics.

The scoring categories include options like:

- **Ones, Twos, Threes, Fours, Fives, Sixes:** Score based on the sum of the dice that show the chosen number.
- **Three of a Kind:** Score the sum of all dice if at least three of them are the same.
- **Four of a Kind:** Score the sum of all dice if at least four of them are the same.
- **Full House:** Score 25 points for a combination of three of one number and two of another.
- **Small Straight:** Score 30 points for four consecutive numbers.
- **Large Straight:** Score 40 points for five consecutive numbers.
- **Yahtzee:** Score 50 points for five dice showing the same number.
- **Chance:** Score the sum of all dice, regardless of combination.

The goal is to achieve the highest score by the end of the game.

The strategy used for the computer player

The computer player's strategy is to first roll each of the five dice and evaluate possible scoring categories. The computer uses these ratings to determine whether to reroll or keep particular dice, giving priority to high-value categories like Large Straight, Small Straight , Yahtzee and so on. It aims to maximise points by the conclusion of its turn and can improve its result with up to “two rerolls”. The highest scoring category is used to make the final decision, ensuring a balance between possible reward and risk.

```
// Function for the computer's turn
void computerTurn(int dice[], int *score) {
    int rolls = 1;

    rollAllDice(dice); //1 st roll
    displayDice(dice); // Display the dice after the  1st roll

    while (rolls < NUM_ROLLS) {
        rollAllDice(dice); //  2 nd Re-roll
        displayDice(dice);
        rolls++;
    }

    int category = rand() % 13 + 1;  // Random choice between 1 and 13
    int scoreForRound = 0;

    switch (category) {
        case 1: scoreForRound = scoreOne(dice); break;
        case 2: scoreForRound = scoreTwo(dice); break;
        case 3: scoreForRound = scoreThree(dice); break;
        case 4: scoreForRound = scoreFour(dice); break;
        case 5: scoreForRound = scoreFive(dice); break;
        case 6: scoreForRound = scoreSix(dice); break;
        case 7: scoreForRound = scoreThreeOfAKind(dice); break;
        case 8: scoreForRound = scoreFourOfAKind(dice); break;
        case 9: scoreForRound = scoreFullHouse(dice); break;
        case 10: scoreForRound = scoreSmallStraight(dice); break;
        case 11: scoreForRound = scoreLargeStraight(dice); break;
        case 12: scoreForRound = scoreYahtzee(dice); break;
        case 13: scoreForRound = scoreChance(dice); break;
        default:
        printf("Invalid choice.\n"); break;
    }

    *score += scoreForRound;
    printf("Computer Score: %d\n", *score);
}
```

Figure 1.1

1. Rolling the Dice
    - The computer rolls all the dice once using the rollAllDice(dice) function
    - While loop shows the dice are displayed after each roll. This loop repeats until the number of rolls reaches NUM_ROLLS
    - After all the loops ,display the final results of the dice

2. Random Category Selection
    - The computer randomly chooses a scoring category from the available options using rand() % 13 + 1
    - This generates a random number between 1 and 13 and it fits into one of possible categories

3. Scoring Based on Category
    - The computer determines the score for that round using a corresponding function,such as scoreOne(dice), scoreTwo(dice), scoreThree(dice) and so on
    - The result of calculation is stored in “scoreForRound”

4. Updating the Score
    - *score += scoreForRound; is used to add the round's score to the total score.

Explanation of challenges and their solutions

1. **Computer Dice Always Showing the Same Result**
   Because the dice were only rolled once at the start of the turn and not updated during subsequent rolls

   How I Addressed It:

   - **To fix this I added "rollAllDice(dice)"**

   Before correcting the code:

```
// Function for the computer's turn
void computerTurn(int dice[], int *score) {
    int rolls = 1;

    rollAllDice(dice);

    while (rolls < NUM_ROLLS) {
        displayDice(dice);
        rolls++;
    }
    displayDice(dice); // Final dice after rolls
```

Figure 1.2

After correcting the code :

```
// Function for the computer's turn
void computerTurn(int dice[], int *score) {
    int rolls = 1;

    rollAllDice(dice); //1 st roll
    displayDice(dice); // Display the dice after the  1st roll

    while (rolls < NUM_ROLLS) {
        rollAllDice(dice); //  2 nd Re-roll
        displayDice(dice);
        rolls++;
    }
```

Figure 1.3

2. **Handling the Display of Dice After Each Roll**
   The same dice values were shown repeatedly during the turn because the dice display was not being refreshed after each roll

   How I Addressed It:

   - **To show the updated dice values after each roll, Used to "displayDice(dice), after each roll to show the updated dice values.**

3. **Managing and Updating the Score After Each Turn**
   It was difficult to calculate and update the score after a randomly selected category because player needed to correctly accumulate the score for each round

   How I Addressed It:

   - **'Switch statement' to handle the different categories of scoring, whereby each category computes the round score by calling a specific function (such as scoreOne(dice), scoreTwo(dice), etc.)**

   After calculating the score for the round and added it to the total score '*score += scoreForRound '

4. **Ensuring Randomness in Dice Rolls**
   If the dice rolls were always the same (because rand() wasn't seeded properly), the game would be predictable, which is not ideal for gameplay

   How I Addressed It:

   - **'srand(time(NULL));' is used to initializes the random number generator with the current time.**

     This ensures that the dice rolls and random category selections are different each time the program is run

5. **Handling invalid scores**
   When calling scoring functions like scoreOne(dice), scoreTwo(dice), etc.Player need to ensure that these functions return valid scores, even if the dice roll doesn't fit perfectly into the category(example player tries to score a yahatzee with using mismatched dices)

   How I Addressed It:

   - **Added logic in each scoring function to handle invalid or non-scoring cases**.

   For example if the dice don't form a "Yahatzee" then return 0 .

Depth of reflection on what was or wasn’t implemented

Based on the code, here are some potential features that were not fully implemented or could be improved

1. **Tracking of Keep Dice**
   There is no clear way to keep track of which dice are kept, but the player can choose which ones to reroll. A feature could be added to track and display which dices are kept during each turn.

2. **Detailed User Input Dices Validation**
   Although the dice rerolling entry is somewhat checked to make sure the dice numbers entered fall within the range, the scoring category field may require more thorough validation to make sure users aren't entering incorrect or out-of-range selections.

3. **Turn and Score Reset**
   There is no reset or guarantee that a player can only score once in a category with the current layout. Players only receive one score for each category in a real Yahtzee game.

4. **Scoreboard / Display of Categories**
   Although the scoring categories are listed at the start but no scoreboard or indication of which categories each player has scored or left. Throughout the game, it would be helpful to track and demonstrate the player's progress, including the categories they have already achieved and the ones they still have available.

5. **Saving and Loading Game**
   Saving and loading a game state is not available. The game restarts every time you run the program. A save/load function would enable gamers to put a game on hold and resume it at a later time.