29th March , 2024

As suggested by the professor I will look into modifying the roll dice code from assignment 4-3 to function within the confines of my final project. Currently I think that the code block below will work as desired during gameplay. The current roll of the dice is passed in alongside how many dice to roll. A switch case then rolls the dice starting with the last dice it should roll, this allows the switch case to cascade without the need to rewrite a lot of code. Probably a better way to do it but this was what popped into my head immediately so I am going with it.

| **void** **RollDice**(**int** diceRoll[6], **int** diceToRoll) {   **switch** (diceToRoll) {  **case** 6:  diceRoll[5] = rand() % 6 + 1;  **case** 5:  diceRoll[4] = rand() % 6 + 1;  **case** 4:  diceRoll[3] = rand() % 6 + 1;  **case** 3:  diceRoll[2] = rand() % 6 + 1;  **case** 2:  diceRoll[1] = rand() % 6 + 1;  **case** 1:  diceRoll[0] = rand() % 6 + 1;  **break**;  **default**:  **break**;  } } |
| --- |

In terms of rolling the dice this works but I still need to consider if the player holds back some dice for their score and rolls the rest of the dice. Currently the program will overwrite whatever dice it wants. As of now I am going to make a function that resorts the dice array to have the ones the player wants to keep at the end of the array and the dice to reroll at the front. In theory this should work with the current RollDice function.

Seems there is a sort function in the algorithm library which by default sorts integers from lowest to highest so I might as well try using that.

The code block works in predetermined rolls. Probably should put it into a function to use it again in the reroll case.

| //Sort the rolls in ascending order sort(currentRoll, currentRoll + 6);  //Loop through all the dice, starting with the highest **for** (**int** i = 5; i >= 0; i--) {  //3 of a kind  **if** (i > 2 &&  currentRoll[i] == currentRoll[i - 1] &&  currentRoll[i - 1] == currentRoll[i - 2]) {   //If the roll is a 1, do different algorithm  **if** (currentRoll[i] == 1) {  tempScore += currentRoll[i] \* 1000;  }  **else** {  tempScore += currentRoll[i] \* 100;  }   i -= 2;  }  //100 points  **else** **if** (currentRoll[i] == 1) {  tempScore += currentRoll[i] \* 100;  }  //50 points  **else** **if** (currentRoll[i] == 5) {  tempScore += currentRoll[i] \* 10;  }   cout << tempScore << endl; } |
| --- |

Modified the previous block to be a function that allows for any amount of dice to be rolled:

| **void** **ScoreDice**(**int** diceToScore = 0) {   //Sort the rolls in ascending order  sort(currentRoll, currentRoll + diceToScore);   //Loop through all the dice, starting with the highest  **for** (**int** i = diceToScore - 1; i >= 0; i--) {  //…  } } |
| --- |

Now I need to add functionality to the reroll and check for a farkle.

Added a local variable to check how many points the player has earned in a roll, if that value is zero then the current uncashed score will be removed.

| **void** **ScoreDice**(**int** diceToScore = 0) {   **int** pointsScored = 0;   //Sort the rolls in ascending order  sort(currentRoll, currentRoll + diceToScore);   //Loop through all the dice, starting with the highest  **for** (**int** i = diceToScore - 1; i >= 0; i--) {  //3 of a kind  **if** (i > 2 &&  currentRoll[i] == currentRoll[i - 1] &&  currentRoll[i - 1] == currentRoll[i - 2]) {   //If the roll is a 1, do different algorithm  **if** (currentRoll[i] == 1) {  pointsScored += currentRoll[i] \* 1000;  }  **else** {  pointsScored += currentRoll[i] \* 100;  }   i -= 2;  diceScored += 3;  }  //100 points  **else** **if** (currentRoll[i] == 1) {  pointsScored += currentRoll[i] \* 100;  diceScored++;  }  //50 points  **else** **if** (currentRoll[i] == 5) {  pointsScored += currentRoll[i] \* 10;  diceScored++;  }   }  //If no points are scored on this roll - FARKLE  **if** (pointsScored == 0) {  cout << "FARKLE" << endl;  //Reset values  tempScore = 0;  diceScored = 0;  for (int i = 0; i < 6; i++) {  currentRoll[i] = 0;  }  }  **else** {  tempScore += pointsScored;  cout << "\nPoints Scored: " << pointsScored << endl;  cout << "Current Score: " << tempScore << endl << endl;  } } |
| --- |

30th March, 2024

Merged Roll and Reroll into one command instead of having an if-else check to determine if it was an initial roll or reroll.

| //Roll **else** **if** (command == "R" || command == "r") {   //If all 6 dice have a score face  **if** (diceScored == 6) {  //Reset current roll  **for** (**int** i = 0; i < 6; i++) {  previousRollSets[savedRolls \* 6 + i] = currentRoll[i];  currentRoll[i] = 0;  }   diceScored = 0; //Reset scored dice count for current roll  }   RollDice(currentRoll, 6 - diceScored);   //Show dice outcome  cout << "Rolled " << 6 - diceScored << " dice: \n" << endl;  **for** (**int** i = 0; i < 6 - diceScored; i++) {  cout << currentRoll[i] << endl;  }   ScoreDice(6 - diceScored); } |
| --- |

Created a dynamic array of players to keep track of player score. Added 2 functions, one to change the turn to the next player in the list and another to reset all current turn values. Removed resetting values from FARKLE roll in ScoreDice.

| //Set the turn to the next player in the list **void** **NextTurn**() {  currentTurn++;   **if** (currentTurn >= numOfPlayers)  currentTurn = 0;   ResetValues(); } |
| --- |

| //Reset all current turn values to their defaults **void** **ResetValues**() {  **for** (**int** i = 0; i < 6; i++) {  currentRoll[i] = 0;  }    score = 0;  diceScored = 0; } |
| --- |

Added code to InitializeGame function to prompt the user for player aliases after the number of players.

| **void** **InitializeGame**() {  **bool** set = false; //Used to determine if number of players has been set   srand(time(NULL)); // initialize random seed:   **//…**  //Make dynamic array of players  players = **new** Player[numOfPlayers];   //Prompt for player aliases  **for** (**int** i = 0; i < numOfPlayers; i++)  {  string input;  cout << "What is Player " << i + 1 << "'s alias? ";  cin >> input;   players[i].SetName(input);  }   currentTurn = 0;   system("cls"); } |
| --- |

Added a method to allow for cashing the players points. Players must have 500 points before they can get on the scoreboard (aka cash for the first time) : in accordance with Farkle’s rules.

| **void** **CashScore**() {  **if** (players[currentTurn].GetScore() == 0 && score < 500) {  cout << "\nYou can not cash your score now!\nYou have to have at least 500 points to get on the scoreboard!" << endl;  **return**;  }    cout << "\nYou banked " << score << " Points!" << endl;  players[currentTurn].AddScore(score);  cout << "You now have " << players[currentTurn].GetScore() << " Points.\n" << endl;  NextTurn(); } |
| --- |

Added a method to display players current turn and statistics (cashed points, current turn points). Called from main after every roll.

| //Prints information on the players current turn and their info **void** **PrintTurnInfo**() {  system("cls");   cout << "It is " << players[currentTurn].GetName() << "'s Turn" << endl << endl;  cout << "You have " << score << " points that can be cashed." << endl;  **if** (players[currentTurn].OnScoreboard())  cout << "You are on the scoreboard with " << players[currentTurn].GetScore() << " points." << endl;  **else**  cout << "You are not on the scoreboard: get 500 points to enter the race!" << endl << endl; } |
| --- |

I had made a system that stores each set of 6 dice rolled when rerolling as a means to keep track of how many points are earned but realized after the fact that I did not need to do that. I was already keeping track of the score with an integer as it is and was overcomplicating things.

I still have a few minor things to do such as

* Filling out the PrintCommands() function to show what the user can actually do
* Check for the win condition to end the game.
* Have the ability for a player to drop out with the game ending if there are less than 2 players.
* Make the scoreboard function.
* Find bugs that may be present when there are more than 2 players