501 Darts Report

Patrick Collins

Who starts first?

To decide who starts first, a minigame is played between the players. This is done by calling a function miniGame() in the 501 darts.cpp. In this minigame the players simply aim for the bullseye, whichever one is closer they win the minigame and start first.

Scoring

At the beginning each player aims for the optimal score of treble 20(60). This is done by calling the function player_name.throw_treble(20, Player success rate) in the player class. The players aim for this until their score drops below 70, which is when the second part of gameplay comes in.

Focus

Once a player's score has dropped below 70, they will enter the focus section, which now makes them aim for singles. This is done by calling the function player_name.throw_single(dartboard number, Player success rate) in the player class. Depending on their score, they will aim for the dartboard to try and get their score to 40, 50 or 60. This is to enter the final stage of gameplay.

Checkout

The final stage of gameplay is winning the game or checking out. The first part to this is if the player's score is on 50. If their score is on 50, they will aim for a bullseye to try and win the game. This is done by calling the function player_name.throw_bull(50, Player success rate) in the player class. However, the player can miss this and needs to hit doubles.

Hitting doubles is done by checking the player score before throwing and aiming for a double which will get their score to 0. This is done by checking the first digit(player_name.getScore() / 10) and second digit of the player's score(player_name.getScore() % 10). For example, if the player's score is 38, the first digit is 3 and second is 8. The player will then enter this part in the IF statements and aim for double 19. If successful, their score will be zero and will have won the set.

Going below zero

If one of the player's score goes below zero due to misses, the score of the last 3 dart throws are added back. This is done by checking the score after each throw and if below 0 the function player_name.addScore(getBelow()) is called in the score class.

Outcomes

To count the outcomes, I just used 14 variables. One for each outcome. There are 14 IF statements that check these outcomes. The variable is incremented if the match ended with the set wins of the players matching the outcome.

For example, outcome one is John winning 7 sets and Sid winning 0. The IF statement will check this outcome and add one to the first outcome's variable. This is checked through all games in the simulation.

Calculating the frequency

To calculate the frequency of each outcome I used the 14 variables.

(Number of outcome divided by number of games) multiplied by 100.

This gives the frequency of each outcome and is then displayed to the user.

Benefit of object orientation

During my time using object orientation I found that the program ran a lot smoother having implemented it. Everything is organised and the separation of classes and its .cpp files really help keep the main .cpp file uncluttered and without errors. If there is an error, which I ran into many, it is easy to single out where the issue is occurring and fix it.

In comparison to the slot machine in term 1, I can definitely see how object orientation can improve it. Instead of all the functions in the one .cpp file, having classes to break up the program could have helped structure how the program was running and to see the flow of gameplay much easier. This could even be a task for myself before second year to further understand the benefits of object orientation.