Report on Darts Assignment – CMP 102 – Isaac Basque-Rice

My code makes use of three classes, Player, Game, and DartBoard.

Player:

Player stores several variables to do with each individual player, i.e. their names, how many points they each have, how many games they have won, how accurate they are, etc.

Additionally, this is the class which stores the ThrowDart function, this function has two parts to it: the Aiming mechanism, which decides where each player will aim on the board based on their current score, and the throwing mechanism, which calls one of four different functions in the DartBoard class (depending on whether the player has aimed for single, double, treble, or bull) which determines what the player actually hits.

Finally, this class also contains a struct, which takes and stores the score value of both players before they take their three throws, this is so if the player in question throws an invalid throw (e.g. the point total goes below 0) the score can be reverted back to its previous state, thereby making that round invalid.

Game:

Game contains five variables. Winner, which is a string, that can either be the name of Player 1 or Player 2 depending on who has won the most recent game. MatchNo, the amount of matches that the game generally will consist of (this value is unlimited), game, which is which number game in this set the players are on (maximum of 5), set, which is the number set in this match the players are on (max 13) and match, which is the match number the players are currently on, this is limited only by matchNo, i.e. when matchNo == match, the game terminates and outputs the statistics

This class also contains three Boolean functions, Check(Game/Set/Match)Winner, these functions take the instantiated player object of either 1 or 2, and check to see if said player has won the game, set, and/or match. If so, the score values are reset to the state they were in at the beginning of the game and the relevant player variables are iterated. Finally, they all return True if the player has won the game, set, or match, and false if they haven't, this is in order to make additional changes in the main function using if statements.

DartBoard:

DartBoard contains only four functions: HitBull, HitTreble, HitDouble, and HitSingle. Each of these serve essentially the same function but for different sections on the dartboard. Each function takes the accuracy of the player (for HitBull it's 80 and for everything else it's user defined), and the target of where the player is aiming (apart from HitBull, there is only one place to aim in that function), and calculates whether the player has hit it or not, the function then returns the value that it calculated and that is where the player has hit. It may be of interest to note that in all functions where it is possible to aim for multiple locations, the dartboard itself is represented by a 2D array, this is so that in the event that the player does, in fact, miss, they will instead hit another, predetermined location that is "next" to the place the player has aimed on the dartboard.

Isaac Basque-Rice

Main:

The main cpp file (Called DartsTask1.cpp) is, of course, where the execution of the program begins and ends, In my solution I make use of 5 functions (including main()), two of which are designed solely to input and output fairly large bodies of text, one designated for the introduction where the values player names, accuracy, and the number of games are taken, and one for Statistics, this is shown after the game is played and displays all the various statistics for each player, as well as who actually won.

The other two functions are for actually playing the game, we have Play(), which is what actually plays the game (one call of Play() is one turn EACH for both players), and CheckWinner, which is where the three functions as described in Game are called.

The Play() function has several interesting properties, the way it is written is to avoid unnecessary repetition and as such makes use of an if/else statement to switch between player 1 and player 2. If playerNumber is odd, then player 1 will take their turn, if it isn't, player two takes their turn. This playerNumber variable is iterated by 1 each round to ensure that the next time it is called, the player who had their go previously doesn't have their go again. Additionally, this function is the first to make use of a globally declared pointer that switches location between players 1 and 2, this is in order to manipulate their attributes individually.

Finally we have the CheckWinner function, which, as it states, runs all three checks found in the game class and if they return true, manipulates the attributes of the relevant player. This area is where the points are reset to 501 (this must be done explicitly for both player1 and player2 as if it isn't only the player it is checking at that moment will be reset).

Object Orientation in this solution:

I feel that Object Orientation is a powerful tool that is perfectly suited for this kind of problem. Whilst usage of procedural or functional programming is technically possible with this program, the ability to compartmentalise and functionally decompose different aspects of the solution through the usage of classes made for a much easier and more manageable programming experience.

In particular, the ability to instantiate multiple objects of the Player class resulted in countless lines of code not needing to be used, without OOP I as a programmer would have possibly had to write the same thing (throwing each dart) once for each player, more than doubling line count and effort in debugging.