CMP102 Unit 2: 501 Darts

When the program is run, the user is presented with a main menu from which they may choose to run a simulation, play an interactive game, test the different strategies, or quit.

Task 1: 501 Darts Simulation

If the user chooses to run a simulation, they then input how many simulations they wish to run and whether Joe should use a basic or enhanced strategy. Strategies are implemented as classes which inherit from the base Strategy class, which declares a pure virtual function for selecting a target based on the player's score, their number of darts left and their score at the start of the round. Objects of the Player class store a pointer to a Strategy object, as well as the player's name and accuracy, and define a function to select and perform a throw using a given Dartboard.

Once the user has input their choices, two Player objects and a SimulatedGame object are created, with the Player objects being passed by reference to the SimulatedGame object's constructor. The SimulatedGame class inherits from the abstract base class Game which stores an array of two pointers to Player objects, a Score object, and a Dartboard object, and defines a function to play a single game of darts. The Score object stores the score of each player and contains functions to update the score and check whether an update would lead to a win or bust, while the Dartboard class statically stores an array of Target objects, which each store the base, multiplier and value of a possible target on a dartboard. The Dartboard class also statically stores the neighbours of each section of the board and defines functions to throw at a target and return the actual target hit. Targets are referred to by their index in the array of targets, with the Dartboard class also defining a function to get the corresponding index based on a base and multiplier.

The SimulatedGame class also defines functions to play a round, a set and a match, and to simulate a number of matches. The function to run a simulation is called after the SimulatedGame object is created. This function stores how many sets each player has won in the current match and records each match result in the simulation_results member variable. This variable is a 2-dimensional vector where the row index is the player index, the column index is the number of sets their opponent won and the element itself is the number of times this result occurred. The number of sets the player won is determined from the size of the vector, which is set based on the match length as this determines how many sets a player needs to win. Finally, the output_results function is called, which formats the simulation_results vector to show how many times (as a percentage of the total simulations run, which is stored as a member variable) each result occurred. The results of a simulation in which Joe uses a basic strategy are shown below.

Results

```
Sid : Joe - Likelihood based on 10000 matches
--------------------
7 : 0 - 0.89%
7 : 1 - 3.66%
7 : 2 - 7.06%
7 : 3 - 11.23%
7 : 4 - 13.38%
7 : 5 - 14.05% [Most likely result]
7 : 6 - 12.54%
0 : 7 - 0.19%
1 : 7 - 1.12%
2 : 7 - 2.67%
3 : 7 - 5.15%
4 : 7 - 7.7%
5 : 7 - 9.82%
6 : 7 - 10.54%
```

Task 2: Enhanced Strategy

The enhanced strategy works by taking into account the possibility of missing, and the EnhancedStrategy class has a static initialize function which calculates the best finish sequence for each score from which finishing is possible. Finish sequences are stored as objects of the FinishSequence class, which store the sequence itself, whether it ends on a double 16 or 20, and its contingency score. The contingency score of a sequence is calculated as 4 minus the number of darts needed to win from the score most likely to be achieved in the case of missing the first throw. Additionally, the

contingency score is doubled if this score allows a finish ending in a double 16 or 20, and is set to 0 if no finish is possible (within 3 darts) in the case of such a miss. The most likely miss is defined as the corresponding single if the throw is a double or triple, or each neighbouring single otherwise, and in this case the contingency scores of each of two misses are averaged. This measure allows the quality of a throw to be assessed based on the options available in the case of a miss.

The best throw for each score is calculated by generating every possible finish sequence excluding any with a higher number of darts than is necessary, removing those which do not end in double 16 or 20 (if any do), removing those whose contingency scores are not the best possible, and then choosing the remaining sequence with the highest value first throw. If a finish is not possible from a certain score, a triple 20 is always thrown, since if a triple 20 does not lead to a score from which a finish is possible, neither does any other throw. Moreover, testing showed that there is no situation where a throw less than triple 20 leads to a score whose finish sequence's contingency score is higher than that which would have been achieved with a triple 20. No consideration of the opponent's score is taken, as taking a slower, ostensibly safer route when the opponent could not finish next turn led to lower performance than even the basic strategy. Finally, if a single dart remains and a bust is possible and would be advantageous in terms of contingency score, and at least equivalent in other respects, the select_target function returns a triple 20.

The performance of each strategy can be tested by choosing the 'test strategies' option at the main menu, which calculates and displays the average number of rounds needed to win using each strategy. This confirms that the enhanced strategy slightly but clearly outperforms the basic one at accuracies below around 90%.

The results of a simulation in which Joe uses the enhanced strategy are shown below. The use of this strategy improves Joe's chances slightly, but Sid still has the clear advantage.

Results

```
Sid : Joe - Likelihood based on 10000 matches
--------------------
7 : 0 - 0.62%
7 : 1 - 2.89%
7 : 2 - 6.22%
7 : 3 - 9.91%
7 : 4 - 12.17%
7 : 5 - 12.99% [Most likely result]
7 : 6 - 12.67%
0 : 7 - 0.19%
1 : 7 - 1.39%
2 : 7 - 3.41%
3 : 7 - 6.29%
4 : 7 - 8.85%
5 : 7 - 10.37%
6 : 7 - 12.03%
```

Task 3: Interactive Game

The user may also choose to play an interactive game against Sid. Interactive gameplay is implemented using the InteractiveStrategy derived class, which selects a target based on console input, and the InteractiveGame derived class, which defines the play_round function to display both players' current scores above an ASCII art dartboard. The current player's score is highlighted in green, and targets hit are highlighted in green for 0.5 seconds before they are coloured white again. When a player wins, the name of the winning player is displayed below the dartboard and the game ends.

Benefits of Object-Oriented Programming

The first benefit observed from structuring programs around classes and objects is that it provides a concrete way to divide a large program into smaller components. This is an easier and clearer development approach than simply grouping related functions and variables with comments or naming conventions. Secondly, programming in terms of objects, or 'things', is less abstract and easier to understand then using a procedural style, especially in large programs with many components. Object orientation also allows the encapsulation of data and functions which should only be available in certain contexts, which helps minimise possible errors and aids maintainability.