Unit 2 – Darts Assignment

Please complete all the pages on this form and upload it along with your .exe, code and report. This form **MUST** **NOT BE ZIPPED.**

Please 🗸 which tasks you have completed:

|  |  |
| --- | --- |
| Task 1 | 🗸 |
| Task 2 | 🗸 |
| Task 3 | 🗸 |

Files for uploading – these must be checked off before uploading

|  |  |
| --- | --- |
|  | 🗸 **to indicate these have been included** |
| Code – **MUST BE COMPRESSED AS A .ZIP** |  |
| .exe – **THIS MUST BE INCLUDED IN THE .ZIP** |  |
| Report - **PDF MUST BE UPLOADED SEPARATELY FROM THE .ZIP** |  |

Please answer the following questions

|  |  |
| --- | --- |
| **Questions** |  |
| Does the code compile without syntax errors? If no explain what the problems are and how you’ve attempted to resolve it. | Yes |
| How many matches does your program simulate? | 10,000 |
| Who plays first? | ~~Joe/Sid~~/User decides **\*delete as applicable** |
| What percentage accuracy have you assigned to the players? | If the user choose to run a fully automated simulation then the players are created with the following accuracies:  "Joe", 71, 73, 80, 78, 80  "Sid", 76, 66, 80, 82, 76  With each column representing the following in order left -right:  - Inner bull success  - Outer bull success  - Single success  - Double success  - Treble success  If the player chooses a ‘manual simulation’ then they get to input each of the above success rates and the simulation runs as per normal.  If the player chooses an interactive game against the computer, then the user enters their own success rates and the computer opponent is generated with the following:  "Computer", 71, 73, 80, 78, 80 |

# Task 1

Explain how your code calculates the frequency across all of your simulations. If no frequency is calculated explain how you have modified your code from the 301 exercise. Please include a screen shot of the output.

|  |
| --- |
| Once a match has been won, this is then added to a vector of type Match, Match in itself is a struct containing only three pieces of data, namely, result\_1, result\_2 and tally.  So each element in the Match vector contains the results from a single game with player 1 result always being result\_1 and likewise for player 2/result\_2, the tally keeps count of how many times a particular result has occurred, with there only being a maximum of 14 possible results if we get repeat results we don’t add this to the vector we simply increment the tally at that results index position in the Match vector. Of course while new results are pushed onto the vector, this vector however should never become larger than 14. Doing it this way saves massively on having to keep a vector containing ALL results, including the repeated ones, which of course would lead to having a vector of size 10,000 instead of size 14, and if were having to loop through this vector, its better to loop over 14 elements rather than 10,000.  Once we have all this data stored we simply loop over each element printing out the index positions result\_1 and result\_2 and then calculate percentage freq by dividing the tally at any given index by the amount of matches played \* 100.  I have an added column of ‘Occurrence/Tally’ just so the user can see the actual numerical amount of times any particular result occurred. All of this produces the following output:  A screen shot of a computer  Description automatically generated |

# Task 2 – if attempted/completed

Explain how you have developed your solution beyond the basic algorithm in task 1. How does a player now decide on the best target to aim for? What challenges did this present? Please include a screen shot of the frequency output.

|  |
| --- |
| When first thinking about how I would solve this I started to feel a little overwhelmed at the prospect of having to try and come up with a solution that would account for ALL potential routes of finishing once on any given finish number. I realized that a player may choose any way they like to finish within reason as long the player finishes on a double and trying to account for this seemed to me unrealistic. So instead I focused on what were the most common finish ‘routes’ dart players take once on any given finish number. This was helpful as it provided me clear boundaries to work within.  From here I still had to think about how I would be able to implement that, as the players score would need to be checked against some form of table containing all possible finishes but then I would somehow need to be able to get from there to getting the information, not only about how many darts to finish that number may require but what particular darts are required numerically to finish and in what order, Of course hitting your double 16 first then a treble 20 for a finish of 92 is absolutely useless.  After some thought I realised the easiest way to contain all this information in one place is through a class, or as I implemented it, a struct, I chose a struct as I knew that this particular object would have no implementation of its own and simply be pure data.  Once I had my Finishes struct in place containing four pieces of data, namely, a finish total, and three separate 2 element arrays, with the first element to contain the number to throw for and the second element to contain the type to throw for, single(s), double(d), treble(t), inner bull(i), outer bull(o), I then created an allFinishes array of type Finishes in the Dartboard object and populated it with every finish <= 170.  It should have been said at the start but it has been programmed that if the players score is greater than 170 then we need not concern ourselves with anything other than throwing for treble 20. I realise there are circumstances where, as the example provided in the assignment brief, that it would be more wise to aim for other numbers in order to leave certain finishes, however, I seen these as very specific circumstances, and given the success rates of our simulated players felt the complexities of having to integrate these checks into my already existing solution vs how often these circumstance may arrive was simply not worth it, and so left my solution as is, with carrying out a simple check is the players score is greater than or less than 170, then act accordingly.  This is not to say that it could not be improved at a future date, there are many improvements that could be made and of course leaving it this way reduces flexibility should the players success rates become lower.  To continue, once the players score is <= 170 the program checks if they are able to finish and responds accordingly. This is achieved by looping through the allFinishes array, comparing the players remaining score with the finish total at each index position and if a match is found then we pass to the throwDart function the number and type contained at the dart\_1 position of that element, and the program continues to carry out this same check each and every single dart. |

# Task 2 – if attempted/completed, Cont.

|  |
| --- |
| It was noticed that it does not work as originally intended, whereby passing the dart\_1 information, then passing dart\_2 information and so on until we either finish or don’t, instead what happens due to looping and checking each and every single dart(which is required in case we don’t hit the number intended for) is this: a finish total is found, dart\_1 is thrown, whether this dart is successful in its hit or not is irrelevant as the next loop then checks the new score against the allFinishes array and a new finish total is found, then passes to the throwDart function the new dart\_1 and not dart\_2 from the previously found finish total, It works whether any given throw is successful or not which I suppose meets my original intention but is not how I intended it on happening. This was clearly a misunderstanding of how to solve this particular problem in the first place.  As an extra I did develop the solution further with regards to statistical output, I only mention this here due to already having provided in the previous task a screen shot of my frequency output for the match results and so instead of providing either the same image again or the same image again but with slightly different data I shall provide instead a screen shot of the (n) dart finish statistical data output. I realise this was not part of the original brief but it was something that was developed early on and I decided to leave it in.  A screenshot of a computer  Description automatically generated |

# Task 3 – if attempted/completed

Explain how you developed the game beyond the basic interface in task 1. How do users choose what to aim for? How did you design and develop the interface? Please include some screen shots of the user interaction.

|  |
| --- |
| The interface was developed further by creating functions which prompted the user for a response every single dart, the first was to ask which number they would like to throw for and the second was to ask which type of their chosen number they would like to throw for.  Of course some checks were put in place to prevent users trying to throw for treble bull etc but on the whole it was relatively simple to implement and integrate with the already existing code, as once it was know what number and which type, then this information could then be passed to the already existing functions and the program ran as per normal.  The limitations of an interactive game however are 1 match only which even then will still require a lot of individual 501 legs, first to 3 legs wins a set, first to 7 sets wins the match. Dart output and remaining score is printed every single dart and the user cannot switch printed output off like what can be done for the fully automated/manual simulations. It made no sense to have the ability of switching printed output off as the player needs to see their score. The question is still asked but if the player chooses ‘yes’ to try and switch printed output off, they are told they can’t.  In order to assist the player with how to finish an SFML window was created to house a simple rectangle shape with an image showing all the possible finishes from 170 and below. So once the player achieves a certain score they simply look up the table and throw for the required number and the algorithm works exactly as it has done from task 2.  On top of this some graphics were added in the form of ASCII art and a screen buffer was used similar to the fruit machine project in order to provide some nice scrolling effects on the intro and clear the screen ready for the game starting, and lastly the font colour was changed to yellow to provide more contrast. |

# Task 3 – if attempted/completed, Cont.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Code – for each class please copy this page and paste screenshots of the code in.

NOTE – Some of my classes are hundreds of lines long and would never fit in a screen shot, in fact would probably require many screen shots just for one class, and I have 4 classes not incl. Main, then I also have 3 structs. I have instead provided the code directly from each .ccp and .h file. I understand they may be a very specific reason why screenshots were requested that I am not privy to and so if screens are required then I shall of course provide them.

|  |
| --- |
| **Class Name:** |
| **.cpp** |
| **.h** |