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 in a .zip**  **The zip file name must contain your student number.** | 🗸 |
| .exe – **this must be included in the .zip** | 🗸 |
| This form – **uploaded outside of 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. | There were no syntax errors, however there was a logic error. The program would occasionally stop running due to the players’ score going below zero. I believe I have solved this by using watchpoints at specific points and performing a dry-run. As a result of these debugging strategies, I found a part of the code where there was no clause to prevent the player from going “bust”. I added a clause there and the program has since run correctly. |
| How many matches does your program simulate? | The player can choose how many to run in the 301 mode as per the brief, and in 501 it runs 100 times. |
| Who plays first? | User decides in 301 as per the brief, and whoever is closes to the bull in 501 |
| What percentage accuracy have you assigned to the players? | The user decides in 301 as per the brief, and in 501 I have assigned 85% to player one and 87% to player two. |

# Report – please write your report below:

My program contains 301 and 501, so the only interaction required from the user is to select the game mode and to alter the accuracies of the players and who throw first in 301, as well as choosing how many times to run the simulation, as per the brief. For both game modes, a switch statement is used to alternate between the two player objects until one player wins the game, and there is always an if statement to make sure the player’s score never goes below 50 for 301 and 2 for 501. For 301, the code uses a do while loop (while both players’ scores are above 0) inside a for loop for the number of games the user has entered. To select the target in 301, a function is called with the score being passed in to calculate what the player should aim at. The program then aims for this and subtracts the hit from the player’s score, provided it doesn’t take the player below 50. The program runs until one player hits a bullseye to finish. The artificial intelligence involved is calculating the target based on the player’s score. After the specified number of simulations run, the program calculates the average number of successful hits per game for each player as well as each player’s success rate based on the number of successful hits compared to the number of total hits. This is then output. For 501, the approach is more complicated. I use many different if statements to determine what the program should do next. To start with, the program checks if the player has a score below 180 and not on a number that has no finish. This is to avoid the player going bust. If this condition is true but the player’s score is over 170, the program throws for 3 treble 15s to get down to a relatively easy finish. The program then checks if the player has a score under 40 which is even. If this is true, the player can throw for a double to checkout. If the player has a score under 170, a 3 dart finish is possible. To check if the finish is a 3 dart finish, a 2 dart finish, the program calls the checkout function, which contains a 4d array of all the three dart finishes and a 3d array of all the two dart finishes. To check for a one dart finish, the program checks if the score is below 40 and even, in which case it throws for a double of half the player’s score to attempt to checkout. The steps for two and three dart finishes are concatenated into a string and returned to the main function. The main function then splits this up using substring operations. If the first character of the string is a “T” the program throws for a treble, if the first character is a “D” the program throws for a double, and if the first character is a number then the program throws for a single. If the first step is achieved, the player can move onto the second step/final step if it’s a two dart finish, or the final step for a 3 dart finish. If a step of the checkout is not hit, the alternate checkout function is called to check if the player can still finish even though a step was not achieved. If the player can finish, this process repeated using the new required score. If the player cannot finish, the function returns an empty string and the program throws for a single 20. This process repeats until one player reaches 0. Once a player reaches 0, the number of games they have won is incremented by one, and both players scores are reset to 501. Once a player has won three games, their set wins increments by one and each players’ score resets to 0. Once a player wins 7 sets, the game is over. Each players’ set wins are concatenated together to create a score, which is scored in the scores array. Once the set amount of simulations have been run, in this case I’ve elected to run it 100 times, the frequency for each score is calculated. This is done by looping through the scores array and counting the number of occurrences for each score. Once a score has been counted, it is stored in another array which is checked every iteration of the loop to avoid counting a score twice. After this is completed, the frequencies for each score are output. The artificial intelligence for my 501 game mode is demonstrated when calculating the finishes using my arrays and the current score. I think that object orientation is much more beneficial than procedural programming for this project because classes keep everything together and there can be as many instances of those classes as possible, rather than having far too many different variables for each player, which would be the case if this was done using procedural programming. My debugging approach consisted of using breakpoints to test certain parts of the code, but mostly watchpoints on variables to make sure everything is updating correctly.

# Pseudocode – please write your pseudocode below

Source.cpp:

1. Include the required libraries

2. Initialise random function

3. Declare playOrder variable to store which player’s turn it is

4. Declare gameDecision variable

5. Take in gameDecision from user input

6. If gameDecision = 1

6.1 Call the constructor to initialise the players’ names, total throws, game wins, and current score

6.2 Initialise an object of the Board class

6.3 Declare variables for the players’ accuracy

6.4 Take in the players’ accuracy from user input

6.5 Call the set\_accuracy function and pass in the accuracy for each player

6.6 Call the set\_successful\_hits function and pass in 0 for each player

6.7 Declare the variable to store the number of games to be simulated and the variable to store the player’s score for a throw

6.8 Take in playOrder from user input

6.9 Loop for the number of games to be simulated

6.9.1 loop until either player reaches 0

6.9.2 Switch statement for playOrder

6.9.3 If the case is 1 player one will throw first

6.9.3.1 Call set\_total\_throws function to increment the player’s total throws by one

6.9.3.2 Call set\_target function passing in the player’s score

6.9.3.3 If target = 50

6.9.3.3.1 If the player’s score = 50

6.9.3.3.1.1 If the score for that throw = 50

6.9.3.3.1.1.1 Call set\_successful\_hits to increment successful hits by one

6.9.3.3.1.1.2 Call set\_current\_score and set the player’s score to 0

6.9.3.3.1.1.3 Break

6.9.3.3.1.2 Else if the score for that throw is 50

6.9.3.3.1.2.1 call set\_successful\_hits to increment the successful hits by one

6.9.3.3.1.2.2 call set\_current\_score to update the player’s score

6.9.3.3.1.3 Else

6.9.3.3.1.3.1 if the player’s score minus the score for that throw is at least 50

6.9.3.3.1.3.1.1 call set\_current\_score to update the player’s score

6.9.3.4 If the target is 20

6.9.3.4.1 Call single\_throw to throw for single 20

6.9.3.4.1 if the throw score = 20

6.9.3.4.1.1 call set\_successful\_hits to increment the player’s successful hits by one

6.9.3.4.2 else

6.9.3.4.2 if the player’s score minus the throw score is at least 50

6.9.3.4.2.1 call set\_current\_score to update the player’s score

6.9.3.5 else

6.9.3.5.1 throw for the target

6.9.3.5.2 if the target is hit

6.9.3.5.2.1 call set\_successful\_hits to increment the player’s successful hits by one

6.9.3.5.2.1 call set\_current\_score to update the player’s score

6.9.3.5.2 if the target is not hit and the score will remain at least 50

6.9.3.5.2.1 call set\_current\_score to update the player’s score

6.9.3.2 break

6.9.4 If the case is 2 player two will throw first

6.9.4.1 Call set\_total\_throws function to increment the player’s total throws by one

6.9.4.2 Call set\_target function passing in the player’s score

6.9.4.3 If target = 50

6.9.4.3.1 If the player’s score = 50

6.9.4.3.1.1 If the score for that throw = 50

6.9.4.3.1.1.1 Call set\_successful\_hits to increment successful hits by one

6.9.4.3.1.1.2 Call set\_current\_score and set the player’s score to 0

6.9.4.3.1.1.3 Break

6.9.4.3.1.2 Else if the score for that throw is 50

6.9.4.3.1.2.1 call set\_successful\_hits to increment the successful hits by one

6.9.4.3.1.2.2 call set\_current\_score to update the player’s score

6.9.4.3.1.3 Else

6.9.4.3.1.3.1 if the player’s score minus the score for that throw is at least 50

6.9.4.3.1.3.1.1 call set\_current\_score to update the player’s score

6.9.4.4 If the target is 20

6.9.4.4.1 Call single\_throw to throw for single 20

6.9.4.4.1 if the throw score = 20

6.9.4.4.1.1 call set\_successful\_hits to increment the player’s successful hits by one

6.9.4.4.1 else

6.9.4.4.2 if the player’s score minus the throw score is at least 50

6.9.4.4.2.1 call set\_current\_score to update the player’s score

6.9.4.5 else

6.9.4.5.1 throw for the target

6.9.4.5.2 if the target is hit

6.9.4.5.2.1 call set\_successful\_hits to increment the player’s successful hits by one

6.9.4.5.2.1 call set\_current\_score to update the player’s score

6.9.4.5.2 if the target is not hit and the score will remain at least 50

6.9.4.5.2.1 call set\_current\_score to update the player’s score

6.9.4.2 break

6.9.5 Increment playOrder by one

6.9.5.1 if playOrder = 3

6.9.5.1.1 set playOrder to 1

6.9.6 if player one’s score is 0

6.9.6.1 call set\_game\_wins and increment player one’s game wins by one

6.9.6.2 call set\_current\_score to reset both players back to 301

6.9.7 if player two’s score is 0

6.9.7.1 call set\_game\_wins and increment player two’s game wins by one

6.9.7.2 call set\_current\_score to reset both players back to 301

6.9.8 call set\_actual\_success\_rate and calculate the actual success rate for both players based on their scores

6.9.9 call set\_avgSuccess and calculate the average number of successful throws each player had per turn

6.9.9.1 output the actual success rate and average number of successful throws

7 If gameDecision = 2 the user has chosen 501

7.1 call the constructor to initialise two objects of the player class

7.2 Set the players’ accuracies

7.3 Initialise an object of the Board class

7.4 Initialise the variable to store the number of games to simulate

7.5 declare the variable to hold the score of a match and the array to store the scores for each match

7.6 declare the variables to hold the scores for throwing at the bull and throw for the bull

7.7 if player one throws closer to the bull

7.7.1 set playOrder to 1

7.8 if player two throws closer to the bull

7.8.1 set playOrder to 2

7.9 loop for the number of games to simulate

7.9.1 set each player’s set wins to 0

7.9.2 set each player’s current score to 501

7.9.3 loop until either player reaches 7 set wins

7.9.3.1 switch statement for playOrder

7.9.3.2 if case is 1

7.9.3.2.1 if score is less than 180 and not equal to 169, 168, 166, 165, 163, 162, 159

7.9.3.2.1.1 if the score is over 170

7.9.3.2.1.1.1 loop three times

7.9.3.2.1.1.1.1 throw for treble 15

7.9.3.2.1.1.1.1.1 if the player’s score – the throw score

7.9.3.2.1.1.1.1.1.1 call set\_current\_score to update the score

7.9.3.2.1.2 if the score is below 40 and is even

7.9.3.2.1.2.1 throw for a double to finish

7.9.3.2.1.2.2 increment total throws by one

7.9.3.2.1.2.3 if the target is hit

7.9.3.2.1.2.3.1 call set\_current\_score to set the score to 0

7.9.3.1.2.4 if the target is not hit but the score will be at least 2

7.9.3.1.2.4.1 call set\_current\_score to update the score

7.9.3.2.1.3 else

7.9.3.2.1.3.1 call checkout to initialise remainingScore to a string containing the required steps to checkout

7.9.3.2.1.3.2 initialise commaPos to the position of the first comma in remainingScore

7.9.3.2.1.3.3 loop for the length of remainingScore

7.9.3.2.1.3.3.1 if there is a comma at the current index

7.9.3.2.1.3.3.1.1 increment numCommas by one

7.9.3.2.1.3.4 if there are 2 commas found

7.9.3.2.1.3.4.1 initialise variable throw1 to the first step of the checkout

7.9.3.2.1.3.4.2 initialise variable throw1Hit and throw2Hit as false

7.9.3.2.1.3.4.3 throw for the treble value throw1

7.9.3.2.1.3.4.4 if the current score – the throw score is at least 2

7.9.3.2.1.3.4.4.1 call set\_current\_score to update the player’s score

7.9.3.2.1.3.4.5 if the target is hit

7.9.3.2.1.3.4.5.1 set throw1Hit to true

7.9.3.2.1.3.4.6 if throw1Hit is true

7.9.3.2.1.3.4.6.1 remove the first part of the checkout from remainingScore

7.9.3.2.1.3.4.6.2 set commaPos to the position of the next comma in the string

7.9.3.2.1.3.4.6.3 set throw2 to the second part of the checkout

7.9.3.2.1.3.4.6.4 If the first character of throw2 is a “T”

7.9.3.2.1.3.4.6.4.1 throw for the treble value of throw2

7.9.3.2.1.3.4.6.4.2 call set\_total\_throws to increment score by one

7.9.3.2.1.3.4.6.4.3 if the player’s current score – the throw score is at least 2

7.9.3.2.1.3.4.6.4.3.1 call set\_current\_score to update the player’s score

7.9.3.2.1.3.4.6.4.3 if the target is hit

7.9.3.2.1.3.4.6.4.3.1 set throw2Hit to true

7.9.3.2.1.3.4.6.5 if the first character of throw2 is a “D”

7.9.3.2.1.3.4.6.5.1 throw for the double value of throw2

7.9.3.2.1.3.4.6.6 call set\_total \_throws to increment the total throws by one

7.9.3.2.1.3.4.6.7 if the player’s score – the throw score is at least 2

7.9.3.2.1.3.4.6.7.1 call set\_current\_score to update the player’s score

7.9.3.2.1.3.4.6.7.2 If the target is hit

7.9.3.2.1.3.4.6.7.2.1 set throw2Hit to true

7.9.4.3.1.3.4.6.6 else

7.9.4.3.1.3.4.6.6.1 throw for the value of throw2

7.9.4.3.1.3.4.6.6.2 call set\_total\_throws to increment the total throws by one

7.9.4.3.1.3.4.6.6.3 if the player’s current score – the throw score is at least 2

7.9.4.3.1.3.4.6.6.3.1 call set\_current\_score to update the score

7.9.4.3.1.3.4.6.6.4 if the target was hit

7.9.4.3.1.3.4.6.6.4.1 set throw2Hit to true

7.9.4.3.1.3.4.6.7 if throw2Hit is true

7.9.4.3.1.3.4.6.7.1 set remainingScore to the final step of the checkout

7.9.4.3.1.3.4.6.7.2 initalise throw3 to remainingScore

7.9.4.3.1.3.4.6.7.3 increment total throws by one

7.9.4.3.1.3.4.6.7.4 if score required is 50

7.9.4.3.1.3.4.6.7.5 throw for the bull

7.9.4.3.1.3.4.6.7.5 else

7.9.4.3.1.3.4.6.7.5.1 throw for the double value of throw3

7.9.4.3.1.4.6.7.6 if the required score was 50

7.9.4.3.1.4.6.7.6.1 if the throw score was 50

7.9.4.3.1.4.6.7.6.1.1 set the score to 0

7.9.4.3.1.4.6.7.6.2 else

7.9.4.3.1.4.6.7.2.1 if the target was hit

7.9.4.3.1.4.6.7.2.1.1 set the score to 0

7.9.4.3.1.4.6.7.6.3 if the target was not hit

7.9.4.3.1.4.6.7.6.4 if the current score – the throw score is at least 2

7.9.4.3.1.4.6.7.4.1 update the score

7.9.4.3.1.3.4.6.8 if throw2Hit is false

7.9.4.3.1.3.4.6.8.1 if the current score is less than 40 and even

7.9.4.3.1.3.4.6.8.1.1 throw for a double to attempt to finish

7.9.4.3.1.3.4.6.8.1.2 increment total throws by one

7.9.4.3.1.3.4.6.8.1.3 if the target is hit

7.9.4.3.1.3.4.6.8.1.3.1 set the player’s score to 0

7.9.4.3.1.3.4.6.8.2 if the current score is odd

7.9.4.3.1.3.4.6.8.2.1 throw for a single one to make the score even

7.9.4.3.1.3.4.6.8.3 if the current score is greater than or equal to 62

7.9.4.3.1.3.4.6.8.3.1 throw for treble 20

7.9.4.3.1.3.4.6.8.4 else if the score is greater than 40

7.9.4.3.1.3.4.6.8.5 throw for single 20

7.9.4.3.1.3.4.6.8.5 if the player’s score – the throw score is at least 2

7.4.3.1.3.4.6.8.5.1 update the score

7.9.3.2.1.3.4.7 if throw1Hit is false

7.9.3.2.1.3.4.7.1 set remainingScore to the output of the alternateCheckout function

7.9.3.2.1.3.4.7.2 if remainingScore is empty

7.9.3.2.1.3.4.7.2.1 loop twice

7.9.3.2.1.3.4.7.2.1.1 throw for single 20

7.9.3.2.1.3.4.7.2.1.2 if the player’s current score – the throw score is at least 2

7.9.3.2.1.3.4.7.2.1.2.1 update the score

7.9.3.2.1.3.4.7.3 set commaPos to the position of the first comma found in remainingScore

7.9.3.2.1.3.4.7.4 set throw1 to the first step of the new checkout, ignoring the first character

7.9.3.2.1.3.4.8 remove the first part of the checkout from remainingScore

7.9.3.2.1.3.4.8.1 if the first part of throw1 is a “T”

7.9.3.2.1.3.4.8.1.1 throw for the treble value of throw1

7.9.3.2.1.3.4.8.1.2 if the player’s current score – the throw score is at least 2

7.9.3.2.1.3.4.8.1.2.1 update the score

7.9.3.2.1.3.4.8.1.3 if the target is hit

7.9.3.2.1.3.4.8.1.3.1 set throw1Hit to true

7.9.3.2.1.3.4.8.2 if the first character of remainingScore is a “D”

7.9.3.2.1.3.4.8.2.1 increment the total throws by one

7.9.3.2.1.3.4.8.2.2 throw for the double value of throw1

7.9.3.2.1.3.4.8.2.3 if the current score – the throw score is at least 2

7.9.3.2.1.3.4.8.2.3.1 update the current score

7.9.3.2.1.3.4.8.2.4 if the target was hit

7.9.3.2.3.4.8.2.4.1 set throw1Hit to true

7.9.3.2.1.3.4.8.3 if throw1Hit is true

7.9.3.2.1.3.4.8.3.1 set throw2 to the remaining score / 2

7.9.3.2.1.3.4.8.3.2 increment total throws by one

7.9.3.2.1.3.4.8.3.3 throw for a double to checkout

7.9.3.2.1.3.4.8.3.4 if the target was hit

7.9.3.2.1.3.4.8.3.4.1 set score to 0

7.9.3.2.1.3.4.8.3.5 else if the current score – the throw score is at least 2

7.9.3.2.1.3.4.8.3.5.1 update the score

7.9.3.2.1.3.4.8.4 else

7.9.3.2.1.3.4.8.4.1 set i to 20, loop until i reaches 0

7.9.3.2.1.3.4.8.4.1.1 if the current score – i is at least 2

7.9.3.2.1.3.4.8.4.1.1.1 subtract i from the current score

7.9.3.2.1.3.4.8.4.1.1.2 break

7.9.3.2.1.3.5 if there is 1 comma found

7.9.3.2.1.3.5.1 if the first character of remainingScore is a “T” or “D”

7.9.3.2.1.3.5.1.1 set throw1 to the first part of the checkout, ignoring the first character

7.9.3.2.1.3.5.1.2 set throw1Hit and throw2Hit to false

7.9.3.2.1.3.5.1.3 if the first character of remainingScore is a “T”

7.9.3.2.1.3.5 throw for the treble value of throw1

7.9.3.2.1.3.6 increment total throws by one

7.9.3.2.1.3.7 if the current score – the throw score is at least 2

7.9.3.2.1.3.7.1 update the score

7.9.3.2.1.3.8 if the target is hit

7.9.3.2.1.3.8.1 set throwHit1 to true

7.9.3.2.1.3.5.1.4 if the first character is a “D”

7.9.3.2.1.3.5.1.4.1 throw for the double value of throw1

7.9.3.2.1.3.5.1.4.2 if the current score – the throw score is at least 2

7.9.3.2.1.3.5.1.4.2.1 update the score

7.9.3.2.1.3.5.1.4.3 if the target is hit

7.9.3.2.1.3.5.1.4.3.1 set throw1Hit to true

7.9.3.2.1.3.5.1.5 else

7.9.3.2.1.3.5.1.5.1 throw for the value of throw1

7.9.3.2.1.4.5.1.5.1.2 if the current score – the throw score is at least 2

7.9.3.2.1.4.5.1.5.1.2.1 update the score

7.9.3.2.1.4.5.1.5.2 if the target is hit

7.9.3.2.1.4.5.1.5.2.1 set throw1Hit to true

7.9.3.2.1.3.5.1.6 if throw1Hit is true

7.9.3.2.1.3.5.1.6.1 set throw2 to the remaining score left

7.9.3.2.1.3.5.1.6.2 throw for a double to attempt to checkout

7.9.3.2.1.3.4.1.6.3 if the target is hit

7.9.3.2.1.3.4.1.6.3.1 set the player’s score to 0

7.9.3.2.1.3.4.1.6.4 else if the player score – the throw score is at least 2

7..9.3.2.1.3.4.1.6.4.1 update the score

7.9.3.2.1.3.5.2 else

7.9.3.2.1.3.5.2.1 set throw1 to the first step of the required checkout

7.9.3.2.1.3.5.2.2 throw for the value of throw1

7.9.3.2.1.3.5.2.2.1 if the current score – the throw score is at least 2

7.9.3.2.1.3.4.2.2.1.1 update the score

7.9.3.2.1.3.5.2.3 if the target is hit

7.9.3.2.1.3.5.2.3 set throw1Hit to true

7.9.3.2.1.3.5.2.4 if throw1Hit is true

7.9.3.2.1.3.5.2.4.1 set throw2 to the final step of the checkout

7.9.3.2.1.3.5.2.4.2 throw for a double to attempt to checkout

7.9.3.2.1.3.5.2.4.2 if the target is hit

7.9.3.2.1.3.5.2.4.2.1 set score to 0

7.9.3.2.1.3.5.2.4.3 if the current score – the throw score is at least 2

7.9.3.2.1.3.5.2.4.3.1 update the score

7.9.3.2.1.3.6 if there are no commas found

7.9.3.2.1.3.6.1 throw for a double to attempt to checkout

7.9.3.2.1.3.6.2 if the target is hit

7.9.3.2.1.3.6.2 set score to 0

7.9.3.2.1.3.6.3 else if the current score – the throw score is at least 2

7.9.3.2.1.3.6.3.1 update the score

7.9.3.2.2 else

7.9.3.2.2 loop 3 time

7.9.3.2.2.1 throw for treble 20

7.9.3.2.2.2 if the current score – the throw score is at least 2

7.9.3.2.2.2.1 update the score

7.9.3.3 if case is 2

7.9.3.3.1 if score is less than 180 and not equal to 169, 168, 166, 165, 163, 162, 159

7.9.3.3.1.1 if the score is over 170

7.9.3.3.1.1.1 loop three times

7.9.3.3.1.1.1.1 throw for treble 15

7.9.3.3.1.1.1.1.1 if the player’s score – the throw score

7.9.3.3.1.1.1.1.1.1 call set\_current\_score to update the score

7.9.3.3.1.2 if the score is below 40 and is even

7.9.3.3.1.2.1 throw for a double to finish

7.9.3.3.1.2.2 increment total throws by one

7.9.3.3.1.2.3 if the target is hit

7.9.3.3.1.2.3.1 call set\_current\_score to set the score to 0

7.9.3.1.2.4 if the target is not hit but the score will be at least 2

7.9.3.1.2.4.1 call set\_current\_score to update the score

7.9.3.3.1.3 else

7.9.3.3.1.3.1 call checkout to initialise remainingScore to a string containing the required steps to checkout

7.9.3.3.1.3.2 initialise commaPos to the position of the first comma in remainingScore

7.9.3.3.1.3.3 loop for the length of remainingScore

7.9.3.3.1.3.3.1 if there is a comma at the current index

7.9.3.3.1.3.3.1.1 increment numCommas by one

7.9.3.3.1.3.4 if there are 2 commas found

7.9.3.3.1.3.4.1 initialise variable throw1 to the first step of the checkout

7.9.3.3.1.3.4.2 initialise variable throw1Hit and throw2Hit as false

7.9.3.3.1.3.4.3 throw for the treble value throw1

7.9.3.3.1.3.4.4 if the current score – the throw score is at least 2

7.9.3.3.1.3.4.4.1 call set\_current\_score to update the player’s score

7.9.3.3.1.3.4.5 if the target is hit

7.9.3.3.1.3.4.5.1 set throw1Hit to true

7.9.3.3.1.3.4.6 if throw1Hit is true

7.9.3.3.1.3.4.6.1 remove the first part of the checkout from remainingScore

7.9.3.3.1.3.4.6.2 set commaPos to the position of the next comma in the string

7.9.3.3.1.3.4.6.3 set throw2 to the second part of the checkout

7.9.3.3.1.3.4.6.4 If the first character of throw2 is a “T”

7.9.3.3.1.3.4.6.4.1 throw for the treble value of throw2

7.9.3.3.1.3.4.6.4.2 call set\_total\_throws to increment score by one

7.9.3.3.1.3.4.6.4.3 if the player’s current score – the throw score is at least 2

7.9.3.3.1.3.4.6.4.3.1 call set\_current\_score to update the player’s score

7.9.3.3.1.3.4.6.4.3 if the target is hit

7.9.3.3.1.3.4.6.4.3.1 set throw2Hit to true

7.9.3.3.1.3.4.6.5 if the first character of throw2 is a “D”

7.9.3.3.1.3.4.6.5.1 throw for the double value of throw2

7.9.3.3.1.3.4.6.6 call set\_total \_throws to increment the total throws by one

7.9.3.3.1.3.4.6.7 if the player’s score – the throw score is at least 2

7.9.3.3.1.3.4.6.7.1 call set\_current\_score to update the player’s score

7.9.3.3.1.3.4.6.7.2 If the target is hit

7.9.3.3.1.3.4.6.7.2.1 set throw2Hit to true

7.9.4.3.1.3.4.6.6 else

7.9.4.3.1.3.4.6.6.1 throw for the value of throw2

7.9.4.3.1.3.4.6.6.2 call set\_total\_throws to increment the total throws by one

7.9.4.3.1.3.4.6.6.3 if the player’s current score – the throw score is at least 2

7.9.4.3.1.3.4.6.6.3.1 call set\_current\_score to update the score

7.9.4.3.1.3.4.6.6.4 if the target was hit

7.9.4.3.1.3.4.6.6.4.1 set throw2Hit to true

7.9.4.3.1.3.4.6.7 if throw2Hit is true

7.9.4.3.1.3.4.6.7.1 set remainingScore to the final step of the checkout

7.9.4.3.1.3.4.6.7.2 initalise throw3 to remainingScore

7.9.4.3.1.3.4.6.7.3 increment total throws by one

7.9.4.3.1.3.4.6.7.4 if score required is 50

7.9.4.3.1.3.4.6.7.5 throw for the bull

7.9.4.3.1.3.4.6.7.5 else

7.9.4.3.1.3.4.6.7.5.1 throw for the double value of throw3

7.9.4.3.1.4.6.7.6 if the required score was 50

7.9.4.3.1.4.6.7.6.1 if the throw score was 50

7.9.4.3.1.4.6.7.6.1.1 set the score to 0

7.9.4.3.1.4.6.7.6.2 else

7.9.4.3.1.4.6.7.2.1 if the target was hit

7.9.4.3.1.4.6.7.2.1.1 set the score to 0

7.9.4.3.1.4.6.7.6.3 if the target was not hit

7.9.4.3.1.4.6.7.6.4 if the current score – the throw score is at least 2

7.9.4.3.1.4.6.7.4.1 update the score

7.9.4.3.1.3.4.6.8 if throw2Hit is false

7.9.4.3.1.3.4.6.8.1 if the current score is less than 40 and even

7.9.4.3.1.3.4.6.8.1.1 throw for a double to attempt to finish

7.9.4.3.1.3.4.6.8.1.2 increment total throws by one

7.9.4.3.1.3.4.6.8.1.3 if the target is hit

7.9.4.3.1.3.4.6.8.1.3.1 set the player’s score to 0

7.9.4.3.1.3.4.6.8.2 if the current score is odd

7.9.4.3.1.3.4.6.8.2.1 throw for a single one to make the score even

7.9.4.3.1.3.4.6.8.3 if the current score is greater than or equal to 62

7.9.4.3.1.3.4.6.8.3.1 throw for treble 20

7.9.4.3.1.3.4.6.8.4 else if the score is greater than 40

7.9.4.3.1.3.4.6.8.5 throw for single 20

7.9.4.3.1.3.4.6.8.5 if the player’s score – the throw score is at least 2

7.4.3.1.3.4.6.8.5.1 update the score

7.9.3.3.1.3.4.7 if throw1Hit is false

7.9.3.3.1.3.4.7.1 set remainingScore to the output of the alternateCheckout function

7.9.3.3.1.3.4.7.2 if remainingScore is empty

7.9.3.3.1.3.4.7.2.1 loop twice

7.9.3.3.1.3.4.7.2.1.1 throw for single 20

7.9.3.3.1.3.4.7.2.1.2 if the player’s current score – the throw score is at least 2

7.9.3.3.1.3.4.7.2.1.2.1 update the score

7.9.3.3.1.3.4.7.3 set commaPos to the position of the first comma found in remainingScore

7.9.3.3.1.3.4.7.4 set throw1 to the first step of the new checkout, ignoring the first character

7.9.3.3.1.3.4.8 remove the first part of the checkout from remainingScore

7.9.3.3.1.3.4.8.1 if the first part of throw1 is a “T”

7.9.3.3.1.3.4.8.1.1 throw for the treble value of throw1

7.9.3.3.1.3.4.8.1.2 if the player’s current score – the throw score is at least 2

7.9.3.3.1.3.4.8.1.2.1 update the score

7.9.3.3.1.3.4.8.1.3 if the target is hit

7.9.3.3.1.3.4.8.1.3.1 set throw1Hit to true

7.9.3.3.1.3.4.8.2 if the first character of remainingScore is a “D”

7.9.3.3.1.3.4.8.2.1 increment the total throws by one

7.9.3.3.1.3.4.8.2.2 throw for the double value of throw1

7.9.3.3.1.3.4.8.2.3 if the current score – the throw score is at least 2

7.9.3.3.1.3.4.8.2.3.1 update the current score

7.9.3.3.1.3.4.8.2.4 if the target was hit

7.9.3.3.3.4.8.2.4.1 set throw1Hit to true

7.9.3.3.1.3.4.8.3 if throw1Hit is true

7.9.3.3.1.3.4.8.3.1 set throw2 to the remaining score / 2

7.9.3.3.1.3.4.8.3.2 increment total throws by one

7.9.3.3.1.3.4.8.3.3 throw for a double to checkout

7.9.3.3.1.3.4.8.3.4 if the target was hit

7.9.3.3.1.3.4.8.3.4.1 set score to 0

7.9.3.3.1.3.4.8.3.5 else if the current score – the throw score is at least 2

7.9.3.3.1.3.4.8.3.5.1 update the score

7.9.3.3.1.3.4.8.4 else

7.9.3.3.1.3.4.8.4.1 set i to 20, loop until i reaches 0

7.9.3.3.1.3.4.8.4.1.1 if the current score – i is at least 2

7.9.3.3.1.3.4.8.4.1.1.1 subtract i from the current score

7.9.3.3.1.3.4.8.4.1.1.2 break

7.9.3.3.1.3.5 if there is 1 comma found

7.9.3.3.1.3.5.1 if the first character of remainingScore is a “T” or “D”

7.9.3.3.1.3.5.1.1 set throw1 to the first part of the checkout, ignoring the first character

7.9.3.3.1.3.5.1.2 set throw1Hit and throw2Hit to false

7.9.3.3.1.3.5.1.3 if the first character of remainingScore is a “T”

7.9.3.3.1.3.5 throw for the treble value of throw1

7.9.3.3.1.3.6 increment total throws by one

7.9.3.3.1.3.7 if the current score – the throw score is at least 2

7.9.3.3.1.3.7.1 update the score

7.9.3.3.1.3.8 if the target is hit

7.9.3.3.1.3.8.1 set throwHit1 to true

7.9.3.3.1.3.5.1.4 if the first character is a “D”

7.9.3.3.1.3.5.1.4.1 throw for the double value of throw1

7.9.3.3.1.3.5.1.4.2 if the current score – the throw score is at least 2

7.9.3.3.1.3.5.1.4.2.1 update the score

7.9.3.3.1.3.5.1.4.3 if the target is hit

7.9.3.3.1.3.5.1.4.3.1 set throw1Hit to true

7.9.3.3.1.3.5.1.5 else

7.9.3.3.1.3.5.1.5.1 throw for the value of throw1

7.9.3.3.1.4.5.1.5.1.2 if the current score – the throw score is at least 2

7.9.3.3.1.4.5.1.5.1.2.1 update the score

7.9.3.3.1.4.5.1.5.2 if the target is hit

7.9.3.3.1.4.5.1.5.2.1 set throw1Hit to true

7.9.3.3.1.3.5.1.6 if throw1Hit is true

7.9.3.3.1.3.5.1.6.1 set throw2 to the remaining score left

7.9.3.3.1.3.5.1.6.2 throw for a double to attempt to checkout

7.9.3.3.1.3.4.1.6.3 if the target is hit

7.9.3.3.1.3.4.1.6.3.1 set the player’s score to 0

7.9.3.3.1.3.4.1.6.4 else if the player score – the throw score is at least 2

7.9.3.2.1.3.4.1.6.4.1 update the score

7.9.3.3.1.3.5.2 else

7.9.3.3.1.3.5.2.1 set throw1 to the first step of the required checkout

7.9.3.3.1.3.5.2.2 throw for the value of throw1

7.9.3.3.1.3.5.2.2.1 if the current score – the throw score is at least 2

7.9.3.3.1.3.4.2.2.1.1 update the score

7.9.3.3.1.3.5.2.3 if the target is hit

7.9.3.3.1.3.5.2.3.1 set throw1Hit to true

7.9.3.3.1.3.5.2.4 if throw1Hit is true

7.9.3.3.1.3.5.2.4.1 set throw2 to the final step of the checkout

7.9.3.3.1.3.5.2.4.2 throw for a double to attempt to checkout

7.9.3.3.1.3.5.2.4.3 if the target is hit

7.9.3.3.1.3.5.2.4.3.1 set score to 0

7.9.3.3.1.3.5.2.4.3 if the current score – the throw score is at least 2

7.9.3.3.1.3.5.2.4.3.1 update the score

7.9.3.3.1.3.6 if there are no commas found

7.9.3.3.1.3.6.1 throw for a double to attempt to checkout

7.9.3.3.1.3.6.2 if the target is hit

7.9.3.3.1.3.6.3 set score to 0

7.9.3.3.1.3.6.4 else if the current score – the throw score is at least 2

7.9.3.3.1.3.6.4.1 update the score

7.9.3.3.2 else

7.9.3.3.2.1 loop 3 time

7.9.3.3.2.2 throw for treble 20

7.9.3.3.2.3 if the current score – the throw score is at least 2

7.9.3.3.2.3.1 update the score

7.9.3.2 increment playOrder by one

7.9.3.2.1 if playOrder = 3

7.9.3.2.1.1 set playOrder to 1

7.9.3.3 if player one reaches 0

7.9.3.3.1 increment player one’s game wins by one

7.9.3.3.2 set each player’s score back to 501

7.9.3.4 if player two reaches 9

7.9.3.4.1 increment player two’s game wins by one

7.9.3.4.2 set each player’s score back to 0

7.9.3.5 if player one reaches 3 game wins

7.9.3.5 increment player one’s set wins by one

7.9.3.6 set each player’s game wins to 0

7.9.3.7 set each player’s score to 501

7.9.3.6 if player two reaches 3 game wins

7.9.3.6.1 increment player two’s set wins by one

7.9.3.6.2 set each player’s game wins to 0

7.9.3.6.3 set each player’s score to 501

7.9.4 set score to player one’s set wins concatenated with player two’s set wins

7.9.5 loop for the number of games

7.9.5.1 if the current index of the scores array is empty

7.9.5.1.1 set the current index of the scores array to the current score

8 loop for the number of games

8.1 set variable found to false

8.2 set variable count to 1

8.3 loop for number of games

8.3.1 if current score is in the checked array

8.3.1.1 set found to true

8.3.2 if found is false

8.3.2.1 count the number of occurrences for the current score

8.3.3 calculate and output the frequency of the current score

8.3.3 add the current score to the checked array

# UML – please draw your UML below

# 

# A screenshot of a computer code Description automatically generated with low confidence

# Code – please paste the file which contains the main function.

//includes the required libraries

#include <cstdlib>

#include <ctime>

#include <iostream>

#include <iomanip>

#include <string>

#include <Windows.h>

#include "Player.h"

#include "Board.h"

using namespace std;

int main()

{

//initalises the random function

srand(time(0));

//variable to hold the order in which the players throw

int playOrder;

//variable to hold which game mode the player is playing (301 or 501)

int gameDecision;

//asks the user to decide which game to play

std::cout << "Enter 1 for 301 and 2 for 501" << endl;

cin >> gameDecision;

//if gameDecision is equal to 1, the user has chosen 301

if (gameDecision == 1)

{

//constructor for the player objects

Player Player1("Peter Wright", 0, 0, 301);

Player Player2("Gerwyn Price", 0, 0, 301);

//initalises an object of the Board class

Board Dartboard;

//holds the variables for the players' accuracy

int accuracy1;

int accuracy2;

//asks the user to enter the accuracy for each player

std::cout << "Enter the accuracy for " << Player1.get\_name() << endl;

cin >> accuracy1;

std::cout << "Enter the accuracy for " << Player2.get\_name() << endl;

cin >> accuracy2;

//sets the accuracy for each player

Player1.set\_accuracy(accuracy1);

Player2.set\_accuracy(accuracy2);

//sets the successful hits variable for each player to 0

Player1.set\_successful\_hits(0);

Player2.set\_successful\_hits(0);

//stores the number of games the user chooses to simulate

int numGames;

//stores the score from the player's throw

int score;

//allows the user to enter the number of simulations to run

std::cout << "Enter the number of simulations to run " << endl;

cin >> numGames;

//allows the user to decide who goes first

std::cout << "Enter 1 for " << Player1.get\_name() << " to throw first or 2 for " << Player2.get\_name() << " to throw first" << endl;

cin >> playOrder;

//for loop for the number of games the user has chosen to simulate

for (int i = 0; i < numGames; i++)

{

//switch statement for each player

do {

switch (playOrder)

{

//if playOrder = 1

case 1:

if (Player2.get\_current\_score() != 0)

{

//increments player one's total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

//calculates the target

Dartboard.set\_target(Player1.get\_current\_score());

//if the target = 50

if (Dartboard.get\_target() == 50)

{

//throws for the bull

Dartboard.set\_throw\_score(Dartboard.bull\_throw\_301(Player1.get\_accuracy()));

if (Player1.get\_current\_score() == 50)

{

if (Dartboard.get\_throw\_score() == 50)

{

Player1.set\_successful\_hits(Player1.get\_successful\_hits() + 1);

//sets the score to 0

Player1.set\_current\_score(0);

//breaks

break;

}

}

//if the score for that throw is 50

else if (Dartboard.get\_throw\_score() == 50)

{

//increments successful hits by one

Player1.set\_successful\_hits(Player1.get\_successful\_hits() + 1);

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - 50);

}

else

{

//if the player will not go bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 50)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

} //if the target is 20

else if (Dartboard.get\_target() == 20)

{

//throws for single 20

Dartboard.set\_throw\_score(Dartboard.single\_throw(20, Player1.get\_accuracy(), gameDecision));

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 20)

{

//increments successful hits by one

Player1.set\_successful\_hits(Player1.get\_successful\_hits() + 1);

}

//if the dart did not hit its target and the player will not go bust

else if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 50)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

//otherwise

else {

//throws for a single of the calculated target

Dartboard.set\_throw\_score(Dartboard.single\_throw(Dartboard.get\_target(), Player1.get\_accuracy(), gameDecision));

//if the dart hits its target

if (Dartboard.get\_throw\_score() == Dartboard.get\_target())

{

//increments successful hits by one

Player1.set\_successful\_hits(Player1.get\_successful\_hits() + 1);

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart does not hit its target and the player will not go bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 50)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

break;

case 2:

//if the other player has not already won

if (Player1.get\_current\_score() != 0)

{

//increments player two's total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

//calculates the target

Dartboard.set\_target(Player2.get\_current\_score());

//if the target = 50

if (Dartboard.get\_target() == 50)

{

//throws for the bull

Dartboard.set\_throw\_score(Dartboard.bull\_throw\_301(Player2.get\_accuracy()));

if (Player2.get\_current\_score() == 50)

{

if (Dartboard.get\_throw\_score() == 50)

{

Player2.set\_successful\_hits(Player2.get\_successful\_hits() + 1);

//sets the score to 0

Player2.set\_current\_score(0);

//breaks

break;

}

}

//if the score for that throw is 50

else if (Dartboard.get\_throw\_score() == 50)

{

//increments successful hits by one

Player2.set\_successful\_hits(Player2.get\_successful\_hits() + 1);

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - 50);

}

else

{

//if the player will not go bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 50)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

} //if the target is 20

else if (Dartboard.get\_target() == 20)

{

//throws for single 20

Dartboard.set\_throw\_score(Dartboard.single\_throw(20, Player2.get\_accuracy(), gameDecision));

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 20)

{

//increments successful hits by one

Player2.set\_successful\_hits(Player2.get\_successful\_hits() + 1);

}

//if the dart did not hit its target and the player will not go bust

else if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 50)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

//otherwise

else {

//throws for a single of the calculated target

Dartboard.set\_throw\_score(Dartboard.single\_throw(Dartboard.get\_target(), Player2.get\_accuracy(), gameDecision));

//if the dart hits its target

if (Dartboard.get\_throw\_score() == Dartboard.get\_target())

{

//increments successful hits by one

Player2.set\_successful\_hits(Player2.get\_successful\_hits() + 1);

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart does not hit its target and the player will not go bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 50)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

break;

}

//increments playOrder by one to swap to the next player

playOrder++;

//if playOrder = 3

if (playOrder == 3)

{

//resets it to one so that the two players alternate

playOrder = 1;

}

//loops until one of the players reaches 0

} while (Player1.get\_current\_score() > 0 && Player2.get\_current\_score() > 0);

//if player one's score = 0

if (Player1.get\_current\_score() == 0)

{

//increments number of game wins by one

Player1.set\_game\_wins(Player1.get\_game\_wins() + 1);

//resets both players' scores to 301

Player1.set\_current\_score(301);

Player2.set\_current\_score(301);

}

//if player two's score = 0

else {

//increments number of game wins by one

Player2.set\_game\_wins(Player2.get\_game\_wins() + 1);

//resets both players' scores to 301

Player1.set\_current\_score(301);

Player2.set\_current\_score(301);

}

}

//calculates the success rates for both players

Player1.set\_actual\_success\_rate(float(Player1.get\_successful\_hits()) / float(Player1.get\_total\_throws()) \* 100);

Player2.set\_actual\_success\_rate(float(Player2.get\_successful\_hits()) / float(Player2.get\_total\_throws()) \* 100);

//calculates the average number of successful hits per turn

Player1.set\_avgSuccess(float(Player1.get\_successful\_hits() / numGames));

Player2.set\_avgSuccess(float(Player1.get\_successful\_hits() / numGames));

//outputs the results

cout << "It took " << Player1.get\_name() << " an average of " << Player1.get\_avgSuccess() << " successful throws per game to win " << Player1.get\_game\_wins() << " games, with "<< Player1.get\_successful\_hits() <<" successful throws, and an average success rate of " << Player1.get\_actual\_success\_rate() << " % " << endl;

cout << "It took " << Player2.get\_name() << " an average of " << Player2.get\_avgSuccess() << " successful throws per game to win " << Player2.get\_game\_wins() << " games, with " << Player2.get\_successful\_hits() << " successful throws, and an average success rate of " << Player2.get\_actual\_success\_rate() << "%" << endl;

}

//if gameDecision = 2, the user has chosen 501

else if (gameDecision == 2)

{

//initalises the player objects

Player Player1("Peter Wright", 0, 0, 501);

Player Player2("Gerwyn Price", 0, 0, 501);

//sets the player's accuracies

Player1.set\_accuracy(87);

Player2.set\_accuracy(85);

//initalises an object of the Board class

Board Dartboard;

//sets the number of games the user will play

const int numGames = 100;

//sets up the variable that holds the final score for each set

string score;

//sets up the array that holds the scores

string scores[numGames];

//holds the variable for the order in which the players will throw

int playOrder;

//initalises the variable for the first throw at the bull

int bull1;

//initalises the variable for the second throw at the bull

int bull2;

//throws the first throw at the bull

bull1 = Dartboard.bull\_throw(Player1.get\_accuracy());

//throws the second throw at the bull

bull2 = Dartboard.bull\_throw(Player2.get\_accuracy());

//if the first throw is close to 50 than the second throw

if (bull1 > bull2)

{

//player one goes first

playOrder = 1;

}

//if the second throw if closer to 50 than the first throw

else {

//player two goes first

playOrder = 2;

}

//for loop for the number of games to simulate

for (int i = 0; i < numGames; i++)

{

//sets each player's set wins to 0

Player1.set\_set\_wins(0);

Player2.set\_set\_wins(0);

//sets each player's score to 501

Player1.set\_current\_score(501);

Player2.set\_current\_score(501);

//game on

do

{

//switch statement for each player to take their three throws

switch (playOrder)

{

//player one

case 1:

//if the score is less than 180 and not on a score that does not have a finish

if (Player1.get\_current\_score() <= 180 && Player1.get\_current\_score() != 169 && Player1.get\_current\_score() != 168

&& Player1.get\_current\_score() != 166 && Player1.get\_current\_score() != 165 && Player1.get\_current\_score() != 163

&& Player1.get\_current\_score() != 162 && Player1.get\_current\_score() != 159)

{

//if the score is over 170

if (Player1.get\_current\_score() > 170)

{

//for loop for three throws

for (int i = 0; i < 3; i++)

{

//throws for treble 15, as this will get the score down to a relatively easy finish to checkout in as few throws as possible

Dartboard.set\_throw\_score(Dartboard.treble\_throw(15, Player1.get\_accuracy()));

//if the thrown score does not make the player go bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the player's score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

//breaks, as the player has taken their three throws

break;

}

//if the player's score is below 40 and is even

if (Player1.get\_current\_score() <= 40 && Player1.get\_current\_score() % 2 == 0)

{

//throws for a double of the value of their score to attempt to checkout

int target = Player1.get\_current\_score() / 2;

Dartboard.set\_throw\_score(Dartboard.double\_throw(target, Player1.get\_accuracy()));

//increments the total number of throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

//if the dart hits its target

if (Dartboard.get\_throw\_score() == target)

{

//sets the player's score to 0

Player1.set\_current\_score(0);

//breaks

break;

}

//if the dart has not hit its target but the player has not gone bust

else if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the player's score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

//if a two dart finish or three dart finish is possible

else

{

//calls the checkout function to provide the player with a checkout based on their current score

string remainingScore = Dartboard.checkout(Player1.get\_current\_score());

//finds the first comma, as this will be the first shot the player must take

int commaPos = remainingScore.find(",");

//variable for number of commas, as this will determine whow many throws is required for the finish

int numCommas = 0;

//searches for commas to find the total number of commas in the string

for (int i = 0; i < remainingScore.length(); i++)

{

if (remainingScore[i] == ',')

{

numCommas++;

}

}

//three dart finish

if (numCommas == 2)

{

//sets the variable for the first throw in the checkout to the first part of the string, ignoring the first character as this will be a letter to determine if the program should aim for a treble, double, or single

int throw1 = stoi(remainingScore.substr(1, commaPos));

//sets the variables that determine whether the required shots have been hit to false

bool throw1Hit = false;

bool throw2Hit = false;

//the first shot in a three dart finish will always be a treble, so the program throws for a treble

Dartboard.set\_throw\_score(Dartboard.treble\_throw(throw1, Player1.get\_accuracy()));

//increments the total number of throws by one

Player1.set\_total\_throws(1 + (Player1.get\_total\_throws()));

//if the score has not made the player go bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the player's score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 3 \* throw1)

{

//sets the variable to true

throw1Hit = true;

}

//the player can hit the next part of their finish

if (throw1Hit == true)

{

//looks for the next part of the finish in the string

remainingScore = remainingScore.substr(commaPos + 1);

int commaPos = remainingScore.find(",");

//sets throw2 to the next required shot of the finish

string throw2 = remainingScore.substr(0, commaPos);

//if the shot begins with "T"

if (throw2[0] == 'T')

{

//throws for a treble

Dartboard.set\_throw\_score(Dartboard.treble\_throw(stoi(throw2.substr(1)), Player1.get\_accuracy()));

//increments total throws by one

Player1.set\_total\_throws(1 + (Player1.get\_total\_throws()));

//if the player has not gone bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the player's score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 3 \* stoi(throw2.substr(1)))

{

//sets the variable to true

throw2Hit = true;

}

}

//if the shot begins with "D"

else if (throw2[0] == 'D')

{

//throws for a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(stoi(throw2.substr(1)), Player1.get\_accuracy()));

//increments throws by one

Player1.set\_total\_throws(1 + (Player1.get\_total\_throws()));

//if the player has not gone bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 2 \* stoi(throw2.substr(1)))

{

//sets the variable to true

throw2Hit = true;

}

}

//if it does not begin with "T" or "D"

else {

//throws for a single

Dartboard.set\_throw\_score(Dartboard.single\_throw(stoi(throw2), Player1.get\_accuracy(), gameDecision));

//increments total throws by one

Player1.set\_total\_throws(1 + (Player1.get\_total\_throws()));

//if the player has not gone bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == stoi(throw2))

{

//sets the variable to true

throw2Hit = true;

}

}

//if both shots have been hit, the player can attempt to checkout

if (throw2Hit == true)

{

//gets the last part of the checkout

remainingScore = remainingScore.substr(commaPos + 1);

string throw3 = remainingScore;

//increments total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

//if the score required is 50

if (throw3 == "50")

{

//throws for the bull

Dartboard.set\_throw\_score(Dartboard.bull\_throw(Player1.get\_accuracy()));

}

//otherwise

else {

//throws for a double, as the player must finish on a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(stoi(throw3.substr(1)), Player1.get\_accuracy()));

}

//if the score required was 50

if (throw3 == "50")

{

//if the player hit the bull

if (Dartboard.get\_throw\_score() == 50)

{

//sets the score to 0

Player1.set\_current\_score(0);

//breaks

break;

}

}

//if the score was not 50

else {

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 2 \* stoi(throw3.substr(1)))

{

//sets the score to 0

Player1.set\_current\_score(0);

//breaks

break;

}

//if the dart did not hit its target

else {

//if the player did not go bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

}

//the player has missed the second part of their finish but may still be able to finish

else {

//if the player's score is even and below 40

if (Player1.get\_current\_score() % 2 == 0 && Player1.get\_current\_score() <= 40)

{

int target = Player1.get\_current\_score() / 2;

//throws for a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(target, Player1.get\_accuracy()));

//increments total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

//if the dart hits its target

if (Dartboard.get\_throw\_score() == target)

{

//sets the score to 0

Player1.set\_current\_score(0);

//breaks

break;

}

}

//focuses the score

else if (Player1.get\_current\_score() % 2 != 0)

{

//throws for a single 1 to make the score even - enhance this

Dartboard.set\_throw\_score(Dartboard.single\_throw(1, Player1.get\_accuracy(), gameDecision));

//increments the total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

}

//focuses to the lowest possible finish

else if (Player1.get\_current\_score() >= 62)

{

Dartboard.set\_throw\_score(Dartboard.treble\_throw(20, Player1.get\_accuracy()));

//increments total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

}

//if the score is greater than 40

else if (Player1.get\_current\_score() > 40)

{

//throws for single 20

Dartboard.set\_throw\_score(Dartboard.single\_throw(20, Player1.get\_accuracy(), gameDecision));

//increments total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

}

//if the player will not go bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

//the player has missed the first part of their finish, but still may be able to checkout

else

{

//calls the alternateCheckout function to search for an alternative checkout

remainingScore = Dartboard.alternateCheckout(Player1.get\_current\_score());

//if no checkout was found

if (remainingScore == "")

{

//throws twice at the single 20

for (int i = 0; i < 2; i++)

{

Dartboard.set\_throw\_score(Dartboard.single\_throw(20, Player1.get\_accuracy(), gameDecision));

//increments total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

//if the player will not go bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

//if an alternative finish was found

else {

//searches for the commas to split up the checkout

commaPos = remainingScore.find(',');

//sets throw1 to the first part of the checkout

throw1 = stoi(remainingScore.substr(1, commaPos));

remainingScore = remainingScore.substr(commaPos + 1);

//if the first part begins with "T"

if (remainingScore[0] == 'T')

{

//increments total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

//throws for a treble

Dartboard.set\_throw\_score(Dartboard.treble\_throw(throw1, Player1.get\_accuracy()));

//if the player will not go bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 3 \* throw1)

{

//sets throw1Hit to true

throw1Hit = true;

}

}

//if it begins with "D"

else if (remainingScore[0] == 'D')

{

//increments total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

//throws for a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(throw1, Player1.get\_accuracy()));

//if the player will not go bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 2 \* throw1)

{

//sets throw1Hit to true

throw1Hit = true;

}

}

//if the required parts of the checkout were hit

if (throw1Hit == true)

{

//the player can attempt to hit their checkout

//gets the remaining part of the checkout

int throw2 = stoi(remainingScore.substr(1)) / 2;

//increments total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

//throws for a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(throw2, Player1.get\_accuracy()));

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 2 \* throw2)

{

//sets the score to 0

Player1.set\_current\_score(0);

//breaks

break;

}

//if the target was not hit but the player will not go bust

else if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

else {

for (int i = 20; i > 0; i--)

{

//looks for the highest number that can be hit without going bust

if (Player1.get\_current\_score() - i >= 2)

{

Dartboard.set\_throw\_score(Dartboard.single\_throw(i, Player1.get\_accuracy(), gameDecision));

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Dartboard.get\_throw\_score());

}

break;

}

}

}

}

}

}

//two dart finish

else if (numCommas == 1)

{

//if checkout step begins with a "T" or "D"

if (remainingScore[0] == 'T' || remainingScore[1] == 'D')

{

//throw 1 is set to the first part of the checkout, ignoring the first character

int throw1 = stoi(remainingScore.substr(1, commaPos));

//sets the hit variables to false

bool throw1Hit = false;

bool throw2Hit = false;

//if it begins with "T"

if (remainingScore[0] == 'T')

{

//throws for a treble

Dartboard.set\_throw\_score(Dartboard.treble\_throw(throw1, Player1.get\_accuracy()));

//increments total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

//if the player will not go bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 3 \* throw1);

{//sets the variable to true

throw1Hit = true;

}

}

//if it begins with "D"

else if (remainingScore[0] == 'D')

{

//throws for a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(throw1, Player1.get\_accuracy()));

//increments total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 2 \* throw1);

{

//variable is set to true

throw1Hit = true;

}

}

//if it does not begin with T or D

else {

//throws for a single

Dartboard.set\_throw\_score(Dartboard.single\_throw(throw1, Player1.get\_accuracy(), gameDecision));

//increments total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

//if the player will not go bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == throw1);

{

//sets the variable to true

throw1Hit = true;

}

//if throw1hit is true

if (throw1Hit == true)

{

//the player can attemt to checkout

int throw2 = stoi(remainingScore.substr(commaPos + 1));

//throws for a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(throw2 / 2, Player1.get\_accuracy()));

//increments total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

//if the dart hits its target

if (Dartboard.get\_throw\_score() == throw2)

{

//sets the score to 0

Player1.set\_current\_score(0);

//breaks

break;

}

//if the dart does not hit its target but the player will not go bust

else if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

}

else {

//gets the required checkout

int throw1 = stoi(remainingScore.substr(0, commaPos));

//sets the hits variables to false

bool throw1Hit = false;

bool throw2Hit = false;

//throws for a single

Dartboard.set\_throw\_score(Dartboard.single\_throw(throw1, Player1.get\_accuracy(), gameDecision));

//increments total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

//if the player will not go bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == throw1);

{

//sets the variable to true

throw1Hit = true;

}

//if the variable is true

if (throw1Hit == true)

{

//gets the final part of the checkout

int throw2 = stoi(remainingScore.substr(commaPos + 2));

//throw fors a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(throw2 / 2, Player1.get\_accuracy()));

//increments the total throws by one

Player1.set\_total\_throws(Player1.get\_total\_throws() + 1);

//if the dart hits its target

if (Dartboard.get\_throw\_score() == throw2)

{

//sets the score to 0

Player1.set\_current\_score(0);

//breaks

break;

}

//if the dart does not hit its target but the player will not go bust

else if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

}

//one dart finish

else if (numCommas == 0)

{

//throws for a double to finish

int target = Player1.get\_current\_score() / 2;

Dartboard.set\_throw\_score(Dartboard.double\_throw(target, Player1.get\_accuracy()));

//if the dart hits its target

if (Dartboard.get\_throw\_score() == target)

{

//sets the score to 0

Player1.set\_current\_score(0);

//breaks

break;

}

//if the dart does not hit its target but the player will not go bust

else if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

}

//if the player's score is above 180

else {

//for loop for 3 loops

for (int i = 0; i < 3; i++)

{

//throws for treble 20

Dartboard.set\_throw\_score(Dartboard.treble\_throw(20, Player1.get\_accuracy()));

//increments total throws by one

Player1.set\_total\_throws(1 + (Player1.get\_total\_throws()));

//if the player will not go bust

if (Player1.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player1.set\_current\_score(Player1.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

break;

case 2:

//player two

//if the score is less than 180 and not on a score that does not have a finish

if (Player2.get\_current\_score() <= 180 && Player2.get\_current\_score() != 169 && Player2.get\_current\_score() != 168

&& Player2.get\_current\_score() != 166 && Player2.get\_current\_score() != 165 && Player2.get\_current\_score() != 163

&& Player2.get\_current\_score() != 162 && Player2.get\_current\_score() != 159)

{

//if the score is over 170

if (Player2.get\_current\_score() > 170)

{

//for loop for three throws

for (int i = 0; i < 3; i++)

{

//throws for treble 15, as this will get the score down to a relatively easy finish to checkout in as few throws as possible

Dartboard.set\_throw\_score(Dartboard.treble\_throw(15, Player2.get\_accuracy()));

//if the thrown score does not make the player go bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the player's score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

//breaks, as the player has taken their three throws

break;

}

//if the player's score is below 40 and is even

if (Player2.get\_current\_score() <= 40 && Player2.get\_current\_score() % 2 == 0)

{

//throws for a double of the value of their score to attempt to checkout

int target = Player2.get\_current\_score() / 2;

Dartboard.set\_throw\_score(Dartboard.double\_throw(target, Player2.get\_accuracy()));

//increments the total number of throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

//if the dart hits its target

if (Dartboard.get\_throw\_score() == target)

{

//sets the player's score to 0

Player2.set\_current\_score(0);

//breaks

break;

}

//if the dart has not hit its target but the player has not gone bust

else if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the player's score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

//if a two dart finish or three dart finish is possible

else

{

//calls the checkout function to provide the player with a checkout based on their current score

string remainingScore = Dartboard.checkout(Player2.get\_current\_score());

//finds the first comma, as this will be the first shot the player must take

int commaPos = remainingScore.find(",");

//variable for number of commas, as this will determine whow many throws is required for the finish

int numCommas = 0;

//searches for commas to find the total number of commas in the string

for (int i = 0; i < remainingScore.length(); i++)

{

if (remainingScore[i] == ',')

{

numCommas++;

}

}

//three dart finish

if (numCommas == 2)

{

//sets the variable for the first throw in the checkout to the first part of the string, ignoring the first character as this will be a letter to determine if the program should aim for a treble, double, or single

int throw1 = stoi(remainingScore.substr(1, commaPos));

//sets the variables that determine whether the required shots have been hit to false

bool throw1Hit = false;

bool throw2Hit = false;

//the first shot in a three dart finish will always be a treble, so the program throws for a treble

Dartboard.set\_throw\_score(Dartboard.treble\_throw(throw1, Player2.get\_accuracy()));

//increments the total number of throws by one

Player2.set\_total\_throws(1 + (Player2.get\_total\_throws()));

//if the score has not made the player go bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the player's score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 3 \* throw1)

{

//sets the variable to true

throw1Hit = true;

}

//the player can hit the next part of their finish

if (throw1Hit == true)

{

//looks for the next part of the finish in the string

remainingScore = remainingScore.substr(commaPos + 1);

int commaPos = remainingScore.find(",");

//sets throw2 to the next required shot of the finish

string throw2 = remainingScore.substr(0, commaPos);

//if the shot begins with "T"

if (throw2[0] == 'T')

{

//throws for a treble

Dartboard.set\_throw\_score(Dartboard.treble\_throw(stoi(throw2.substr(1)), Player2.get\_accuracy()));

//increments total throws by one

Player2.set\_total\_throws(1 + (Player2.get\_total\_throws()));

//if the player has not gone bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the player's score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 3 \* stoi(throw2.substr(1)))

{

//sets the variable to true

throw2Hit = true;

}

}

//if the shot begins with "D"

else if (throw2[0] == 'D')

{

//throws for a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(stoi(throw2.substr(1)), Player2.get\_accuracy()));

//increments throws by one

Player2.set\_total\_throws(1 + (Player2.get\_total\_throws()));

//if the player has not gone bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 2 \* stoi(throw2.substr(1)))

{

//sets the variable to true

throw2Hit = true;

}

}

//if it does not begin with "T" or "D"

else {

//throws for a single

Dartboard.set\_throw\_score(Dartboard.single\_throw(stoi(throw2), Player2.get\_accuracy(), gameDecision));

//increments total throws by one

Player2.set\_total\_throws(1 + (Player2.get\_total\_throws()));

//if the player has not gone bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == stoi(throw2))

{

//sets the variable to true

throw2Hit = true;

}

}

//if both shots have been hit, the player can attempt to checkout

if (throw2Hit == true)

{

//gets the last part of the checkout

remainingScore = remainingScore.substr(commaPos + 1);

string throw3 = remainingScore;

//increments total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

//if the score required is 50

if (throw3 == "50")

{

//throws for the bull

Dartboard.set\_throw\_score(Dartboard.bull\_throw(Player2.get\_accuracy()));

}

//otherwise

else {

//throws for a double, as the player must finish on a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(stoi(throw3.substr(1)), Player2.get\_accuracy()));

}

//if the score required was 50

if (throw3 == "50")

{

//if the player hit the bull

if (Dartboard.get\_throw\_score() == 50)

{

//sets the score to 0

Player2.set\_current\_score(0);

//breaks

break;

}

}

//if the score was not 50

else {

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 2 \* stoi(throw3.substr(1)))

{

//sets the score to 0

Player2.set\_current\_score(0);

//breaks

break;

}

//if the dart did not hit its target

else {

//if the player did not go bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

}

//the player has missed the second part of their finish but may still be able to finish

else {

//if the player's score is even and below 40

if (Player2.get\_current\_score() % 2 == 0 && Player2.get\_current\_score() <= 40)

{

int target = Player2.get\_current\_score() / 2;

//throws for a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(target, Player2.get\_accuracy()));

//increments total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

//if the dart hits its target

if (Dartboard.get\_throw\_score() == target)

{

//sets the score to 0

Player2.set\_current\_score(0);

//breaks

break;

}

}

//focuses the score

else if (Player2.get\_current\_score() % 2 != 0)

{

//throws for a single 1 to make the score even - enhance this

Dartboard.set\_throw\_score(Dartboard.single\_throw(1, Player2.get\_accuracy(), gameDecision));

//increments the total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

}

//focuses to the lowest possible finish

else if (Player2.get\_current\_score() >= 62)

{

Dartboard.set\_throw\_score(Dartboard.treble\_throw(20, Player2.get\_accuracy()));

//increments total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

}

//if the score is greater than 40

else if (Player2.get\_current\_score() > 40)

{

//throws for single 20

Dartboard.set\_throw\_score(Dartboard.single\_throw(20, Player2.get\_accuracy(), gameDecision));

//increments total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

}

//if the player will not go bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

//the player has missed the first part of their finish, but still may be able to checkout

else

{

//calls the alternateCheckout function to search for an alternative checkout

remainingScore = Dartboard.alternateCheckout(Player2.get\_current\_score());

//if no checkout was found

if (remainingScore == "")

{

//throws twice at the single 20

for (int i = 0; i < 2; i++)

{

Dartboard.set\_throw\_score(Dartboard.single\_throw(20, Player2.get\_accuracy(), gameDecision));

//increments total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

//if the player will not go bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

//if an alternative finish was found

else {

//searches for the commas to split up the checkout

commaPos = remainingScore.find(',');

//sets throw1 to the first part of the checkout

throw1 = stoi(remainingScore.substr(1, commaPos));

remainingScore = remainingScore.substr(commaPos + 1);

//if the first part begins with "T"

if (remainingScore[0] == 'T')

{

//increments total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

//throws for a treble

Dartboard.set\_throw\_score(Dartboard.treble\_throw(throw1, Player2.get\_accuracy()));

//if the player will not go bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 3 \* throw1)

{

//sets throw1Hit to true

throw1Hit = true;

}

}

//if it begins with "D"

else if (remainingScore[0] == 'D')

{

//increments total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

//throws for a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(throw1, Player2.get\_accuracy()));

//if the player will not go bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 2 \* throw1)

{

//sets throw1Hit to true

throw1Hit = true;

}

}

//if the required parts of the checkout were hit

if (throw1Hit == true)

{

//the player can attempt to hit their checkout

//gets the remaining part of the checkout

int throw2 = stoi(remainingScore.substr(1)) / 2;

//increments total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

//throws for a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(throw2, Player2.get\_accuracy()));

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 2 \* throw2)

{

//sets the score to 0

Player2.set\_current\_score(0);

//breaks

break;

}

//if the target was not hit but the player will not go bust

else if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

else {

for (int i = 20; i > 0; i--)

{

//looks for the highest number that can be hit without going bust

if (Player2.get\_current\_score() - i >= 2)

{

Dartboard.set\_throw\_score(Dartboard.single\_throw(i, Player2.get\_accuracy(), gameDecision));

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Dartboard.get\_throw\_score());

}

break;

}

}

}

}

}

}

//two dart finish

else if (numCommas == 1)

{

//if checkout step begins with a "T" or "D"

if (remainingScore[0] == 'T' || remainingScore[1] == 'D')

{

//throw 1 is set to the first part of the checkout, ignoring the first character

int throw1 = stoi(remainingScore.substr(1, commaPos));

//sets the hit variables to false

bool throw1Hit = false;

bool throw2Hit = false;

//if it begins with "T"

if (remainingScore[0] == 'T')

{

//throws for a treble

Dartboard.set\_throw\_score(Dartboard.treble\_throw(throw1, Player2.get\_accuracy()));

//increments total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

//if the player will not go bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 3 \* throw1);

{//sets the variable to true

throw1Hit = true;

}

}

//if it begins with "D"

else if (remainingScore[0] == 'D')

{

//throws for a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(throw1, Player2.get\_accuracy()));

//increments total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == 2 \* throw1);

{

//variable is set to true

throw1Hit = true;

}

}

//if it does not begin with T or D

else {

//throws for a single

Dartboard.set\_throw\_score(Dartboard.single\_throw(throw1, Player2.get\_accuracy(), gameDecision));

//increments total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

//if the player will not go bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == throw1);

{

//sets the variable to true

throw1Hit = true;

}

//if throw1hit is true

if (throw1Hit == true)

{

//the player can attemt to checkout

int throw2 = stoi(remainingScore.substr(commaPos + 1));

//throws for a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(throw2 / 2, Player2.get\_accuracy()));

//increments total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

//if the dart hits its target

if (Dartboard.get\_throw\_score() == throw2)

{

//sets the score to 0

Player2.set\_current\_score(0);

//breaks

break;

}

//if the dart does not hit its target but the player will not go bust

else if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

}

else {

//gets the required checkout

int throw1 = stoi(remainingScore.substr(0, commaPos));

//sets the hits variables to false

bool throw1Hit = false;

bool throw2Hit = false;

//throws for a single

Dartboard.set\_throw\_score(Dartboard.single\_throw(throw1, Player2.get\_accuracy(), gameDecision));

//increments total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

//if the player will not go bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

//if the dart hits its target

if (Dartboard.get\_throw\_score() == throw1);

{

//sets the variable to true

throw1Hit = true;

}

//if the variable is true

if (throw1Hit == true)

{

//gets the final part of the checkout

int throw2 = stoi(remainingScore.substr(commaPos + 2));

//throw fors a double

Dartboard.set\_throw\_score(Dartboard.double\_throw(throw2 / 2, Player2.get\_accuracy()));

//increments the total throws by one

Player2.set\_total\_throws(Player2.get\_total\_throws() + 1);

//if the dart hits its target

if (Dartboard.get\_throw\_score() == throw2)

{

//sets the score to 0

Player2.set\_current\_score(0);

//breaks

break;

}

//if the dart does not hit its target but the player will not go bust

else if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

}

//one dart finish

else if (numCommas == 0)

{

//throws for a double to finish

int target = Player2.get\_current\_score() / 2;

Dartboard.set\_throw\_score(Dartboard.double\_throw(target, Player2.get\_accuracy()));

//if the dart hits its target

if (Dartboard.get\_throw\_score() == target)

{

//sets the score to 0

Player2.set\_current\_score(0);

//breaks

break;

}

//if the dart does not hit its target but the player will not go bust

else if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

}

//if the player's score is above 180

else {

//for loop for 3 loops

for (int i = 0; i < 3; i++)

{

//throws for treble 20

Dartboard.set\_throw\_score(Dartboard.treble\_throw(20, Player2.get\_accuracy()));

//increments total throws by one

Player2.set\_total\_throws(1 + (Player2.get\_total\_throws()));

//if the player will not go bust

if (Player2.get\_current\_score() - Dartboard.get\_throw\_score() >= 2)

{

//updates the score

Player2.set\_current\_score(Player2.get\_current\_score() - Dartboard.get\_throw\_score());

}

}

}

break;

}

//increments the playOrder variable to swap between the two players

playOrder++;

//if playOrder = 3;

if (playOrder == 3)

{

//resets playOrder to one to allow the program to alternate between the two players

playOrder = 1;

}

//if player one reaches 0

if (Player1.get\_current\_score() == 0)

{

//increments player one's game wins by one

Player1.set\_game\_wins(Player1.get\_game\_wins() + 1);

//sets both players back to 501

Player1.set\_current\_score(501);

Player2.set\_current\_score(501);

}

//if player two reaches 0

else if (Player2.get\_current\_score() == 0)

{

//increments player two's game wins by one

Player2.set\_game\_wins(Player2.get\_game\_wins() + 1);

//sets both players back to 501

Player1.set\_current\_score(501);

Player2.set\_current\_score(501);

}

//if player one wins 3 games

if (Player1.get\_game\_wins() == 3)

{

//increments player one's set wins by one

Player1.set\_set\_wins(Player1.get\_set\_wins() + 1);

//resets both player's game wins back to 0 and their scores to 501

Player1.set\_game\_wins(0);

Player2.set\_game\_wins(0);

Player1.set\_current\_score(501);

Player2.set\_current\_score(501);

}

//if player two wins 3 games

else if (Player2.get\_game\_wins() == 3)

{

//increments player two's set wins by one

Player2.set\_set\_wins(Player2.get\_set\_wins() + 1);

//resets both player's game wins back to 0 and their scores to 501

Player1.set\_game\_wins(0);

Player2.set\_game\_wins(0);

Player1.set\_current\_score(501);

Player2.set\_current\_score(501);

}

//loops until one of the players reaches 7 set wins

} while (Player1.get\_set\_wins() != 7 && Player2.get\_set\_wins() != 7);

//stores the score of the game by concatanating both scores into a string

score = to\_string(Player1.get\_set\_wins()) + ":" + to\_string(Player2.get\_set\_wins());

//loops for the number of games

for (int i = 0; i < numGames; i++)

{

//searches the scores array until it finds an empty position

if (scores[i] == "")

{

//fills the empty position with the game's score

scores[i] = score;

//breaks

break;

}

}

}

//outputs the results

//variable to hold the number of times each score appeared

float count;

//array to hold the scores that have already been counted

string checked[numGames];

//variable to determine if the score has been counted

bool found;

//loops for the number of games

for (int i = 0; i < numGames; i++)

{

//sets found to false

found = false;

//sets count to 1

count = 1;

//loops through the checked array to see if the current score has already been counted

for (int x = 0; x < numGames; x++)

{

//if the score has been counted

if (checked[x] == scores[i])

{

//sets found to true

found = true;

//breaks

break;

}

}

//if the score has not been counted

if (found == false)

{

//loops through the scores array to count how many times the score appears

for (int j = i + 1; j < numGames; j++)

{

//if the score appears

if (scores[i] == scores[j])

{

//increments count by one

count++;

}

}

//outputs the result

cout << "The frequency of " << scores[i] << " is " << float((count / numGames) \* 100) << "%" << endl;

}

//adds the current score to the checked array

checked[i] = scores[i];

}

}

//ensures that the console window does not close before the results are read

system("pause");

};

# Code – for each class please copy this page and copy in the code. It does not matter if this goes over one page. Make sure the code is easily readable.

**Class Name: Player**

**Player.cpp**

#include "Player.h"

#include <string>

//constructor

Player::Player(string n, int t, int g, int c)

{

name = n;

totalThrows = t;

gameWins = g;

currentScore = c;

}

//destructor

Player::~Player()

{

}

//setters and getters

void Player::set\_name(string s)

{

name = s;

}

string Player::get\_name()

{

return name;

}

void Player::set\_current\_score(int n)

{

Player::currentScore = n;

}

int Player::get\_current\_score()

{

return currentScore;

}

void Player::set\_accuracy(int n)

{

accuracy = n;

}

int Player::get\_accuracy()

{

return accuracy;

}

void Player::set\_total\_throws(int n)

{

totalThrows = n;

}

int Player::get\_total\_throws()

{

return totalThrows;

}

void Player::set\_successful\_hits(int n)

{

successfulHits = n;

}

int Player::get\_successful\_hits()

{

return successfulHits;

}

void Player::set\_actual\_success\_rate(int n)

{

actualSuccessRate = n;

}

int Player::get\_actual\_success\_rate()

{

return actualSuccessRate;

}

void Player::set\_score\_required(int s)

{

scoreRequired = s;

}

int Player::get\_score\_required()

{

return scoreRequired;

}

void Player::set\_game\_wins(int n)

{

gameWins = n;

}

int Player::get\_game\_wins()

{

return gameWins;

}

void Player::set\_set\_wins(int n)

{

setWins = n;

}

int Player::get\_set\_wins()

{

return(setWins);

}

void Player::set\_avgSuccess(int n)

{

avgSuccess = n;

}

int Player::get\_avgSuccess()

{

return avgSuccess;

}

**Player.h**

#pragma once

#include <string>

using namespace std;

class Player

{

private:

//variable to store the player's name

std::string name;

//variable to store the player's score

int currentScore;

//variable to store the required score to checkout

int scoreRequired;

//stores the player's accuracy

int accuracy;

//stores the players's total throws

int totalThrows;

//stores the player's successful hits

int successfulHits;

//stores the player's actual success rate

float actualSuccessRate;

//stores the player's game wins

int gameWins;

//stores the player's set wins

int setWins;

//stores the player's average successful throws

int avgSuccess;

public:

//constuctor

Player(string n, int t, int g, int c);

//destructor

~Player();

//setters and getters

void set\_name(string s);

string get\_name();

void set\_current\_score(int n);

int get\_current\_score();

void set\_accuracy(int n);

int get\_accuracy();

void set\_total\_throws(int n);

int get\_total\_throws();

void set\_successful\_hits(int n);

int get\_successful\_hits();

void set\_actual\_success\_rate(int n);

int get\_actual\_success\_rate();

void set\_score\_required(int n);

int get\_score\_required();

void set\_game\_wins(int n);

int get\_game\_wins();

void set\_set\_wins(int n);

int get\_set\_wins();

void set\_avgSuccess(int n);

int get\_avgSuccess();

};

**Class Name: Board**

**Board.cpp**

**#include "Board.h"**

**#include "Player.h"**

**#include <cstdlib>**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**int Board::single\_throw(int t, int a, int gm)**

**{**

**//2d array with each segment of the board to allow the dart to go left or right of the intended target**

**int board[2][21] = { {0,20,15,17,18,12,13,19,16,14,6,8,9,4,11,10,7,2,1,3,5},**

**{0,18,17,19,13,20,10,16,11,12,15,14,5,6,9,2,8,3,4,7,1} };**

**//generates a random number between 0-100 to compare with the player's accuracy**

**int r = rand() % 100;**

**//if the target is the outer bull**

**if (t == 25)**

**{**

**//if statement executes if the random number is smaller than the player's accuracy**

**if (r < a)**

**{**

**//the outer bull has been hit**

**return 25;**

**}**

**//if the random number is larger than the player's accuracy**

**else**

**{**

**//determines whether the inner bull is hit or the dart misses entirely**

**r = rand() % 2 + 1;**

**if (r == 2)**

**{**

**//the inner bull has been hit**

**return 50;**

**}**

**//if the bull is missed entirely**

**else if (r == 1)**

**{**

**//generates a random number to determine which number the dart hits**

**r = rand() % 20 + 1;**

**int miss = rand() % 3 + 1;**

**//hits the single segment of that numnber**

**if (miss == 1)**

**{**

**return r;**

**}**

**//hits the double segment of that number**

**else if (miss == 2)**

**{**

**return 2 \* r;**

**}**

**//hits the treble segment of that number**

**else {**

**return 3 \* r;**

**}**

**}**

**}**

**}**

**//if the target is not the outer bull**

**else**

**{**

**//if the random number is smaller than the accuracy**

**if (r < a)**

**{**

**//the intended target has been hit**

**return t;**

**}**

**//if the random number is larger than the accuracy**

**else**

**{**

**//if the game mode = 1, the player is playing 301 so the options to hit the outer bull, trebles, and doubles are removed**

**if (gm == 1)**

**{**

**r = rand() % 100 + 1;**

**//45% chance of going left**

**if (r <= 45)**

**{**

**return board[0][t];**

**}**

**//45% chance of going right**

**else if (r > 45 && r <= 90)**

**{**

**return board[1][t];**

**}**

**//10% chance of missing entirely**

**else**

**{**

**return 0;**

**}**

**}**

**r = rand() % 100 + 1;**

**//25% chance of going left**

**if (r <= 25)**

**{**

**return board[0][t];**

**}**

**//25% chance of going right**

**else if (r > 25 && r <= 50)**

**{**

**return board[1][t];**

**}**

**//15% chance of left double**

**else if (r > 50 && r <= 65)**

**{**

**return 2 \* (board[0][t]);**

**}**

**//15% chance of right double**

**else if (r > 65 && r <= 80)**

**{**

**return 2 \* (board[1][t]);**

**}**

**//5% chance of left treble**

**else if (r > 80 && r <= 85)**

**{**

**return 3 \* (board[0][t]);**

**}**

**//5% chance of right treble**

**else if (r > 85 && r <= 90)**

**{**

**return 3 \* (board[1][t]);**

**}**

**//7% chance of missing the board**

**else if (r > 90 && r <= 97)**

**{**

**return 0;**

**}**

**//3% chance of anywhere else on the board**

**else {**

**//generates a random number to decide if the dart hits the bull or another segment**

**r = rand() % 2 + 1;**

**if (r == 1)**

**{**

**//bull**

**r = rand() % 2 + 1;**

**if (r == 1)**

**{**

**//outer bull**

**return 25;**

**}**

**else {**

**//bull**

**return 50;**

**}**

**}**

**//any other segment on the board**

**else {**

**//input validation so that the algorithm does not accidentally return the intended target**

**do {**

**r = rand() % 20 + 1;**

**} while (r != t);**

**int segment = rand() % 3 + 1;**

**//the dart has hit a single segment**

**if (segment == 1)**

**{**

**return r;**

**}**

**//the dart has hit a double segment**

**else if (segment == 2)**

**{**

**return 2 \* r;**

**}**

**//the dart has hit a treble**

**else {**

**return 3 \* r;**

**}**

**}**

**}**

**}**

**}**

**}**

**int Board::double\_throw(int t, int a)**

**{**

**//2d array with each segment of the board to allow the dart to go left or right of the intended target**

**int board[2][21] = { {0,20,15,17,18,12,13,19,16,14,6,8,9,4,11,10,7,2,1,3,5},**

**{0,18,17,19,13,20,10,16,11,12,15,14,5,6,9,2,8,3,4,7,1} };**

**//generates a random number between 0-100 to compare with the player's accuracy**

**int r = rand() % 100;**

**if (r < a)**

**{**

**//the intended target has been hit**

**return 2 \* t;**

**}**

**//if the random number is larger than the accuracy**

**else**

**{**

**r = rand() % 100 + 1;**

**//25% chance of hitting the left double**

**if (r <= 25)**

**{**

**return 2 \* (board[0][t]);**

**}**

**//25% chance of hitting the right double**

**else if (r > 25 && r <= 50)**

**{**

**return 2 \* (board[1][t]);**

**}**

**//15% chance of hitting the left single**

**else if (r > 50 && r <= 65)**

**{**

**return board[0][t];**

**}**

**//15% chance of hitting the right single**

**else if (r > 65 && r <= 80)**

**{**

**return board[1][t];**

**}**

**//5% chance of left treble**

**else if (r > 80 && r <= 85)**

**{**

**return 3 \* (board[0][t]);**

**}**

**//5% chance of right treble**

**else if (r > 85 && r <= 90)**

**{**

**return 3 \* (board[1][t]);**

**}**

**//7% chance of missing the board**

**else if (r > 90 && r <= 97)**

**{**

**return 0;**

**}**

**//3% chance of anywhere else on the board**

**else {**

**//generates a random number to decide if the dart hits the bull or another segment**

**r = rand() % 2 + 1;**

**if (r == 1)**

**{**

**//bull**

**r = rand() % 2 + 1;**

**if (r == 1)**

**{**

**//outer bull**

**return 25;**

**}**

**else {**

**//bull**

**return 50;**

**}**

**}**

**//any other segment on the board**

**else {**

**//input validation so that the algorithm does not accidentally return the intended target**

**do {**

**r = rand() % 20 + 1;**

**} while (r != t);**

**int segment = rand() % 3 + 1;**

**//the dart has hit a single segment**

**if (segment == 1)**

**{**

**return r;**

**}**

**//the dart has hit a double segment**

**else if (segment == 2)**

**{**

**return 2 \* r;**

**}**

**//the dart has hit a treble**

**else {**

**return 3 \* r;**

**}**

**}**

**}**

**}**

**}**

**int Board::treble\_throw(int t, int a)**

**{**

**//2d array with each segment of the board to allow the dart to go left or right of the intended target**

**int board[2][21] = { {0,20,15,17,18,12,13,19,16,14,6,8,9,4,11,10,7,2,1,3,5},**

**{0,18,17,19,13,20,10,16,11,12,15,14,5,6,9,2,8,3,4,7,1} };**

**//generates a random number between 0-100 to compare with the player's accuracy**

**int r = rand() % 100;**

**if (r < a)**

**{**

**//the intended target has been hit**

**return 3 \* t;**

**}**

**//if the random number is larger than the accuracy**

**else**

**{**

**r = rand() % 100 + 1;**

**//25% chance of hitting the left treble**

**if (r <= 25)**

**{**

**return 3 \* (board[0][t]);**

**}**

**//25% chance of hitting the right treble**

**else if (r > 25 && r <= 50)**

**{**

**return 3 \* (board[1][t]);**

**}**

**//15% chance of hitting the left single**

**else if (r > 50 && r <= 65)**

**{**

**return board[0][t];**

**}**

**//15% chance of hitting the right single**

**else if (r > 65 && r <= 80)**

**{**

**return board[1][t];**

**}**

**//5% chance of left double**

**else if (r > 80 && r <= 85)**

**{**

**return 2 \* (board[0][t]);**

**}**

**//5% chance of right double**

**else if (r > 85 && r <= 90)**

**{**

**return 2 \* (board[1][t]);**

**}**

**//7% chance of hitting anywhere else on the board**

**else if (r > 90 && r <= 97)**

**{**

**//generates a random number to decide if the dart hits the bull or another segment**

**r = rand() % 2 + 1;**

**if (r == 1)**

**{**

**//bull**

**r = rand() % 2 + 1;**

**if (r == 1)**

**{**

**//outer bull**

**return 25;**

**}**

**else {**

**//bull**

**return 50;**

**}**

**}**

**//any other segment on the board**

**else**

**{**

**//input validation so that the algorithm does not accidentally return the intended target**

**do {**

**r = rand() % 20 + 1;**

**} while (r != t);**

**int segment = rand() % 3 + 1;**

**//the dart has hit a single segment**

**if (segment == 1)**

**{**

**return r;**

**}**

**//the dart has hit a double segment**

**else if (segment == 2)**

**{**

**return 2 \* r;**

**}**

**//the dart has hit a treble**

**else {**

**return 3 \* r;**

**}**

**}**

**}**

**//3% chance of missing the board entirely**

**else**

**{**

**return 0;**

**}**

**}**

**}**

**int Board::bull\_throw(int a)**

**{**

**int r = rand() % 100 + 1;**

**if (r < a)**

**{**

**return 50;**

**}**

**else {**

**r = rand() % 100 + 1;**

**//80% chance that the outer bull is hit instead**

**if (r <= 80)**

**{**

**return 25;**

**}**

**else {**

**r = rand() % 20 + 1;**

**int segment = rand() % 100 + 1;**

**//80% chance to hit a single**

**if (segment <= 80)**

**{**

**return r;**

**}**

**// 10% chance to hit a treble**

**else if (segment > 80 && segment <= 90)**

**{**

**return 3 \* r;**

**}**

**//5% chance to hit a double**

**else if (segment > 90 && segment <= 95)**

**{**

**return 2 \* r;**

**}**

**//5% chance to miss entirely**

**else {**

**return 0;**

**}**

**return r;**

**}**

**}**

**}**

**string Board::checkout(int c)**

**{**

**//4d array to store the steps for a 3 dart finish**

**string threeDartFinish[4][65] = {**

**//scores**

**{"170", "167", "164", "161", "160", "158", "157", "156", "155", "154", "153", "152", "151",**

**"150", "149", "148", "147", "146", "145", "144", "143", "142", "141", "140", "139", "138",**

**"137", "136", "135", "134", "133", "132", "131", "130", "129", "128", "127", "126", "125",**

**"124", "123", "122", "121", "120", "119", "118", "117", "116", "115", "114", "113","112",**

**"111", "110", "109", "108", "107", "106", "105", "104", "103", "102", "101", "99"},**

**//first shots**

**{"T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20","T20", "T20", "T20",**

**"T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20",**

**"T20", "T20", "25", "T20", "T20", "25", "T20", "T20", "T19", "T18", "T20", "T19", "T18",**

**"T20", "T19", "T18", "T20", "T20", "T19", "T20", "T20", "T19", "T19", "T20", "T19","T20",**

**"T19", "T20", "T20", "T20", "T20", "T20", "T19", "T16", "T19", "T20", "T20", "T19"},**

**//second shots**

**{"T20", "T19", "T18", "T17", "T20", "T20", "T19", "T20", "T19", "T18", "T19", "T20", "T17",**

**"T18", "T19", "T20", "T17", "T18", "T19", "T20", "T17", "T14", "T19", "T20", "T13", "T18", "T15",**

**"T20", "T20", "T14", "T19", "T19", "T13", "T20", "T12", "T14", "T17", "T19", "T13", "T16", "T16",**

**"T20", "T11", "20", "T12", "18", "17", "19", "18", "14", "16", "20", "14", "10", "9", "16", "15",**

**"6", "16", "16", "6", "10", "9", "10"},**

**//final shots**

**{"50", "50", "50", "50", "D20", "D19", "D20", "D18", "D19", "D20", "D18", "D16", "D20", "D18", "D16",**

**"D14", "D18", "D16", "D14", "D12", "D16", "D20", "D12", "D10", "D20", "D12", "D16", "D8", "50", "D16",**

**"D8", "50", "D16", "D5", "D18", "D16", "D8", "D6", "D16", "D8", "D9", "D4", "D14", "D20", "D13", "D20",**

**"D20", "D20", "D20", "D20", "D20", "D16", "D20", "D20", "D20", "D16", "D16", "D20", "D16", "D20", "D20",**

**"D16", "D16", "D16"} };**

**//3d array to store the steps for a 2 dart finish**

**string twoDartFinish[3][78]{**

**//scores**

**{"100", "98", "97", "96", "95", "94", "93", "92", "91", "90", "89", "88", "87", "86", "85", "84", "83",**

**"82", "81", "80", "79", "78", "77", "76", "75", "74", "73", "72", "71", "70", "69", "68", "67", "66",**

**"65", "64", "63", "62", "61", "60", "59", "58", "57", "56", "55", "54", "53", "52", "51", "50", "49",**

**"48", "47", "46", "45", "44", "43", "42", "41", "39", "37", "35", "33", "31", "29", "27", "25", "23",**

**"21", "19", "17", "15", "13", "11", "9", "7", "5", "3"},**

**//first shots**

**{"T20", "T20", "T19", "T20", "T19", "T18", "T19", "T20", "T17", "T18", "T19", "T16", "T17", "T18", "T15",**

**"T20", "T17", "T14", "T19", "T16", "T13", "T18", "T19", "T20", "T17", "T14", "T19", "T16", "T13", "T18",**

**"T19", "T20", "T17", "T14", "T15", "T16", "T13", "T10", "T15", "20", "19", "18", "17", "16", "15", "14",**

**"13", "20", "19", "18", "17", "16", "15", "6", "13", "12", "11", "10", "9", "7", "5", "3", "17", "15",**

**"13", "11", "9", "7", "5", "3", "1", "7", "5", "3", "1", "3", "1", "1",},**

**//final shots**

**{"D20", "D19", "D20", "D18", "D19", "D20", "D18", "D16", "D20","D18", "D16", "D20", "D18", "D16", "D20",**

**"D12", "D16", "D20", "D12", "D16", "D20", "D12", "D10", "D8", "D12", "D16", "D8", "D12", "D16", "D8",**

**"D6", "D4", "D8", "D12", "D10", "D8", "D12", "D16", "D8", "D20", "D20", "D20", "D20", "D20", "D20", "D20",**

**"D20", "D16", "D16", "D16", "D16", "D16", "D16", "D20", "D16", "D16", "D16", "D16", "D16", "D16", "D16",**

**"D16", "D8", "D8", "D8", "D8", "D8", "D8", "D8", "D8", "D8", "D4", "D4", "D4", "D4", "D2", "D2", "D1"}, };**

**//loops for the length of the 4d array**

**for (int i = 0; i < 65; i++)**

**{**

**//if the score matches the player's remaining score**

**if (threeDartFinish[0][i] == to\_string(c))**

**{**

**//assigns the required steps to checkout to the string scoreRequired**

**string scoreRequired(threeDartFinish[1][i] + "," + threeDartFinish[2][i] + "," + threeDartFinish[3][i]);**

**//returns scoreRequired**

**return scoreRequired;**

**}**

**}**

**//loops for the length of the 4d array**

**for (int i = 0; i < 78; i++)**

**{ //if the score matches the player's remaining score**

**if (twoDartFinish[0][i] == to\_string(c))**

**{ //assigns the required steps to checkout to the string scoreRequired**

**string scoreRequired(twoDartFinish[1][i] + "," + twoDartFinish[2][i]);**

**//returns scoreRequired**

**return scoreRequired;**

**}**

**}**

**}**

**string Board::alternateCheckout(int s)**

**{**

**//4d array to store the steps for a 3 dart finish**

**string threeDartFinish[4][65] = {**

**//scores**

**{"170", "167", "164", "161", "160", "158", "157", "156", "155", "154", "153", "152", "151",**

**"150", "149", "148", "147", "146", "145", "144", "143", "142", "141", "140", "139", "138",**

**"137", "136", "135", "134", "133", "132", "131", "130", "129", "128", "127", "126", "125",**

**"124", "123", "122", "121", "120", "119", "118", "117", "116", "115", "114", "113","112",**

**"111", "110", "109", "108", "107", "106", "105", "104", "103", "102", "101", "99"},**

**//first shots**

**{"T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20","T20", "T20", "T20",**

**"T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20", "T20",**

**"T20", "T20", "25", "T20", "T20", "25", "T20", "T20", "T19", "T18", "T20", "T19", "T18",**

**"T20", "T19", "T18", "T20", "T20", "T19", "T20", "T20", "T19", "T19", "T20", "T19","T20",**

**"T19", "T20", "T20", "T20", "T20", "T20", "T19", "T16", "T19", "T20", "T20", "T19"},**

**//second shots**

**{"T20", "T19", "T18", "T17", "T20", "T20", "T19", "T20", "T19", "T18", "T19", "T20", "T17",**

**"T18", "T19", "T20", "T17", "T18", "T19", "T20", "T17", "T14", "T19", "T20", "T13", "T18", "T15",**

**"T20", "T20", "T14", "T19", "T19", "T13", "T20", "T12", "T14", "T17", "T19", "T13", "T16", "T16",**

**"T20", "T11", "20", "T12", "18", "17", "19", "18", "14", "16", "20", "14", "10", "9", "16", "15",**

**"6", "16", "16", "6", "10", "9", "10"},**

**//final shots**

**{"50", "50", "50", "50", "D20", "D19", "D20", "D18", "D19", "D20", "D18", "D16", "D20", "D18", "D16",**

**"D14", "D18", "D16", "D14", "D12", "D16", "D20", "D12", "D10", "D20", "D12", "D16", "D8", "50", "D16",**

**"D8", "50", "D16", "D5", "D18", "D16", "D8", "D6", "D16", "D8", "D9", "D4", "D14", "D20", "D13", "D20",**

**"D20", "D20", "D20", "D20", "D20", "D16", "D20", "D20", "D20", "D16", "D16", "D20", "D16", "D20", "D20",**

**"D16", "D16", "D16"} };**

**//3d array to store the steps for a two dart finish**

**string twoDartFinish[3][78]{**

**//scores**

**{"100", "98", "97", "96", "95", "94", "93", "92", "91", "90", "89", "88", "87", "86", "85", "84", "83",**

**"82", "81", "80", "79", "78", "77", "76", "75", "74", "73", "72", "71", "70", "69", "68", "67", "66",**

**"65", "64", "63", "62", "61", "60", "59", "58", "57", "56", "55", "54", "53", "52", "51", "50", "49",**

**"48", "47", "46", "45", "44", "43", "42", "41", "39", "37", "35", "33", "31", "29", "27", "25", "23",**

**"21", "19", "17", "15", "13", "11", "9", "7", "5", "3"},**

**//first shots**

**{"T20", "T20", "T19", "T20", "T19", "T18", "T19", "T20", "T17", "T18", "T19", "T16", "T17", "T18", "T15",**

**"T20", "T17", "T14", "T19", "T16", "T13", "T18", "T19", "T20", "T17", "T14", "T19", "T16", "T13", "T18",**

**"T19", "T20", "T17", "T14", "T15", "T16", "T13", "T10", "T15", "20", "19", "18", "17", "16", "15", "14",**

**"13", "20", "19", "18", "17", "16", "15", "6", "13", "12", "11", "10", "9", "7", "5", "3", "17", "15",**

**"13", "11", "9", "7", "5", "3", "1", "7", "5", "3", "1", "3", "1", "1",},**

**//final shots**

**{"D20", "D19", "D20", "D18", "D19", "D20", "D18", "D16", "D20","D18", "D16", "D20", "D18", "D16", "D20",**

**"D12", "D16", "D20", "D12", "D16", "D20", "D12", "D10", "D8", "D12", "D16", "D8", "D12", "D16", "D8",**

**"D6", "D4", "D8", "D12", "D10", "D8", "D12", "D16", "D8", "D20", "D20", "D20", "D20", "D20", "D20", "D20",**

**"D20", "D16", "D16", "D16", "D16", "D16", "D16", "D20", "D16", "D16", "D16", "D16", "D16", "D16", "D16",**

**"D16", "D8", "D8", "D8", "D8", "D8", "D8", "D8", "D8", "D8", "D4", "D4", "D4", "D4", "D2", "D2", "D1"}, };**

**//loops for the length of the 4d array**

**for (int i = 0; i < 65; i++)**

**{**

**//if the score in the array matches the player's remaining score**

**if (threeDartFinish[1][i] == to\_string(s))**

**{**

**//assigns the required checkout steps to the string scoreRequired**

**string scoreRequired = threeDartFinish[2][i] + threeDartFinish[3][i];**

**//returns scoreRequired**

**return scoreRequired;**

**}**

**}**

**//loops for the length of the 3d array**

**for (int i = 0; i < 78; i++)**

**{**

**//if the score in the array matches the player's remaining score**

**if (twoDartFinish[1][i] == to\_string(s))**

**{**

**//assigns the required checkout steps to the string scoreRequired**

**string scoreRequired = twoDartFinish[2][i];**

**//returns scoreRequired**

**return scoreRequired;**

**}**

**}**

**//if a checkout is not found, scoreRequired is empty**

**string scoreRequired = "";**

**//returns scoreRequired**

**return scoreRequired;**

**}**

**//sets throw score**

**void Board::set\_throw\_score(int t)**

**{**

**throwScore = t;**

**}**

**//returns throw score**

**int Board::get\_throw\_score()**

**{**

**return throwScore;**

**}**

**//bull throw for playing 301**

**int Board::bull\_throw\_301(int a)**

**{**

**//generates a random number between 1-100**

**int r = rand() % 100 + 1;**

**//if the randomly generated number is less than the accuracy**

**if (r < a)**

**{**

**//returns 50**

**return 50;**

**}**

**//otherwise**

**else {**

**//returns a random number between 1-20**

**r = rand() % 20 + 1;**

**return r;**

**}**

**}**

**//calculates the players target based on the score that is passed in**

**void Board::set\_target(int n)**

**{**

**//if the players score is greater than or equal to 100 or equal to 50**

**if (n >= 100 || n == 50)**

**{**

**//sets target to 50**

**target = 50;**

**}**

**// if the score is between 70-100**

**else if (n < 100 && n >= 70)**

**{**

**//sets target to 20**

**target = 20;**

**}**

**//otherwise**

**else {**

**//sets target to the difference between the player's score and 50**

**target = n - 50;**

**}**

**}**

**int Board::get\_target()**

**{**

**//returns target**

**return target;**

**}**

**Board.h**

#pragma once

#include <string>

using namespace std;

class Board

{

private:

//holds the score for a throw

int throwScore;

int target;

public:

//functions

int single\_throw(int t, int a, int gm);

int double\_throw(int t, int a);

int treble\_throw(int t, int a);

int bull\_throw(int a);

int bull\_throw\_301(int a);

string checkout(int c);

string alternateCheckout(int ts);

//setter and getter

void set\_throw\_score(int n);

int get\_throw\_score();

void set\_target(int n);

int get\_target();

};