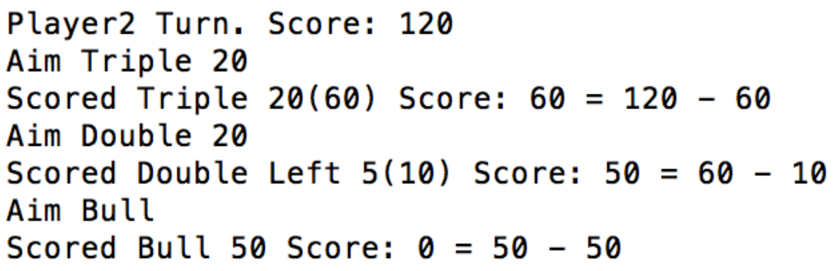
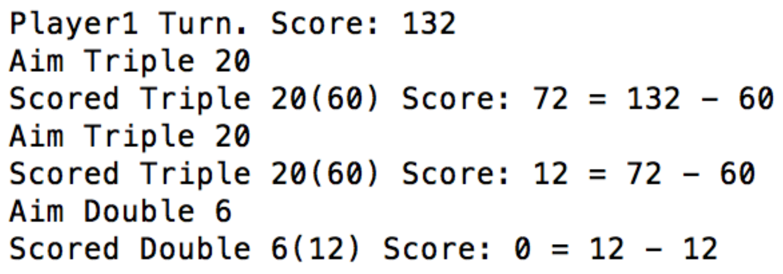
Matthew Wallace, 1502616, AG0701A (Programming in C++)

In my solution I used OOD, inheritance and polymorphism. The program starts asking you how many players you want to have, how to name them and have many games you want to simulate. Every simulation starts with “nearest the bull” game, to see who throws first. The winner is picked and the simulation continues in the established order, e.g.: if you have four players and ‘Player3’ won “nearest the bull” game he will start first and then accordingly ‘Player4’ (the program will loop the player’s vector to the beginning), ‘Player1’ and ‘Player2’. The simulation is based on the ‘Nine Darts Finish’ strategy. The players aim for the T20 to score the highest possible score which is 180. They keep aiming triple as long as it will leave them with the lowest and still possible to win score. If it’s possible to win by aiming for the bull or the double (in this order) they will do it. Otherwise, they will keep aiming for the highest possible score. If the score is odd and it’s impossible to win by aiming for the double or the bull they will aim for the single to make it even but it will never be smaller than 2. In other words, if they keep missing, they will be left with the lowest winning score which is 2. And if they bust the score will go back to the one they had before the turn started. The idea for the checkout was to that recreate a three 167s (T20-T19-Bull) which is considered a pure or perfect nine dart finish by some players and a 141 checkout (T20-T19-D12). It is possible to get a very similar effect in my program.
Here’s sample output of a checkout on the bull:



```
Player2 Turn. Score: 120                          //Players name and score
Aim Triple 20                                     //Player’s aim
Scored Triple 20(60) Score: 60 = 120 - 60         //What he actually scored
Aim Double 20                                     //Player’s aim
Scored Double Left 5(10) Score: 50 = 60 - 10      //What he actually scored
Aim Bull                                          //Player’s aim
Scored Bull 50 Score: 0 = 50 - 50                 //What he actually scored
```

Sample output of a checkout on the double (close to 141 checkout):



```
Player1 Turn. Score: 132
Aim Triple 20
Scored Triple 20(60) Score: 72 = 132 - 60
Aim Triple 20
Scored Triple 20(60) Score: 12 = 72 - 60
Aim Double 6
Scored Double 6(12) Score: 0 = 12 - 12
```

The whole game is handled by the ‘Game’ class. The function member ‘Play’ calls ‘NineDartFinish’ function which receives vector of pointers to ‘GenericPlayer’, that points to the ‘Player’ object, which are created on the heap. It is possible because ‘Player’ class inherits from ‘GenericPlayer’ class so ‘Player’ object is also a member of ‘GenericPlayer’ class. It allows me to have functions that accept a pointer to ‘GenericPlayer’ and can work with either ‘Player’ or ‘GenericPlayer’ object. This approach allows me to easily add different types of players and also having the user to play with the computer would take a lot of less changes in the program.