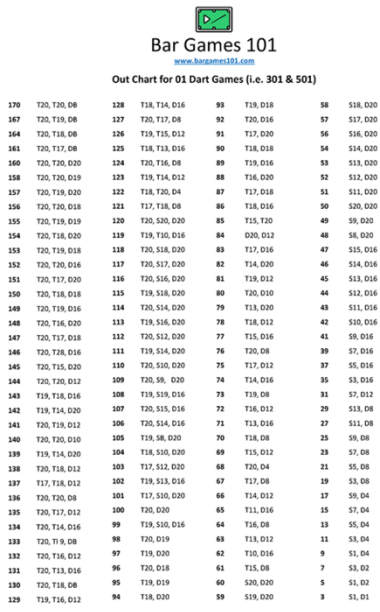
Report: Dart Project (incl. some enhanced strategy from Task 2 and throw function improvements)

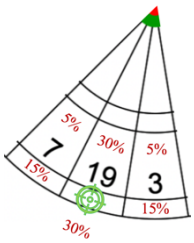
I have created a Monte Carlo simulation of darts matches between two players, which are stored in an array of two player objects from the Player class. Each match consists of a loop over 13 sets. Player 0 is assumed to start throwing in each match and thereafter the players alternate throwing first in the sets of a match. Each set is a loop over games, played until one player has won a total of 3 games. Finally, during each game the players take turns throwing up to 3 darts at the dartboard. This 'game' loop is exited when one player has reduced their score from 501 to 0, finishing by hitting a double field or the Bull's Eye. At any point, the players' scores are stored using an array of two score objects from the Score class.


Bar Games 101
www.bargames101.com
Out Chart for 01 Dart Games (i.e. 301 & 501)

170	T20, T20, DB	128	T18, T14, D16	93	T19, D18	58	S18, D20
167	T20, T19, DB	127	T20, T17, D8	92	T20, D16	57	S17, D20
164	T20, T18, DB	126	T19, T15, D12	91	T17, D20	56	S16, D20
161	T20, T17, DB	125	T18, T13, D16	90	T18, D18	54	S14, D20
160	T20, T20, D20	124	T20, T16, D8	89	T19, D16	53	S13, D20
158	T20, T20, D19	123	T19, T14, D12	88	T16, D20	52	S12, D20
157	T20, T19, D20	122	T18, T20, D4	87	T17, D18	51	S11, D20
156	T20, T20, D18	121	T17, T18, D8	86	T18, D16	50	S20, D20
155	T20, T19, D19	120	T20, S20, D20	85	T15, T20	49	S9, D20
154	T20, T18, D20	119	T19, T10, D16	84	D20, D12	48	S8, D20
153	T20, T19, D18	118	T20, S18, D20	83	T17, D16	47	S15, D16
152	T20, T20, D16	117	T20, S17, D20	82	T14, D20	46	S14, D16
151	T20, T17, D20	116	T20, S16, D20	81	T19, D12	45	S13, D16
150	T20, T18, D18	115	T19, S18, D20	80	T20, D10	44	S12, D16
149	T20, T19, D16	114	T20, S14, D20	79	T13, D20	43	S11, D16
148	T20, T16, D20	113	T19, S16, D20	78	T18, D12	42	S10, D16
147	T20, T17, D18	112	T20, S12, D20	77	T15, D16	41	S9, D16
146	T20, T28, D16	111	T19, S14, D20	76	T20, D8	39	S7, D16
145	T20, T15, D20	110	T20, S10, D20	75	T17, D12	37	S5, D16
144	T20, T20, D12	109	T20, S9, D20	74	T14, D16	35	S3, D16
143	T19, T18, D16	108	T19, S19, D16	73	T19, D8	31	S7, D12
142	T19, T14, D20	107	T20, S15, D16	72	T16, D12	29	S13, D8
141	T20, T19, D12	106	T20, S14, D16	71	T13, D16	27	S11, D8
140	T20, T20, D10	105	T19, S8, D20	70	T18, D8	25	S9, D8
139	T19, T14, D20	104	T18, S10, D20	69	T15, D12	23	S7, D8
138	T20, T18, D12	103	T17, S12, D20	68	T20, D4	21	S5, D8
137	T17, T18, D12	102	T19, S13, D16	67	T17, D8	19	S3, D8
136	T20, T20, D8	101	T17, S10, D20	66	T14, D12	17	S9, D4
135	T20, T17, D12	100	T20, D20	65	T11, D16	15	S7, D4
134	T20, T14, D16	99	T19, S10, D16	64	T16, D8	13	S5, D4
133	T20, Ti 9, D8	98	T20, D19	63	T13, D12	11	S3, D4
132	T20, T16, D12	97	T19, D20	62	T10, D16	9	S1, D4
131	T20, T13, D16	96	T20, D18	61	T15, D8	7	S3, D2
130	T20, T18, D8	95	T19, D19	60	S20, D20	5	S1, D2
129	T19, T16, D12	94	T18, D20	59	S19, D20	3	S1, D1

Each throw is simulated with artificial intelligence—like in real life, the optimal score to target on the dartboard (e.g. 38) is determined according to the respective player's current score using the throwDecision function in the ThrowDecision class. Next, using the actualTargetHit function from the same class, the optimal field on the dartboard to achieve this optimal score is determined (e.g. double 19). To ensure realistic intelligent optimal strategies, the 'Out Chart' from bargames101.com/wp-content/uploads/2016/10/The-01-Out-Chart_750x970.png (see right) underlies the construction of the two functions in the ThrowDecision class.

The throw at the determined target field is then implemented using the functions from the Board class, which simulate a throw at the (outer) Bull's Eye or single/double/treble fields. The targeting accuracy when aiming at the (outer) Bull's Eye and single/double/treble fields is a pre-determined percentage stored in each player object. These percentages allow to mimic players with varying performance. Together with the target field determined by the actualTargetHit function, this accuracy is passed to the Board class functions. These functions return the actual score hit by a player when targeting.

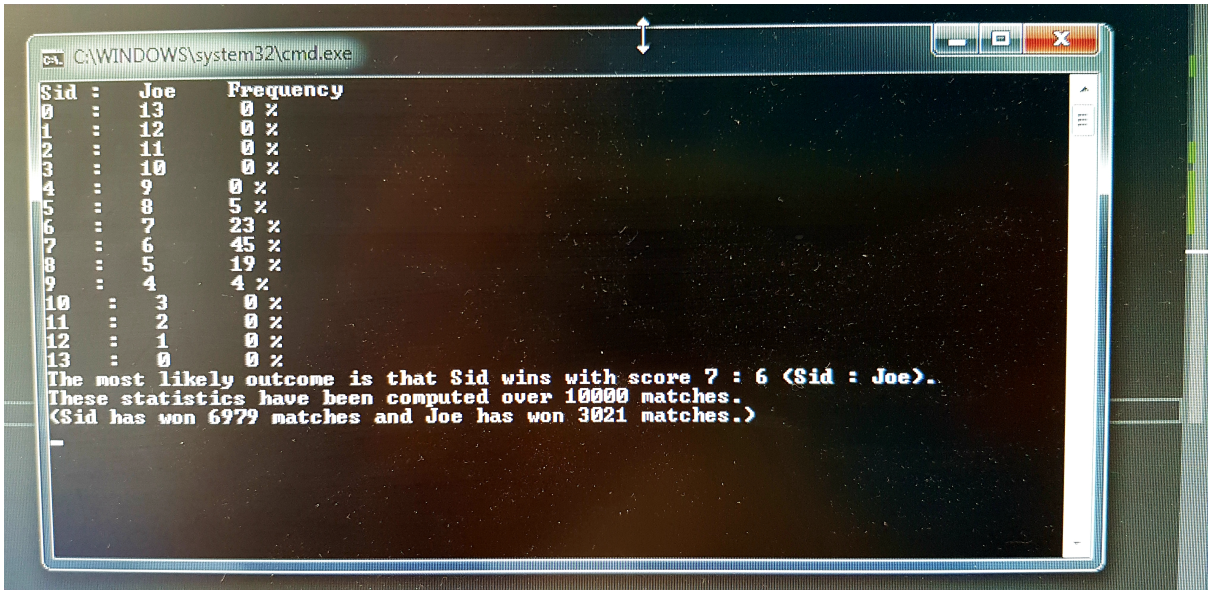


The actual score hit is obtained as follows: If the respective player hits doubles accurately with probability *p*, the probability to miss any double (1-*p*) is distributed amongst the neighbouring fields in approximate correspondence to their surface area (example on left for missed "double 19"). This area-dependant percentage has the advantage that the relative probability to hit a neighbour field is preserved when the targeting accuracy of a player is adjusted, mimicking reality. Further, if the actual target hit was double or Bull's Eye, the throw is registered as a valid finishing throw. Should the actual throw have reduced a player's game score to 0, the player has won the respective game. If however, the score 0 was achieved by a treble, single, outer Bull's Eye target hit, or if the player scores 1 or negatively, their score is reset to the score before their turn to throw and it's the respective other player's turn to throw up to thrice.

The number of sets won for player 1 are stored in an array (the other player's matches won are not needed as sets played add to 13 and it is more efficient to only score player 1's sets won). From this, the occurrences of final match scores over all of the matches are counted. This allows to output the frequency of match score occurrence over 10,000 matches and to find the most likely outcome (see "results").

OOP allows us to separate data and functions for different aspects of the code, providing greater control than in procedural programming. Similar objects from a class can be constructed (and destructed) easily—such as our two player objects, each of which has a name and hitting accuracies for (outer) Bull's Eye and single/double/treble fields. Also, inheritance is possible between classes (not used here). Thus, lines of code, development time and errors are reduced compared to procedural programming. Further, security from unwanted manipulation of data can be increased by declaring the respective as "private" in a class: This way, data, such as the player's names, cannot be changed from out with the class.

Sample output



N.B. The above MC simulation is run with both players having the same hitting probabilities. This shows how much of an advantage Sid has from starting each match. When creating the player objects in source.cpp, the targeting probabilities can easily be modified in a realistic* fashion.

Further, I have noted that the simulations for a match in the example we were given stopped as soon as one player had won 7 sets. However, in the instructions it said that 13 sets are played per match. I hence stuck to what it said in the instructions in my simulations.

*) See project report