DEGREE PROJECT, IN COMPUTER SCIENCE , FIRST LEVEL
STOCKHOLM, SWEDEN 2015

Optimal Yahtzee

A COMPARISON BETWEEN DIFFERENT ALGORITHMS FOR PLAYING YAHTZEE

DANIEL JENDEBERG, LOUISE WIKSTÉN

KTH ROYAL INSTITUTE OF TECHNOLOGY

CSC SCHOOL

Abstract

The game of Yahtzee is a semi-strategic luck based game, which means it should be possible to maximize the score with an optimal strategy. The main purpose of the study is to compare the results of the optimal algorithm with other useable strategies when playing Yahtzee. To receive interesting results, the strategies and decisions made by the algorithms will be compared and analyzed in performance as well as by creating an environment where they have to choose from the same set of dices. To further see how well the different algorithms performed, Human Trials were conducted where 6 humans contributed by playing the game of Yahtzee 300 times. These test subjects were familiar with the game of Yahtzee and in this study it is concluded that these subjects had through reinforcement learning created an almost optimal play style.

Our conclusion is that the Optimal Algorithm performs better than other algorithms that because it does not take any risks while playing while it tries to maximize the score but doing this uses a great amount of computation power, approximately 13.8GB of RAM and around 22 hours to complete.

Table of Contents

1. INTRODUCTION **4**
1.1 PROBLEM DEFINITION 4
1.2 PROBLEM STATEMENT 4
1.2.1 Optimal algorithm *4*
1.2.3 Other algorithms *4*
1.2.3 Comparison *5*

2. BACKGROUND **6**
2.1 SCANDINAVIAN YAHTZEE RULES (YATZY) 6
2.2 OPTIMAL ALGORITHM 7
2.2.1 Technical Specifications *8*
2.3 OTHER YAHTZEE ALGORITHMS 9
2.3.1 Scorecard-based Human Algorithm *9*
2.3.2 Dice-based Human Algorithm *10*
2.3.3 Forced-scoring *10*
2.5 REINFORCEMENT LEARNING 12

3. METHOD **13**
3.1 DICE 13
3.1.1 Test for the virtual dices *13*
3.1.2 Test for real dices *13*
3.2 SCORECARD 13
3.3 PERFORMANCE TESTS 14
3.4 SAME-VALUE TESTS 14
3.5 HUMAN TRIALS 14

4. RESULTS **15**
4.1 PERFORMANCE TESTS 15
4.1.1 Optimal Algorithm *15*
4.1.1 Scorecard-based Human Algorithm *17*
4.1.2 Dice-Based Human Algorithm *18*
4.1.3 Forced Scoring Algorithm *19*
4.2 HUMAN TRIALS 21
4.2.1 Manual test of the dices *21*
4.2.2 Results of humans playing *21*
4.3 SAME-VALUE TESTS 23

5. DISCUSSION **28**
5.1 THE ALGORITHMS 28
5.2 DICE TESTS 28
5.2.1 Virtual Dices *28*
5.2.2 Real Dices *28*
5.3 PERFORMANCE TEST 28
5.3.1 Analysis of graphs and data *29*

5.3.2 Conclusions ... *30*
5.4 SAME-VALUE TEST ... 30
5.4.1 Comparison of the algorithms ... *31*
5.4.2 Conclusions ... *31*
5.5 HUMAN TRIALS ... 32
6. CONCLUSIONS ... 32
7. REFERENCES ... 33
APPENDIX A - PERFORMANCE TEST TABLES ... 34
APPENDIX B - HUMAN TRIALS SCORES ... 42

1. Introduction

Yahtzee is a semi strategic luck-based dice game that was developed in the 1940's. The game's objective is maximizing your score from throwing five dices, all with six sides, and obtaining different dice combinations. The desired combinations are given by categories on a Yahtzee scorecard, and for each category there are requirements. Each category can only be scored once, and that is done by summarizing the values on the dice faces or zero if the dices don't match the requirements. The game is over when all categories are scored.

For each turn, the player starts by rolling the dices. Based on the outcome, the player can save one or more dices for receiving a certain combination, and re-roll the ones that are not saved. The total number of rolls for one round is three. When all categories are filled in, the final score is the sum of the values in each category.

Because of this being a game that is somewhat based on strategy, when selecting which dice or dices to save between rolls and deciding where to put the score, it should be possible to find an algorithm that maximizes the score. In this study we use the Scandinavian rules for Yahtzee.

1.1 Problem Definition

The main purpose of this study is to measure an efficient Yahtzee-playing algorithm against other algorithms to receive readings on performance for different play styles. More importantly, how much would a mathematically optimal algorithm score and does one benefit from using that algorithm despite its obvious inefficiency?

For this study several different algorithms, one of them being a mathematically efficient one (meaning the algorithm developed by J. Glenn [1]), will be developed and tested to determine how they differ in score as well as the overall time consumption. The game that is played will be Yahtzee solitaire, a solo version of Yahtzee, scored using the Scandinavian rules of scoring. If successful, this study should be able to determine how well the algorithms score in Yahtzee solitaire as well as rank them in order of scoring and time consumption.

Since Yahtzee is predefined in size it is known how many turns, throws, dices and how many scorecard rows there are, calculating and determine the complicacy should be straightforward. Any comparisons of time consumption in different algorithms should therefore return comprehensible and clear results.

One extra thing to take into consideration, when analyzing these algorithms and strategies, is that when a human plays Yahtzee enough times a state of reinforcement learning will affect their strategy and how would this affect the outcome of the game?

1.2 Problem Statement

1.2.1 Optimal algorithm

- How well does an optimal algorithm perform, in both scoring and time consumption?

1.2.3 Other algorithms

- How well does other algorithms perform, in both scoring and time consumption?

1.2.3 Comparison

- How well does the optimal algorithm perform against non-optimal algorithms, both in scoring and time consumptions?
    - What were the differences in performance?
    - What choices create a difference in scoring?
- Is the optimized algorithm suitable for usage or should other algorithms be chosen when playing as a human?
    - What is the decisive factor when choosing the algorithm?

How does these algorithms play in comparison to human trials?

2. Background

A Yahtzee solitaire algorithm is based on two questions:

- Which dices should or should not be rerolled?
- In which category should I score the currently obtained combination?

From these questions the optimal algorithm should choose the best path, if we consider the game as a set of states with choices connecting them [1], according to probability theory. The other algorithm should use other logical principles to make their choices; these choices could also be set with certain restrictions.

2.1 Scandinavian Yahtzee rules (Yatzy)

The Scandinavian Yahtzee rules are different from the international rules, and contain new categories on the scorecard and different scoring rules. The scorecard can be viewed in Table 1 where each category is described and the category's maximum score.

The game is based on 15 turns, and each turn the player may roll the dice up to three times. For each turn the player has to obtain a combination of dice values and select a row on the scorecard where to put it, in no particular order. However, there are rules for each category in the scorecard.

Name	Maximum Scores	Description
Ones	5	Number of ones (only ones scored).
Twos	10	Number of twos (only twos scored).
Threes	15	Number of threes (only threes scored).
Fours	20	Number of fours (only fours scored).
Fives	25	Number of fives (only fives scored).
Sixes	30	Number of sixes (only sixes scored).
Sum (first six rows)	105	Total score above
Bonus (more than 63 points in upper bracket)	50	Obtained if total score above is more than 63, else zero points
Pair	12	Two dices showing the same number.
Two pairs	22	Two different pairs of dices i.e. 2,2,3,6,6.
Three of a kind	18	Three dices showing the same number.
Four of a kind	24	Four dices showing the same number.
Small straight	15	The five dice shows 1,2,3,4,5.
Large straight	20	The five dice shows 2,3,4,5,6.
Full house	28	Two of one kind and three of another, i.e. 2,2,5,5,5
Chance	30	Anything goes.
Yahtzee (Yatzy)	50	All dice the same number, always 50 points.
Total sum	374 (324 without bonus)	Total score, points awarded in game

Table 2.1: *Presents a detailed view of the scorecard.*

In the first section, there are six rows and the sum that is written in the scorecard is the sum of dice that shows ones, twos, etc. If the sum of these six rows is more than or equal to 63, the player is awarded with a bonus of 50 points. That means that the top rows are important when trying to score well.

The following four rows are *Pair*, *Two pairs*, *Three of a kind* and *Four of a kind*. In all these, only the dices that are a part of the combination are to be used in the sum, and the sum will be higher if the selected dices show higher numbers. For example will a pair of sixes give a score of 12, but a pair of fours is 8 points. However, the other dices may display any number and will not affect the score. It is important to notice that the *Two pairs* must contain two different pairs.

The *Small straight* and *Large straight* rows include all dices, and always gives the same scores. To be able to fill out these rows the dices have to display a straight from 1 to 5 (*Small straight*) or 2 to 6 (*Large straight*). The *Small straight* sum is 15 points, which is the sum of the values that the dices shows, and the *Large straight* sum is the sum of the values of the dices, 20 points.

The *Full house* category includes all dices and requires three of them to be one and the same value and the other two dices to display equal values, but different from the three dices' values. In other words, this category combines the *Three of a kind* and the *Pair* categories. In this category it is therefore preferable to use dices with high values, as the maximum score is 28 points.

The following category, *Chance*, is a category where each dice value counts but without any requirements, and is often used as a discard box for a roll that will not fit in any other category. If it is not used as a discard box it is suitable to try to receive a high score in this category, because the value can have a great impact on the final score due to its maximum being 30 points.

Yahtzee is when all dice shows the same value after three throws. Not all throws have to be used however, if a desired combination is rolled in one throw, the player can decide to only use one of the throws. No matter which value that the dice shows, the *Yahtzee* is always 50 points. That means that the player can receive a high score in this category and receiving a *Yahtzee* is preferred.

If the player rolls the dice and they show values that does not fit any category and the *Chance* category is filled in, they will have to place a zero in one of the unused boxes and it will be regarded as filled in. That means that if the player rolls something that fits in this category later on, it will not count.

This is a summary of the Yahtzee rules from Alga [2].

2.2 Optimal algorithm

The problem of finding an optimal Yahtzee algorithm is about maximizing the expected score. In probability theory it is possible to calculate an expected value, for example the expected value of a thrown dice is 3.5, in this case the expected value is roughly the average value obtained from large number of dice rolls.

The problem of finding the optimal Yahtzee algorithm is a problem that is solved. James Glenn created an optimal Yahtzee algorithm for the international Yahtzee by visualizing the game as a graph, where the vertices are the different states of the game and from each state there are edges to subsequent states. [1]

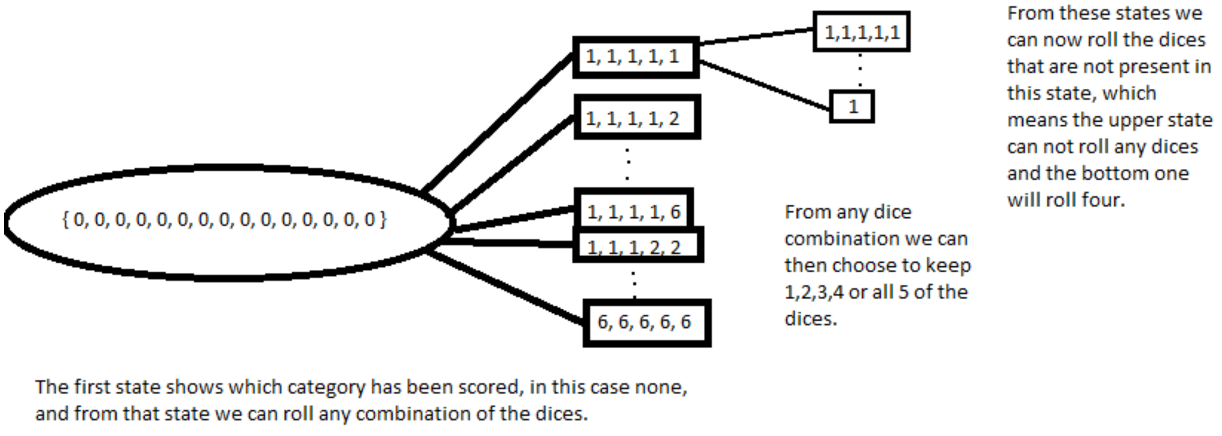

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
1, 1, 1, 1, 1
1, 1, 1, 1, 2
1, 1, 1, 1, 6
1, 1, 1, 2, 2
6, 6, 6, 6, 6
1,1,1,1,1
1
From these states we can now roll the dices that are not present in this state, which means the upper state can not roll any dices and the bottom one will roll four.
From any dice combination we can then choose to keep 1,2,3,4 or all 5 of the dices.
The first state shows which category has been scored, in this case none, and from that state we can roll any combination of the dices.

Figure 2.1: *Shows some states from V1, V2 and V3 to illustrate the concept of the graph from the optimal algorithm.*

By creating this graph James Glenn created an algorithm that scored 245.87 on average without *Yahtzee* bonuses, 100 points extra for every “extra” *Yahtzee*, and 254.59 with. This study uses slightly different Yahtzee rules but the optimal algorithm will be based on the computations described in James Glenn’s thesis.

2.2.1 Technical Specifications

These are a summary of the technical specifications given in J.Glenn’s report [3]. The optimal algorithm computes a six-partite graph, G = {V1 ∪ V2 ∪ V3 ∪ V4 ∪ V5 ∪ V6, E}, of different states. An edge, e ∊ E, can only reach between two neighboring vertex groups, a state from V(i) has edges to V(i+1) where i ∊ {1,2,3,4,5,6} = Z(6)\{0}, which means a state from V6 has edges to states in V1. As stated in the previous heading, we use statistics to calculate the expected values of every state.

2.2.1.1 Mathematical Definition: Vertices

The different vertices have different definitions, which are as follow.

I. u ∊ V1, u = {C, upper} where C is a set of Booleans representing the different categories in the scorecard and whether or not they are scored and upper is the current score in the upper categories. There are therefore a fixed amount of states in V1 which is $|V1| = 2^{15} * 64 = 2097152$.

II. v ∊ V2|V4|V6, v={u, dices}, u ∊ V1, dices={i1,i2,i3,i4,i5}, i1,i2,i3,i4,i5 ∊ (1,2,3,4,5,6). Since there are no correlation between dices there will be 252 distinct dice combinations. $|V2| = |V4| = |V6 = |2^{15} * 64 * 252 = 528482304$.

III. v ∈ V3|V5, v={u, dices}, u ∈ V1, keeps is a set of one to five integer values between 1 and 6. The set represents the dices kept in between two rolls. There will be 462 distinct combinations of dices to keep, $|V3| = |V5| = 2^{15} * 64 * 462 = 968884224$.

2.2.1.2 Mathematical Definition: Edges

The edges in the graphs are specified as possible movement from a state. The edges are therefore directed edges from a vertex that simulates either a reroll or a scoring action.

(1) Reroll Action

$$e = (x, y), x \in V(i), y \in V(i+1), i \in Z5\backslash\{0\}$$

(2) Scoring Action

$$e = (x, y), x \in V6, y \in V1$$

2.2.1.3 Mathematical Definition: Expected Values Calculations

The equations to calculate the expected values, X(u), have a base case: For any terminal position u, X(u)=0, for any non terminal position these three equations are followed:

(1) u ∈ V1|V3|V5, $X(u) = \sum_{(u,v)\in E} Proll(u, v) * X(v)$
(2) u ∈ V2|V4, $X(u) = max_{(u,v)\in E} X(v)$
(3) u ∈ V6, $X(u) = S(u, v) * X(v)$, v ∈ V1, S(u, v) is the score gained for moving from a state u to v.

These are the basic technical specifications of the optimal algorithm. Since the graph described has 1499463680 states which each have a set of dices and a Boolean array, this would take approximately 52GB of storage, not taking into account the amount of edges. Of course that is not reasonable, since computers today rarely have more than 16GB of ram, but through some optimizations this algorithm is executable.

2.3 Other Yahtzee algorithms

When not calculating the chance of a certain outcome of the game, there are other strategies that are possible to use while playing Yahtzee. Yahtzee players more commonly use them because it is too difficult to always calculate what is optimal in each situation. The algorithms below are based on a human that is not to familiar with the game of Yahtzee, since playing the game several times might, with high probability, create a state of reinforcement learning.

2.3.1 Scorecard-based Human Algorithm

The Scorecard-based human algorithm is an algorithm that for each turn of the game calculates which row on the scorecard that will generate the best score and the probability of receiving a higher dice combination. This is a greedy approach to solving the problem since it does not take future expected scores into account, except when there is a sufficiently large chance of obtaining a higher score for that category later. Without further analysis of human behavior, this is as close to a optimal human play style our algorithms comes.

2.3.2 Dice-based Human Algorithm

A strategy is to either find out which number is most frequent in the first roll and save the dices that displays the number, if those can be scored in a category. In this study this approach is called the "Dice-based Human Algorithm" meaning that it only calculates the options it has from the most common dice number rolled, and does not take the scorecard in consideration. Because of that, the algorithm never tries to score in a free category and the strategy is therefore not optimal.

The algorithm is based on certain rules that are applied and the dices are rerolled accordingly. Firstly, the algorithm iterates through the dice values and tries to find two matching dices. When a pair is found, the algorithm continues comparing the other dices to find more of the kind or a dice pair with other values. If no two dices display the same value, the algorithm explores whether the dices displays a *Large straight* or *Small straight*, otherwise all dices, except the one that possibly displays a six, are rerolled to find a better score.

Pseudo code:
```
if pair
    if three of a kind
        if four of a kind
            if Yahtzee
                return
        else if other two dices alike //full house
            return
        else
            reroll dices not matching the three of a kind
    else if another pair amongst the rest of the dices
        reroll the dice with no match
    else if pair is better than 3 or its top row free
        reroll dice not in pair
    else
        reroll all dice except the 6es (best to keep
else if close to large straight
    if large straight
        return
    else
        reroll dice that does not fit
else if close to small straight
    if small straight
        return
    else
        reroll dice that does not fit
else
    reroll all dice except the 6's
```

When calculating where to place the result in the scorecard, the algorithm chooses the category that will maximize the final score. Therefore it prefers placing the score in the upper categories when there are more than three of each number because that behavior leads to the upper bonus.

2.3.3 Forced-scoring

The forced-scoring approach attempts to maximize the score of each category in order, from top to bottom, of the scorecard. The algorithm is not using probability theory and therefore no expected values or future score is calculated. This is a less complex algorithm, a brute force attempt to the game of Yahtzee and used as a method of comparison in this study.

Pseudo code:

```
for each category
    if(category is (1,6))
        save all dices that matches the category and reroll others
    else if(category == 7)
        if(got pair better than 3)
            save
        else
            save highest number and reroll others
    else if(category == 8)
        if(one pair)
            if(two pair)
                save all
            else
                save highest and pair and reroll others
        else
            save the two highest numbers and reroll others
    else if(category == 9)
        if(three of a kind)
            save all
        else if(pair)
            reroll all other
        else
            save highest and reroll others
    else if(category == 10)
        if(four of a kind)
            save all
        if(three of a kind)
            save the three reroll others
        else if(pair)
            save the pair reroll others
        else
            save highest and reroll others
    else if(category == 11)
        save one of each dice between (1,5) and reroll all duplicates and 6’s
    else if(category == 12)
        save one of each dice between (2,6) and reroll all duplicates and 1’s
    else if(category == 13)
        if(three of a kind)
            if(other two dices is pair)
                save all
            else
                save highest and reroll last dice
        if(pair)
            if(two pair)
                save two pair and reroll last dice
            else
                save highest and pair
                reroll others
    else if(category == 14)
        save all above 3 reroll others
    else if(category == 15)
```

```
if(pair)
      if(three of a kind)
            if(four of a kind)
                  if(Yahtzee)
                        save
                  else
                        save four and reroll last
            else
                  save three and reroll others
      else
            save pair and reroll others
else
      save highest and reroll others
```

2.5 Reinforcement learning

Reinforcement learning is learning through interacting with the environment several times and learning/recognizing which choices contribute the most preferable outcome [4]. Playing this way might make a human score close to or even rivaling the optimal algorithm. One thing that could be the differing factor, between the optimal algorithm and the human trials, is that a human can remember the patterns in which the dices come and if certain numbers are more frequent than others. While that might generate certain positive results it might also generate a false feeling of knowledge, in other words; that the subject expect a certain outcome that does not happen.

3. Method

Under this heading we have different tests as well as implementation details for the Scorecard (3.2) and the description for the human trials (3.5).
The testing method includes tests to validate dices (3.1), virtual and real dices, as well as the methods used to test the algorithms, Performance Tests (3.3) and a Same-Value test (3.4).

The algorithms were written in Java, except for the Optimal Algorithm. The Optimal Algorithm could not be written in Java due to constraints like the Java Virtual Machines allocation size as well as the time consuming Java Garbage Collector, instead the Optimal Algorithm was written in C++.

3.1 Dice

To construct a dice that was useable for the Yahtzee-playing algorithm, each dice needed three values for each turn in the game and for each round there are 15 turns. That means that the dices needed 15*3 values for every game, even though not all might be used.

To save these values and use the same values for the different algorithms, the dices were printed to a file with three numbers between 1 and 6 on each row. These three numbers represent the values of each dice for a turn. To randomize these values the Java Random.nextInt() method was used. This method was chosen over the C++ rand() method because the random number generator supplied by Java gave different random values between runs while the C++ version gave the same random values over and over again.

The Optimal Algorithm was, as mentioned above, programmed in C++, therefore it had to read the dice values from files for the tests in section 3.3 as well as 3.4.

3.1.1 Test for the virtual dices

As for probability scenarios, in this study that is the simulation of a dice throw, the program used the Java API's Random.nextInt() which is a random number generator. This random number generator is not perfectly random, but sufficiently random to test the algorithms and receive realistic results, the dices received an expected value of 3.50011 after three million test runs, which is sufficiently random for this study. This is a precondition for the randomness of the algorithms to have a fair performance test, and the test shows that it is fulfilled.

3.1.2 Test for real dices

To test the dices used to play Yahtzee, to assure that the dices used for manually playing were sufficiently random for this study, a manual test was conducted. This test consisted of throwing 5 dices 100 times while saving the data for comparison.

3.2 Scorecard

The scorecard was implemented as a triple {sum, S, M} where sum is the accumulate sum of scores, S is a list of scores in each category and M is a matrix of the dimension (15x5) where the dice combinations scored in each category is stored.

3.3 Performance Tests

These tests were developed to test the algorithms over 100 000 rounds of Yahtzee solitaire. For each round the total score of the round was calculated and added to an array and the total sum of all rounds. To receive the mean score of the algorithm the summarized result was divided by the number rounds. The array was printed to an Excel-file, for easy access. For each possible score in Yahtzee, the number of occurrences in the array was calculated. These values were then presented in a graph to see the score distribution. 100 000 rounds were enough to receive an estimation of the algorithm's performance.

All the different algorithms were used in the performance tests; the Optimal Algorithm, the Scorecard-based Human Algorithm, the Dice-based Human Algorithm and the Forced-Scoring Algorithm. They were all used to notice changes in the performance of different approaches.

These performance tests were a good estimation of the algorithms average score, because with 100 000 rounds the dice values will reach the expected score and therefore these tests will be a good estimation. The test data generated was suitable for diagrams that show trends in data generated.

3.4 Same-value Tests

These Same-value tests will be developed to test the algorithms with the same predefined random values. These tests where developed to see where the algorithms differ in choices, therefore only a small amount of tests will generate a large amount of data. The data generated will be 15*3*n = 45*n where 15 is the number of turns per game, 3 is the number of choices per turn and n is the number of tests run. By comparing the results from these tests it is easy to describe how and when different algorithms make correct or wrong decision.

The algorithms used in this test were the Optimal Algorithm, the Scorecard-based Human Algorithm and the Dice-based Human Algorithm. The Forced-Scoring Algorithm was not used in these tests because its decisions for the dices only applies to the category it has to score in and therefore it is not of interest in the study.

The data generated from the Same-value tests were the dice values for each turn of the game, the dice values that the algorithm wanted to save from each throw and where it placed the score. The total score for the round was saved as well to compare the algorithms and what different decisions that were made.

3.5 Human Trials

To compare the algorithms with humans a few test subjects, 6 persons, were gathered to play 300 rounds of solitaire Yahtzee. The test subjects had played Yahtzee regularly during their childhood and were therefore considered familiar to the game. Due to time constraints, the test subjects played together but were instructed to only try to maximize their own score to receive more realistic results for solitaire Yahtzee.
These human tests were time-consuming and therefore not an efficient method. The contribution made by this data might not be statistically valid but it will still give an indication to whether or not humans can play optimally through reinforcement learning.

4. Results

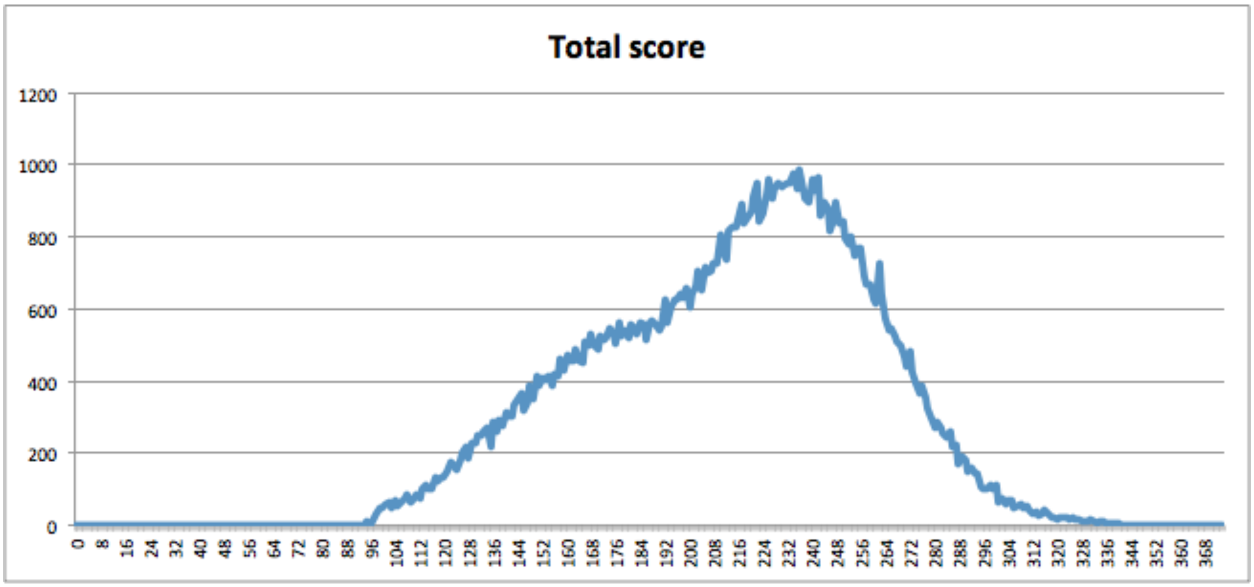
4.1 Performance tests

The results consist of;

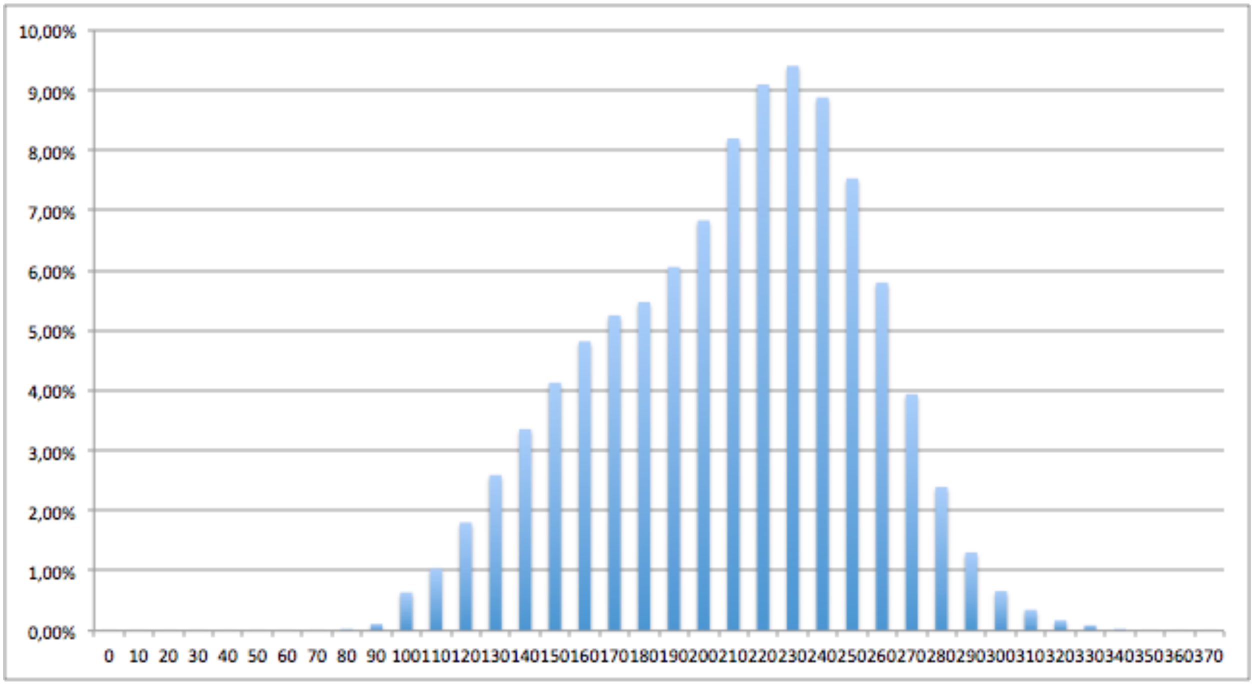
- Graphs depicting results over all test runs.
- Mean and Median over the results
- Maximum and Minimum scores.

4.1.1 Optimal Algorithm

The optimal algorithm has been run in a performance test that ran the algorithm 100 000 times and gave a mean score of 213 and a median of 219. The maximum score for these 100 000 runs was 341 and the minimum score was 88, which gives a range of 253 points. Time consumption for 100'000 runs was over 22 hours, 2 hours where for building the graph.



Graph 4.1: *Showing the data generated by 100'000 runs of the optimal algorithm. The Y-axis is the number of occurrences and the X-axis is the total score.*



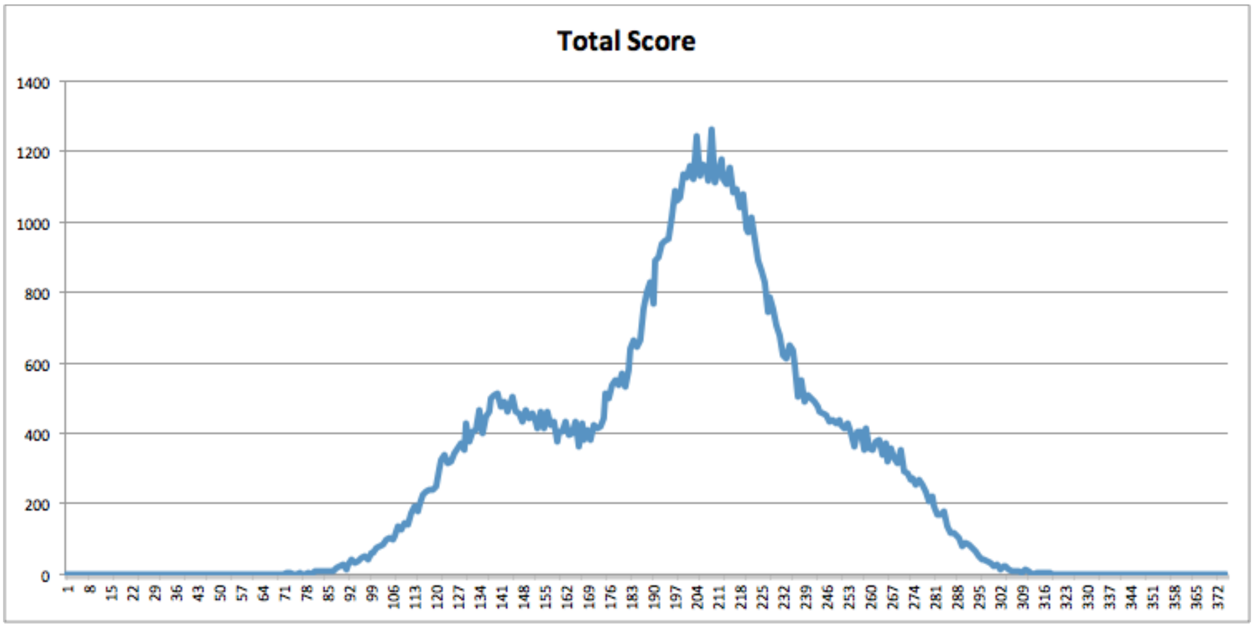
Graph 4.2: *Showing the data generated by 100'000 runs of the optimal algorithm in percentages in distinct ranges. The Y-axis is the percentage of occurrences and the X-axis is the total score in ranges of 10.*

150		159		4133	4,13%	13,69%
160		169		4821	4,82%	18,51%
170		179		5255	5,26%	23,76%
180		189		5477	5,48%	29,24%
190		199		6061	6,06%	35,30%
200		209		6836	6,84%	42,14%
210		219		8202	8,20%	50,34%
220		229		9102	9,10%	59,44%
230		239		9411	9,41%	68,85%
240		249		8885	8,89%	77,74%
250		259		7536	7,54%	85,27%
260		269		5801	5,80%	91,07%

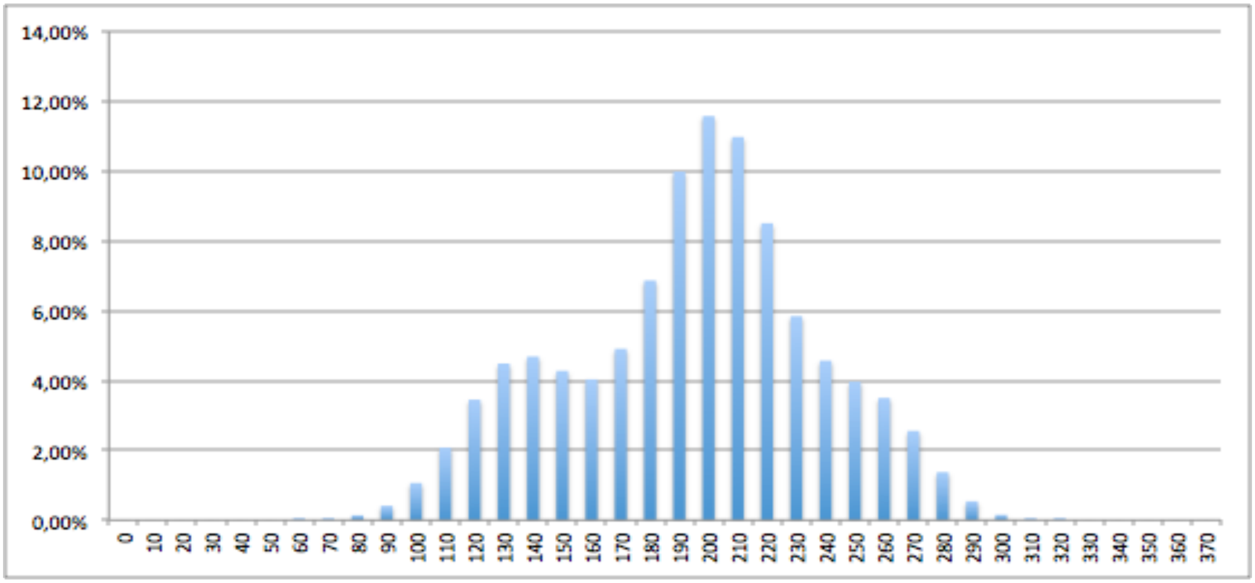
Table 4.1: *Showing the interesting data range collected from the optimal algorithm.*

4.1.1 Scorecard-based Human Algorithm

The scorecard-based human algorithm has been run in a performance test that ran the algorithm 100 000 times and gave a mean score of 199 and a median of 203. The maximum score for these 100 000 runs was 327 points and minimum was 65 which gives a range of 262 points. Time consumption for 100'000 runs was 3.48s.



Graph 4.3: *Showing the data gathered from 100'000 runs of the Scorecard-based Human algorithm. The Y-axis is the number of occurrences and the X-axis is the total score.*



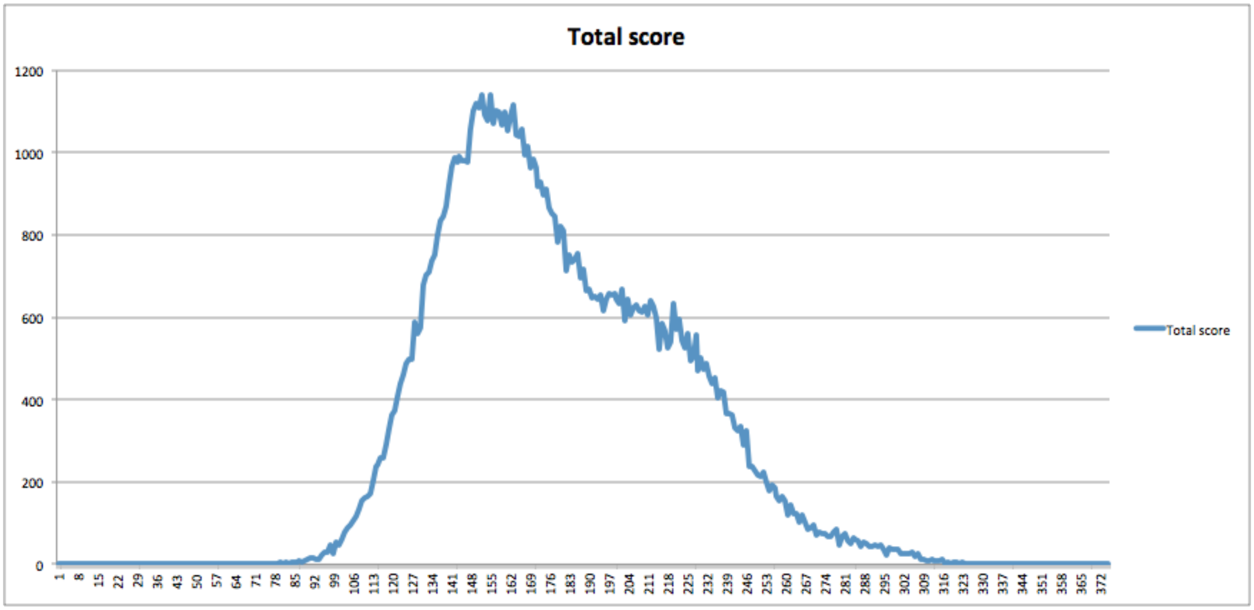
Graph 4.4: *Showing the data generated by 100'000 runs of the Scorecard-Based algorithm, the X-axis consists of distinct ranges of 10 and the Y-axis is the percentage in those ranges.*

Range of score				Number of occurrences	Percentage	Percentage (Cumulative)
180		189		6866	6,87%	36,42%
190		199		9987	9,99%	46,40%
200		209		11577	11,58%	57,98%
210		219		10977	10,98%	68,96%
220		229		8505	8,51%	77,46%
230		239		5842	5,84%	83,31%

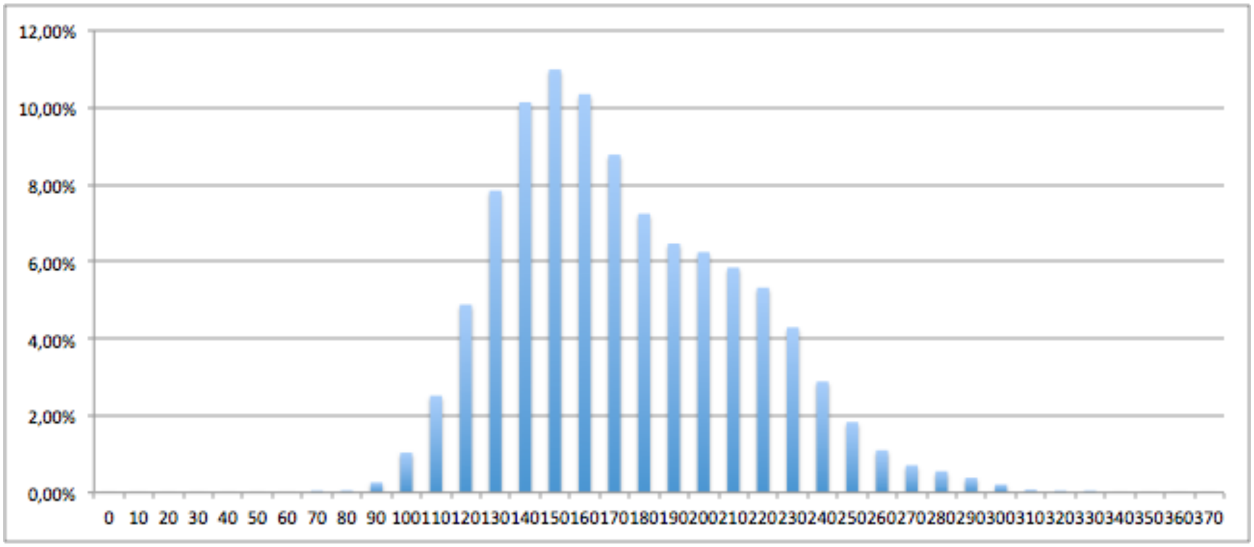
Table 4.2: *Showing the interesting data range for the Scorecard-based Human algorithm.*

4.1.2 Dice-Based Human Algorithm

The *Dice-Based Human Algorithm* has been run in a performance test that ran the algorithm 100 000 times and that gave a mean score of 178 and a median of 172. The test saved information about the maximum score for the 100 000 runs which was 331 points and the minimum was 79 which gives a range of 252 points. Time consumption for 100'000 runs was 2.14s.



Graph 4.5: *Showing the data generated by 100'000 runs of the Dice-Based algorithm. The Y-axis is the number of occurrences and the X-axis is the total score.*



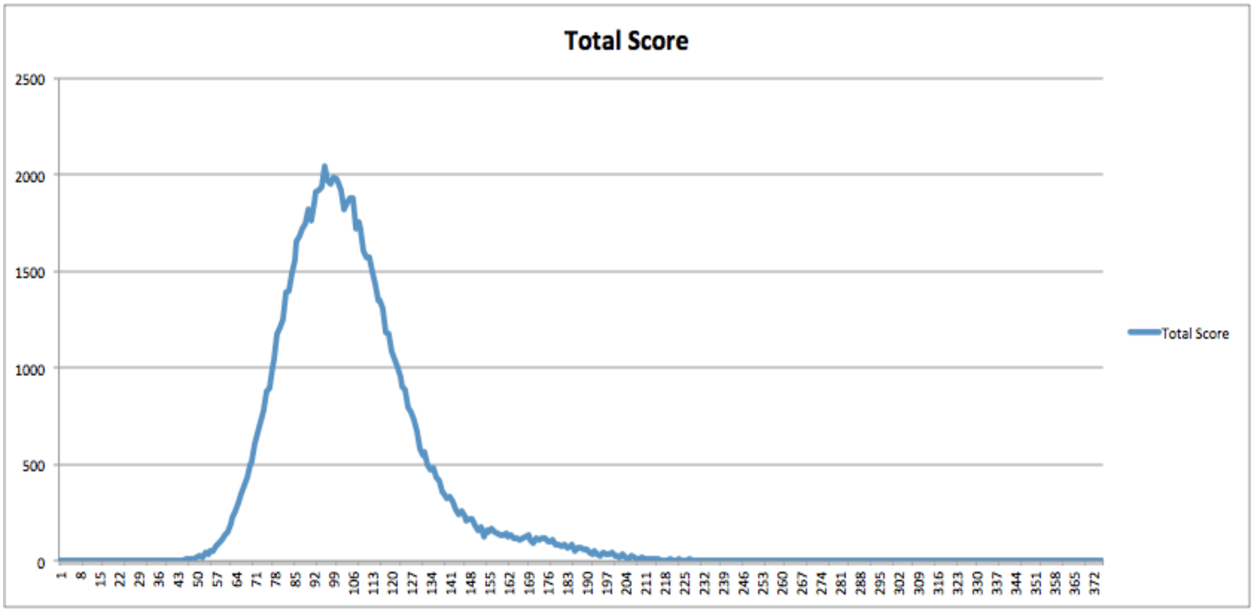
Graph 4.6: *Showing the data generated by 100'000 runs of the Dice-Based algorithm, the X-axis consists of distinct ranges of 10 and the Y-axis is the percentage in those ranges.*

120		129		4881	4,88%	8,75%
130		139		7842	7,84%	16,60%
140		149		10137	10,14%	26,73%
150		159		10989	10,99%	37,72%
160		169		10349	10,35%	48,07%
170		179		8776	8,78%	56,85%

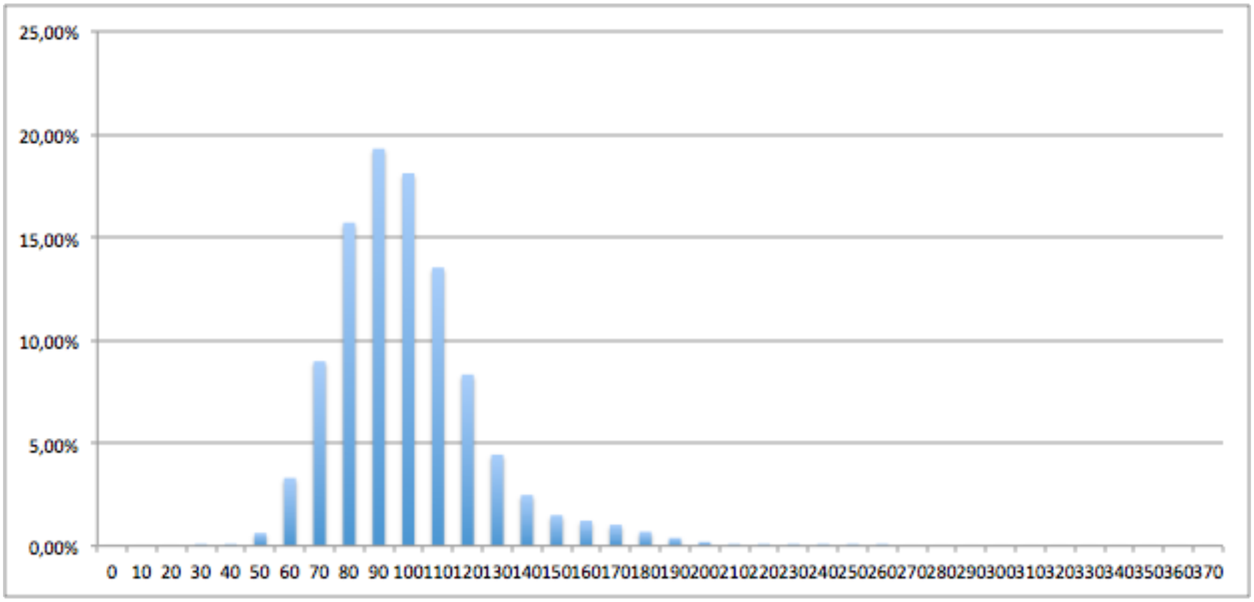
Table 4.3: *Showing the interesting data range for the Dice-based Human algorithm.*

4.1.3 Forced Scoring Algorithm

The *Forced Scoring Algorithm* has been run in a similar performance test that ran the algorithm 100 000 times and that gave a mean score of 104 and a median of 101. The maximum score for the 100 000 runs was 263 points and minimum was 38 which gives a range of 225. Time consumption for 100'000 runs was 1.23s.



Graph 4.7: *Showing the data generated by 100'000 runs of the Forced Scoring algorithm. The Y-axis is the number of occurrences and the X-axis is the total score.*



Graph 4.8: *Showing the data generated by 100'000 runs of the Forced-Scoring algorithm, the X-axis consists of distinct ranges of 10 and the Y-axis is the percentage in those ranges.*

Range of score				Number of occurrences	Percentage	Percentage (Cumulative)
70		79		8971	8,97%	12,96%
80		89		15707	15,71%	28,67%
90		99		19300	19,30%	47,97%
100		109		18110	18,11%	66,08%
110		119		13526	13,53%	79,60%

Table 4.4: *Showing the interesting ranges for the Forced Scoring Algorithm.*

4.2 Human Trials

4.2.1 Manual test of the dices

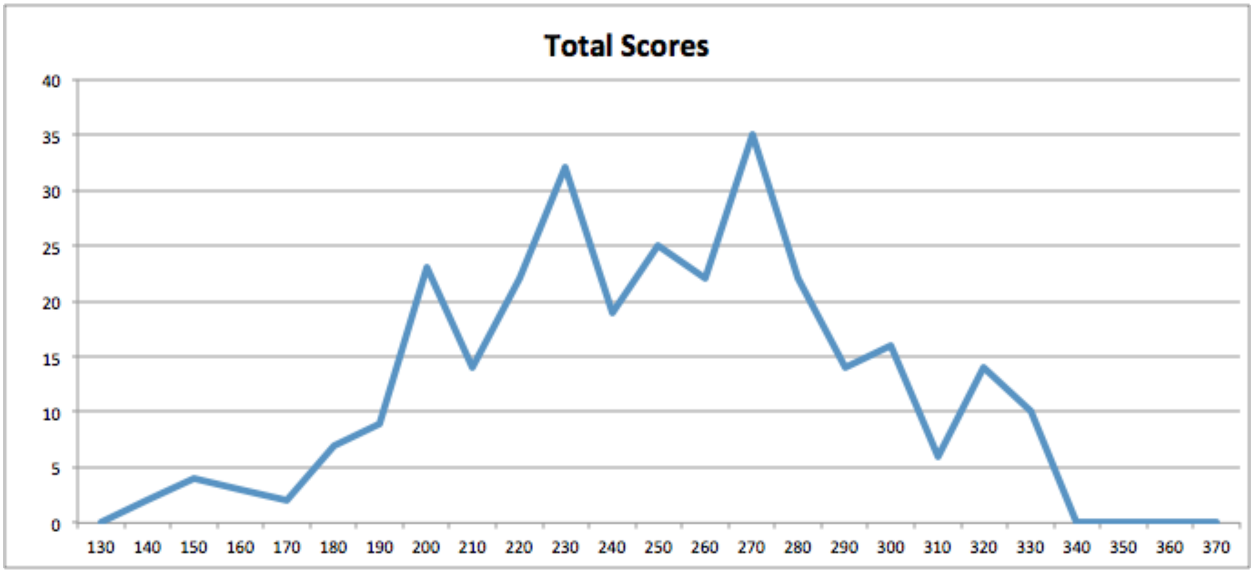
Before the human trials could be conducted the materials that would be used, the dices, were checked to avoid invalidities in the randomness of the dices. The expected score of this trial was 3.64. The complete results are presented in the table below.

Dice face	Number of occurrences	Percentages
1	76	15,2 %
2	84	16,8 %
3	84	16,8 %
4	91	18,2 %
5	90	18,0 %
6	85	17,0 %
Total	500	100,0 %

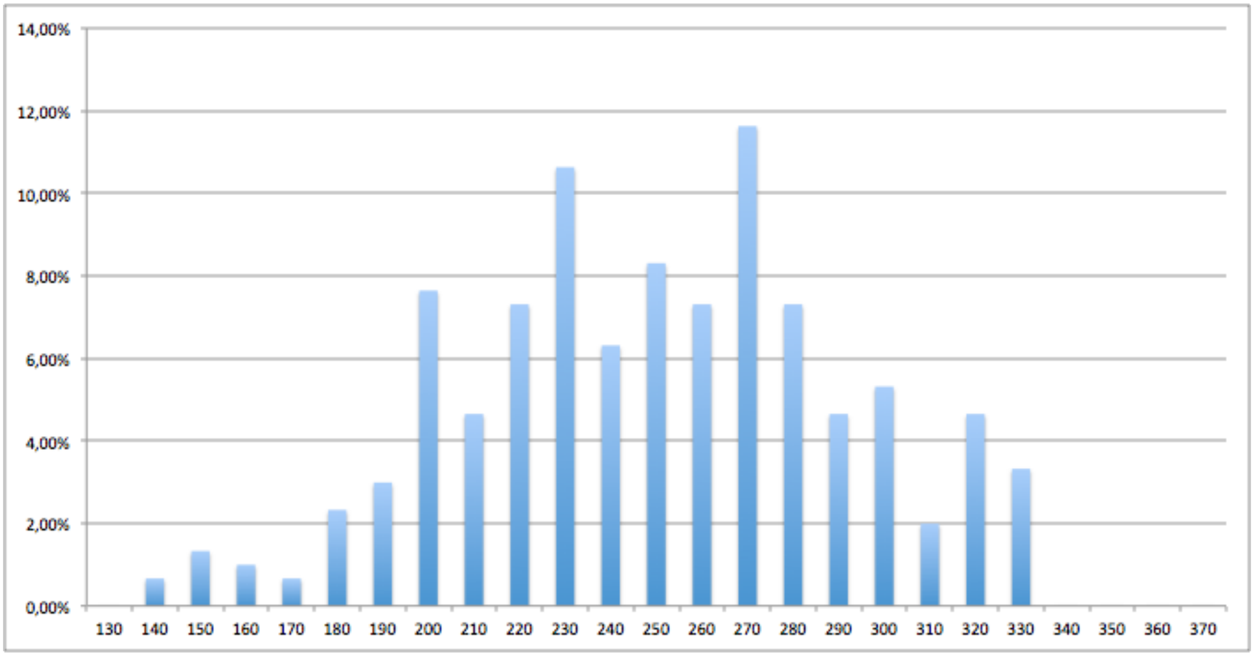
Table 4.5: *Showing the frequency at which the values occurred on the dice faces.*

4.2.2 Results of humans playing

The Human trials consisted of 6 people playing the game and generating 300 results. The mean from the human trials was 253.82 and the median was 253, so both valid averages. The maximum score was 333 and the minimum score was 148.



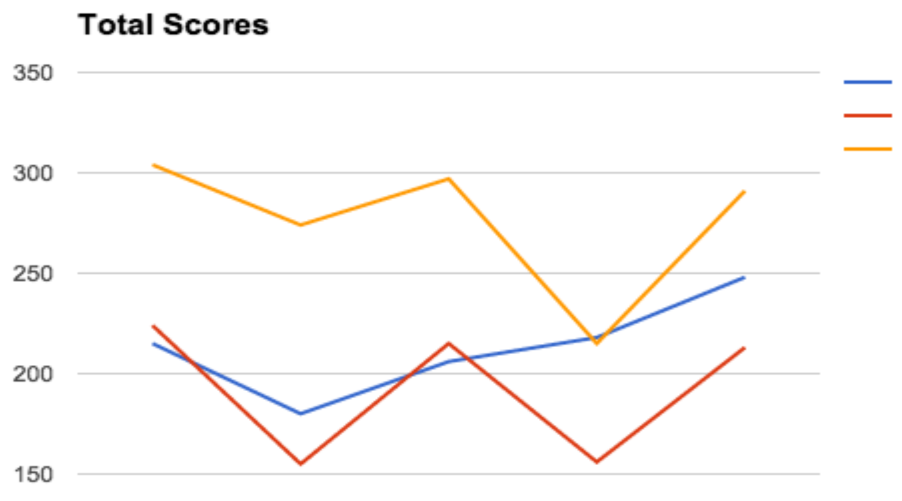
Graph 4.9: *Showing the score obtained from 300 games of Yahtzee played by Humans. The Y-axis is the number of occurrences and the X-axis is the total score.*



Graph 4.10: *Showing the score obtained from 300 games of Yahtzee played by Humans, the X-axis is distinct ranges of 10 and the Y-axis is the percentage of scores in the range.*

4.3 Same-value tests

The same-value tests were run 5 times. During these tests, the algorithms made 5*15*3 = 225 different selections.



Graph 4.11: *Showing the score obtained from 5 runs with the three algorithms. The blue line represents the Scorecard-based Human Algorithm, the red line represents the Dice-based Human Algorithm and the orange line is the Optimal Algorithm.*

Game number	*Dice-based Human Algorithm*	*Scorecard-based Human Algorithm*	*Optimal Algorithm*
1	*224*	*215*	*304*
2	*155*	*180*	*274*
3	*215*	*206*	*297*
4	*156*	*218*	*215*
5	*213*	*248*	*291*

Table 4.6: *Showing the score obtained from 5 runs with the three algorithms.*

Round 1		Throw result	Dice-based algorithm	Scorecard-based algorithm	Optimal algorithm
	1st throw	5,5,1,6,6			
	Keep		5,5,6,6	6,6	6,6
	2nd throw		5,5,6,6,6	6,1,6,6,6	6,1,6,6,6
	Keep		5,5,6,6,6	6,6,6,6	6,6,6,6
	Result		5,5,6,6,6	6,6,6,6,6	6,6,6,6,6
	Score set in category:		Sixes	Sixes	Yahtzee

Table 4.7: *Showing the decisions made by the three algorithms for Game 1 and the first round.*

Table 4.7 shows decisions made by the different algorithms during the first round. That means all categories were available and the conditions for each algorithm were alike. However, they made different decisions when deciding what to keep and how to score. The Dice-based Human Algorithm decided to try for a *Full house*, but when receiving one (note: highest possible *Full house*), it changed to score the combination in *Sixes*. The Scorecard-based Human Algorithm and the Optimal Algorithm wanted to keep the sixes and received five sixes, but chose different scoring options. The Optimal Algorithm received three sixes in *Sixes* during the fourth round and the bonus limit was reached even though it scored in *Yahtzee* instead of *Sixes*.

Round 4		Throw result	Dice-based algorithm	Scorecard-based algorithm	Optimal algorithm
	1st throw	1,3,2,4,1			
	Keep		1,2,3,4	4	1,1
	2nd throw		1,2,3,4,5	5,6,6,4,1	1,6,6,6,1
	Keep		1,2,3,4,5	5	6,6,6
	3rd throw / Result		1,2,3,4,5	1,5,6,6,6	2,6,6,6,6
	Score set in category:		Small straight	Three of a kind	Four of a kind
Round 6					
	1st throw	6,3,6,1,2			
	Keep		6,6	2	6,6
	2nd throw		6,4,6,6,2	2,4,3,6,2	6,6,6,4,2
	Keep		6,6,6	2,6,2	6,6,6,4
	3rd throw / Result		4,6,6,6,6	2,2,6,6,6	4,4,6,6,6
	Score set in category:		Four of a kind	Full house	Full house

Table 4.8: *Showing the decisions made by the three algorithms for Game 2 and the 4th and 6th round.*

Table 4.8 shows decisions made by the three algorithms for the second game and its fourth and sixth round. The Dice-based Human Algorithm had all categories available, except *Ones*, *Sixes* and *Two pairs*. The Scorecard-based Human Algorithm had results in *Sixes*, *Two pairs* and *Four of a kind* and the Optimal Algorithm had scored in *Fives*, *Sixes* and *Large straight*. Even though no algorithm had *Sixes* available they decided to keep the sixes to receive a high score in some other category. The Dice-based Human Algorithm decided to try to receive a *Small straight* due to having to re-roll only one of the dices. After the fourth round all the algorithms score in *Threes*.

During the sixth round, the Optimal Algorithm and the Dice-based Human Algorithm chose to save the sixes again to receive a high value in some category. When receiving three sixes the algorithm takes in consideration that the *Four of a kind* is scored and therefore it also saves the four to increase the chances of a *Full house* and succeeds. The Scorecard-Based Human Algorithm saved the dice displaying 2, but chose to re-roll dices to receive a *Full house* after the first roll, but with a lower score.

Round 3		Throw result	Dice-based algorithm	Scorecard-based algorithm	Optimal algorithm
	1st throw	6,4,3,4,4			
	Keep		6,4,4,4	4,4,4	6
	2nd throw		6,4,6,4,4	3,4,6,4,4	6,4,1,6,6
	Keep		6,4,6,4,4	4,4,4	6,6,6
	3rd throw / Result		4,4,4,6,6	2,4,4,4,6	6,6,6,6,3
	Score set in category:		Full house	Three of a kind	Sixes

Table 4.9: *Showing the decisions made by the three algorithms for Game 3 and the 3rd round.*

During the third game the three algorithms all scores in *Threes* and *Fours* the first two rounds. The third round is displayed in Table 4.9. During the third round the Optimal Algorithm decides not to save the three fours because of the *Fours* being scored and saves the dice displaying a six. This is rewarding and the Optimal Algorithm receives four sixes, while the Dice-Based Human Algorithms saves for a *Full house*, but the Scorecard-Based Algorithm does not receive any other four, and uses three fours for a *Three of a kind* which is not profitable.

The Dice-Based Human Algorithm scores 12 points in *Sixes* during the 7th round and the Scorecard-Based Human Algorithm scores 18 points in *Sixes* in during the 6th round. That means that the Dice-Based Human Algorithm might not receive the upper bonus and the Scorecard-Based Human Algorithm received 91 points less than the Optimal Algorithm, partly based on this move.

Round 1		Throw result	Dice-based algorithm	Scorecard-based algorithm	Optimal algorithm
	1st throw	3,4,4,2,1			
	Keep		4,4	4,4	4,4
	2nd throw		5,4,4,3,2	5,4,4,3,2	5,4,4,3,2
	Keep		4,4	4,4	4,4
	3rd throw / Result		4,4,4,6,6	4,4,4,6,6	4,4,4,6,6
	Score set in category:		Fours	Fours	Full house

Table 4.10: *Showing the decisions made by the three algorithms for Game 4 and the 1st round.*

Table 4.10 shows the first round for the fourth game. All categories are available for all algorithms, and all algorithms choose to save the fours during the throws. They all receive three fours and two sixes in the final throw, but they score in different categories. The Dice-based and Scorecard-Based algorithms decides to score in *Fours* because three of the same values is a good start for bonus, but the Optimal Algorithm scores in *Full house*.

Round 6		Throw result	Dice-based algorithm	Scorecard-based algorithm	Optimal algorithm
	1st throw	3,2,5,3,2			
	Keep		2,2	2,2	2,2
	2nd throw		3,2,6,3,2	3,2,6,3,2	3,2,6,3,2
	Keep		2,2	6	2,2
	3rd throw / Result		1,2,2,2,6	2,3,6,6,6	1,2,2,2,6
	Score set in category:		Twos	Sixes	Twos
Round 7					
	1st throw	2,6,4,6,2			
	Keep		2,2,6,6	2,6,2,6	6,6
	2nd throw		2,6,2,6,2	2,6,2,6,2	5,6,2,6,3
	Keep		6,6	2,6,2,6,2	6,6
	3rd throw / Result		3,4,5,6,6	2,2,2,6,6	5,6,4,6,2
	Score set in category:		Sixes	Twos	Sixes

Table 4.11: *Showing the decisions made by the three algorithms for Game 4 and its 6th and 7th round.*

5. Discussion

5.1 The Algorithms

The algorithms that aim to simulate human behaviour might be too simple, mostly in terms of not using any probability. They might be too calculation heavy, since humans regularly do not compute probabilities and expected values by themselves.
One source of error might be that even though we know that our optimal algorithm calculates the correct expected values for the different states, it might not make the correct choices based on this.

Runtime issues occurred mostly for the optimal algorithm. These problem occurred because the algorithm needed 13,8GB of RAM and the environment in which it was run had 16GB of RAM, so if too many other programs ran at the same time the algorithm was started it would crash. These issues were addressed mostly with a reboot or closing of other programs using high amount of memory.

5.2 Dice Tests

Since dices are the main, and only, source of randomization in Yahtzee, we performed test to assure that the dices were sufficient for this study.

5.2.1 Virtual Dices

Tests where conducted on the random number generators from both Java and C++ and from these tests we concluded that while both where sufficiently random, the C++ rand() method generated the same dice values each time we built the program whereas the Java Random.NextInt() generated different values from different runs which gave the impression that it was more randomized and therefore Javas random method was chosen. The dices values generated by the Random.NextInt() was validified through a test that generated 3 million dice values and the expected value from these tests were 3.5001, which is close enough to the theoretical expected value of 3.5. From this it is possible to draw the conclusion that the environment produced was sufficient for this study, and preconditions for the performance test is fulfilled.

5.2.2 Real Dices

The tests conducted on the real dices suggest that they had a fair distribution between the different values and that they therefore have an equal weight distribution. What could have been done further would have been to attempt to roll each dice separately to avoid individual dices differing in weight distribution. To draw perfect conclusions more tests would have been optimal to remove any error factors. In this study however it was decided that the test conducted was enough to prove the validity of the dices since the human trial was not the focus of the study.

From the data under heading 4.1.4 Human Trials we can draw the conclusion that these dices are sufficient for this study.

5.3 Performance test

Due to the dices being sufficiently random these tests were a great source of data. We can conclude that this is the best way to generate large amounts of data to examine trends in the way the different algorithms score.

5.3.1 Analysis of graphs and data

To analyze the different graphs, we look for certain key characteristics, such as local maximas, and draw conclusions based on these.

Optimal Algorithm

The Optimal Algorithm has the highest mean, median, maximum and minimum values. From this, the conclusion that it was the best performing algorithm can be drawn. However, it lacked in the fact that it was time consuming; it was more than 22’759 times slower than the slowest of the non-optimal algorithms. Another interesting result is that the most common range of values was not 210-219 (consisted of 8.20 %) where the mean and median is located; instead the most common range is 230-239, which contained 9.41 % of the scores. This suggest that we have more data in a shorter range above 219 than below it, where the data is more equally distributed and this is supported by Graph 4.2 as well.

The most distinctive characteristic of Graph 4.1 is that the values seem to have two extreme values, one is at approximately 180 and is neither a local maxima nor is it a minima and the other is a global maxima at approximately 235 points. The distance between these are 55 points and therefore it can be concluded that most of that difference is receiving the upper category bonus or not. It could be argued that the score for *Yahtzee* is 50 points and that it could be the difference, but since the chance of receiving *Yahtzee* is so small (1/7776 in one roll and 1/29 in three rolls) it is not likely that the second maxima is from scoring the *Yahtzee* category.

While this is a strong strategy it is not usable at all for humans playing casually, the thought of trying to compute several millions of states or checking through a graph containing all these states to find the correct one and then choose a path is hardly recommended, but as far as computer strategies go it is still optimal.

Scorecard-based Human Algorithm

The Scorecard-based algorithm has lower mean, median, maximum and minimum values compared to the optimal algorithm, but this algorithm was not completely outperformed. It scored 14 points less as mean and its median was 16 points lower. This is great for an algorithm that does not calculate expected score for future turns but only cares about maximizing the current round’s score in the scorecard.

As for Graph 4.3 it provides us with an interesting scoring trend, the graph depicted has three stationary points, one local maxima (at approximately 140), one local minima (at approximately 170) and one global maxima (at approximately 210). This mimics the Optimal algorithm in having different stationary points and that these points are most likely the results of either the upper category bonus or from receiving a lucky *Yahtzee*. Since the difference from the first maxima to the second is roughly 70 points it also suggest that over obtaining the upper bonus it probably scored higher in the categories *Fives* and *Sixes* since this means that the algorithm has a higher chance of obtaining bonus as well as a higher overall score.

Dice-Based Human Algorithm

The Dice-based algorithm has a mean and median which is significantly lower than that of the Optimal but a maximum score of only 10 points lower than that of the optimal. This suggest that this approach is more greedy in the way that it wants to maximize the score but does so without taking the probability of failure into consideration, Although this is also contradicted by this algorithm having the lesser range in comparison to the Optimal and the Scorecard-based Algorithms.

In Graph 4.5 and 4.6 we can see that this algorithm, like the Optimal algorithm has two stationary points, this algorithm has one that is not a local maxima nor is it a minima and one global maxima. When compared to the Graph 4.1, the graph of the Optimal Algorithm, we can see that not only are the values lower but it is a mirror image Graph 4.1 which means that the global maxima is lower than that of the other stationary point. Therefore the conclusion is drawn that Dice-Based approach fall short of the Optimal Algorithm not only because of lower average scores which is a result of missed bonuses.

If this approach could be set to maximize the score in the upper categories while maintaining the score in the lower categories it would rival the Scorecard-Based Algorithm especially since the range of all the values is the least.

Forced Scoring Algorithm

This algorithm performed, as expected, by far the worst of all algorithms, but because of it we had a worst-case scenario measurement to compare all the other algorithms against. No one should use this strategy while playing, if the goal is to receive a high score.

The Graph 4.7 and 4.8 shows a close to normal distribution curve and that suggest this algorithm does not achieve bonus nor anything other than a lucky score once in a while.

5.3.2 Conclusions

To conclude the performance test part it would be all too easy to call the Optimal Algorithm a winner. Although it had the highest scores the time consumption is major which is why the recommended algorithm from this analysis differ depending on your preferences. If your goal is solely to maximize the score, use the Optimal Algorithm, if you do not have days to wait use the slightly less optimal but more efficient Scorecard-Based Algorithm, which works to maximize the score for each category.

5.4 Same-value test

The Same-value tests collect data about the different choices made by the different algorithms and the decisions they make that affect the final score. This data is difficult to present in a way that is easy to read. Even though the same-value test was not run many times, a large amount of data was generated.

For most rounds of the game, the same-value test might not be perfect, because the algorithms must choose to score an available category. This means that the decisions are not only based on the algorithm’s decision making for this turn, but also on the previous choices. However, this test is still a good measurement for where the algorithms choose differently and which the decisions are that affect the total score in a positive or negative way and a few of the most important decisions are made by the algorithms with the same conditions. This means that by analyzing what the algorithm that scored the best and the other algorithms it should be feasible to find where the incorrect decisions were made.

5.4.1 Comparison of the algorithms

Graph 4.11 and Table 4.6 shows the total scores during the 5 rounds, and it is clear that the optimal algorithm makes the best decisions when playing. Only during the fourth game is its total score beaten and it is by the Scorecard-based Human Algorithm with three points. This means that the Optimal Algorithm makes the best decisions when playing.

The data presented in Table 4.7 is interesting in the study because of the different scoring options. The Optimal Algorithm decides that because this being the first round of the game, it is possible to receive at least three sixes later on and therefore being granted 50 points at the end of the game is preferable to scoring in *Sixes* with a bonus. The algorithm receives three sixes in another round and reaches the bonus limit. That means that by scoring in *Yahtzee* during the first round, the Optimal Algorithm earns approximately 50 points.

During the second game and the data presented in Table 4.8, the Optimal Algorithm did not only go for the available upper categories (*Ones* to *Sixes*), but saved sixes to score well in another category. The Scorecard-based Human Algorithm prefer to save the values that it needs to receive a bonus, while the Optimal Algorithm also considers maximizing the score all categories.

The Optimal Algorithm trends to save sixes if it is possible to always maximize its score. That is presented in Table 4.9 when it does not keep the *Three of a kind* the fours represent, but re-rolls all dices except the six. This is not always profitable, but the sixes are preferred when playing Yahtzee and if they are saved the score will probably be higher.

In Table 4.10 is the first round during the fourth game presented. The *Full house* is almost the best while three fours is not the best. This is the game where its score is beaten by another algorithm, but that is probably based on the data presented in Table 4.11. These the two rounds where the algorithms score in *Twos* and *Sixes*. The Scorecard-based Human Algorithm decides to change from saving the dices displaying two to saving the six, which is computationally the incorrect move. During the last throw it receives three sixes and is able to score well in *Sixes*. The Optimal Algorithm scored well in the second category, but did not receive more than two sixes during round 7 and had to score only 12 in *Sixes*. That is why the Scorecard-based Human Algorithm had a better total score in the fourth game.

5.4.2 Conclusions

The Optimal Algorithm is slow to run, but if maximizing the score is desired it is the best algorithm to use. It does not have a predefined strategy because the decisions are carefully calculated and every decision is important. The Optimal Algorithm does not take any unnecessary risks, which is why it rarely scores extremely low but that is the reason the other algorithms might beat its score. The strategy the Optimal Algorithm uses is that if it is reasonable, saving sixes is a good choice to maximize the score. It also tries to receive a score that is close to or over the expected value in each category to have the best possible score while the other algorithms considers any value a good choice. This technique is usually referenced as minimizing the deviation.

5.5 Human Trials

In order to measure the human trials against the other algorithms we think of these six people playing as an algorithm using reinforcement learning, a sort of prediction learning [4] which is also a technique in machine learning and artificial intelligence [5].
The people chosen for this trial were a homogenous group of people that are all students at university level. A certain set, half of the group, are also related which might suggest that they have similar cognitive patterns. Whilst these factors might have affected the outcome of these tests, the test subjects quite clearly outperform our optimal algorithm, which can be deducted from Graphs 4.9 and 4.10.
As to why these test subject outperformed the optimal algorithm the two answers must be that the optimal algorithm does not use a perfectly optimal strategy and that the data generated from the 200 trials is not a statistically valid sample, at least not in comparison with the amount of data collected for the algorithms, and therefore luck might have influenced the end results.
To further the study of human Yahtzee strategies and reinforcement learning, which is a field in machine learning and artificial intelligence [5], the implementation of an algorithm learning as the test subject have done, over a large amount of games, as well as further trials with humans who play the game of Yahtzee regularly.

6. Conclusions

To conclude the analysis of this study we would like to point out the strength of the different algorithms:

- The Optimal Algorithm provides a sure way of obtaining a relatively high score but with great time consumption.
- The Scorecard-Based Human Algorithm provides a relatively strong score while being calculation modest.
- The Dice-Based Human Algorithm provides a lesser score while being extremely calculation modest, with a few optimizations, most of the consisting of probability calculations, it could be a strong competitor.
- The Forced Scoring Algorithm provides nothing while playing and only provided this study with a lowest scoring algorithm that was never expected to score above average. This algorithm is not suitable for solitaire Yahtzee and might be used in the multiplayer version, but that would also eliminate all need of a strategy and base the score only on luck.

The Human Trials clearly proves the validity of using human cognitive thinking versus the different algorithms. As previously stated it is believed in this study that reinforcement learning, learning from the consequences of previous actions, is the algorithm used by frequent Yahtzee participants. Since the human trials were not the main focus of this study it would be interesting to do a larger study in this area and see if the hypothesis that reinforcement learning can perform optimally. Testing that theory on humans might prove difficult since it takes a lot of time to play a significant amount of games, but testing this would be easier with the help of reinforcement learning algorithms from the area of machine learning.

7. References

[1] Glenn, J (2006), 'An optimal strategy for Yahtzee', Technical Report, Loyola College in Maryland. Available at: http://www.cs.loyola.edu/~jglenn/research/optimal_Yahtzee.pdf [Accessed: 15th March 2015]

[2] Alga Spel (2009), 'Super Yatzy', Swedish game rules for yatzy, Brio AB/ALGA. Available at: http://www.algaspel.se/~/media/Alga/Files/Rules/flersprakiga/38018949_QUBE_Superyatzy.ashx [Accessed: 20th March 2015]

[3] Glenn, J (2007), 'Computer Strategies for Solitaire Yahtzee', Technical Report, Loyola College in Maryland. Available at: http://cswww.essex.ac.uk/cig/2007/papers/2046.pdf [Accessed: 27th March 2015]

[4] Niv, Y (2009), 'Reinforcement learning in the brain', Scientific Report, Princeton University Available at: http://www.princeton.edu/~yael/Publications/Niv2009.pdf [Accessed: 15th April 2015]

[5] Kaelbling, L. Littman, M. Moore, A. (1996) *Reinforcement Learning: A Survey*. Journal of Artificial Intelligence Research 4 (1996). p. 237-285.
Available at: http://www.jair.org/media/301/live-301-1562-jair.pdf [Accessed: 20th April 2015]

Appendix A - Performance Test Tables

Table presenting values from the Optimal Algorithm.

Range				Optimal Algorithm		
from		to		Count	Percent %	Accumulative %
80		89		2	0,00%	0,00%
90		99		114	0,11%	0,12%
100		109		639	0,64%	0,76%
110		119		1041	1,04%	1,80%
120		129		1804	1,80%	3,60%
130		139		2594	2,59%	6,19%
140		149		3360	3,36%	9,55%
150		159		4133	4,13%	13,69%
160		169		4821	4,82%	18,51%
170		179		5255	5,26%	23,76%
180		189		5477	5,48%	29,24%
190		199		6061	6,06%	35,30%
200		209		6836	6,84%	42,14%
210		219		8202	8,20%	50,34%
220		229		9102	9,10%	59,44%
230		239		9411	9,41%	68,85%
240		249		8885	8,89%	77,74%
250		259		7536	7,54%	85,27%
260		269		5801	5,80%	91,07%
270		279		3941	3,94%	95,02%

280		289		2396	2,40%	97,41%
290		299		1305	1,31%	98,72%
300		309		661	0,66%	99,38%
310		319		350	0,35%	99,73%
320		329		174	0,17%	99,90%
330		339		91	0,09%	99,99%
340		349		8	0,01%	100,00%

Table with the values from the Scorecard-based Human Algorithm.

Range				Scorecard-based Human Algorithm		
from		to		Count	Percent %	Accumulative %
60		69		6	0,01%	0,01%
70		79		20	0,02%	0,03%
80		89		139	0,14%	0,17%
90		99		411	0,41%	0,58%
100		109		1062	1,06%	1,64%
110		119		2075	2,08%	3,71%
120		129		3451	3,45%	7,16%
130		139		4487	4,49%	11,65%
140		149		4686	4,69%	16,34%
150		159		4275	4,28%	20,61%
160		169		4032	4,03%	24,64%
170		179		4907	4,91%	29,55%
180		189		6866	6,87%	36,42%
190		199		9987	9,99%	46,40%
200		209		11577	11,58%	57,98%
210		219		10977	10,98%	68,96%
220		229		8505	8,51%	77,46%
230		239		5842	5,84%	83,31%
240		249		4567	4,57%	87,87%
250		259		3967	3,97%	91,84%
260		269		3499	3,50%	95,34%
270		279		2555	2,56%	97,89%

280		289		1373	1,37%	99,27%
290		299		540	0,54%	99,81%
300		309		151	0,15%	99,96%
310		319		36	0,04%	99,99%
320		329		7	0,01%	100,00%

Table with the values from the Dice-based Human Algorithm

Range				Dice-based Human Algorithm		
from		to		Count	Percent %	Accumulative %
70		79		4	0,00%	0,00%
80		89		56	0,06%	0,06%
90		99		262	0,26%	0,32%
100		109		1036	1,04%	1,36%
110		119		2514	2,51%	3,87%
120		129		4881	4,88%	8,75%
130		139		7842	7,84%	16,60%
140		149		10137	10,14%	26,73%
150		159		10989	10,99%	37,72%
160		169		10349	10,35%	48,07%
170		179		8776	8,78%	56,85%
180		189		7241	7,24%	64,09%
190		199		6467	6,47%	70,55%
200		209		6245	6,25%	76,80%
210		219		5843	5,84%	82,64%
220		229		5321	5,32%	87,96%
230		239		4288	4,29%	92,25%
240		249		2883	2,88%	95,13%
250		259		1829	1,83%	96,96%
260		269		1098	1,10%	98,06%
270		279		707	0,71%	98,77%
280		289		551	0,55%	99,32%

290		299		382	0,38%	99,70%
300		309		207	0,21%	99,91%
310		319		70	0,07%	99,98%
320		329		20	0,02%	100,00%
330		339		1	0,00%	100,00%

Table with values from the Forced-Scoring Algorithm

Range				Forced-Scoring Algorithm		
from		to		Count	Percent %	Accumulative %
30		39		1	0,00%	0,00%
40		49		58	0,06%	0,06%
50		59		629	0,63%	0,69%
60		69		3300	3,30%	3,99%
70		79		8971	8,97%	12,96%
80		89		15707	15,71%	28,67%
90		99		19300	19,30%	47,97%
100		109		18110	18,11%	66,08%
110		119		13526	13,53%	79,60%
120		129		8327	8,33%	87,93%
130		139		4434	4,43%	92,36%
140		149		2472	2,47%	94,84%
150		159		1493	1,49%	96,33%
160		169		1227	1,23%	97,56%
170		179		1027	1,03%	98,58%
180		189		691	0,69%	99,27%
190		199		385	0,39%	99,66%
200		209		189	0,19%	99,85%
210		219		74	0,07%	99,92%
220		229		34	0,03%	99,96%
230		239		21	0,02%	99,98%
240		249		11	0,01%	99,99%

250		259		6	0,01%	99,99%
260		269		2	0,00%	100,00%

Appendix B - Human Trials Scores

Range						
From		to		count	%	Accumulative %
140		149		2	0,66%	0,66%
150		159		4	1,33%	1,99%
160		169		3	1,00%	2,99%
170		179		2	0,66%	3,65%
180		189		7	2,33%	5,98%
190		199		9	2,99%	8,97%
200		209		23	7,64%	16,61%
210		219		14	4,65%	21,26%
220		229		22	7,31%	28,57%
230		239		32	10,63%	39,20%
240		249		19	6,31%	45,51%
250		259		25	8,31%	53,82%
260		269		22	7,31%	61,13%
270		279		35	11,63%	72,76%
280		289		22	7,31%	80,07%
290		299		14	4,65%	84,72%
300		309		16	5,32%	90,03%
310		319		6	1,99%	92,03%
320		329		14	4,65%	96,68%
330		339		10	3,32%	100,00%