Optimal Yahtzee performance in multi-player games

Andreas Serra
aserra@kth.se

Kai Widell Niigata
kaiwn@kth.se

April 12, 2013

Abstract

Yahtzee is a game with a moderately large search space, dependent on the factor of luck. This makes it not quite trivial to implement an optimal strategy for it. Using the optimal strategy for single-player use, comparisons against other algorithms are made and the results are analyzed for hints on what it could take to make an algorithm that could beat the single-player optimal strategy.

Statement of Collaboration

We planned the project together and came up with which parts each one of us would do. Kai focused more on the AI bot while Andreas who already knew Python focused more on implementing the other strategies. The other parts of the report and implementation was done with us working in pairs.

Contents

1 Introduction

What makes the topic of optimal Yahtzee interesting is the way that it demonstrates the potential of being able to traverse a full search space in order to find an optimal solution. It is somewhat different from most other optimal solution finding problems in the way that the topic at hand also is dependent on the factor of luck and therefore an optimal algorithm does not per se imply perfect results. Our task will be to develop an optimal algorithm, compare it against other Yahtzee algorithms and analyze the resulting differences.

1.1 Problem Statement

A topic such as optimal Yahtzee can be researched from many different angles. In order for us to limit the range of research, we will focus mostly on answering the following questions.
1. How long does it take for our implemented optimal Yahtzee algorithm to run?
2. Does an optimal Yahtzee algorithm for single-player use necessarily also mean it being an optimal algorithm for multi-player use?
3. If not, what does it take to make an algorithm that can beat the single-player optimal algorithm in multi-player?
4. What are the average win-lose ratios for the single-player optimal algorithm compared to simpler algorithms such as greedy or random algorithms?

2 Background

Yahtzee is a dice game made by Milton Bradley. It got inspiration from such games such as Yacht and Generala and has been released in several different versions. The version that is used here is the non-Scandinavian variant.

2.1 Yahtzee Rules

Yahtzee is a game played with five dice and a scorecard. The goal of the game is to gather as many points as possible, which is done by getting different combinations of dice. There are in total thirteen different combinations divided in a lower section and an upper section.

The lower section is made up of the combinations:
Three-Of-A-Kind: Three of the five dice are of the same value, e.g. (1, 1,

1, 4, 3). The sum of the dice determines the points received.
Four-Of-A-Kind: Four of the five dice are of the same value, e.g. (2, 2, 2, 2, 5). The sum of the dice determines the points received.
Full House: Three are of the same value and the other two are of another value, e.g. (4, 4, 4, 2, 2). 25 points are given for this combination.
Small Straight: Four consecutive numbers, e.g. (1, 2, 3, 4, 3). The number of points given are 30.
Large Straight: Five consecutive numbers, e.g. (2, 3, 4, 5, 6). The number of points given are 40.
Yahtzee: Five of the same value, e.g. (6, 6, 6, 6, 6). The number of points given are 50.
Chance: Any combination. The number of points received is determined by the sum of the dice, e.g. the combination (3, 3, 5, 6, 2) gives 19 points.

The upper section is made up of six categories, one for each of the faces of a die. In these categories the points received are equal to the amount of eyes on the die face times how many dice are showing that specific face. If the total of points received in the upper section are 63 points or more, then 35 points are added to the total score, which is called a Yahtzee bonus [2].

Each combination can only be filled in once, i.e. if the player gets full house twice, he can only receive the points for it once. However, if the player gets another Yahtzee he receives 100 points on top of the points received in the matching upper category if it is not already filled in. This is one part of the so called Joker rule which also states that if the matching upper category is already filled in, then Yahtzee can be used as a wild card. I.e. the user can choose one of the currently unused lower categories and receive the maximum possible points of that category. If no lower category is available then an upper category must be filled in with a zero. If the player gets a second Yahtzee and have already filled in the Yahtzee category with a zero then the above Joker rule still apply, though the extra 100 points are not awarded [2].

Each turn starts with the player throwing all the five dice together. The player can now choose to either fill in a category or throw a subset of the dice again, which can be done up to two times. For example, the player throws (3, 3, 3, 2, 5) on the first throw, which is a three-of-a-kind, but an even better combination is e.g. Yahtzee, so the player can keep the three threes and hope for getting even more threes.

The player can always choose to fill in a zero in one of the unused categories if he wants to. Also if none of the available categories gives any points this has to be done.

There are thirteen turns in total so with luck and by having a good strategy the player can try to maximize his total score. A simple strategy can e.g. be to never go for a category where the points are determined by the sum of the dice if the first throw contains low values (1, 2, 3), but rather go for a category with fixed points, and vice versa.

2.2 Optimal Strategy

The current state of research considering the game of Yahtzee is that an optimal strategy for maximizing the average amount of points has been developed [1]. Furthermore research in developing an optimal multi-player strategy, an algorithm for maximizing the amount of victories, has been done but as of yet, a fully optimal strategy has not been found [3]. How the single optimal algorithm works is basically that we start at the end state of the game and then layer by layer calculate us towards the beginning state by means of dynamic programming. Through calculating the probability of getting specific dice rolls and going from one state to another we can for each state know what the potential (expected value) will be and in order to maximize ones score we then only need to make the decisions that makes the potential as high as possible.

3 Method

The first thing that has to be done is the implementation of the single optimal algorithm as described by Glenn, James article. After that, in order to evaluate the performance of the algorithm, alternative algorithms will also need to be implemented. When all of this is done, data about win-lose statistics will be gathered through battling the algorithms against each other. Statistics about what typical scores the algorithms get will also be gathered by letting the algorithms play single-player Yahtzee a large amount of times.

One could also try to implement an improved version of the single optimal algorithm using multi-player specific information, but since this makes the search space increase by a large amount we believe that the required resources to finish such a task would be out of scope for this project.

Comparing the single optimal algorithm against the other algorithms is done in order to evaluate the effectiveness of the algorithm when playing multi-player Yahtzee and gather data about whether it really can be a multi-player optimal algorithm or not.

3.1 Problem Representation

Yahtzee can be represented as a graph of states and so called roll-states being the vertices and the edges being the transition possibilities from one state/roll-state to another state/roll-state.

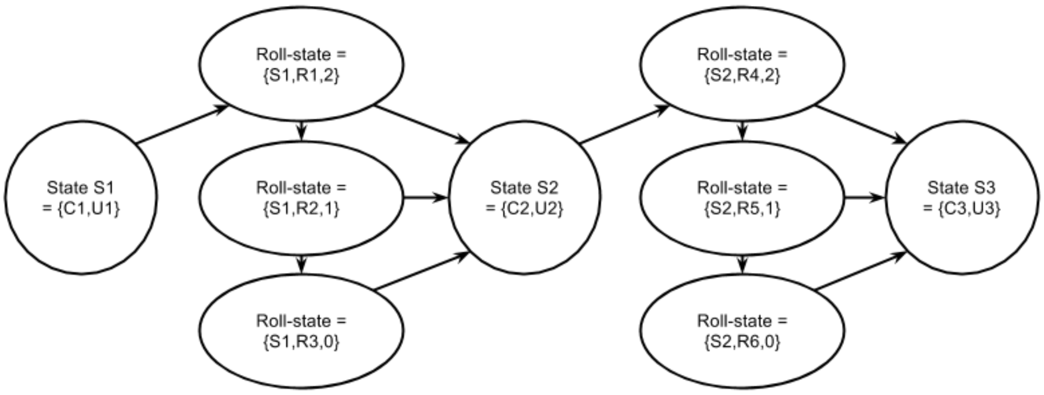

Figure 1: Connection between states and roll-states

The graph shows how a state moves to a roll-state and then finally to another state. Here S_1, S_2, S_3 are different states and $C_1, C_2, C_3, U_1, U_2, U_3$ are different categories and upper section scores. The roll-states are like states but also has information of what roll was obtained and how many rerolls are left.

3.2 State Definition

As we saw in the previous section, states play an important role in the optimal Yahtzee strategy, therefore a more formal definition is required:

Definition:

A state is defined as the categories that are used and how many points there are currently in total in the upper section. Also a flag indicating whether one has got Yahtzee is classified as a category. If we define the set of all category combination as C and the set of all possible upper point total scores as U, then the set of all states S_a, is just the Cartesian product of the two sets, i.e.

$$S_a = C \times U$$

so a state is just a tuple of a binary string representing the used categories and a number indicating what the upper section total score is.

To compute the number of states we only have to compute the size of the sets C and U. The size of U is 64 since the upper points total score can be any value in the range $0, 1, 2, ..., 63$. For the size of C we can just think of a 14 digit binary string where each digit corresponds to each category so that if the digit is 1, then the category is used and otherwise it is not. So the total number of such combinations are 2^{14}. The total number of states S_a are therefore:

$$|S_a| = |C \times U| = |C| \cdot |U| = 2^{14} \cdot 2^6 = 2^{20} = 1048576$$

However, every state is not reachable, since e.g. if one has not used any of the upper categories, then the upper section total score cannot be anything but 0. Finding the reachable states mathematically is rather hard, and therefore a programming approach was used. By using dynamic programming one can compute the reachable states more efficiently [1]. The number of reachable states $|S|$ were computed to 726016.

3.3 Roll-state Definition

Since the potentials for different states are highly dependent on probabilities of throwing certain rolls, there is also a need of having states which store information on what roll is obtained and how many rerolls can be done. It can also be seen in the graph, that between the states there are different roll-states which are necessary for the computation of the probabilities to go from a certain state to another. Formally we define a roll-state as:

Definition:
A roll-state is a triple containing a state, a roll and how many rerolls there are left, so symbolically:

$$p = (s, r, n) \in P, \quad s \in S, \quad r \in R, \quad n \in \{0, 1, 2\}$$

where R is the set of all distinct rolls, e.g. the roll (1, 2, 3, 4, 5) is the same as the roll (5, 4, 3, 2, 1). The total number of such rolls can be determined with the following reasoning: We want to choose five values (one for each die) out of six values with repetition and where the order doesn't matter. This is calculated by using the following expression:

$$\binom{m+n-1}{m}$$

where $m = 5$ is the number of values to choose, and $n = 6$ is the number of options to choose from, so:

$$|R| = \binom{5+6-1}{5} = \binom{10}{5} = 252$$

Therefore the total number of roll-states are:

$$|P| = |S \times R \times \{0,1,2\}| = |S| \cdot |R| \cdot |\{0,1,2\}| = 726016 \cdot 252 \cdot 3 = 548868096$$

3.4 Bots

In order to evaluate the performance of the single optimal strategy for multi-player use, a few other strategies will be implemented, which we call bots. The name bot is used since they are thought to compete against each other in multi-player battles. The different strategies are described below.

3.4.1 Single Optimal Bot

The single optimal bot works in the way that, given a state, find the maximum potential from that state. Therefore, the average score that this bot will get is just the potential score given the state that has 0 points in the upper section and no category used. To compute the potential score for each state, one can use recursion and compute it effectively with dynamic programming [1]. Since the computation of the potential scores is rather time consuming, computing all the potential scores beforehand and dumping them to a file is done.

To increase the efficiency of the bot, the probability of going from one roll or a subset of it to any other roll is calculated beforehand and stored in an array.

In the state definition it was mentioned briefly that just combining upper section points and categories available would include many unreachable states. In order to find the reachable states a dynamic programming algorithm was used as described in the article by Glenn, James.

3.4.2 Greedy Bot

The greedy bot plays like the single optimal bot but instead of going through the whole game, it just checks the next state. Since the greedy bot only checks the next state, generating the data file becomes unnecessary and therefore the algorithm can be implemented faster than the single optimal bot.

3.4.3 Random Bot

The random bot is, just as the name suggests, a bot which plays randomly, i.e. it has not a set rule of what dice to keep. What dice to keep is instead determined at random, with the probability of 1/4 of keeping each die.

When choosing which category to fill in, it will use the category that gives the most points and thus category selecting uses a greedy approach.

3.4.4 AI Bot

The AI bot is designed to play like a regular human, and therefore has a set of rules and priorities of which categories to use and what rolls to keep.

For instance, if the roll contains a three-of-a-kind, the algorithm now aims for a four-of-a-kind or Yahtzee, and therefore saves only the dice belonging to the three-of-a-kind. Since the score received in both three-of-a-kind and four-of-a-kind is determined by the sum of the dice, it is a bad choice to go for these categories if the dice contained in the combination are of low values (1, 2, 3) etc. Instead the algorithm tries to go for full-house or one of the upper categories in this case. What is considered low values is controlled by a threshold value. If a roll instead contains a straight, i.e. three consecutive numbers, the algorithm aims to get a small straight or large straight. In most cases the categories are already in use, and if that is the case, the algorithm can as the last resort reroll all the dice.

After obtaining the final roll, the algorithm now has to choose a category to fill in. Generally the algorithm just goes for the category that gives the most points. Though, there are exceptions to this rule, e.g. when a roll such as (1, 1, 1, 1, 6) is obtained, using the chance category, it will give 10 points, while the category ones will only give 4 points. If this is the case, the algorithm fill in the ones category instead (if it's not already used) since 10 points is considered a low score in the chance category. When the points exceed the average score of chance $(30+5)/2 = 17.5$ and the chance category gives the most of all categories, then the chance category is used.

4 Results

The gathering of data between the different algorithms has been done using a battling platform that we have called the battlefield. Using the battlefield, we were able to gather data both about the single-player statistics as well as of the multi-player statistics and as it supports arbitrarily many simultaneously playing bots we could use it to take statistics over how well each bot played against all the other bots at the same time as well.

4.1 Implementation

Our implementation was written in Python and therefore the running of the algorithm was quite slow. The first runs of the algorithm took a little over 24 hours to finish on a computer with an Intel Core i7 2600K 3.4 GHz CPU. We then proceeded to optimize our solution through pruning impossible states, generating all data such as the likelihood of getting a specific combination of dice beforehand, store temporary results such as roll-states in memory instead of having to generate them over and over as well as optimizing the state class and implementing more efficient memory structures.

The final implementation of the algorithm run on the same computer as above, took a total of 7 hours to generate the solution which we output to a file of size 9 MB. In order to use the actual algorithm we only need this file and therefore the generator is not required anymore once the data has been successfully generated.

The information that is stored in the file is the potential given each possible state. This is the bare minimum that the algorithm requires, one could also store intermediate steps such as roll-states but we noticed that the algorithm in actual games just needed a few seconds in order to come up with the best available move even when this information was not available. Having to store this information as well would also make the data file much larger since there are many more roll-states than there are states in the solution, so we decided not to store it.

4.2 Expected Score

The expected score (potential) of the single optimal algorithm was with bonus Yahtzees 254.589 points and without 245.871 points. These values were generated by the algorithm itself and could possibly be wrong, therefore in order to prove that they were correct we made the algorithm play a large amount of games and we noticed that the resulting mean values gradually converged towards the expected values and the algorithm was therefore deemed correct.

4.3 Comparison

After having played 10000 games between the single optimal bot and all of the other bots individually the following results were found:
Single optimal bot vs random bot: 99.68% win chance.
Single optimal bot vs AI bot: 80.86% win chance.
Single optimal bot vs greedy bot: 72.99% win chance.

This means that it is extremely unlikely that the random strategy will beat the optimal strategy, AI strategy will win every fifth game or so and the Greedy strategy will win 27% of the games.

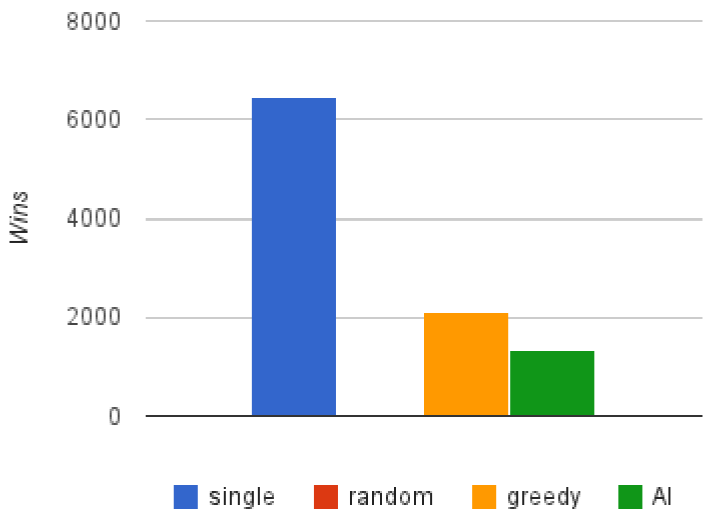

Figure 2: Diagram showing wins and losses for the different algorithms after 10000 games played

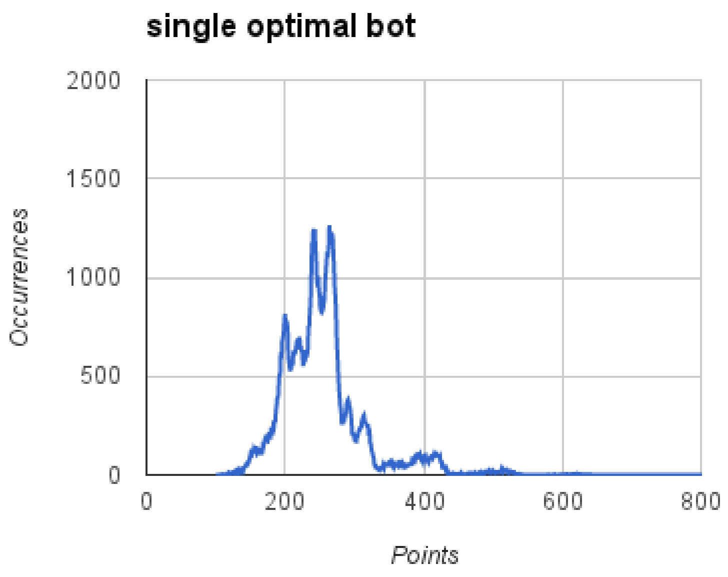

Figure 3: Graph showing how many times each score are obtained for the single optimal bot when run 100000 times

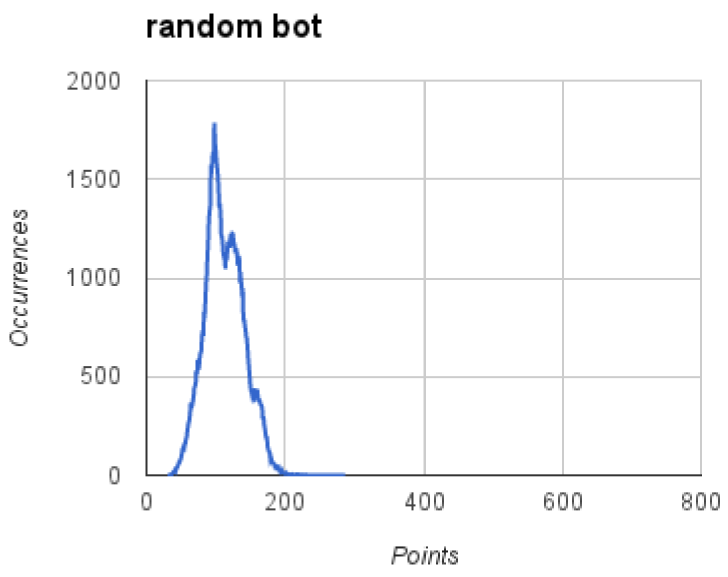

Figure 4: Graph showing how many times each score are obtained for the random bot when run 100000 times

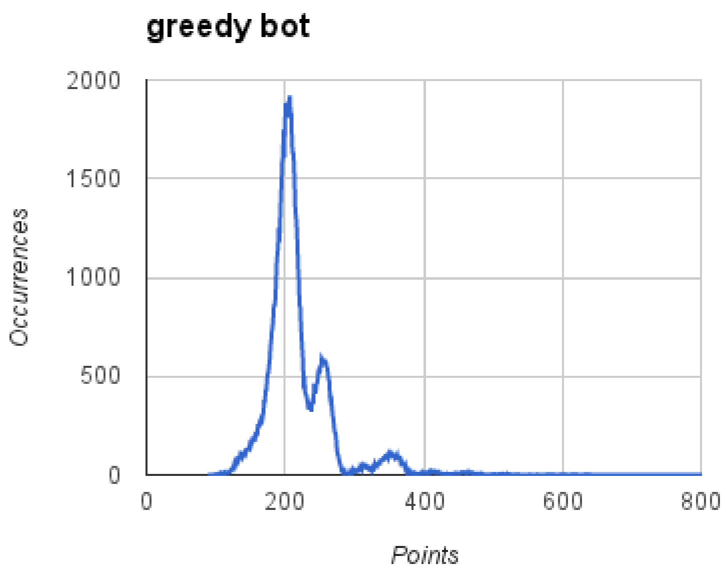

Figure 5: Graph showing how many times each score are obtained for the greedy bot when run 100000 times

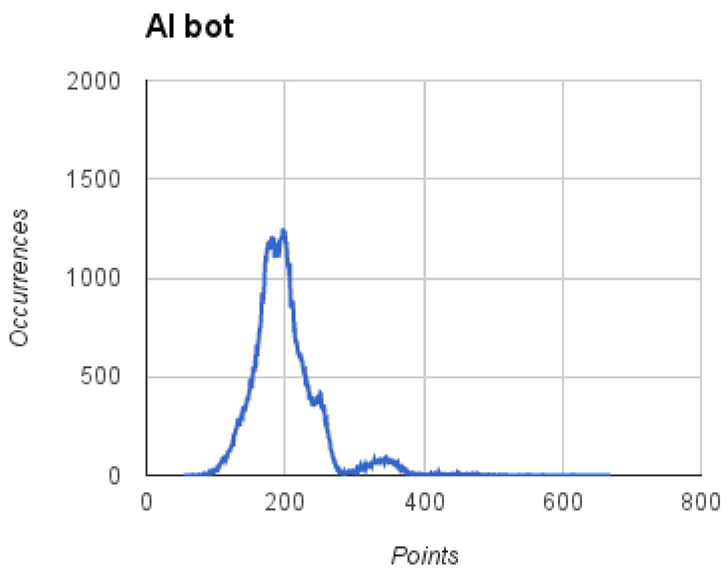

Figure 6: Graph showing how many times each score are obtained for the AI bot when run 100000 times

As can be observed in the graphs, the different bots have their peaks accordingly to the average scores which was expected, but a phenomenon we see in every bot except the random bot is that there are several peaks. The reason why this happens could be because, there are many ways to obtain a certain score. For example, getting 5 points as a total score can only be done in one way, that is only filling in the chance category with the roll (1, 1, 1, 1, 1). In the same way there are only a few ways to get a certain score for the higher scores as well.

One can also see that all bots besides the random bot has a great variance of scores with the single optimal bot having the most. This is probably because getting scores higher than 200 points often requires a Yahtzee, but getting a Yahtzee for the random bot is so unlikely since it has no strategy to go for a Yahtzee, but rather have to hope for the low probability of getting it.

5 Discussion

As can be seen in the data, random was the worst performing bot which is only natural since what it essentially does is roll all the dice once and then take the category that gives the highest amount of points. Since it does not save any dice and also does not try to get the upper-bonus it is no surprise that it also can not compete well against other bots. Even implementing such a simple heuristic as to save all of the dice when you have a Yahtzee makes the expected score raise by about 10 points so it does not take much to improve it.

The AI bot had several parameters which after some adjustment improved the score. For instance, one parameter controlled when the algorithm would consider going for a three-of-a-kind or four-of-a-kind over other categories. The performance was at best when this value was set to five, i.e. if a roll does not contain more than three fives or sixes, then go for other categories. This means that, e.g. for the roll (3, 3, 3, 3, 2) the algorithm would go for a full house or Yahtzee rather than four-of-a-kind. Setting the value to 6 didn't make any difference at all. The reason why 5 and 6 gives the best results might be because, with this rule, the worst possible three-of-a-kind that the algorithm would score would be $5 + 5 + 5 + 1 + 2 = 18$ points, which is higher than the average three-of-a-kind score of $(30 + 5)/2 = 17.5$ points. With the same reasoning the other parameters, like when to consider using chance over other categories was determined. The AI bot is just a simple heuristic so its performance is far from being optimal. As of now the AI bot does not even consider the upper section total score at all, so with some more thought the algorithm could be improved.

The greedy bot which is basically the single optimal bot without the data file uses the same algorithm as the single optimal bot and this therefore makes it play quite well but since it only takes the potential of the next state and not the potential of the whole game into consideration, it will make greedy decisions sometimes that are quite far from optimal.

None of the bots uses the multi-player feature of being able to accommodate ones play in order to score higher than the opposing players. It is a requirement that an algorithm that is able to win over the single optimal strategy use this information since without this information the algorithm can at best play just as well as the single optimal one, thus itself also being a single optimal algorithm. If an algorithm using that feature were implemented then it would be able to beat the single optimal strategy and since we know that such an algorithm exists we can therefore conclude that the single optimal strategy is not a multi-player optimal strategy [3].

6 Conclusion

Our results show that the single optimal strategy is also a well-playing strategy for multi-player use. Since the single optimal strategy maximizes the average score, the only way for an algorithm to be able to beat it is to have it being able to take the opponents current score and available categories into consideration when determining what to do. Research about this has already been done but unfortunately no statistics about win-lose ratios has been published [3]. What is certain is that an algorithm that maximizes the likelihood of reaching a specific score will win more than 50% of the time against the single optimal strategy. Since there still exists a lack of statistics and that the fully optimal multi-player strategy has not been developed yet we can therefore conclude that there exists further potential in the research of this topic.

References

[1] James Glenn. *An Optimal Strategy for Yahtzee*. May 2006. URL: `http://www.cs.loyola.edu/~jglenn/research/optimal_yahtzee.pdf`.

[2] Hasbro. *Instructions 1 or more players Yahtzee*. 2010. URL: `http://www.hasbro.com/common/instruct/Yahtzee.pdf`.

[3] Jakub Pawlewicz. *A Nearly Optimal Computer Player in Multi-player Yahtzee*. 2010. URL: `http://www.mimuw.edu.pl/~pan/papers/yahtzee.pdf`.