

# Optimal Yahtzee: An Artificially Intelligent Approach

Elwin Brown\*, Jason Gorelik†, R. Bowie Smith‡, and Justin Wilmot§

\*School of Computer Science and Electrical Engineering  
University of Maryland, Baltimore County  
elwin.brown3@gmail.com

†School of Computer Science and Electrical Engineering  
University of Maryland, Baltimore County  
jasong2@umbc.edu

‡School of Computer Science and Electrical Engineering  
University of Maryland, Baltimore County  
rsmith11@umbc.edu

§School of Mathematics and Statistics  
University of Maryland, Baltimore County  
justin.wilmot@gmail.com

**Abstract**—Yahtzee is a dice game based on the rules of poker. Like poker, the game is a combination of chance and strategy. This paper describes our research modeling the game of Yahtzee and applying various AI techniques to generate an autonomous agent that plays an optimal game of Yahtzee.

## I. BACKGROUND

Yahtzee is a dice game based on the rules of poker. Over the course of thirteen rounds, players take turns rolling a set of five 6-sided dice in an attempt to satisfy one of thirteen possible dice combinations. For each round, each player is allowed up to two re-rolls of any of the five dice before applying the dice to one of their remaining thirteen scores. Each of the thirteen dice combinations can only be played once, so at the end of the thirteen rounds each dice combination has an associated score. The objective of the game is to maximize your score.

Since the game of Yahtzee was first introduced in the 1940's, many variations of the game have been used. For our research, we assume the original version of Yahtzee with no Yahtzee Bonus and no Joker rules. This means Yahtzee may only be scored once, and additional Yahtzee scores may not be used as a “Joker” to fulfill other dice combinations. Furthermore, we will focus on solitaire Yahtzee—where the player solely tries to maximize their own score.

For reference, a scorecard for the game of Yahtzee is provided in Figure 1.

## II. GAME MODEL

The game of Yahtzee can be modeled as a series of decisions made by an agent. At each decision point the agent must decide to either apply the current dice to one of their remaining scores, or choose to re-roll any of the five dice. If the agent has already used two re-rolls for the current turn, the agent has no choice but to apply the dice to one of their remaining scores.

**Yahtzee** Name \_\_\_\_\_







UPPER SECTION	HOW TO SCORE	GAME #1	GAME #2	GAME #3	GAME #4	GAME #5	GAME #6
Aces  = 1	Count and Add Only Aces						
Twos  = 2	Count and Add Only Twos						
Threes  = 3	Count and Add Only Threes						
Fours  = 4	Count and Add Only Fours						
Fives  = 5	Count and Add Only Fives						
Sixes  = 6	Count and Add Only Sixes						
<b>TOTAL SCORE</b>	→						
<b>BONUS</b> If total score is 63 or over	SCORE 35						
<b>TOTAL</b> Of Upper Section	→						
<b>LOWER SECTION</b>							
<b>3 of a kind</b>	Add Total Of All Dice						
<b>4 of a kind</b>	Add Total Of All Dice						
<b>Full House</b>	SCORE 25						
<b>Sm. Straight</b> Sequence of 4	SCORE 30						
<b>Lg. Straight</b> Sequence of 5	SCORE 40						
<b>YAHTZEE</b> 5 of a kind	SCORE 50						
<b>Chance</b>	Score Total Of All 5 Dice						
<b>YAHTZEE BONUS</b>	1 FOR EACH BONUS SCORE 100 PER 1						
<b>TOTAL</b> Of Lower Section	→						
<b>TOTAL</b> Of Upper Section	→						
<b>GRAND TOTAL</b>	→						

Fig. 1. Yahtzee Scorecard

Each decision made by the agent is based entirely on the current state of the game. The state of game is entirely determined by four values: the current round number (1-13), how many re-rolls have been used (1-2), the current state of the dice (array of 5 values (1-6)), and the current state of the agents scorecard (array of 13 values representing score for each of the 13 dice combinations). To make the model of the game simpler, each of the 13 scores starts with a value of -1 to

indicate that the score has not yet been played. This eliminates the need for an array of boolean values to flag each score as played/unplayed, and thus reduces the dimensionality of the input elements needed for the agent to make a decision.

In formal terms, the task environment for Yahtzee is fully observable, stochastic, sequential or episodic (depending on the model), static, discrete, and single agent. The environment is fully observable because the agent has complete knowledge of the state of the game given by the round number, re-roll count, state of dice, and the state of the scorecard. As a dice based game, the environment is stochastic, as the next state of the game is completely determined by rolling dice. The environment is either sequential or episodic depending on the game model used. Each decision to roll the dice or apply the dice to a score effects future scoring options and probabilities, and all decisions are evaluated together to determine the final score, so the environment is inherently sequential. However, we can treat the game as episodic by modeling each re-roll as an independent event. This drastically reduces the size of the game state space and can be used as a useful approximation to the “true” sequential environment. The environment is static because the state of the game does not change while the agent deliberates on a decision, and discrete because there are a finite number of game states (albeit a very large finite number!) Finally, the game is single agent because of our decision to analyze solitaire Yahtzee. A multiplayer game of Yahtzee would be an interesting future research project for a multi-agent environment.

To aid the development of an autonomous agent, we designed our model of Yahtzee to be flexible, allowing for any number of rounds to be played. A standard game of Yahtzee has 13 rounds and 13 accompanying scores for each round. Likewise, any smaller subset of a game of yahtzee with  $n$  rounds will only have  $n$  scoring options. Our game model is designed in such a way that the  $n$  scoring options are the first  $n$  options available on a standard Yahtzee playing card. Therefore, a three round game would only have scoring options for Ones, Twos, and Threes; a six round game would only have scoring options for Ones, Twos, Threes, Fours, Fives, and Sixes; a nine round game would only have scoring options for Ones, Twos, Threes, Fours, Fives, Sixes, Three of a Kind, Four of a Kind, and Full House; etc. This model of the game allows us to reduce the complexity of the game (by reducing the state space), thus making it easier to test various strategies and implementations. Once a successful strategy is found, it is easy to scale up to a full game of Yahtzee.

### III. COMPLEXITY

#### A. Sequential Model

Based on our model of Yahtzee as a series of decisions, the game complexity can be viewed as the size of the decision tree representing all possible decisions the agent could make. We can derive an upper-bound on the size of the game tree by assuming the agent will always use both re-rolls on each of their 13 turns. (This is a reasonable assumption to make, given that most human players use nearly all of their re-rolls

attempting to maximize their scores). Given this assumption, the decision tree has 39 levels (decision points)—13 decisions to decide which score to apply the dice to after the second re-roll of each turn, and 26 decisions to decide which dice to re-roll (using two re-rolls for each of 13 turns).

The number of possible decisions the agent has to make when applying the dice to a score is given by (1):

$$d_s(t) = 13 - t + 1 \quad (1)$$

Where  $d_s(t)$  is the number of possible decisions the agent can make when applying the dice to a score at turn  $t$ , and  $t \in [1, 13]$ .

Unlike the number of possible decisions the agent can make when scoring, the number of possible decisions the agent has to make when choosing which dice to re-roll is independent of what turn number it is. The total number of re-roll decisions is give by (2):

$$d_r = \sum_{k=1}^5 \binom{5}{k} = 31 \quad (2)$$

Where  $d_r$  is the number of possible decisions the agent can make when re-rolling the dice (the total number of ways to re-roll), and  $k$  is the number of dice chosen to re-roll.

As a sequential game, the decision tree is multiplied in size at each of the 39 decisions made by the agent. Using our assumptions above, we can calculate the size of the decision tree as (3):

$$d_t = \left( \prod_{t=1}^{13} d_s(t) \right) * (d_r^{26}) = 13!(31^{26}) \quad (3)$$

Where  $d_t$  is the size of the decision tree.

#### B. Episodic Model

Another way to view our model of Yahtzee is as series of independent decisions. In other words, rather than modeling the game in its true from (which is a sequential environment), we can model the game as series of independent episodic events. This is a useful way to approximate the true sequential nature of the game, and it drastically reduces the complexity.

The episodic model of the Yahtzee game has two different types of events: deciding which dice to re-roll based on the game state, and deciding which score to apply the dice to based on the game state. The number of possible decisions an agent has to decide amongst when choosing which score to apply the dice to is the same as during the sequential model of the game, given by equation (1). However, the number of possible decisions when deciding which dice to re-roll is reduced to all possible combinations of: moves the agent has played, state of the dice, and possible re-roll combinations. The number of possible combinations of moves the agent has played is  $2^n$ , where  $n$  is the number of rounds (and possible moves available). For a standard 13 round game of Yahtzee this gives  $2^{13}$  possible combinations of scores played or unplayed. The possible combinations of the 5 dice can be calculated by using k-combination with repetition (4):

$$\binom{6}{5} = 252 \quad (4)$$

The total number of possible re-roll combinations is given by  $2^5 = 32$ , which represents a choice of the 32 possible combinations of re-rolling or not re-rolling each of the 5 dice. In all, this gives a total state space for all possible decisions the agent must decide on as (5):

$$d_n = 2^n * \binom{6}{5} * 2^5 = 252 * 2^{5+n} \quad (5)$$

For a standard 13 round game of Yahtzee, this gives approximately 66 million different states of played-Moves/diceCombinations/rollActions. This state space is nearly 40 orders of magnitude smaller than the state space of the sequential model of Yahtzee represented as a decision tree of all possible games. Furthermore, because our model of Yahtzee is flexible, we can reduce the game to a lesser number of rounds while we experiment with techniques to train the agent.

#### IV. STRATEGY

Given the enormous size of the decision tree, game strategies will invariably involve methods and heuristics for reducing the dimensionality of the problem. The most obvious way to achieve this is to treat each turn of the game as an individual event. This effectively turns the task environment from sequential to episodic. Although we lose the foresight of how individual decisions affect future decisions, we can approximate this behavior in an episodic environment by giving decisions a weight corresponding to their effect on future scoring options.

##### A. Pipeline Implementation

The first step in implementing any AI model for Yahtzee is to implement the game in such a way that allows for the training of and input by an AI agent. A second major consideration is the ability to “truncate” a game to  $n \leq 13$  rounds, accordingly limiting scoring to the first  $n$  scoring options as well. This modification allows for the training of an agent on a smaller search space, a necessity when attempting to tackle a problem of this magnitude with the computational power offered by a personal computer.

The game has been implemented in such a way that a combination of AI and human players may play against one another, with the interface harness giving both types of players the same game state information and input options. The dice are rolled, and then the state of the dice and remaining scoring options are available to the player. The player may then choose to score their roll, or reroll all or some of their dice. If the roll is scored, the round passes to the next player until all players in the game have scored. If the player chooses to reroll, the new state of the dice is passed to the player, as well as remaining scoring options. The player once more has the choice of rerolling or scoring the dice, at which point the player MUST score their dice.

As is detailed in the following section regarding Reinforcement learning, the gamestate and actions taken by the agent can be easily tracked and stored by an agent, and allows for this information to be stored and mined. Further, the implementation of the Yahtzee game allows for many AI agents to play simultaneously for many games, allowing data to be collected and learning to occur as fast as possible using the simulation.

##### B. Expert System

Rather than searching an enormous state space, we have achieved an efficient and reasonably well performing solution with the use of an expert system. The expert system is designed to take advantage of a few simple heuristics gleaned from expert Yahtzee players. Except for two dice combinations (small straight and large straight), all dice combinations need multiples of one dice number in order to achieve a high score. This can be achieved by re-rolling dice in a priority order; first re-rolling those dice that are not of the highest frequency of occurrence, and second breaking a tie of duplicate dice frequency combinations with the highest of the two dice. For example, if the dice combination [3,3,3,5,6] were rolled, the last two dice would be re-rolled to try to increase number “3” dies. However, if the dice combination were [3,3,5,6,6], then the first three dice would be re-rolled. The 6’s have a higher value and break the tie when duplicates occur.

This is a very simplistic approach, and in order to produce a more intelligent agent more subtle rules are required. In the spirit of ripple-down expert systems, we add additional rules by analyzing the game log and noticing cases where the system clearly made the wrong decision—so called “cornerstone” cases. One example would be when the agent applies the dice to the small straight score after rolling the combination [2,3,4,5,5]. Although this dice combination was correctly identified by the agent as a small straight, the agent did not check to see if a large straight had been scored. If not, it would be wise to use both re-rolls in attempt to turn one of the “5” dice into a “6”. In this manner we can continue to run the expert system, analyze the game log, determine misclassification’s, and add cornerstone events to improve the game logic.

Thus far, using an expert system approach in the spirit of ripple-down rules has performed remarkably well. On a trial run of 1000 agents, the average agent is score 212 and the average high score is over 330. Although not quite optimal, these are competitive scores that could often outperform a human player when the dice are in favor.

##### C. Reinforcement Learning

Rather than programming an agent to make decisions according to our own logic, as in the case of an expert system programmed with the ripple-down rules methodology, we would prefer to have the AI learn to play the game of Yahtzee. The primary motivation for this strategy is that a self-learning AI may find optimal game play strategies that are counterintuitive to the way humans play the game.

There are two problems inhibiting the general training of an AI Yahtzee agent. First, the sequential nature of the game of Yahtzee has an extremely large search space, and is computationally intractable for a small machine. In order to get around this problem, we can approximate the game of Yahtzee by treating each turn as an independent event. This model sacrifices the holistic nature of the game, where all turns culminate to give a single score, for a simplistic model where each turn is individually evaluated. However, as stated in the above section on game modeling, we can approximate the holistic influence of certain scores by introducing biases into the individual turn scoring function.

The second problem with training an agent to play Yahtzee is that many strategies for training an AI require a very large corpus of data. This is problem for strategies such as Deep Neural Networks, which learn to classify decisions based on large amounts of data.

In order to get around these problems, we chose to implement an agent that uses Reinforcement Learning (RL) to learn to play the game of Yahtzee. We used the episodic model of the game to get around the search space complexity, and RL has the unique advantage that an agent can learn to play the game without any knowledge of previous game play data. Our implementation is heavily influenced by the work of Graham [9], who developed a RL framework for a dice game called YamSlam. The RL model is very similar to the use of the Monte Carlo method, which uses random sampling to develop a macroscopic view of the system. The details of our implementation are outlined below.

Our RL implementation is based around a data structure called an action table. The goal of action table is to enumerate all possible game states and actions, and to assign relative values to each action given a large series of random trials—in essence, a Q-value store. Specifically, the action table is a key-value store, where the key and the value are both 3-tuples. The key-tuple represents a codified version of which scores the agent has and has not played, the state of the dice, and the action the agent will take. The value-tuple represents the sum total of all game trials played and evaluated for the given key-tuple, the total number of trials for the given key-tuple, and the average score given these two values (used for making decisions during game play). The following example(6) illustrates an example of an action table entry for a 6-round game of Yahtzee:

$$(9, (2, 2, 3, 4, 4), 11) = (35, 70, 0.5) \quad (6)$$

In this example, the 9 is the decimal encoded version of the bit-string “001001”, representing that 3s and 6s have been played, but 1s, 2s, 4s, and 5s have not. (2,2,3,4,4) represents the state of the dice at the beginning of the agents turn. The 11 represents the decimal encoded version of the bit-string “01011”, representing the action for the agent is to only re-roll dice 2, 4, and 5, and to keep (not re-roll) dice 1 and 3. The 35 represents the total score the action table has accumulated over all trials from performing the given re-roll action “01011”, on the given game state “001001” with the current state of the

dice (2,2,3,4,4). The 70 represents the total number of trials thus far, and the 0.5 represents the average score, which is simple calculated by the quotient of score sum total and total trials. Strictly speaking it isn’t necessary to keep this value in the table, but it’s used for making decisions during game play and makes the look up fast.

Once the action table has been set up, using the action table is a straight forward process by the RL agent. If the agent has remaining re-rolls, the agent will iterate through all 32 possible possible re-roll actions (all possible bit strings of length 5), and choose the action that has the highest average scoring potential. If the agent does not have remaining re-rolls, the agent will simply choose to apply the dice to one of the unplayed scoring options which gives the agent the greatest number of points.

One subtle issue we encountered with our RL implementation was that high scoring options were favored when lower scoring but rarer scores were available. For instance, if an agent has the dice combination (3,3,5,5,5), it would make sense to apply the dice to “5s” or “Full House”. However, by standard scoring rules, a higher score can be achieved by playing these dice against “Chance”. Another example is that the agent may apply this score “Three of a Kind” as opposed to “5s”, because the agent does not have knowledge of the “Bonus” score. Both of these issues are related to the fact that our model of the game is episodic, evaluating the score on a turn-by-turn basis, when in reality the game is holistic, receiving one score for the sum total of all actions played during the game. In order to get around this issue, our RL implementation also consists of a custom scoring function that adds biases to the standard scoring rules of a game of Yahtzee. For example, a bias of -10 is added to all “Chance” scores, to reduce the appeal of “Chance” and save it for tight situations when nothing else can be played. Also, a bias of  $1.666 * s$ , where  $s$  is one of the six scoring option digits at the top of the card, is introduced to account for the bonus. The custom scoring function is used during both training trials and game play to bring our episodic model of Yahtzee closer to a strategy that would be used in a genuine sequential model of Yahtzee.

#### D. Neural Network

One idea we initially had for building a Yahtzee agent was to use a Neural Network approach. Using the dice values, available re-rolls, and available scoring options as features, we thought we could train a neural network to produce decisions which optimize the final game score. Unlike the RL agent, we would use final scores (rather than score per turn), since certain scoring options are statistically more likely to score highly than others. For instance, if after re-rolling twice the dice (1,1,1,1,6), then choosing “1s” would yield 4 points, while “6s” would yield 6 points. However, this is clearly the incorrect decision, since “6s” is highly likely to earn more points on some other turn and “1s” is unlikely to ever score more than 4 points.

Unfortunately, this idea fell short because we could not find data to train our model with. In a future implementation

we could produce data for this model by beginning with random edge weights and playing a large number of games. As this data is generated, backpropagation could be used in the network to modify the graph weights until higher scores are achieved. Due to the large number of possible game states and random chance, training this model would be computationally expensive. However, as training progresses, such a model should theoretically converge on optimal play.

Due to the randomness of play, reweighting of edges should be done with care. For example, rerolling all 5 dice and scoring a yahtzee might “persuade” the model that rerolling 5 dice is a good idea since it got a large reward in that instance. Clearly, rerolling all 5 dice is rarely a good strategy. Hence, in order to play optimally, the model would need to observe a statistically significant number of outcomes for each possible situation.

## V. EVALUATION

An evaluation of a Yahtzee agent is difficult without a formal analysis of the mathematically verified optimal strategy and score. We did not have this available, so we instead relied on comparisons of our models against one another and amongst themselves as they were iteratively improved.

The expert system showed mediocre performance when the initial heuristics were implemented after gleaning strategies from expert players. The initial implementation scored an average of 135 per ten thousands agent trials. After analyzing game logs and implementing new rules to sway decisions in so-called “cornerstone decisions”, we incrementally increased the average score of the agent to over 212 with an average high score of 330 per ten thousand agent trials.

Evaluating the performance of the RL agent was much easier than the expert system agent. Unlike the expert system, which had cornerstone cases that were hard to measure the impact of, the RL agent was easy to see growth in performance as the number of trials increased. Due to the prohibitive cost/time of training the game on a full game of Yahtzee, we used a reduced game of Yahtzee with 6 rounds to demonstrate that as the number of trials increases the game asymptotically approaches an optimal strategy for a given model. We trained our agent with 100 million trials, saving an action table every 1 million trials to test agent score and performance. This process took about 6 hours running on a 2.3 GHz Intel Core i5. The results of the average score for ten thousands agents at each point along the training are plotted in figure (2) and displayed in chart (3). Using the same process we estimate that it would take nearly a thousand hours to train a full game of Yahtzee!

To compare the two agents against one another, its necessary to evaluate them on the the same size game of Yahtzee. Since we only have a sufficiently trained RL agent for a 6 round game, we must also see the average score for ten thousand agents playing a 6-round game using the expert system agent. It turns out that the expert system agent received an average score of 58.1735, nearly 4 points more than the RL trained agent. We predict this is due to the episodic approximation of the game used to train the RL agent.

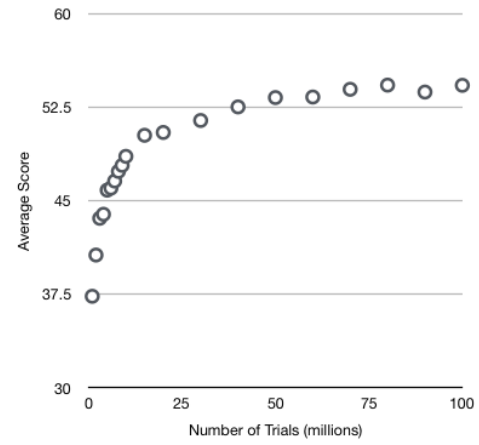


Fig. 2. Results of RL Implementation, 6-Round Game

trials (millions)	average score (per ten thousand AI plays)
1	37.352
2	40.657
3	43.605
4	43.926
5	45.862
6	46.025
7	46.59
8	47.364
9	47.849
10	48.571
15	50.26
20	50.487
30	51.447
40	52.5349
50	53.2835
60	53.3272
70	53.9468
80	54.2827
90	53.7266
100	54.2668

Fig. 3. Results of RL Implementation, 6-Round Game

## VI. CONCLUSION

While often looked at as a casual game of chance, the size of the search space and the amount of skill needed in a game

of Yahtzee is deceiving. We have been humbled by the game of Yahtzee, and are surprised at the computational requirements of training an autonomous agent to play the game.

Looking forward, there are many alternative strategies we would like to experiment with. The RL agent, while being entirely unsupervised and “learning” the game of Yahtzee, was reliant on artificial “biases” that we introduced into the scoring function. The biases were selected with a little bit of math and intuition, but like the expert system, suffer from our limited vantage point. It would be interesting to explore options for generating these bias constants using another model, a so-called “meta-AI” to help generate the constants to drive our RL agent.

Another strategy we would like to consider is combining multiple AI strategies. For example, we did not use a neural network approach because we don’t have game play data, but our RL agent could be used to generate data to train a neural network.

Another optimization technique we could use in future analysis is a way to prematurely prune our RL search space. If it became obvious that certain actions were not desirable early in the training process, we could remove them from the action table to gradually reduce the size of the action table. This would speed up the training process and make it feasible to scale up to larger games of Yahtzee.

## VII. RELATED WORK

### A. Decision Trees/Expert Systems

A general solution for games that can be modeled with a decision tree such as Yahtzee is the expert systems methodology called ripple-down rules[1]. Ripple-down rules use an incremental approach to knowledge acquisition. When cases are incorrectly classified by the current system they are added to a database to incrementally refine the knowledge base. These misclassified cases are referred to as “cornerstone event”. The methodology makes use of a true-false binary tree to categorize knowledge from human experts.

A similar approach can be used for Yahtzee where each misclassified case can be marked as a cornerstone decision. After consulting with a human expert, classification errors are determined and added to the database. The cornerstone decisions are used to traverse each node until a leaf node is reached, at which point a classification (decision) is made. Generally, this would be a difficult task to accomplish given the multiple classification tasks the agent will need to cross check against. This will lead to a tree that will multiply in size and eventually become unmanageable. A variation of ripple-down rules called Multiple Classification Ripple Round Rules (MCRRR) attempts to solve this problem[2]. MCRRR extends ripple-down rules to multiple classification tasks by altering the underlying knowledge representation structure to that of an n-tree. The cornerstone case approach is altered so that multiple cornerstone cases might apply to a single case. Every node attached to the root node will be explored, and only best option at the end of each node will be added to the result.

		Neuron-2	
		Fire	Rest
Neuron-1	Fire	R, R	P, P
	Rest	P, P	R, R

Fig. 4. Neuron-2

### B. Neural Networks

The application of game theory to neural networks has recently gained attention [3]. The combination is well suited to games involving human and non-human players such as Yahtzee. Game theory is applied to neural networks by using a paired neuron system. Matrices are used to model the neurons, and the paired matrices are used to achieve a payoff matrix within the given context.

The decisions made throughout a game such as Yahtzee can be represented in neuron matrices to represent a payoff matrix. A general necessity for game theoretic interpretations to apply to Yahtzee is static information extraction. This means that complete information needs to be available to the human/AI at each available turn. Decisions need to be made independently with bound information to receive an accurate payoff. A player can pursue a strategy by observing another players decisions and acting accordingly. At each decision event, such a matrix can be applied weighing by non-benefit/benefit outcomes. Such a matrix can be represented as Neuron-1, and Neuron-2 for both players. (Figure 4)

Game theory will use matrices such as these and apply it to probability distributions, recording possible uncertainty verse likelihood payoff matrices that would determine possible conclusions. Each player looking for the maximum payoff traverses down a game tree of communicating neurons and analyzes the different payoffs at every path. It is obvious that the decision-making process for this model involves input and output with a reward/punishment mechanism function.

### C. Reinforcement Learning

While neural networks are a powerful tool for classifying data and making decisions, they rely on a large corpus of data to train their models. In the case of game like Yahtzee, a large corpus of data from games of expert players may not be available. Fortunately, in a game like Yahtzee where the score of the game is a natural way to evaluate the performance of the AI, a technique called reinforcement learning can be used to iteratively improve the AI’s performance. A framework for reinforcement learning in the context of dice games was developed by software engineer Greg Graham [9].

Graham’s dice game reinforcement learning framework is loosely based on the Monte Carlo method. After the dice game has been defined, random trials are played using all possible states of the game and all possible actions (dice rolls) for

each given game state. A table is kept to record the number of trials for each state/action combination, the running total of points won for the state/action combination, and the average score over time. After sufficiently many trials, the best action to take for any state of the game will converge on the action with the highest average score.

In Graham's work, he developed a model of the game Yamslam and used his reinforcement learning framework to train an AI to play the game. Using the method outlined above, Graham showed that the average score of the AI will converge on an optimal strategy for playing Yamslam. It should be noted that this method treats the game as a strictly episodic environment, maximizing the points gained on each roll. This may not be the best strategy when a game involves cumulative bonuses (such as Yahtzee), but it nevertheless is good approximation to an optimal game and a good strategy for reducing the state space and therefore game complexity.

#### *D. Other Work Optimizing Yahtzee*

Many attempts have been made to optimize Yahtzee play. Some of the literature describes brute force approaches that can produce mathematically correct solutions, while other approaches use heuristics and machine learning to achieve less computationally demanding yet accurate approximate solutions. One such approach from researcher James Glenn [4] analyzes the entire state graph that can be traversed in a game of solitaire Yahtzee. While achieving a near optimal score, the decisions required minutes to hours of CPU time. Another approach taken by researchers Markus Felldin and Vinit Sood [5] reduce Yahtzee to an episodic game, treating each decision as an individual event. They used statistical methods to determine the optimal move at each decision point. It's interesting to note that in all cases a tradeoff was made between performance (final score) and cost (time to run algorithm).

There are several primary strategies that have been implemented with the goal of playing a "nearly-optimal" game of Yahtzee. Approaches have included optimizing the probability of beating a particular given score, maximizing average solitaire score, using an expert-designed decision model, and heuristically applying these strategies to multi-player games. As previously discussed, there must be trade-offs between a truly optimal play and a reasonable computation time. The primary downfall of setting a target score and optimizing the probability of surpassing that score is indeed extreme computational complexity [6]. Heuristics tend to be the weapons of choice for problems with state spaces this large, so approximation methods reign supreme. Shrinking the problem space is critical for these methods, e.g. minimizing dice rolls per turn in order to maximize scores while minimizing risk [7]. Each turn, the agent determines which scoring option will offer the most points for the fewest rerolls, and then acts accordingly. A genetic algorithmic approach is explored by researcher James Glenn, using statistical analysis of the agent's performance to adjust certain parameters and priorities. The heuristic of breaking the game into subproblems and solving

those is also explored. Maximizing the "upper box" values allows for a bonus score of 35, whereas after an initial "lower box" "Yahtzee" score of 50 points, 100 points are obtained for each additional "Yahtzee" rolled, in conjunction with the standard scoring lower section rolls. These subproblems can be solved independently, significantly shrinking the search space [8].

#### REFERENCES

- [1] Debbie Richards, Two decades of Ripple Down Rules research, The Knowledge Engineering Review, Vol 24:2, 159184. 2009, Cambridge University Press.
- [2] Ivan Bindoff, Byeong Ho Kang, Applying Multiple Classification Ripple Round Rules to a Complex Configuration Task. AI 2011: Advances in Artificial Intelligence.
- [3] Alfons Shuster, Yoko Yamaguchi, Application of Game Theory to Neuronal Networks, Advances in Artificial Intelligence, 2009 Volume 2010, Article ID 521606, 12 pages .
- [4] James Glenn, An Optimal Strategy for Yahtzee. 2006. Loyola College of Maryland.
- [5] Markus Felldin and Vinit Sood, Optimal Yahtzee. KTH Vetenskap Och Konst.
- [6] Pawlewicz, J. and van den Herik, H.J. and Plaat, A. and Iida, H. Nearly Optimal Computer Play in Multi-player Yahtzee. 2011, Warsaw University, Institute of Informatics, Warsaw, Poland.
- [7] Maynard, Ken and Moss, Patrick and Whitehead, Marcus and Narayanan, S. and Garay, Matt and Brannon, Nathan and Kantamneni, Raj Gopal and Kustra, Todd, Modeling expert problem solving in a game of chance: a Yahtzee case study. Expert Systems 2001, Volume 18, ISSN 02664720
- [8] James Glenn, Computer Strategies for Solitaire Yahtzee. 2007 IEEE Symposium on Computational Intelligence and Games
- [9] Greg Graham, A Reinforced Learning Framework for Dice Games. RStudio Publications. Tuesday, October 13, 2015.