Jackson Hamburger - jah562

Qinglin Chen – qc59

Jiang Yu – jy424

Artificially Intelligent Yahtzee Agent Using Game Tree Search and Heuristic Board Evaluation Function

**Introduction**

Invented in 1954 by a Canadian couple and currently sold by Hasbro, Yahtzee is a family game intended for ages eight and up. Yahtzee is easy to learn and fun to play. It has achieved tremendous success selling over 50 million sets each year, making it one of the most popular games in America. The game of Yahtzee revolves around throwing five standard six sided die and can be played with any number of people including just one. The goal of the game is to achieve the highest possible total score over a total of 13 turns. While some players rely heavily on luck for Yahtzee success, thoughtful game play makes a tremendous difference when playing high numbers of games. Our project was to develop an artificially intelligent agent to play the game of Yahtzee as well as possible to maximize expected total score (it is important to note that the goal of the agent is to maximize the total score rather than to win the game, which are two different goals when playing the game with more than one player). We went about this by creating an agent that would use a game tree to search the space of possible moves and possible probabilistic outcomes for each move on each turn, and a heuristic function to estimate expected score at each leaf (this method will be explained further). This technique was sufficient for our agent to beat a human player and a simpler greedy agent.

**Yahtzee Gameplay**

The game of Yahtzee consists of 13 rounds of gameplay. Each round a player rolls five six-sided die, decides which die they want to keep, and which die they want to re-roll. They then re-roll the die they decided to, and repeat the process of deciding which die to keep. The player then again re-rolls the die they decided not to keep. At this point the player is left with 5 rolled die.

With these 5 die the player selects one of the 13 categories, each of which has a different point value based on the 5 die that they have rolled. After a player selects a category, the player cannot use that category for the remainder of the game. The 13 categories reward points as follows.

* Yahtzee – The player receives 50 points if all 5 die show the same face (i.e. 4,4,4,4,4), 0 otherwise
* Ones – The player receives the sum of all the ones in the five die rolled
* Twos – The player receives the sum of all the twos in the five die rolled
* Threes – The player receives the sum of all the threes in the five die rolled
* Fours – The player receives the sum of all the fours in the five die rolled
* Fives – The player receives the sum of all the fives in the five die rolled
* Sixes – The player receives the sum of all the sixes in the five die rolled
* Three-Of-A-Kind – The player receives the sum of all 5 die if at least three show the same face, 0 otherwise
* Four-Of-A-Kind – The player receives the sum of all 5 die if at least three show the same face, 0 otherwise
* Full House – The player receives 25 points is the die show three of a kind and a pair, 0 otherwise
* Small Straight – The player receives 30 points if 4 sequential dice show (i.e. 2,3,4,5), 0 otherwise
* Large Straight - The player receives 40 points if 5 sequential dice show (i.e. 1,2,3,4,5), 0 otherwise
* Chance – The player receives the sum of all die

After the 13th turn all categories are used and the game ends. The player’s final score is the sum of all of his or her turn scores. The goal in a single player game (our case) is to maximize your score, the goal in the multiplayer game would be to have a higher score than all other players.

**Performance Measure**

Due to the innate randomness in the game of Yahtzee we will be judging our intelligent Yahtzee playing agent of the basis of average score and standard deviation of score.

As a result of its popularity they is a decent amount of online literature (academic and otherwise) on the subject of Yahtzee and Yahtzee game play. Nearly all of this literature revolves around the game of Yahtzee in its bonus form. It must be noted however that the game of Yahtzee we are playing does not involve the traditional bonus that many forms of the game do. Our agent was designed to score as high as possible according to the rules described above. Thus we have to be careful when comparing results from bonus Yahtzee to standard Yahtzee, though because there are no penalties in bonus Yahtzee, the scores achieved in bonuses Yahtzee are strictly greater than scores achieved in the standard Yahtzee game we are playing.

Published in May of 2006 by James Glenn out of Loyola College in Maryland’s Computer Science department *An Optimal Strategy for Yahtzee*, provides a thorough look into optimal Yahtzee play (in the case of bonus Yahtzee). Along with the details of a design for an optimally playing Yahtzee agent the paper goes through a statistically analysis of optimal strategy and other non-optimal strategies. The results for 4 of their agents are as follows

|  |  |  |
| --- | --- | --- |
| *Strategy* | *Expected Score* | *Standard Deviation* |
| Greedy | **218.05** | **46.87** |
| Heuristic | **240.67** | **60.90** |
| Better Heuristic | **244.87** | **57.39** |
| Optimal | **254.59** | **59.61** |

Along with this we wanted to compare our agent to human players. This however is not as easy as it may seen as due to the high standard deviations of scores in Yahtzee games we would need many more trials than we could feasibly perform. Given the average Yahtzee game takes about 5 minutes if we were to play for a total of 12 hours we would get approximately 150 games recorded. Given a similar standard deviation to the heuristic games of 60 and 150 trials assuming the trial scores are normally distributed we would still get a 95% confidence interval for our mean of size 19.2 (realistically the human score standard deviation is likely to be much higher due to inconsistent strategy and the human tendency towards risk in some scenarios). Judging success and failure on a confidence interval this large would be little more than conjecture. Thus unfortunately we will have great data human data to compare our agent to. In the absence of our own data we have found a site publishing very basic statistics about bonus Yahtzee games. Extrapolating from the mean information of games with any number of Yahtzee we have determined that an estimate for the average human Yahtzee score is 210 with a standard deviation of 60.

|  |  |  |
| --- | --- | --- |
| *Strategy* | *Expected Score* | *Standard Deviation* |
| Human | **205** | **60** |

Along with this data we have found we have used a Greedy Agent that we have coded to generate more in depth data about how our intelligent agent preforms.

**Agent Design**

An agent playing a game of Yahtzee must make three types of decisions each turn. One, it must decide which die to hold after the first roll. Two, it must decide which die to hold after the second roll. And three, it must decide which category to use given a final set of 5 die. Each of these three decisions is non-trivial and we will examine our decision choices for each one. Do to the recursive nature of our agent’s rationality we will start with deciding which category to use and work backwards.

**Category Selection**

Given a set of 5 rolled die with a values we want to pick a category , where is the set of remaining, unused category that maximizes our score for not just the round but the game. For the greedy agent this decision is simple. It iterates through and calculates the score of using each for die and picks the maximum. This is a traditionally greedy method and is short sighted as greedy algorithms have the tendency to be. It is short sighted in the sense that the expected score of a reaming game using categories () is far lower than that of (). Thus we needed to develop a technique for our agent to make intelligent decisions regarding which category to pick when given a set of die and a set of remaining categories . What we decided to do is develop a heuristic function, that takes as a parameter the set of remaining categories after using category (let us call this set ). Now, instead of doing what the greedy agent does by finding the that maximizes the score of using on the intelligent agent maximizes the sum of the current turn score and the expected score for the rest of the game ().

By using our heuristic function we take into account the effect that choosing a specific category will have on the expected score for the rest of the game. Thus if we have the optimal heuristic function our agent would be play the optimal category picking strategy. We will discuss later our choice of heuristic function.

**Second Round Die Hold**

At the point in a player’s turn where they have already rolled the initial time, and having selected which die to hold have rolled a second time, we have one more chance to hold certain die and re-roll others. At each point where we decided which die to hold, we have 32 different moves we can make (for each of 5 die we make a binary decision on whether or not to hold the die, thus there are 25=32 possible choices). We can represent each of the choices as an edge on a graph going from a decision node to a roll node, at each roll node we can represent the probability of moving ending up with a certain die combination as another edge to a die value node. i.e.

Once we have constructed this tree we are able to start evaluating which move maximizes our expectation of the final score. To calculated the expectation we take the probability that a particular die vector is reached given a certain move and multiply it by the max() term from the category selection. We then sum each of these products over all possible outcomes of a given move node. From preforming this calculation on all move nodes we can get the expected score we will receive for the rest of the game given each possible move. After this we simply iterate the move nodes to find which one maximizes our expectation, and select that move as the one to use as our move for the current turn.

**First Round Die Hold**

For each of the 13 turns after we roll the 5 die we must select which of the 5 die we would like to keep. This decision builds off of our previous two decisions. One could think of this decision in a similar fashion to the second round die hold decision. From the current set of rolled die we have, V’, we again have 32 different possible moves to make. So we build a graph similar to the one above. The only thing we change about this graph is how we evaluate the base case. Last time we evaluated the base case with respect to the final category choice decision, and picked the move that maximized this expectation. This time we will pick that move that maximizes the expectation of the expectation each node would have at the second turn if its die array were the one in the base case. So at every base node we construct another graph that we evaluate the exact same way we do in the second round die hold circumstance. In this way we are calculating the expectation of our score for the rest of the game given we make one of the 32 hold moves.

**Heuristic Function**

At this point each of our three decision’s all bottoms out at the base case calculation . We know that is the score that we will receive using category ci with die V from the rules of Yahtzee. So we must decide how we are going to determine what the expectation of the score is for the rest of a game with category set remaining. First we need to look at the domain of , since there are only 13 possible categories the number of possible combinations of used and unused categories is 213 or 8192. While this number is relatively large, we can easily hold in memory a table with 8192 slots mapping each category combination to an expected value. If expected values are stored as doubles they each take up 8 bytes. Which means the table is 64KB large. This will indeed take up multiple pages in memory but it will have tremendously fast access. So we decided we would use a lookup table based heuristic function. The tricky part is how to populate the table. If we think about it for the last turn of the game there will be no remaining categories so the expectation of the base case of our function will be or just because is the empty set. So we can calculate the expectation of when just by calculating the expectation of a one turn game of Yahtzee with only one category available, repeating for each category. Once we have all where we can use the same process to find all where and so on until we have generated the entire table and have values of for all possible . The calculation of for a particular we need a large number of trials as score have a high standard deviation. As get larger so will the standard deviation of so we will need more trials each round as we progressively grow our heuristic table. Given we used 1,000 trials for the base case of and 10,000 trials for we needed to run over 40 million partial Yahtzee games, over 300 million individual turns. This learning computation took 10 computers running 4 cores at 3.40Ghz with 16GB of RAM nearly 10 hours to complete. Given the complete heuristic table, table lookup is near instantaneous for the agent.

**Implementation**

This intelligent Yahtzee playing agent was implemented completely in Java. The decision to use java was one based mostly off of our groups comfort with the language, rather than any performance issues. This decision to use java was made before we decided to use simulation to build our heuristic function, so perhaps choosing a more performant language might have been prudent of us. We did not see the massive computation we ended up performing coming, and by the time we decided to build the table via simulation, our code was nearly complete.

The java is broken down by the abstraction of a game engine, an agent and basic game functions. The game engine and basic game functions were trivial to implement and most of the interesting work was done in the agent classes. As we alluded to before, while we made a human agent, playable from the command line, we were not able to use it to generate enough significant data for it to have been useful. The two agents that we did create were the Greedy and Intelligent agents. These two agents are nearly identical, save for the intelligent agent using the sum of the heuristic function and turn expectation in the category selection base case while the greedy agent uses just the turn expectation.

The code for this project is attached to this report. We have decided only to highlight one aspect of the implantation as it mostly follows the design described above.

Initially, as the described about at each leaf node, the system would calculate the expectation and the effect would propagate up the tree. This implementation preformed an exponential number of calculations of the base case as identical base cases were repeatedly calculated. With this implementation the Greedy agent would take on the order of 10 minutes to play just a single game of Yahtzee. Once we figured out we could memorize each result as it was calculated we improved run time dramatically enabling a thousand games to be played every 2 minutes. Without this improvement it would have taken thousands of years to accurately train the heuristic table. Once we made this improvement we tweaked the game engine to play repeated games to train its self and build and save the heuristic table. Building the actual heuristic table was the process of running 8 threads on 10 machines in Carpenter Hall, constantly over the course of a day and concatenating the large csv, index to heuristic value sub tables they created.

**Results**

After completing the training of our heuristic table we decided the best way to test our agent was to run 100,000 simulated Yahtzee games for both our heuristic and our greedy agents. The results we got were very encouraging and in line with what we had expected.

|  |  |  |
| --- | --- | --- |
| *Strategy* | *Average Score* | *Standard Deviation* |
| Our Greedy | **209.3131** | **32.97616** |
| Our Heuristic | **227.15** | **32.19662** |

Compare that with the published results for bonus Yahtzee:

|  |  |  |
| --- | --- | --- |
| *Strategy* | *Expected Score* | *Standard Deviation* |
| Greedy | **218.05** | **46.87** |
| Heuristic | **240.67** | **60.90** |
| Better Heuristic | **244.87** | **57.39** |
| Optimal | **254.59** | **59.61** |

As you can see our intelligent agent outperformed our greedy agent in non-bonus Yahtzee by an average of about 18 points, quite a substantial margin.

One thing to note is how our greedy agent averaged 9 points lower than the Loyola greedy agent, we think this is a product of the fact that they played bonus Yahtzee, which allows for higher scores than our standard Yahtzee and not a product of a mistake in our greedy implementation.

Also to note is the dramatic differences in our standard deviations which again we believe to be due to the difference styles of Yahtzee being played. On the note of standard deviations, though there is little difference between our greedy and heuristic standard deviations there is a large gap between Loyola’s heuristic and greedy standard deviations which may be due the their heuristic agent taking better advantage of the bonus leading to some extremely high scores.

Given our two agents average scores and standard deviations and the central limit theorem our heuristic agent will beat our greedy agent 65.03% of the time. Also based on our estimation for the human score our heuristic agent will beat a human opponent 62.75% of the time. These percentages are not tremendously higher than 50% but again this is a product of the substantial randomness factor in Yahtzee.

If we further break down our agent’s performance we begin to see some very interesting results as to the differences in their two game play strategies.

Consider the following graph of the average cumulative score as the game progresses:

Here we can see that out of the gate the greedy agent takes the lead (on average) and late in the game the heuristic agent performs better than the greedy agent. As late in the game as the 10th turn of 13 they have a near identical average score. This makes sense as our heuristic agent will sacrifice points in the short term for the expectation of points in the long term.

This can also be seen in the turn by turn average score graph here:

Also but analyzing the average turn each category was played we see that the heuristic agent plays far differently than the greedy agent. The greedy agent will play categories like chance, three-of-a-kind and sixes far earlier than the heuristic agent, as these categories have high point values but also have potential for higher points later, the latter of which the heuristic agent considers.

By looking at the average score achieved per category we can see on where the margin between the two agent scores lies.

Interestingly while the margin is slight for most categories, 6.3 points of the 18 points advantage come from the Yahtzee category alone, with another 3 coming from Large Straight. We are not certain as to what these differences result from.

**Conclusion**

Without a doubt our game tree search, heuristic based artificially intelligent Yahtzee was an overwhelming success. Not only did our agent outperform human and greedy players we could see the different ways in which the players operated. Through this project we were able to dive deep into some fascinating AI subjects while creating something that on its own is intelligent. With these results we can see that applying the concepts learned in CS 4700 lecture can provided real, significant results for problems without clear solutions.

**Loyola Paper:** <http://www.cs.loyola.edu/~jglenn/research/optimal_yahtzee.pdf>

**Human Yahtzee Stats:** <http://yahtzee.silentmatt.com/statistics>