# a2

# Part 1: Raichu

# 1) The Formulation of the Problem

I began with my daughter creating a physical version of Raichu to play, and after a few games I determined the following actions would move you closer to victory:

- progress toward evolving into Raichu (opposite row)

- taking pieces

- blocking enemy progress

I decided to view each move as a scoring opportunity, with the value of the move determined by a combination of those three factors. A given move would earn you a certain amount of points, and an enemy move would subtract points from your total. Thus, the point differential was the way the expected value for a given move was determined. Since the scoring values were equal and opposite, this scoring method was a zero-sum game, and I used minimax with alpha/beta pruning to search for the best move.

# 2) How the Program Runs

- Sets initial search tree depth to 2

- Finds all viable moves and store them in a list of lists of the following structure:

- [expected value, staring (row,column) coordinate, ending (row,column) coordinate, [jumped piece (row,column) coordinate]]

- Switches which player is in control

- runs recursive minimax algorithm with alpha/beta pruning on each viable move

- replaces expected value with propogated expected value from minimax

- finds new maximum expected value move

- yields new best move

- increases depth by 2 and repeats (as long as time allows)

# 3) Difficulties and Design Decisions

- The main difficulty was determining how to implement the minimax algorithm. I spent a lot of time trying to pass a the entire move structure through the recursive algorithm, and this led to terminal Min values being associated with Max moves and vice versa. The solution for this challenge was to get a full list of viable moves prior to entering the minimax algorithm and only let the alpha and beta values propogate up the tree. After all, the moves along the way didn't matter, just the expected values that would be propogated up from terminal nodes.

- The next big challenge was fine tuning the scoring rubric. This didn't truly come into play until the "tank" arena option came online. It became clear quickly that no defense made a victory unlikely, as the existence of a white and black Raichu typically ends in a draw. At the same time, offense is what earned Riachus, and the quicker these were on your roster, the better. This is the balance I designed based on those matches:

MOVEMENT POINTS

- If no Raichus exist on your team, forward progress earns a percentage of 3 points for Pikachus and 2 points for Pichus, based on how many rows the move advances the piece

- If at least one Raichu exists in your team, forward progress scoring is turned off

- Forward progress scoring reactivates when opponent has a Raichu

- Raichu moves earn points equal to (number of pieces enemy owns less 1) divided by 3

JUMPING POINTS

- Jumping a Pichu is worth 6 points (stacks with movement points)

- Jumping a Pikachu is worth 8 points (stacks with movement points)

- Jumping a Raichu is worth 10 points (stacks with movement points)

- Jumping the last piece is worth an additional 200 points (ending game)

# Part 2: Tetris Bot

# 1) The Formulation of the Problem

In this component of the project, we were instructed to create an AI that could play a game similar to the 2048 tile game. To construct our AI, we experimented with a game search tree and an adversarial search technique.

Before deciding on which algorithm to utilize or which heuristic to employ, we spent a large amount of time playing the game to learn the do's and don'ts of game play. There were several possibilities for both the deterministic and non-deterministic versions of the games. However, we proceeded to build the algorithm and heuristics with the deterministic form of the game in mind, because non-deterministic selections are difficult to foresee. but can be done using genetic algorithm or particle swarm optimization techniques as we did not cover those algorithms we just implemented a combination of heuristics

# 2) How the Program Runs

- It will copy the original Quintris to a variable as we should not do any operations on original quintris

- For a given quintris piece it will flip the piece as the first rotation and check for all other rotations i.e 90 degree, 180 degree, 270 degree and 360 degree and 2 horizontal flips which constitutes of 2 flips and 4 rotations

- Given a flip or a rotation the piece will move either extreme right or extreme left and iterate over the possible moves and get the successors

- For each successor we are attaching 9 heuristics, overall heuristic value, and the move it is taking to a dictionary

- Heuristics play an important role in this code i.e number of holes, height of the board, bumpiness, row transitions, column transitions, rows occupied by "x", number of lines cleared, number of pits.

- We gave every heuristic a coefficient value which follows dot product to find the heuristic value of that particular move and attach it to a dictionary

- x1\*(aggregate height)+x2\*(number of holes)+x3\*(bumpiness)+x4\*(pits)+....

- After getting all moves with its heuristic values attached and its path given we will move forward to calculate the best move among all the moves

- We are calculating best move based on greedy as the heuristic which is we are selecting the move with highest heuristic value other words best first.

- After this we are converting the move to a string as we need to return a string of moves in case of computer and simple

- Same as above We are implementing this for Computer animated.

# 3) Difficulties and Design Decisions

- The main difficulty We faced in this problem is to find the best move from the possible moves. We tried doing genetic algorithm and PSO optimization but those were giving wrong assumptions

- We also faced problem on how to implement Expecti minimax to this problem as it only takes max nodes and expectation is the probability of the next piece

- One more difficulty is how to change current piece location based on the probability of next piece or the heuristic value of the next piece

-So after viewing these difficulties, we just wrote in a simple way of a dot product of heuristics to calculated the cost function of the problem

Highest score - 5440

lowest score - 10

Here I am attaching screen shots of this

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidenceText

Description automatically generated with medium confidenceText

Description automatically generated with medium confidenceText

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

# Part 3: Truth be Told

# 1) The Formulation of the Problem

To address this problem, I used a Naive Bayes model wherein every word in a given review was treated independently of the others. Running through the training data, I tabulated frequency of words in true comments, deceptive comments, and overall. I also kept track of total deceptive comments and true comments in the event that future training data would not be perfectly split.

- Given this Bayesian classifier:

- P(A|w1,w2,...wn,) / P(B|w1,w2,...,wn) > 1

- I applied Bayes law to both numerator and denomenator, and the denomenators in each application of Bayes Law cancelled out nicely:

- [P(w1,w2,...wn|A)P(A)] / [P(w1,w2,...,wn|B)P(B)] > 1

- I then used the Naive Bayes assumption to make the Bayesian classifier look like this:

- [P(w1|A)P(w2|A)...P(wn|A)P(A)] / [P(w1|B)P(w2|B)...P(wn|B)P(B)]

- To improve accuracy, I included a function called "word\_cleaner" which strips each word of punctuation, capitalizaiton, and pluralization. One final step to improve accuracy was to remove from the calculation words that did not necessary deserve the influence they had on the accuracy of the classifier, such as "I", "To", "We", "At", and "Service".

# 2) How the Program Runs

- Creates dictionaries for all the words, words in true comments, and words in deceptive comments. The entries are word:frequency (total count) pairs

- Initialized counters for the number of true comments, deceptive comments, true words, and deceptive words.

- Cycles through the training data, "cleaning" words and updating the appropriate dictionaries and counters

- Removes words that are undeservedly influential on the accuracy of the classifier

- Initializes a probability dictionary that pairs every word that is in both a true comment and a false comment with the following ratio:

- P(word|true comment) / P(word|deceptive comment)

- Cycles through testing data, initializing a "truthiness" variable for each comment in the testing data

- truthiness = P(True Comment) / P(Deceptive Comment)

- "Cleans" words as it did for training data

- If a cleaned word is in probability dictionary, the program multiplies the given comment's truthiness by the ratio associated with that word

- When all words in a comment have been cycled through, classifier tags comments with a truthiness > 1 as 'truthful' and those with a truthiness <= 1 as 'deceptive'

- returns a list of the labels assigned by the classifier corresponding positionally to the comments in the testing data

# 3) Difficulties and Design Decisions

- The main difficulty I bumped into initially was that the probabilities of any of these words given a true/deceptive comment status were all very small. Thus, multiplying them together for my Bayesian classifier was causing divide by zero errors. The probability dictionary fixed this, and since multiplication is commutative, dividing each P(wi|True) by its corresponding P(wi|Deceptive) prior to multiplying the numerators and denomenators did not mathematically change the result. It did fix the divide by zero problem.

- The biggest design choice was deciding which words to remove from the calculations. I checked to see which words were causing a big impact on accuracy, and by disqualifying them was able to improve accuracy from 86% to 87.5%. At the same time, removing words without a good reason could cause a reduction in accuracy given a different training data or testing data. I made the choice that words like "I", "to", "we", and "at" had no reason to reduce my accuracy, and these words were removed under the assumption that their influece was an artifact of the particular training data I was given. I was less confident that "service" was such a word, but given that service is precisely the type of word a true critic would use to praise or criticize a hotel while also being precisely the type of word a business or disgruntled indivudual would use to falsify reviews, I concluded that this was a word that would not be a great indication of truthfulness and excluded it from calculations.