AI- Final Project Report

Lakshmi Sahithi Yalamarthi

PSU ID – 907580033

Introduction:

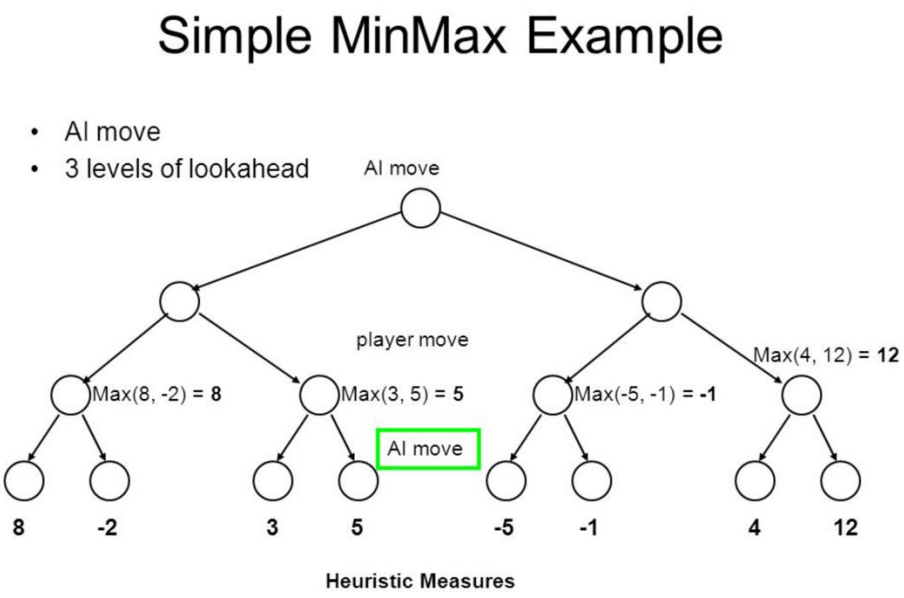
The project designs a 2048 puzzle, with MinMax algorithm. This puzzle is developed by Gabriele Cirulli. The process of the game is to add adjacent tiles with same numbers to get a tile with doubled number. The maximum number on the tile that can be reached with this game is 2048. This project focuses on reaching a tile with 2048, using minmax algorithm. In this project the Agent picks a tile and makes moves in random manner. Agent predicts the moves and calculates average of end score for every move and picks the move that has highest score. Each tile can be moved in four possible directions - up, down, left, and right. The agent can move the tile in any of four directions, until they collide with another tile in grid or it touches the edge of the grid. If it encounters any tile on its path with same number, then both gets added to form a single tile.

Algorithm:

In this project, there are two agents who acts as players. The goal of first player is to move the tile in such a way that it maximizes the value of the grid, whereas second player attempts to sends a new input tile to the grid to minimize the score achieved at that state. The player who tries to maximize the score is Maximizer, and the player who performs counter act is Minimizer. Both players get their chances in alternate manner to reach their respective goals. Thus, MinMax algorithm achieves the goal of this puzzle with moves that Maximizes or Minimizes the score. The outcome of this Algorithm is either of the agents must win resulting the other agent to lose. Agents of this algorithm uses Depth-First search to find their best moves. This algorithm uses backtracking to minimize the loss and proceeds with its best move. It simulates the grid as a perfect binary search tree, it uses recursion technique in trees, to find its successor state.

Algorithm in trees:

Algorithm starts from root node of tree and traces each node in tree and reaches deep down, and backed up through the tree, when the stack unwinds and reaches the root node back. Thus, the agent looks for all possible moves throughout the tree and when unwinding it decides a successor state based on current state of the grid. The score for each move is calculated through a simple evaluation function. The time complexity of algorithm depends on recursion through the tree, as the recursion is through depth first search, then space complexity is given by O(mb) where m is depth of tree and b is branching factor, time complexity is $O(b^m)$. The completion state of this algorithm guarantees a solution in finite search tree, and each player plays optimally and unbiased to reach their goals.



Pseudo code:

Function Minimax(node, depth, player):

If player = “Maximizer” then

Evaluation = -infinity;

For each child in tree

Evaluation = (child, depth -1)

Max Evaluation = Minimax (Max Evaluation, Evaluation, “Minimizer”)

Return Max Evaluation;

```
Else if player = “Minimizer” then
        Evaluation = +infinity;
        For each child in tree
                Evaluation = (child, depth -1)
                Min Evaluation = Minimax (Min Evaluation, Evaluation, “Maximizer”)
        Return Min Evaluation;
```

For instance, In a game tree with 4 layers, if there are terminal nodes are with values {(1,4), (2,5), (-1,-8), (7,9)} then

if maximizer starts the game then it returns values as {(4,5),(-1,9)}

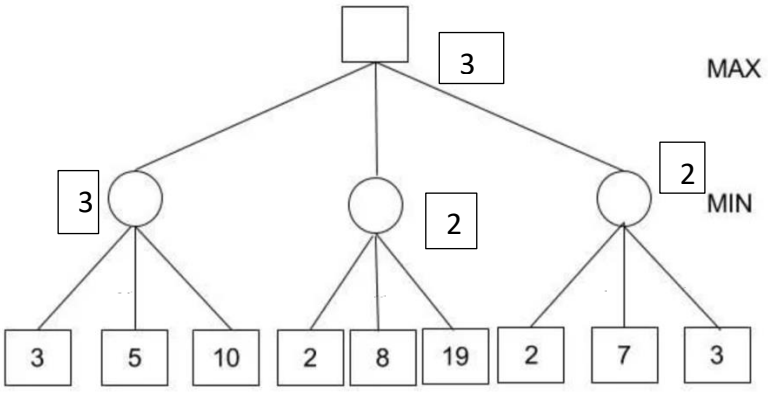
minimizer to play next turn where it picks {4,-1}.

maximizer picks {4} from resulting set, thus reaching the root node, and ending the game.

In this example, we discussed about 4 layers of tree, but in 2048 puzzle branching factor is huge which improves complexity of the Minmax Algorithm. To reduce this limitation, we use Alpha-beta Pruning, as it improves performance of agent.

Alpha-Beta Pruning:

This process removes nodes in the tree that does not influence the final decision of agent. Every move is alternated between Minimizer and Maximizer, so there are few nodes picked by one player are not considered in decision making of other player. These nodes are either highest weighted nodes or lowest weighted nodes of the tree. Pruning of these nodes, reduces complexity and does not change output of regular Minimax algorithm. This is implemented as extra step in solving 2048 puzzle, to enhance time and space complexity of the algorithm.



Implementation:

In this project, we have an agent and opponent playing the game. Agent has a state of action set in four directions {up, down, left, right}. When opponent is playing it has an action to place a randomly numbered tile which is multiples of 2 on an empty position in the grid. Evaluation of the game at current state is average score when reaching maximum depth or filling the grid with tiles.

There are three heuristics defined in solving this puzzle.

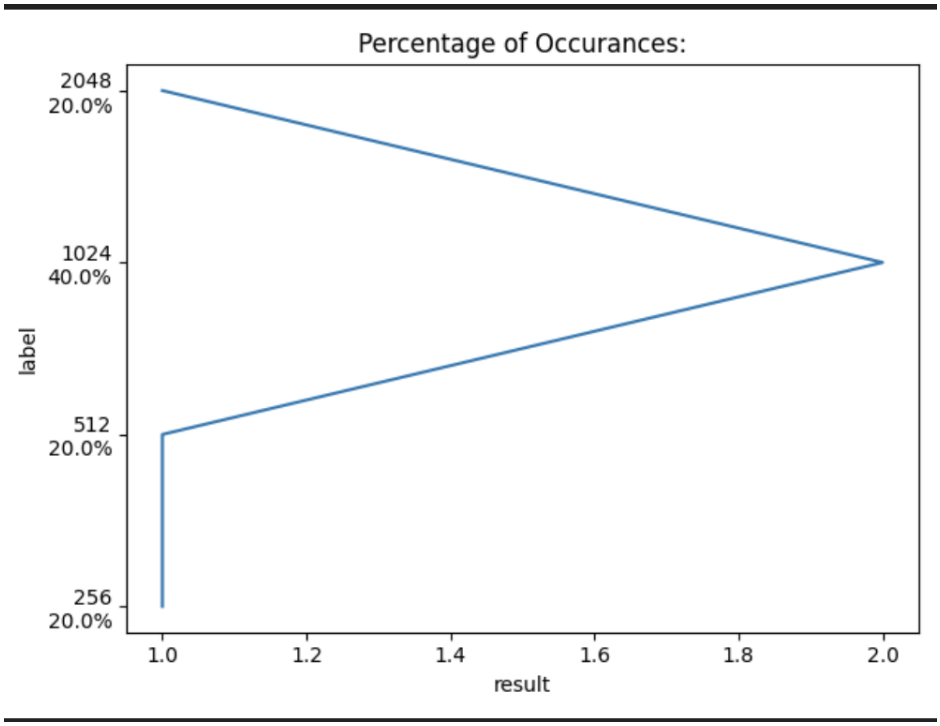
- The number of empty tiles in grid, this gives opponent to pick a random position to place the tile
- Heuristic to place two lowest numbered tiles, together so that it gets merged.
- The last heuristic is average highest score calculation, this shows the progress of the game

Results:

The results are obtained after 50 games, where each time a highest score of 2048 is achieved. The occurrences of 2048 in puzzle is more, with alpha beta pruning when compared to regular Minimax algorithm. Out of 50 games, 14 games have reached the goal 2048 whereas 19 games reached the goal with alpha-beta pruning. Comparing the time complexity, Minimax algorithm took 4.50 mins whereas alpha beta pruning took 1.45 mins.

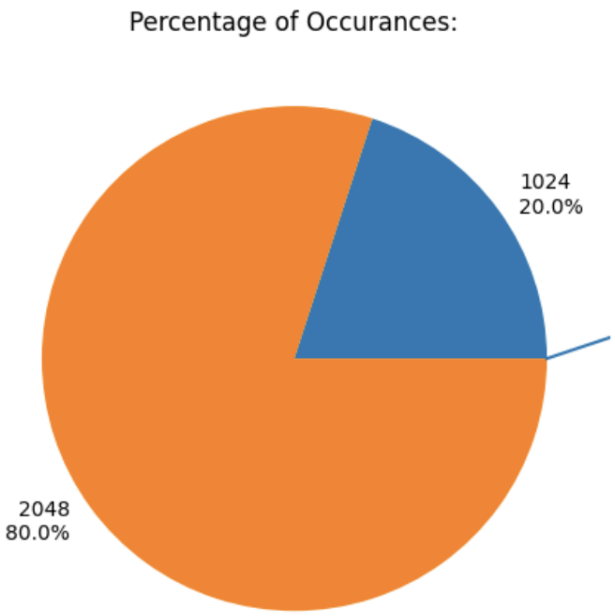
For 5 games it did not reach the goal 2048:

```
Highest Score for this game: 256
Max tiles array [2048, 512, 1024, 1024, 256]
Result [1 1 2 1]
Labels ['256\n20.0%', '512\n20.0%', '1024\n40.0%', '2048\n20.0%']
Percentage of 32's 0.0
Percentage of 64's 0.0
Percentage of 128's 0.0
Percentage of 256's 20.0
Percentage of 512's 20.0
Percentage of 1024's 40.0
Percentage of 2048's 20.0
```



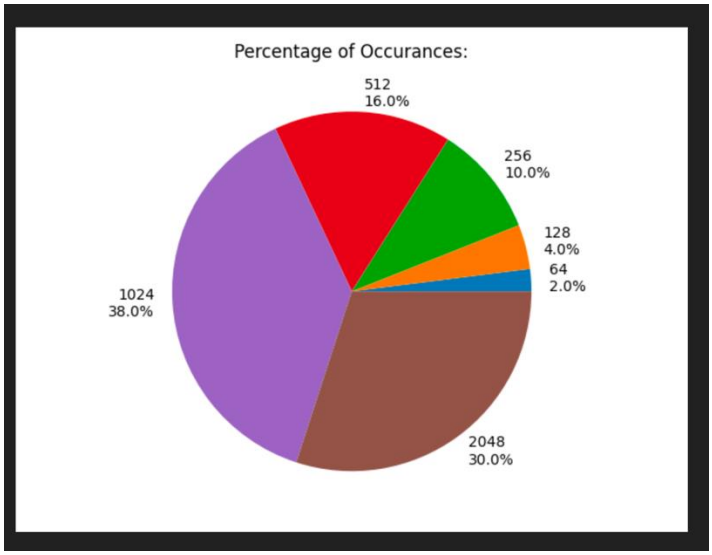
For 10 games:

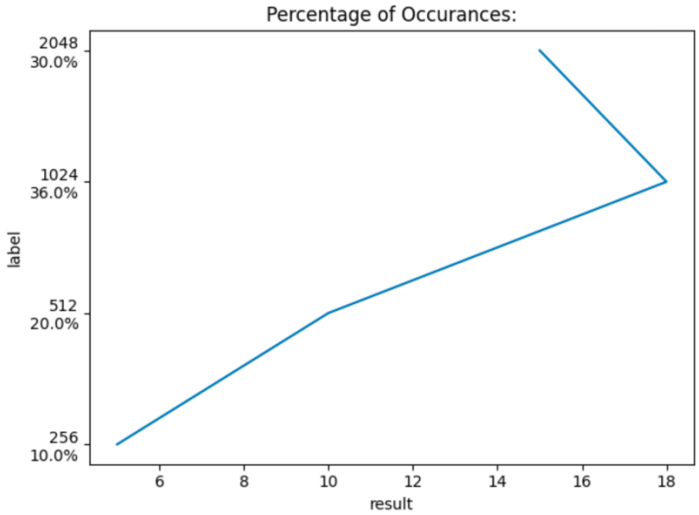
```
Highest Score for this game: 1024
Max tiles array [2048, 2048, 2048, 2048, 1024]
Result [1 4]
Labels ['1024\n20.0%', '2048\n80.0%']
Percentage of 32's 0.0
Percentage of 64's 0.0
Percentage of 128's 0.0
Percentage of 256's 0.0
Percentage of 512's 0.0
Percentage of 1024's 20.0
Percentage of 2048's 80.0
```



For 50 games it reached goal state of 2048, there are 22% of tiles on the grid that are numbered with 2048.

```
Labels ['64\n2.0%', '128\n2.0%', '256\n8.0%', '512\n22.0%', '1024\n44.0%', '2048\n22.0%']
Percentage of 32's 0.0
Percentage of 64's 2.0
Percentage of 128's 2.0
Percentage of 256's 8.0
Percentage of 512's 22.0
Percentage of 1024's 44.0
Percentage of 2048's 22.0
```





Conclusion:

MiniMax Algorithm worked well for solving 2048 puzzle. Alpha-beta pruning has reduced the complexity of algorithm and increased efficiency. However, it has reached goal of reaching 2048 out of 50 trials. Algorithms like Expectimax will show better efficiency and more probability of winning when compared to Minimax.

Code link:

https://github.com/ylsahithi/MiniMax/

References:

https://www.javatpoint.com/mini-max-algorithm-in-ai

https://www.upgrad.com/blog/min-max-algorithm-in-ai/

https://github.com/SrinidhiRaghavan/AI-2048-Puzzle

https://github.com/yangshun/2048-python

http://www.cs.columbia.edu/~sedwards/classes/2020/4995-fall/reports/Minimax.pdf

Images are from:

https://www.hackerearth.com/blog/developers/minimax-algorithm-alpha-beta-pruning/
https://slideplayer.com/slide/10448386/