Artificial Intelligence 2048 Minimax

Ertuğrul Yılmaz - 041701030

Introduction

2048 is a puzzle game which build on a 4x4 grip or more. The game grid can have one or more tiles and these tiles are numbered as the power of two.A tile can hold two as minimum value and 2048 as maximum value. In this puzzle the goal is to reach the tile which has the value 2048. Player can have at most four different moves left, right, down and up. Player chooses a possible move then all the tiles on grid moves as far as possible according to input direction. For a moving tile there are three possible outcome and these are tile can collide with the grid or tile can collide with another tile or tile stays on the current location. If a tile collide with another tile than there is two possible outcome, these are moving tile holds a different value than collided tile then stays in the location or moving tile holds the same value with collided tile then they merge into one tile which holds the sum of old two tiles. After every action the puzzle inserts a tile which holds the values of two or four to a random empty location. In this puzzle minimax algorithm can be used to reach the 2048 tile. Can be assumed that puzzle has two player, first player is trying reach 2048 tile by making the possible moves and second player is trying to make harder to reach by putting the tiles on grid. While implementing the algorithm we say that first is trying maximizing the score and second is trying to minimizing the score.

Explanation of Minimax Algorithm

Minimax algorithm is a backtracking algorithm which means it recursively analyzes the all possible future by considering both players expected actions. The idea is algorithm assumes that there are two players against each other and one of the players is taking action for maximizing the score but other player is trying minimize the score by its actions. Minimax algorithm can be represented in tree structure for better understanding. It generates every possible action as a node after generating a depth then it is continuously keeping that way for next possible future.

For 2048 puzzle algorithm assumes agent is maximizer and the random tile generator is minimizer. Maximizer makes its move according to algorithm then minimizer makes its move. This process is continues until there is no possible future. For maximizing move the algorithm can generate at most four(left, right, down, up) nodes. When minimizer makes a move it can put two different numbers(2, 4) to one of an empty cell then 2*k node can be generated for the minimizing move where k is the number number of empty cells on the grid. Algorithm computes a score for winning chance for every node.

Heuristic Function

For this project, it is observed that player will have better chance to win if the the biggest tile is in a corner. Three four possible corner for this puzzle, these are left-top, right-top, left-bottom and right-bottom. By using this approach a small value tile distance from the corner would be bigger. It increases chance of merging with new tiles because a random tile can only have the values of two or four. Heuristic function is defined according to this approach. The score matrix defined as:

```
[
[[ 3,  2,  1,  0],[ 2,  1,  0, -1],[ 1,  0, -1, -2],[ 0, -1, -2, -3]],#first line
[[ 0,  1,  2,  3],[-1,  0,  1,  2],[-2, -1,  0,  1],[-3, -2, -1, -0]],#second line
[[ 0, -1, -2, -3],[ 1,  0, -1, -2],[ 2,  1,  0, -1],[ 3,  2,  1,  0]],#third line
[[-3, -2, -1,  0],[-2, -1,  0,  1],[-1,  0,  1,  2],[ 0,  1,  2,  3]]#fourth line
]
```

Figure 1 : Heuristic in Code

First line stands for left-top. Second line for right-top. Third line for left-bottom. Fourth line for right-bottom. It can be represented with much clear look:

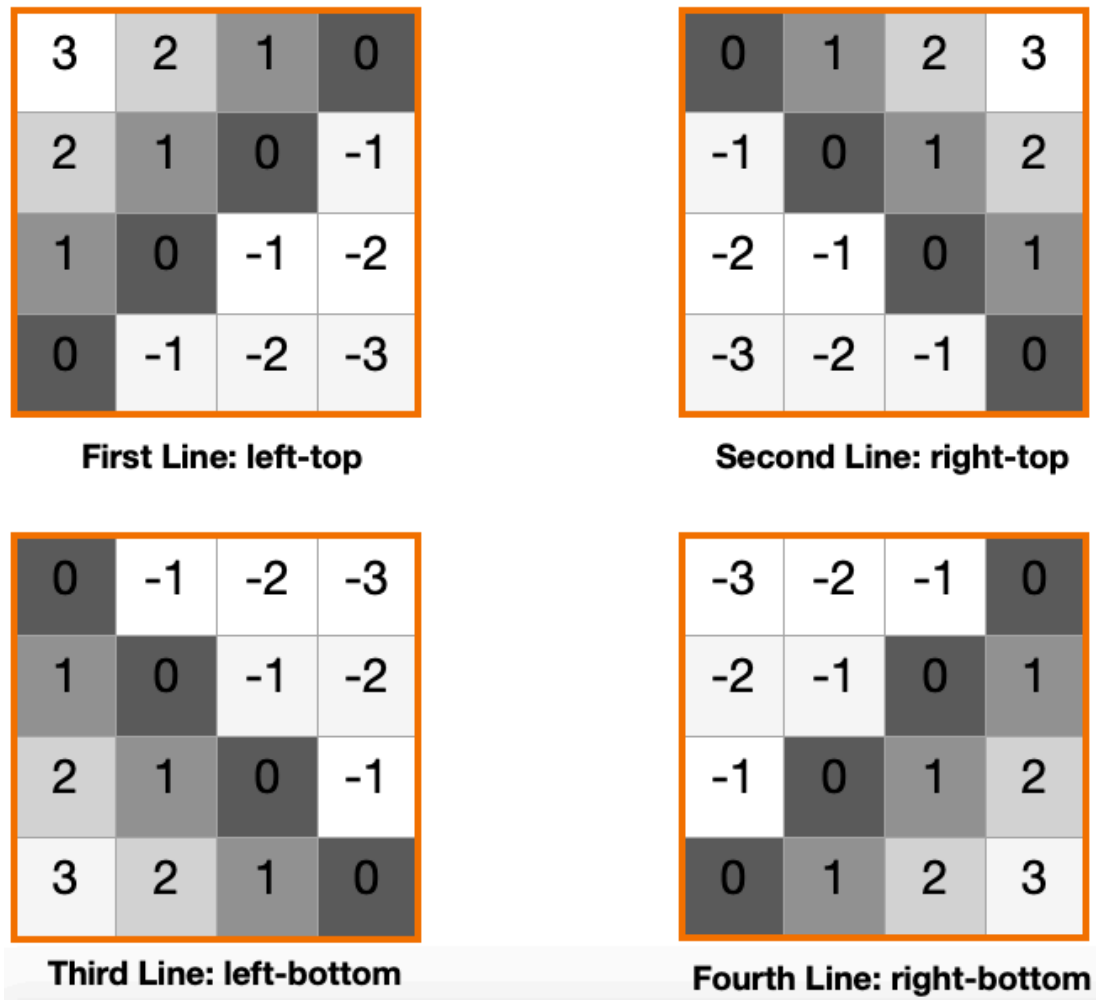

Figure 2 : Heuristic on Puzzle Grid

Code

For this puzzle after deciding the algorithm which is going to be used to solve and defining the heuristic cleverly would make the AI more efficient. Algorithm also has pruning with tree depth four for time efficiency. Here is some explanation about the classes which was used:

Helper.py

Class is the main class for the basic tools which is needed to implement the algorithm. Explanations of the some important functions is below.

- terminal(grid): It returns true if there is no available moves for the current grid.
- Eval(grid): It evaluates the heuristic(calculates the score) for given four situation. Returns the scores as array.

Grid.py

Class includes all the possible movement functions for a grid. Functions can be listed as:

- clone(self) : Return a copy of the current grid.
- setCellValue(self, pos, value): Changes the desired tile value to given value.
- insertTile(self, pos, value): Calls the setCellValue function for the desired change.
- getAvailableCells(self): Returns locations of all the empty cells with an array which is named as ‘cells’.
- getMaxTile(self): It find the tile which has the maximum value on the grid and returns it.
- canInsert(self, pos): Returns true if desired location is empty.
- move(self, dir): It moves the whole according to desired direction(dir).
- moveUD(self, down(boolean)): Moves the grid up or down.
- moveLR(self, right(boolean)) Moves the grip left or right.
- merge(self, cells): It merges the two cell if it is possible.
- canMove(self, dirs): It returns true if grid can move through the desired direction. Otherwise returns false.
- getAvailableMoves(self, dirs): It returns the available moves as an array for current grid.
- getCellValue(self, pos): It returns the specific tile value from the desired position o grid.

BaseAI_3.py

All AI's (minimizer and maximizer) are inherited from this class. Because algorithm recursively makes the calculations. It only calls the getMove(self, grip) function and passes to next class.

ComputerAI_3.py

It is the opponent players' class which gets the available cells first then return a random location from empty cells to place the random tile.

PlayerAI_3.py

This class for our main agent. It is the class for player AI who makes the move(left, right, up, down). AI first gets the available moves then calls the minimax algorithm for every available move. After it makes the move which has higher maximum score than other available moves.

Minimax_3.py

Includes the implementation of minimax algorithm. Maximize and minimize function are calling each other according to desired depth limit for search tree. Class has tree function.

- Decision(gird, max): The depth of the search is defined in a variable named 'limit'. If parameter 'max' is true then it call the maximize function first, otherwise it calls the minimize first. Returns the max score or min score respectively.
- Maximize(grid, depth, start): Calculates the score first then calls the minimize function. Returns the score if the node is a terminal or depth limit is reached.
- Minimize(grid, depth, start): Calculates the score first then calls the maximize function. Returns the score if the node is a terminal or depth limit is reached.

Minimaxab_3.py

Includes the implementation of minimax algorithm but it also includes pruning. Pruning can be known as simplification, it give us better time efficiency. The idea behind the pruning is; While looking for maximizing, if a node has less score the old node then algorithm stops working for current node and goes to the next node. While looking for minimizing algorithm does the same step but if the next score is smaller than old score. Pruning is very important for time efficiency because algorithm doesn't need look for the nodes which isn't going to be used. This project uses the alpha-

beta pruning. Alpha and beta are the parameters which are used to keep track of the best scores or worst scores. Alpha comes from maximizing and beta comes from minimizing. While maximizing is alpha is bigger than beta, so it is obvious that current node is not going to be used and no need to calculate the recursive call for the node. For minimizing the idea is the same. Functions are same with Minimax_3.py but also has the pruning part.

GameManager_3.py

Them main class of the puzzle. The computer AI and the opponent AI is created in this class. Class is respectively creates the computer AI, opponent AI, displayer(basicUI). Puzzle starts with the insertion of a random tile grid then computer AI start playing and class checks if the desired move is valid. After opponent places a tile. It goes like that until there is no available move for the grid. Explanation of some functions are below.

- isGameOver(self): Returns true is current grid has no available move.
- insertRandomTile(self): Inserts a random tile to the grid.
- start(self): Starts the whole process.

The main is simple:

```
def main():
    gameManager = GameManager()
    playerAI    = PlayerAI()
    computerAI  = ComputerAI()
    displayer   = Displayer()

    gameManager.setDisplayer(displayer)
    gameManager.setPlayerAI(playerAI)
    gameManager.setComputerAI(computerAI)

    gameManager.start()
```

Figure 3 : Inside of the main function

Conclusion

In conclusion, the Minimax algorithm is very useful algorithm for two opposite player tasks. It can be easily implemented to any project after understanding of the recursive nature of minimax. For better results depth of three algorithm can be increased but it comes with less time efficiency. That is the reason for the usage of pruning. Pruning is a very import tool for simplifying the time complexity of the algorithm because the number number of nodes in the tree are not increasing linearly most of the cases and without pruning there will huge time penalty.

References

- A.L. Aradhya, “Minimax Algorithm in Game Theory”, GeeksForGeeks, May. 28, 2019. Accessed on: Jan. 18, 2021. [Online]. Available: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/
- javaTpoint, “Alpha-Beta Pruning”, javaTpoint, Accessed on: Jan. 18, 2021. [Online]. Available: https://www.javatpoint.com/ai-alpha-beta-pruning
- S. Raghavan, “AI-2048-Puzzle”, GitHub, Jul. 15, 2017. Accessed on: Jan. 18, 2021. [Online]. Available: https://github.com/SrinidhiRaghavan/AI-2048-Puzzle
- S. Lague, “Algorithm Explained - minimax and alpha-beta pruning”, YouTube, Apr. 20, 2018. Accessed on: Jan. 18, 2021. [Online]. Available: https://www.youtube.com/watch?v=l-hh51ncgDI