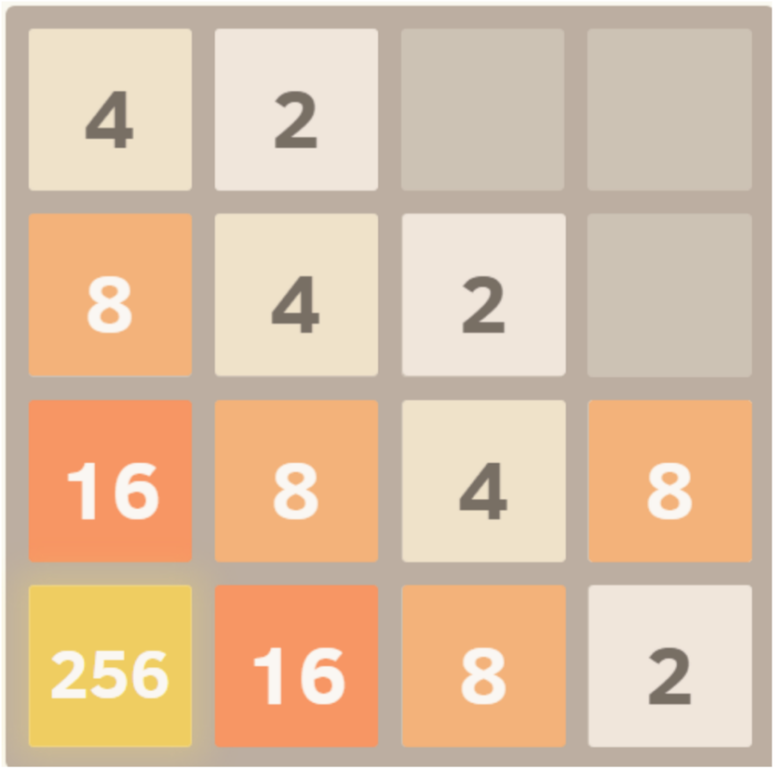
Homework 2 - Report

AI for solving 2048

An Artificial Intelligence program which solves 2048 using Minimax with Alpha Beta Pruning. The report elucidates the implementation of the program, the rationality behind the heuristics and the performance statistics.



Files Submitted : PlayerAI.py

Implementation

MiniMax with Alpha-Beta pruning

The program implements Adversarial searching using MiniMax with AlphaBeta pruning. The playerAI is considered to be the Max player and plays to maximize the score (achieve 2048) and the Min player is the computer AI. The computer AI randomly generates 2 or 4 in a random position with its associated probability.

The Max player, searches through all possibilities moves (a 2 or 4 in each empty spot available) made by the computer AI and chooses the best move to maximize the score. The The Max player prunes the search tree using a heuristic score and tries to move best for worst case scenarios.

Also, in order to match the time constraint imposed, I iterated over different depths and compared iterative depth search with fixed depth search. It was observed that, to maximize the chances, the program needs at least a depth of 3 and a depth of 4 slows it down and does not meet the 1 second constraint. Hence iterative depth search does not provide significant gains as it always runs at depth of 3 and runs at 4 only when empty cells are less than 2. Only towards the very end of the game does it move to depths of 6 or 7.

So instead of using IDS, I am using a depth selective search. If the number of empty cells is less than or equal to 3, I increase the depth to 4 and otherwise run it at depth of 3. This works faster and produces the same results. When the number of cells are less than 3, either the game is approaching an end or several merges might be needed. Either way it’s a crucial point and an increased depth helps make a better choice.

Heuristics

The AI uses 5 parameters to provide a heuristics score or evaluation of the grid at any state of the game.

1. **Pattern:** This heuristic calculates the difference between a cell and its neighbors in all directions and sums them up along individual directions in both increasing and decreasing orders. This heuristic makes sure that the grid is optimized and numbers are merged in a particular pattern.
2. **MaxTile:** Tihs heuristic is simply the larges tile on the board. This is to move the grid towards the highest possible tile.
3. **EmptyCell:** In order for the AI to move around the tiles easily and merge them, there needs to be sufficient empty cells. When large number of empty cells are present, the AI can even be capable of correcting itself when the pattern or smoothness is disturbed. Hence this heuristic is for maximizing the number of empty tiles present.
4. **Score:** This heuristic tries to encourage the AI to merge more cells. It also gives preference to merging cells with larger values.
5. **Smoothness:** This heuristic measures difference between adjacent tiles. The idea is to minimize the difference between neighbors to make it easier to merge the tiles. This way this can help maximize the score.
6. **Islands:** When there are isolated tiles of high value present in the middle of the grid, it gets difficult to merge and move tiles around the grid around. This heuristic gives a score to minimize such isolated tiles.

The heuristics are merged by assigning weights to each of the heuristic so that the utility function can generate an overall score for the grid based on which the next decision can be made. The weights are chosen over trial & error

The Utility function returns a total score of all heuristics.

Performance

The performance of the AI was evaluated by running the program over several iterations (at least 400 times).

The AI performs pretty well. It achieved 2048 about 30-40% of the times. It also achieves 1024 90% of the time. The AI also achieved 4096 twice.

The AI plays well and whenever it achieves 1024, at sometimes due to chance, the order gets distorted and hence it is unable to merge to 2048. The grid will contain 1024,512, 256 and 128 typically.