Report:

Implementation Details:

Heuristics Used:

I have tried the following following heuristics: a) Monotonicity b) Smoothness c) MaxTile value 4) Empty Tiles 5) Gradient Method 6) Penalty.

The best one that seemed to work for me is a combination of Gradient Method and adding penalty for having clusters.

About Gradient Method:
It assigns weights to each cell as shown below - enforcing that the Max Tile always tile at top left corner and pattern is also enforced for the merger of tiles.

```
W = [[6, 5, 4, 1],
     [5, 4, 1, 0],
     [4, 1, 0, -1],
     [1, 0, -1, -2]];
```

Score = ΣW_{ij} * grid.map[i][j]

Penalty method:
Basic Idea: Neighbors to be of almost comparable size to be together and penalty for not having it. This is useful for merging and complements the above method.

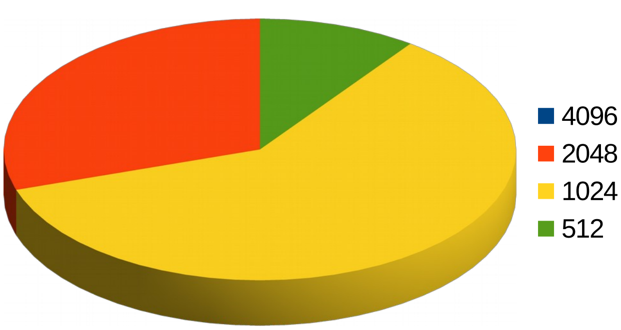
```
for each "cell":
    for each neighbor of cell:
        penalty += (cell.value – neighbor.value)
```

Final_Score = score – penalty --→ My Heuristic

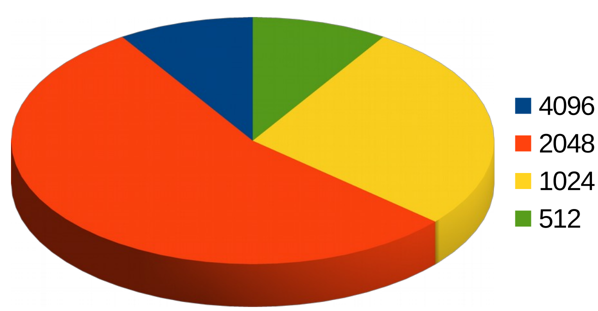
Success Percentage:

With Alpha-Beta Pruning I was able to achieve **60-75%** success rate on my runs and my best is **4096**.

Comparison between Vanilla Minimax vs Minimax with Alpha Beta pruning:



MaxTile distribution



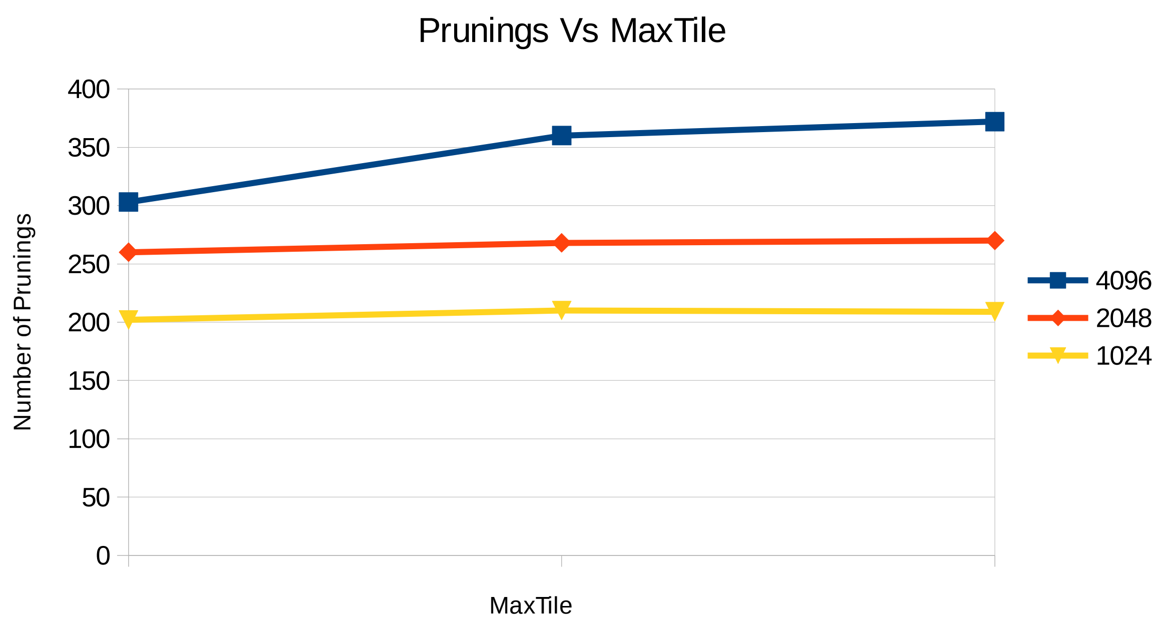
Vanilla – Minimax Implementation

MiniMax with AlphaBeta Pruning

Advantage of Alpha Beta pruning I noticed over Plain vanilla minimax:

Maximum depth reached as result of pruning was much higher compared to vanilla version and I could explore more game states, which in turn helped me reach better scores compared to normal version.

Number of Pruning's vs MaxTile reached:



As can be seen from the above graphs higher number of pruning's lead to better game scores.

Note about implementing Iterative scheme:

Initially I was under the impression that using IDS will waste time as it redoes the same computations again. Then I realized as the number of available tiles decreases I can go further down the search tree and explore more states. Adding in IDS helped me in achieving 2048 with 60-70% success rates.