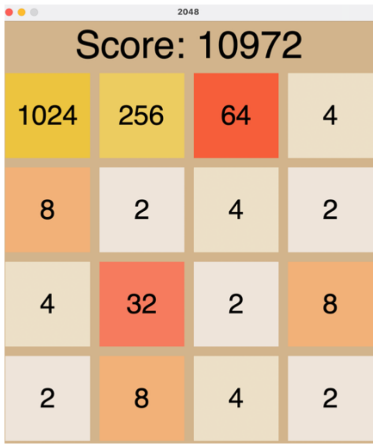
Project: 2048

Objective: To build an agent that plays 2048 effectively.

Strategy: Use search algorithms, iterative deepening, and heuristics.

Development Phases:

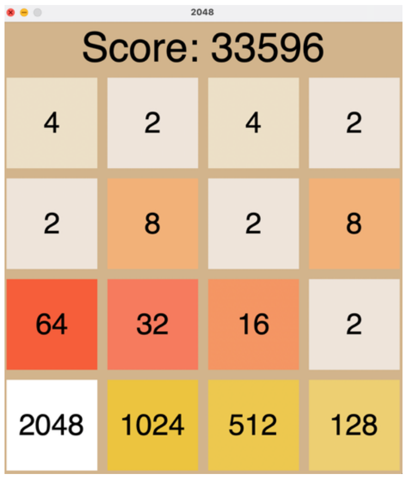
1. Started with MinMax.py and improved heuristics, the average score was 7276, max score was 10972.



Score: 10972

1024	256	64	4
8	2	4	2
4	32	2	8
2	8	4	2

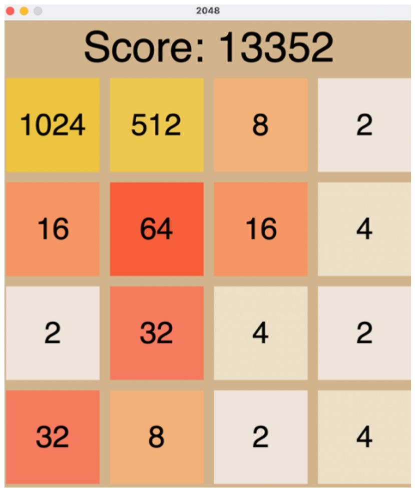
2. I added Expectimax with a fixed depth search of 4, I also added move ordering. The code worked great, my average score was over 30,000 but it never made it on the leaderboard



Score: 33596

4	2	4	2
2	8	2	8
64	32	16	2
2048	1024	512	128

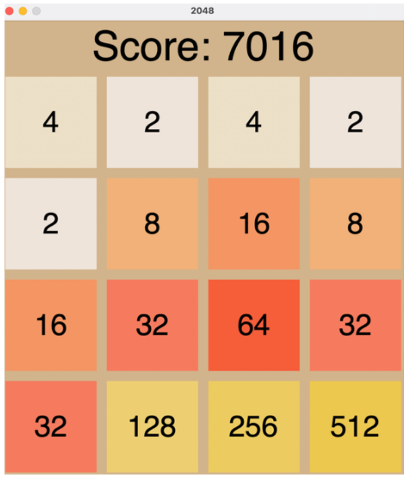
3. I brought down the fixed depth to 3 and the average score decreased; it still didn't make it on the leaderboard.


2048

Score: 13352

1024	512	8	2
16	64	16	4
2	32	4	2
32	8	2	4

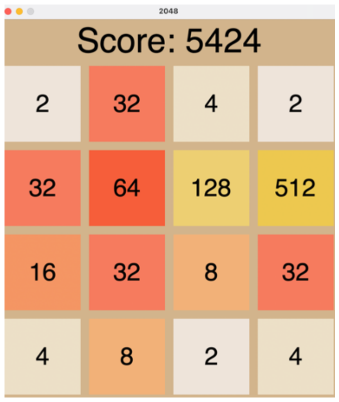
4. I brought down the fixed depth to 2 just to make it on the leaderboard and the average score decreased further.


2048

Score: 7016

4	2	4	2
2	8	16	8
16	32	64	32
32	128	256	512

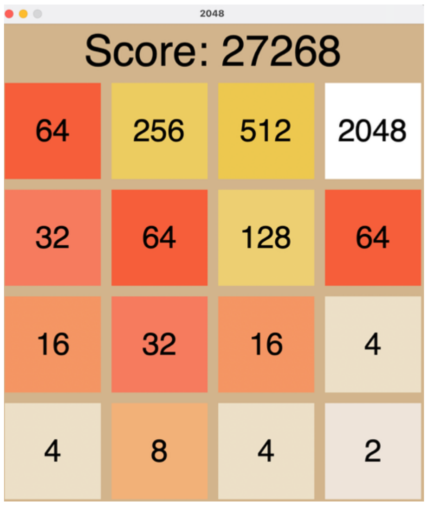
5. I tried 4-Tuple, but my score wasn’t consistent


2048

Score: 5424

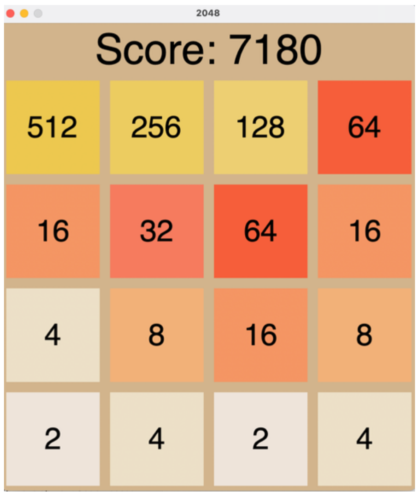
2	32	4	2
32	64	128	512
16	32	8	32
4	8	2	4

6. I switched back to Expectimax, added reward for empty tiles, reward for high value tiles in the corner, reward rows and columns for tile ordering, negative reward for large difference between neighboring tiles. This would perform well 5/10 times.



Score: 27268

64	256	512	2048
32	64	128	64
16	32	16	4
4	8	4	2

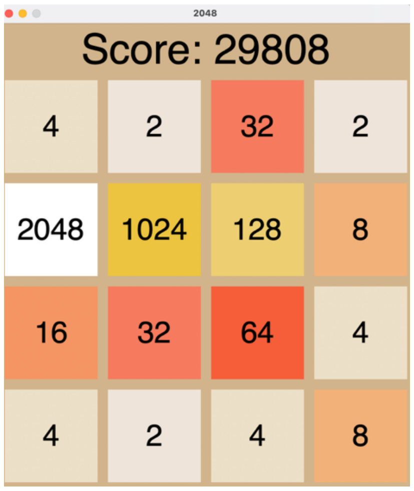


Score: 7180

512	256	128	64
16	32	64	16
4	8	16	8
2	4	2	4

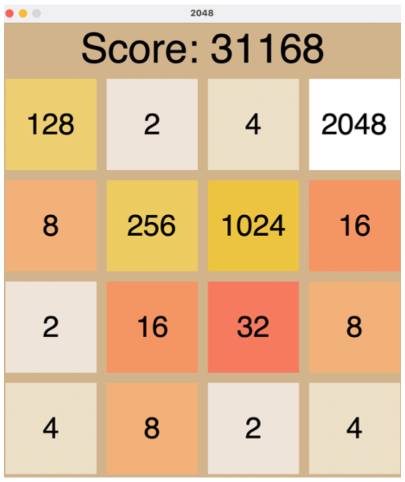
7. Instead of defining depth I used iterative deepening, and started with depth = 2, and tried going deeper if it doesn't time out - for each depth - I added move ordering and added fall back to random move if it would time out before decision.

8. Then, I used Expectimax, added one logic to average the value of states. I improved my heuristics by rewarding more empty tiles, max tiles in corner, negative reward for differences in adjacent files, negative reward for rows and columns that are not sorted based on values, reward high value tiles on the top left, and limited branching to speed up computation. I also added if-else for move ordering to choose the best mover order. This is my best version so far and scored over 28000 consistently.



Score: 29808

4	2	32	2
2048	1024	128	8
16	32	64	4
4	2	4	8



Score: 31168

128	2	4	2048
8	256	1024	16
2	16	32	8
4	8	2	4