Project: 2048

Name: Khevna Vadaliya

Banner ID: 001395699

Course: CSCI - 4740 - 01

OBJECTIVE: *To develop an intelligent 2048 AI Agent iteratively that uses MinMax with best heuristics, Expectimax, TD-Learning or Q Learning and Monte Carlo Tree Search to explore deep strategic lines and iteratively integrating these methods so the agent continually sharpens its play, maximizes expected cumulative score and geometric mean, and reliably reaches and surpasses the 2048 tile with highest score each time it runs.*

INITIAL SETUP

WHAT DID I DO?

Initially, I decided to use the Rule.py code and enhance it with the best heuristics for the 2048 game. At first, all I could think of was implementing better heuristics and checking their performance stats after playing 100k games. This could give me an indication of where to work and what to improve. Based on that, I will try to do Expectimax in that code

TESTING SUMMARY

1st Code: MinMax with Heuristics

- Approach: This agent tries out all four possible moves one step ahead, uses a handy formula to score each resulting board—counting empty spaces, penalizing big jumps between tiles, rewarding nice ordering (monotonicity), giving a big bonus if the largest tile sits in a corner, adding up weighted tile positions, and even factoring in the size of the biggest tile—and then just picks the move with the highest score.

- Results (100+ runs):
    - Average: 2333
    - Median: 2332
    - Best: 3684

Observations:

- The baseline Expectimax agent comfortably exceeds 2 000 on average and peaks near 3 700.
- However, it suffers from high variance—occasionally stalling below 1 000 when the board turns unfavorably.
- These results establish a clear baseline: future iterations must raise the average toward 3 000+, tighten the score distribution, and eliminate sub-1 000 outliers to reliably secure the 2048 tile every run.

DEVELOPMENT LOG 1

WHAT DID I DO?

I implemented Expectimax with iterative deepening after analysing the testing stats of my initial code. It is better at this point, out of 5 runs, this version reaches 2048 twice which is an improvement. In this iteration, I mainly improved structure with better decision logic. I implemented heuristics such as corner and center indices, the 16-entry gradient map, penalty ordering, and move-preference list class-level constants so they're allocated just once and clearly documented by name. The primary search method, findMove will always seeds a valid default move before entering its iterative-deepening loop and then refines that choice at each depth, ensuring setMove() is never ruled out. Additionally, I also streamlined the max_node and chance_node. However, I maintained evaluation function mathematically identical but I divided it to well-labeled sections: an empty-tile bonus, a corner placement reward and penalty, our tile gradient score, monotonicity on rows and columns, merge-potential counting, and off-pattern penalties. My another short term goal at this point is to reach 2048 tile in each run.

TESTING SUMMARY (Development Log 1 vs Initial Code)

2nd Code: Expectimax with Tuned Heuristics

- Approach: Depth-3 Expectimax with the same structure as the baseline, but using aggressively retuned evaluation weights (heavier empty-tile bonus, stronger smoothness/monotonicity factors, steeper gradient scores, and harsher corner/center penalties).

- Results (100+ runs):

    - Average: 13832

    - Median: 14054

    - Best: 33,696

Observations:

- By fine-tuning the heuristic constants, this version jumps the average from ~2 300 to ~13 800, dramatically raises the ceiling (into the mid-30 000's), and reaches 2048 almost every time.

CONCLUSION

In this Project, Customary 2048 agents that operate mostly on heuristics have one of two shortcomings. They either execute very fast and simple policies yet do not create notable results because of their simplicity or powerful search algorithms but are implemented with inefficient heuristic evaluation functions. To uncover the possible performance of Expectimax, I began by comparing Expectimax with the heuristic agent; next, I used careful heuristic engineering to craft what would be in my judgment an expert policy, which I then reverse-engineered into a heuristic from the sample plays of Expectimax. The expert heuristic tries to place weights on every tile depending on whether it is on one of the corners of the grid, whether the tile is new and empty, and even evaluation of combinations of low or high tiles in a gradient in that corner reference frame. The aim is to encourage the agents to coalesce higher tiles in one corner and keep lower tiles free in the opposite corner. Using these heuristics has yielded average scores well over 13,700, with rare falls below 1000-that only occurred very little-a guarantee of the 2048 tile in most of its runs, and peak scores well above 30,000, all achieved without touching a single line of search code! Also, again through trial and error on each parameter separately, utilising the reward of these heuristics and penalties by coarse adjustments led to a substantial increase both in stability and in top performance, all while leaving the search algorithm unchanged! Going forward, combining Monte Carlo Tree Search and reinforcement learning will improve strategic look-ahead, and by generation, we have stood by evidencing the importance of well-balanced heuristics on AI game play; in most cases, they are what makes a mediocre entrant go through to the top shelf agent.