Developments for ‘2048’ Project

This write-up documents the entire development process of a 2048 agent, including test results, proposed improvements, and design decisions.

MinMax Tree Search

1. Version / Baseline (`MinMax.py`): MinMax with Simple Heuristic

- Method:
  Minimax Tree Search; Simple Heuristic
- Results:

Average Score	Maximum Score	Median Score	2048 Percent	8192 Percent	Number of Games
9,872.67	25,876	7,578	1.4	0.0	72

- Observations:
  - MinMax is a poor match for 2048’s stochastic nature. It treats the random tile spawns too pessimistically, which limits average performance.
- Plan:
  - Replace Minmax Tree Search with ExpectiMax Tree Search

ExpectiMax Tree Search

2. Version (`MyAgent_Version02.py`): ExpectiMax with Simple Heuristic

- Methodological improvements over the previous version:
  Minmax Tree Search → ExpectiMax Tree Search
  - The “max” nodes are the AI choosing a move; the “chance” nodes represent random tile spawns.
  - At a max node, the action with the highest expected value is taken; at a chance node, the expectation (weighted average) over all random outcomes is computed.
- Results:

Average Score	Maximum Score	Median Score	2048 Percent	8192 Percent	Number of Games
10,317.12	26,460	10,632	3.2	0.0	250

- Observations:
  - Replacing MinMax with ExpectiMax gave a clear, measurable improvement: ExpectiMax models random tile spawns correctly and boosts the median and 2048 rate.
  - The gain is modest (≈4.5%) because the heuristic still remains simple; the search model improved but leaf evaluation still limits quality.
  - This confirms the hypothesis that modeling chance is necessary but insufficient – better leaf evaluation is the main lever.
- Plan:
  - Replace the simple heuristic by an extended heuristic.

3. Version (`MyAgent_Version03.py`): ExpectiMax with Extended Heuristic

- Methodological improvements over the previous version:
  Simple Heuristic → Extended Heuristic
  - For the heuristic, a composite score is built by summing several heuristic components in a weighted linear manner.
    1. Heuristic component (`base_game_score`)
       - It is the game’s own score (sum of merged tile values that have been created so far).
       - It gives baseline progress toward winning.
    2. Heuristic component (`corner_score`)
       - It is a big positive bonus if the largest tile sits in one of the four corner indices; otherwise a penalty (and a special-case penalty if the board is empty).
       - The standard 2048 strategy is to keep the max tile in a stable corner to avoid breaking the ordered gradient and to reduce the chance of losing mobility.

3. Heuristic component (`empty_tile_score`)
   - It is the weighted number of empty tiles on the board. More empty cells mean more legal moves and higher chance to combine tiles later; empties strongly correlate with survival.

- Results:

Average Score	Maximum Score	Median Score	2048 Percent	8192 Percent	Number of Games
24,398.63	59,232	26,020	55.0	0.0	131

- Observations:
  - Introducing structured heuristic components (base score, corner, empties) caused a dramatic improvement. This shows heuristic design is the largest driver of performance so far.
  - The 2048 attainment rate jumped to 55% - strong evidence that encouraging corner-max and maintaining empties pays off.
  - Median and mean are both much higher and max also increased, indicating the heuristic improves both typical and best-case play.
- Plan:
  - Replace the extended heuristic by a further-extended heuristic.

4. Version (`MyAgent.py`): ExpectiMax with Further-Extended Heuristic

- Methodological improvements over the previous version:
  Extended Heuristic → Further-Extended Heuristic
  - For the heuristic, a composite score is built by summing several heuristic components in a weighted linear manner.
    1. Heuristic component (`base_game_score`): *Refer to version 3 for more details.*
    2. Heuristic component (`corner_score`): *Refer to version 3 for more details.*
    3. Heuristic component (`empty_tile_score`): *Refer to version 3 for more details.*
    4. Heuristic component (`snake_pattern_score`)
       - Is a weighted sum of tile values arranged in a predefined snake ordering (weights in `SNAKE_WEIGHTS`), then normalized by `max_tile_exponent`.
       - The intent is to reward boards that follow a strong descending snake gradient (largest tile at the start of the snake).
       - The "snake" (or gradient) arrangement keeps tiles ordered and makes merges predictable while preserving the max tile location.
    5. Heuristic component (`monotonicity_score`)
       - It counts rows/columns that are monotonic (non-decreasing or non-increasing) and multiplies by `MONOTONICITY_WEIGHT`.
       - Monotonic rows/columns indicate a gradient where tiles consistently increase or decrease across a line – that reduces the chance of blocking merges and helps keep the big tile in place.
    6. Heuristic component (`merge_score`)
       - It counts adjacent equal non-zero tiles (horizontally and vertically) – each potential merge gets `MERGE_WEIGHT`.
       - This indicates immediate opportunities to increase score and create higher-value tiles. A board with many adjacent equal tiles is tactically stronger.
    7. Heuristic component (`smoothness_score`):
       - This penalizes large differences between neighbouring tiles (summing absolute differences of exponents along rows and columns), multiplied by `SMOOTHNESS_SCALE`.
       - Smoother gradients are less likely to leave isolated high tiles that are hard to merge; smoothness encourages gradual value transitions.
- Results:

Average Score	Maximum Score	Median Score	2048 Percent	8192 Percent	Number of Games
41,182.10	82,760	35,712	89.6	0.0	250

- Observations:
  - Adding snake pattern, monotonicity, merge and smoothness components produced another substantial jump. These features capture long-term board geometry and merge likelihood.

- ◦ Very high 2048 rate (≈90%) shows the agent reliably reaches the basic goal. Max score nearly doubled from Version 03, indicating better deep play.
  - ◦ Median < mean and a very large max indicate there is still high variance / room for spectacular runs; but typical play (median) is now strong.
- Plan:
  - Add methodology for storing the best move.

5. **Version (`MyAgent_Version05.py`): ExpectiMax with Further-Extended Heuristic and Best Move Memory**
   - Methodological improvements over the previous version:
     + Best Move Memory
     - Remembers the single move that the search previously judged best and tries it first next time.
     - If the search tries the most promising move first, it often finds a strong (or the strongest) line earlier. In MiniMax/Alpha-Beta this increases pruning; in ExpectiMax (here) it helps the search discover high values earlier which should improve the anytime result under a time limit.
   - Results:

Average Score	Maximum Score	Median Score	2048 Percent	8192 Percent	Number of Games
39,845.09	75,488	35,782	77.3	0.0	77

   - Observations:
     - Introducing a single “best move memory” produced a small decline in average and a noticeable drop in 2048 rate (89.6 → 77.3).
     - One possible reason for the drop is stale-move bias: storing one global “best move” (rather than per-state/transposition) can bias the search toward moves that were good under different board contexts, reducing adaptability.
   - Outlook:
     - During development, various versions of move ordering were tested, but none were found to improve the overall performance of the agent.
     - Hybrid Search: Run full ExpectiMax to some shallow depth d, then stop and evaluate the leaf with a learned value function (neural net) instead of the handcrafted heuristic. This should give more accurate leaf evaluations and enables to search deeper effectively.

Summary of Results

Ver-sion	Description	Avg. Score	Relative Change in ‘Avg. Score’	Max. Score	Med. Score	2048 %	8192 %	Num. of Games
1	Minimax Tree Search with Simple Heuristic	9,872.67	-	25,876	7,578	1.4	0.0	72
2	ExpectiMax Tree Search with Simple Heuristic	10,317.12	4.50%	26,460	10,632	3.2	0.0	250
3	ExpectiMax Tree Search with Extended Heuristic	24,398.63	136.49%	59,232	26,020	55.0	0.0	131
4	ExpectiMax Tree Search with Further-Extended Heuristic	41,182.10	68.79%	82,760	35,712	89.6	0.0	250
5	ExpectiMax Tree Search with Further-Extended Heuristic, Best Move Memory,	39,845.09	-3.25%	75,488	35,782	77.3	0.0	77

Discussion of Results

The heuristic design matters most. Replacing MinMax with ExpectiMax gave small gains (Version 01 → Version 02); adding relevant heuristic features (Version 02 → Version 03 and Version 03 → Version 04) produced very large improvements. Structural features (snake, monotonicity, smoothness, merge potential) are highly effective. The search model matters too, but mainly insofar as it exploits a good leaf evaluator. ExpectiMax + good heuristic is far better than Minimax + simple heuristic. But even at Version 04 the distribution is skewed: high maxima indicate the agent can occasionally exploit excellent sequences. Median and mean analysis together shows improvements are both in typical games and best-case runs.