Implementation and In-depth Analysis of Advanced AI Algorithms for Automating the 2048 Game

Yağız Can Akay
TOBB ETÜ Artificial Intelligence Engineering
Söğütözü caddesi TOBB ETÜ Konukevi Yenimahalle/Ankara
yagizakay11@gmail.com

Abstract*—The game 2048 is a well-known puzzle involving strategic tile movements and mergers on a 4x4 grid to achieve increasingly high-valued tiles. This study presents an artificial intelligence (AI) based automated solution for 2048, employing five distinct algorithms: Minimax, Greedy, A*, Expectimax, and Hill Climbing. Each algorithm utilizes specialized heuristics, such as tile monotonicity, smoothness, clustering, and merge potential, carefully weighted to optimize decision-making. To enhance efficiency and decision quality, advanced techniques including alpha-beta pruning, Monte Carlo simulations, memoization via LRU caching, and randomized restarts were implemented. Performance evaluations demonstrate that Expectimax achieves the highest average and maximum scores, while Minimax delivers a balanced trade-off between performance and computational efficiency. Greedy and Hill Climbing algorithms provide faster yet less optimal results. The comprehensive analysis reveals insights into heuristic effectiveness and optimization impacts, guiding future research in enhancing automated decision systems for puzzle games.

Keywords*—2048 game, artificial intelligence, heuristic algorithms, Expectimax, alpha-beta pruning, decision-making optimization

I. Introduction

In recent years, the popularity of puzzle games has significantly grown, driven by their simplicity, challenging nature, and the cognitive engagement they provide. Among these games, the puzzle known as 2048, originally developed by Gabriele Cirulli, stands out due to its strategic depth and accessibility. Played on a 4×4 grid, the core objective of 2048 is straightforward: merge tiles of the same value to generate higher-valued tiles, ultimately aiming to achieve the elusive 2048 tile. However, simplicity in mechanics belies a complex decision-making landscape, making it an ideal environment for applying and testing artificial intelligence (AI) algorithms.

Given the combinational explosion of possible moves and the presence of uncertainty through random tile placements, developing efficient, robust AI systems for 2048 represents a significant challenge. Researchers and enthusiasts alike have explored various computational approaches, including heuristic-driven search methods and advanced probabilistic simulations, to automate optimal play. Despite substantial progress, achieving consistent high-performance gameplay remains computationally demanding, requiring sophisticated heuristics and optimization techniques.

This paper presents a comprehensive analysis and practical implementation of five AI algorithms specifically tailored to automate 2048 gameplay: Minimax, Greedy, A*, Expectimax, and Hill Climbing. Each algorithm integrates advanced heuristic evaluation functions—such as tile monotonicity, smoothness, clustering of high-valued tiles, and potential merges—meticulously optimized through weight adjustments. Additionally, we introduce and apply computational enhancements such as Alpha-Beta pruning, Monte Carlo simulations, memoization (LRU caching), and randomized restarts to balance the algorithms' strategic accuracy with computational efficiency.

We systematically analyze the effectiveness of these heuristic strategies and optimizations, benchmarking their performance through extensive empirical testing. Results provide valuable insights into the capabilities and limitations of each algorithm, revealing clear trade-offs between decision accuracy, computational overhead, and real-time applicability.

Fig. 1: Screenshot of 2048 Gameplay

The subsequent sections of this paper detail the methodologies and heuristic formulations, provide comprehensive descriptions and pseudocode representations of each algorithm, evaluate their performance, and discuss their comparative strengths and weaknesses. Finally, we suggest potential areas for future work, aiming to further refine AI capabilities in puzzle game domains.

II. Methodology

This study employs five distinct AI-based search and decision algorithms to automate gameplay for the puzzle game 2048: Minimax, Greedy, A*, Expectimax, and Hill Climbing. These algorithms differ significantly in their fundamental strategies, computational complexity, heuristic evaluation approaches, and optimization methods.

A. Heuristic Evaluation Funcitons

At the core of each algorithm lies a carefully crafted heuristic evaluation function designed to quantify the desirability of different board states. Four principal heuristics are defined and consistently applied across algorithms, each weighted strategically to reflect their relative importance:

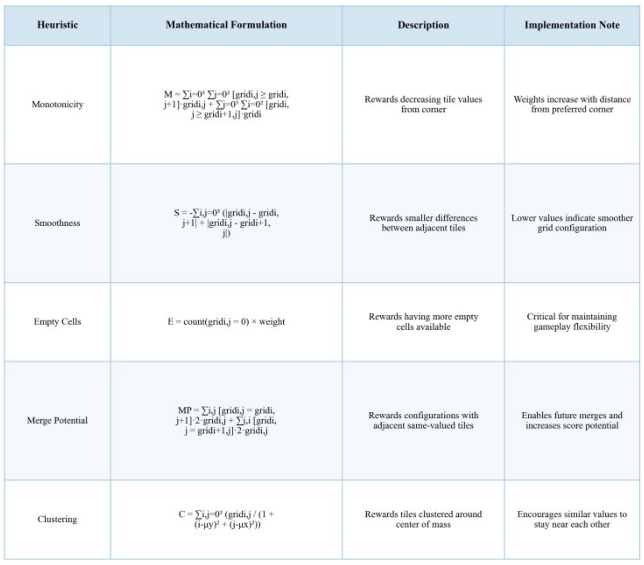
- Monotonicity:
  Rewards grid configurations in which tile values consistently decrease or increase along rows and columns, facilitating easier tile merging and management.
- Smoothness:
  Encourages configurations where adjacent tiles exhibit minimal numerical differences, supporting frequent merges and minimizing isolated high-value tiles.

- Tile Clustering:
  Evaluates and rewards arrangements that position higher-valued tiles in proximity, enhancing the probability of efficient future merges.
- Merge Potential:
  Scores boards higher if adjacent tiles have equal values, indicating immediate merging opportunities and thus potential for substantial scoring.

Additionally, two position-specific heuristics enhance evaluations further:

- Empty Tile Count and Ratio:
  Reward board states with numerous empty tiles, preserving board flexibility and increasing the probability of successful future moves.
- Max Tile Placement:
  Gives higher scores to boards positioning the highest-valued tile in strategically favorable positions—primarily corners and edges—maximizing tile stability and manageability.

These heuristic evaluations are formalized mathematically, providing clear, quantitative criteria for decision-making and allowing consistent algorithmic comparisons



Heuristic	Mathematical Formulation	Description	Implementation Note
Monotonicity	$M = \sum_{i=0}^{3} \sum_{j=0}^{2} [grid_{i,j} \geq grid_{i,j+1}] \cdot grid_{i,j} + \sum_{j=0}^{3} \sum_{i=0}^{2} [grid_{i,j} \geq grid_{i+1,j}] \cdot grid_{i}$	Rewards decreasing tile values from corner	Weights increase with distance from preferred corner
Smoothness	$S = -\sum_{i,j=0}^{3} (\lvert grid_{i,j} - grid_{i,j+1} \rvert + \lvert grid_{i,j} - grid_{i+1,j} \rvert)$	Rewards smaller differences between adjacent tiles	Lower values indicate smoother grid configuration
Empty Cells	$E = \mathrm{count}(grid_{i,j} = 0) \times weight$	Rewards having more empty cells available	Critical for maintaining gameplay flexibility
Merge Potential	$MP = \sum_{i,j} [grid_{i,j} = grid_{i,j+1}] \cdot 2 \cdot grid_{i,j} + \sum_{j,i} [grid_{i,j} = grid_{i+1,j}] \cdot 2 \cdot grid_{i,j}$	Rewards configurations with adjacent same-valued tiles	Enables future merges and increases score potential
Clustering	$C = \sum_{i,j=0}^{3} (grid_{i,j} / (1 + (i-\mu y)^2 + (j-\mu x)^2))$	Rewards tiles clustered around center of mass	Encourages similar values to stay near each other

Table 1: Heuristic Formulas

B. Optimization Techniques

To improve computational efficiency and gameplay performance, several optimization methods were implemented across the algorithms:

- Alpha-Beta Pruning:
  Integrated within the Minimax algorithm, this pruning technique eliminates unnecessary branches in the search tree, significantly reducing computational load and allowing deeper lookahead capability.
- Memoization (LRU Cache):
  Implemented primarily in Minimax and Expectimax algorithms, memoization stores previously evaluated board states, preventing redundant computations and decreasing runtime.
- Monte Carlo Simulations (Rollouts):
  Integrated within Expectimax, Monte Carlo simulations statistically estimate future states' expected values, balancing accuracy with computational cost.
- Randomized Restarts:
  Specifically applied in the Hill Climbing algorithm, this technique reduces local optimum stagnation by periodically introducing random moves, promoting exploration of the search space.
- Move Ordering:
  Utilized in algorithms like Minimax and A*, move ordering evaluates the most promising moves first, optimizing pruning effectiveness and enhancing overall computational efficiency.

C. Performance Evaluation Metrics

Algorithmic performance was empirically evaluated using multiple standardized metrics across numerous test runs:

- Average Score: Average achieved points indicating typical algorithm performance.
- Maximum Score: Highest points reached, indicating best-case capability.
- Average Highest Tile: Average value of the largest tile, indicating algorithm effectiveness in strategic tile management.
- Computation Time: Average time required per move or game, directly reflecting computational efficiency.

These comprehensive metrics offer clear comparative insights, balancing performance accuracy against practical computational demands.

III. Detailed Explanation of Algorithms

A. Minimax Algorithm

Minimax is a well-established decision-making algorithm primarily designed for turn-based, adversarial games involving two players. In the context of the 2048 game, we adapt Minimax to the scenario where one player (the maximizer) is actively making strategic moves, while the second player (the minimizer) is represented by the game itself, randomly placing new tiles on the grid.

Minimax functions by constructing a game tree where nodes represent game states, and edges denote possible moves. Each level alternates between the maximizing player's moves and minimizing random tile placements by the game:

- Maximizing Level: Chooses the move that maximizes potential future rewards.
- Minimizing Level: Assumes worst-case placement of new tiles to evaluate the robustness of decisions.

To address the computational complexity inherent to the Minimax search, we implemented several optimization strategies:

- Alpha-Beta Pruning:
  Significantly reduces search complexity by pruning branches that do not influence the final decision, thus limiting unnecessary exploration.

- Memoization (LRU Cache):
  Caches previously evaluated board states to avoid redundant computations, resulting in substantial runtime improvements.

- Move Ordering:
  Prioritizes moves that appear more promising based on heuristic evaluations, effectively enhancing alpha-beta pruning efficiency.

Minimax decisions are guided by a weighted combination of multiple heuristic functions:

- Monotonicity: Encourages sequences of descending tile values.

- Smoothness: Rewards minimal differences between adjacent tiles.

- Tile Clustering: Rewards positions where higher-value tiles cluster together.

- Max Tile Placement: Favors boards where the highest-value tile occupies a corner or edge.

Component	Description	Impact
Monotonicity	Evaluates how well tiles are ordered in decreasing values from corner (same as in A*)	High
Smoothness	Measures differences between adjacent tiles (same as in A*)	Medium
Clustering	Evaluates how well similar values are clustered together (same as in A*)	Medium
Merge Potential	Rewards configurations with adjacent same-valued tiles	High
Max Tile Position	Prefers max tile in corner > edge > center (same as in A*)	High
Alpha-Beta Pruning	Eliminates branches that cannot influence final decision	Speed: +50%
Memoization	Caches previously computed states using LRU cache	Speed: +16%
Move Ordering	Evaluates most promising moves first to improve pruning	Speed: +15%

Table 2: Minimax Algortihm Optimizations

```
function minimax(grid, depth, alpha, beta, maximizingPlayer):
    if depth == 0 or no moves available:
        return evaluate(grid)

    if maximizingPlayer:
        maxEval = -∞
        for each move in possible_moves(grid):
            new_grid = apply_move(grid, move)
            eval = minimax(new_grid, depth - 1, alpha, beta, False)
            maxEval = max(maxEval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break  # alpha-beta pruning
        return maxEval
    else:
        minEval = +∞
        for each possible tile placement:
            new_grid = place_tile(grid, tile)
            eval = minimax(new_grid, depth - 1, alpha, beta, True)
            minEval = min(minEval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break  # alpha-beta pruning
        return minEval
```

Pseudo Code 1: Minimax Algorithm

The Minimax algorithm integrates sophisticated heuristic evaluations with advanced optimization techniques. It provides robust performance by balancing deep lookahead capability with computational efficiency, effectively handling the inherent uncertainty within 2048 gameplay.

B. Greedy Algorithm

The Greedy algorithm for 2048 selects moves based solely on their immediate heuristic evaluation, without considering possible future states. At each step, it identifies and selects the move that provides the maximum instantaneous heuristic benefit, thus significantly reducing computation time.

The algorithm uses a composite heuristic function combining several critical metrics:

- Score Difference: Measures the immediate increase in game score resulting from a move.

- Empty Cells Bonus: Prioritizes moves that increase or maintain the number of empty cells, thereby enhancing board flexibility for future moves.

- Tile Clustering: Rewards moves resulting in closer proximity between high-valued tiles, maximizing future merge potential.

- Max Tile Placement: Provides extra scoring benefits if the highest-value tile occupies strategic positions, such as corners or edges.

```
function greedy_best_move(grid):
    best_move = null
    max_score = -∞

    for each possible_move in moves ["Up", "Down", "Left", "Right"]:
        temp_grid, score_diff = simulate_move(grid, possible_move)

        if move is valid:
            empty_cells = count_empty_cells(temp_grid)
            clustering = calculate_clustering(temp_grid)
            max_tile_pos = evaluate_max_tile_placement(temp_grid)

            total_score = score_diff +
                          (10 * empty_cells) +
                          (clustering / 100) +
                          (10 * max_tile_pos)

            if total_score > max_score:
                max_score = total_score
                best_move = possible_move

    return best_move
```

Pseudo Code 2: Greedy Algorithm

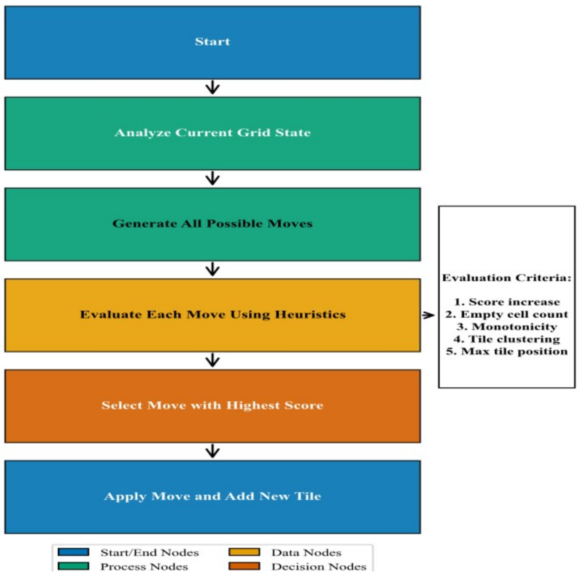


Fig. 2: Greedy Flowchart

C. *A* Algorithm*

The A* algorithm is a widely recognized heuristic-driven search technique traditionally utilized in pathfinding and graph traversal problems. In the context of the 2048 game, we adapt A* to select optimal moves by evaluating each potential game state using a combination of heuristics, thereby systematically searching for the move that leads to the most favorable immediate outcome.

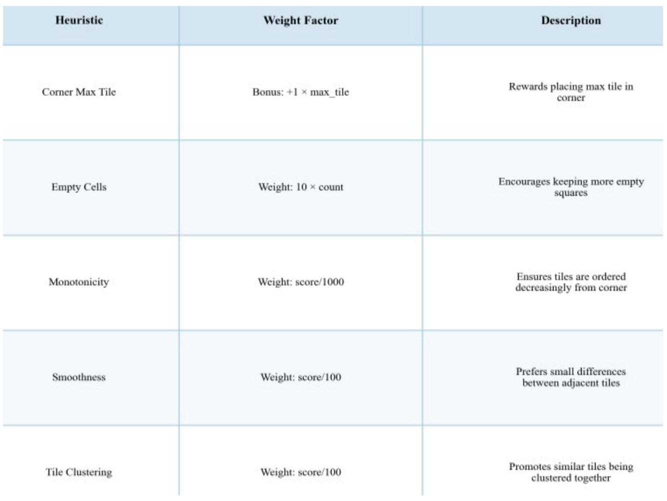
A* algorithm combines two evaluation components:

- Actual cost (g): In this context, this is the immediate gain (score difference) obtained after performing a move.
- Heuristic estimate (h): Estimated desirability of resulting board state, calculated through combined heuristic evaluations.

The algorithm systematically computes a weighted sum of these values for each possible move and selects the one yielding the maximum combined evaluation.

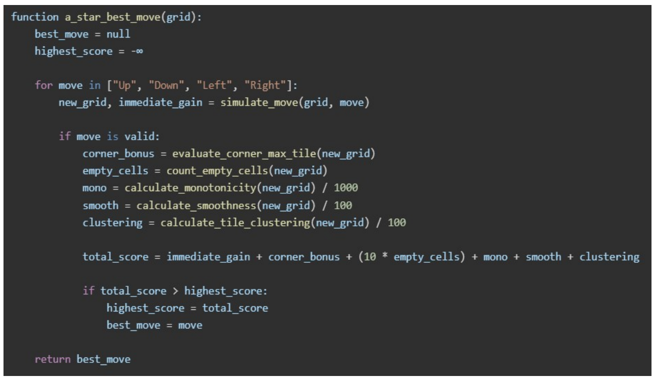
The A* algorithm employs multiple sophisticated heuristic functions that comprehensively assess the desirability of board states:

- Corner Max Tile:
  Gives significant positive weighting if the highest-value tile is placed in a corner.
- Monotonicity:
  Evaluates sequences that consistently increase or decrease across rows and columns, facilitating easier merges.
- Smoothness:
  Rewards configurations where adjacent tiles have minimal value differences, promoting frequent merges and stable configurations.
- Tile Clustering:
  Scores arrangements with higher-value tiles grouped closely together, increasing future merge potential.
- Empty Tile Count:
  Favors configurations with a high number of empty cells, enhancing flexibility for subsequent moves.



Heuristic	Weight Factor	Description
Corner Max Tile	Bonus: +1 × max_tile	Rewards placing max tile in corner
Empty Cells	Weight: 10 × count	Encourages keeping more empty squares
Monotonicity	Weight: score/1000	Ensures tiles are ordered decreasingly from corner
Smoothness	Weight: score/100	Prefers small differences between adjacent tiles
Tile Clustering	Weight: score/100	Promotes similar tiles being clustered together

Table 3: A* Algorithm Heuristics



```
function a_star_best_move(grid):
    best_move = null
    highest_score = -∞

    for move in ["Up", "Down", "Left", "Right"]:
        new_grid, immediate_gain = simulate_move(grid, move)

        if move is valid:
            corner_bonus = evaluate_corner_max_tile(new_grid)
            empty_cells = count_empty_cells(new_grid)
            mono = calculate_monotonicity(new_grid) / 1000
            smooth = calculate_smoothness(new_grid) / 100
            clustering = calculate_tile_clustering(new_grid) / 100

            total_score = immediate_gain + corner_bonus + (10 * empty_cells) + mono + smooth + clustering

            if total_score > highest_score:
                highest_score = total_score
                best_move = move

    return best_move
```

Pseudo Code 3: A* Algortihm

The adapted A* algorithm strategically combines immediate gains with sophisticated heuristic assessments, balancing immediate and future board considerations effectively. Though more computationally demanding than Greedy methods, it achieves significantly improved gameplay performance by integrating broader strategic evaluations.

D. *Expectimax Algorithm*

Expectimax is an extension of the traditional Minimax algorithm tailored specifically for environments involving uncertainty and probabilistic outcomes. In the context of the 2048 game, Expectimax evaluates moves based on expected values rather than worst-case scenarios, accurately modeling the inherent randomness associated with tile placements.

Expectimax alternates between two decision types in its tree structure:

- Max nodes: Represent the player's optimal move selections aiming to maximize expected score.
- Chance nodes: Represent the probabilistic placement of new tiles (typically 2 or 4) by the game, calculated by taking the weighted average of all possible outcomes based on their likelihood.

Expectimax calculates expected outcomes, allowing it to adapt more effectively to the game's inherent randomness compared to deterministic Minimax.
To maintain computational feasibility, Expectimax employs the following optimization strategies:

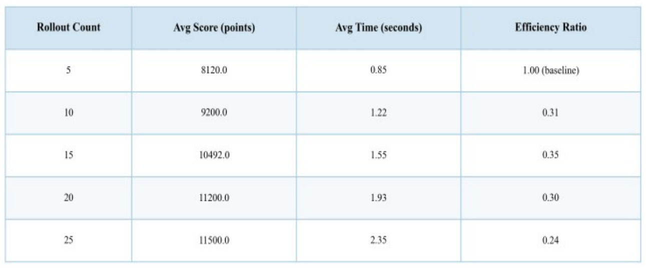
- Monte Carlo Rollouts:
  Utilizes random simulations to approximate expected outcomes efficiently, significantly reducing computation time while maintaining high-quality decision-making.
- Weighted Tile Placement:
  Assigns greater importance to tile placements near corners, optimizing sampling quality during Monte Carlo simulations.
- Memoization (LRU Cache):
  Implements caching to store previously evaluated board positions, eliminating redundant computations and enhancing runtime efficiency.

Expectimax utilizes an extensive set of heuristics:

- Empty Cell Count and Ratio:
  Strongly favors board states with more empty cells for higher flexibility.
- Monotonicity and Smoothness:
  Similar to previous algorithms, encourages uniform and

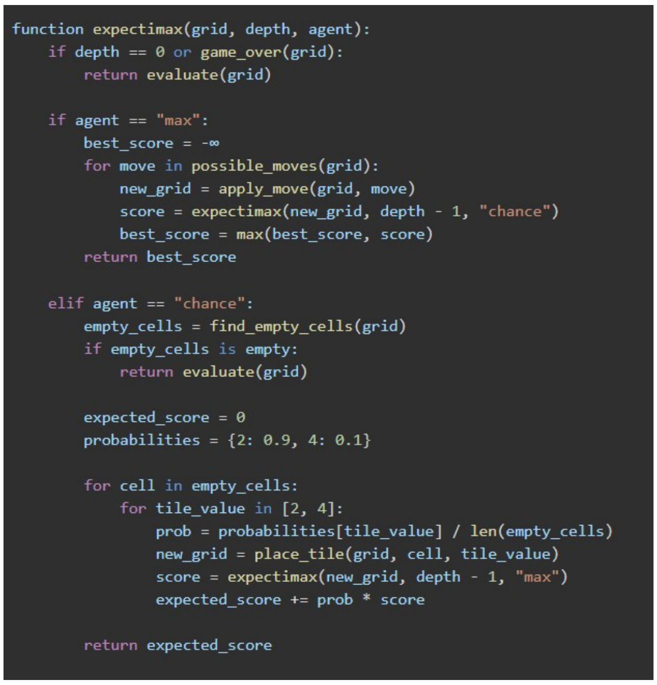
smooth transitions between adjacent tiles to facilitate merging.

- Tile Clustering:
  Evaluates how effectively high-valued tiles are grouped.
- Merge Potential:
  Rewards board states exhibiting immediate tile-merging opportunities.
- Corner Tile Bonus:
  Rewards highest-value tiles positioned strategically in corners, promoting stability.



Rollout Count	Avg Score (points)	Avg Time (seconds)	Efficiency Ratio
5	8120.0	0.85	1.00 (baseline)
10	9200.0	1.22	0.31
15	10492.0	1.55	0.35
20	11200.0	1.93	0.30
25	11500.0	2.35	0.24

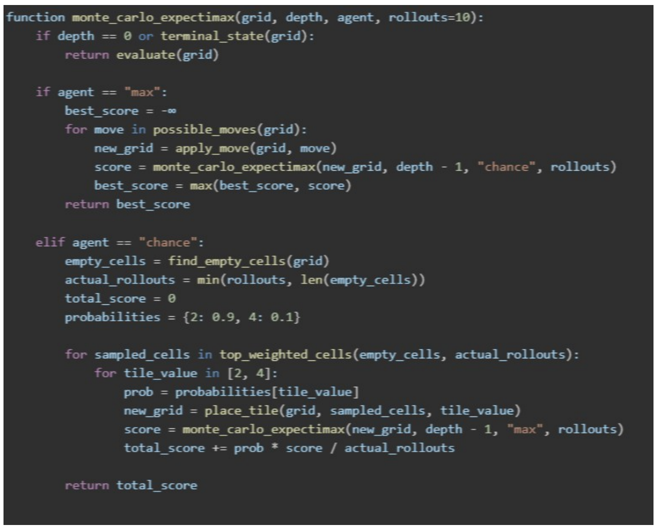
Table 3: Monte Carlo Rollout Performance on Expectimax



```
function expectimax(grid, depth, agent):
    if depth == 0 or game_over(grid):
        return evaluate(grid)

    if agent == "max":
        best_score = -∞
        for move in possible_moves(grid):
            new_grid = apply_move(grid, move)
            score = expectimax(new_grid, depth - 1, "chance")
            best_score = max(best_score, score)
        return best_score

    elif agent == "chance":
        empty_cells = find_empty_cells(grid)
        if empty_cells is empty:
            return evaluate(grid)

        expected_score = 0
        probabilities = {2: 0.9, 4: 0.1}

        for cell in empty_cells:
            for tile_value in [2, 4]:
                prob = probabilities[tile_value] / len(empty_cells)
                new_grid = place_tile(grid, cell, tile_value)
                score = expectimax(new_grid, depth - 1, "max")
                expected_score += prob * score

        return expected_score
```

Pseudo Code 3: Expectimax Algorithm

Expectimax is well-suited to the probabilistic nature of 2048, delivering superior average performance compared to deterministic approaches. However, its computational demands are considerable, making its optimization strategies (Monte Carlo rollouts and memoization) essential to ensure practical applicability.



```
function monte_carlo_expectimax(grid, depth, agent, rollouts=10):
    if depth == 0 or terminal_state(grid):
        return evaluate(grid)

    if agent == "max":
        best_score = -∞
        for move in possible_moves(grid):
            new_grid = apply_move(grid, move)
            score = monte_carlo_expectimax(new_grid, depth - 1, "chance", rollouts)
            best_score = max(best_score, score)
        return best_score

    elif agent == "chance":
        empty_cells = find_empty_cells(grid)
        actual_rollouts = min(rollouts, len(empty_cells))
        total_score = 0
        probabilities = {2: 0.9, 4: 0.1}

        for sampled_cells in top_weighted_cells(empty_cells, actual_rollouts):
            for tile_value in [2, 4]:
                prob = probabilities[tile_value]
                new_grid = place_tile(grid, sampled_cells, tile_value)
                score = monte_carlo_expectimax(new_grid, depth - 1, "max", rollouts)
                total_score += prob * score / actual_rollouts

        return total_score
```

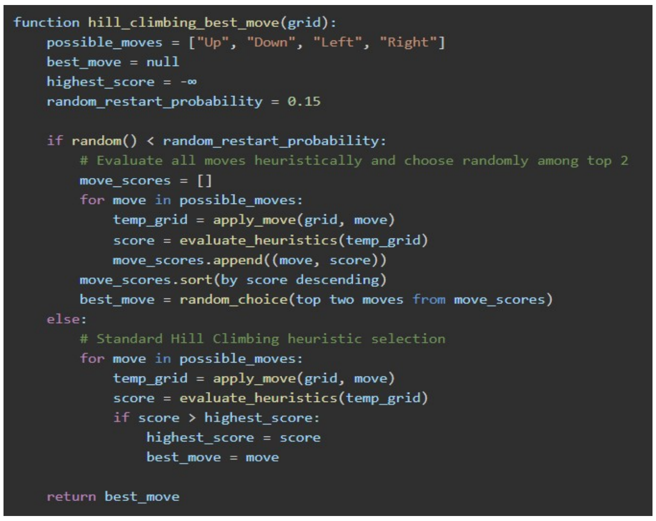
Pseudo Code 4: Monte Carlo Rollout on Expectimax

E. Hill Climbing Algortihm

Hill Climbing is a straightforward, heuristic-driven local search algorithm primarily utilized for optimization tasks. In the context of the 2048 game, Hill Climbing systematically improves its current position by iteratively moving to neighboring states with higher heuristic evaluations. Due to its nature, it can quickly become trapped in local maxima; therefore, additional strategies such as random restarts are applied to mitigate this issue.

The algorithm begins at an initial board state and repeatedly selects the move with the highest immediate heuristic improvement. Unlike deeper search algorithms, Hill Climbing does not perform exhaustive or predictive searches. To avoid becoming trapped in suboptimal positions, a randomized restart mechanism is employed, periodically exploring less immediately promising moves to increase global search space coverage.

- Random Restart Strategy (15% probability):
  Introduces stochastic decision-making to reduce local optimum stagnation. Occasionally selects the next move randomly from the top two available options based on heuristic evaluations.
- Enhanced Heuristic Weights:
  Adjusted heuristic scoring provides balanced decision-making, including increased weights on empty cell count, monotonicity, smoothness, and tile clustering.
- Empty Cell Count:
  Prioritizes states maintaining a larger number of empty tiles.
- Monotonicity and Smoothness:
  Encourages configurations that facilitate efficient merging, similar to previously described heuristics.
- Merge Potential:
  Rewards boards exhibiting immediate tile merging opportunities.
- Max Tile Placement:
  Scores highest when the top tile occupies strategic positions such as corners.



```
function hill_climbing_best_move(grid):
    possible_moves = ["Up", "Down", "Left", "Right"]
    best_move = null
    highest_score = -∞
    random_restart_probability = 0.15

    if random() < random_restart_probability:
        # Evaluate all moves heuristically and choose randomly among top 2
        move_scores = []
        for move in possible_moves:
            temp_grid = apply_move(grid, move)
            score = evaluate_heuristics(temp_grid)
            move_scores.append((move, score))
        move_scores.sort(by score descending)
        best_move = random_choice(top two moves from move_scores)
    else:
        # Standard Hill Climbing heuristic selection
        for move in possible_moves:
            temp_grid = apply_move(grid, move)
            score = evaluate_heuristics(temp_grid)
            if score > highest_score:
                highest_score = score
                best_move = move

    return best_move
```

Pseudo Code 4: Hill Climbing

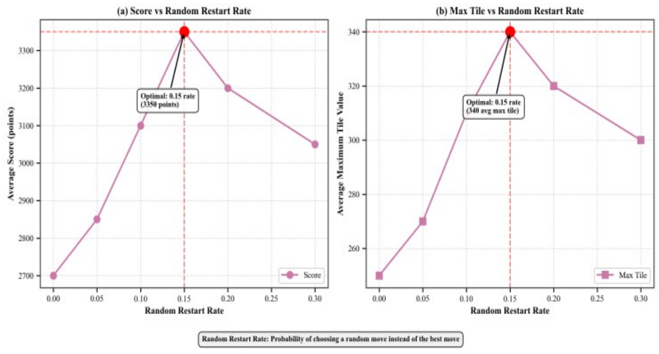


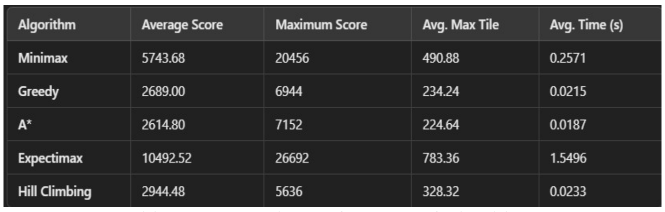
Fig. 3: Effect of Random Restart on Hill Climbing

Hill Climbing offers an effective yet computationally lightweight approach for automating 2048 gameplay. Despite its simplicity, carefully designed heuristics and randomized restarts significantly mitigate its main limitation—the tendency to settle into local optima. This strategy provides a practical balance between computational efficiency and gameplay performance, suitable for real-time applications where responsiveness is critical.

IV. Performance Evaluation and Results

To rigorously assess and compare the effectiveness of the proposed algorithms (Minimax, Greedy, A*, Expectimax, and Hill Climbing), comprehensive performance tests were conducted. Each algorithm underwent extensive testing under identical conditions, running 100 simulations per algorithm. The performance was measured using four critical metrics:

- Average Score: Represents the algorithm's typical gameplay performance.
- Maximum Score: Reflects the best achievable scenario within tests.
- Average Maximum Tile: Indicates how effectively each algorithm generates higher-value tiles.
- Average Computation Time: Measures the computational efficiency of each algorithm in seconds.



Algorithm	Average Score	Maximum Score	Avg. Max Tile	Avg. Time (s)
Minimax	5743.68	20456	490.88	0.2571
Greedy	2689.00	6944	234.24	0.0215
A*	2614.80	7152	224.64	0.0187
Expectimax	10492.52	26692	783.36	1.5496
Hill Climbing	2944.48	5636	328.32	0.0233

Table 3: Comparative Performance of Algorithms

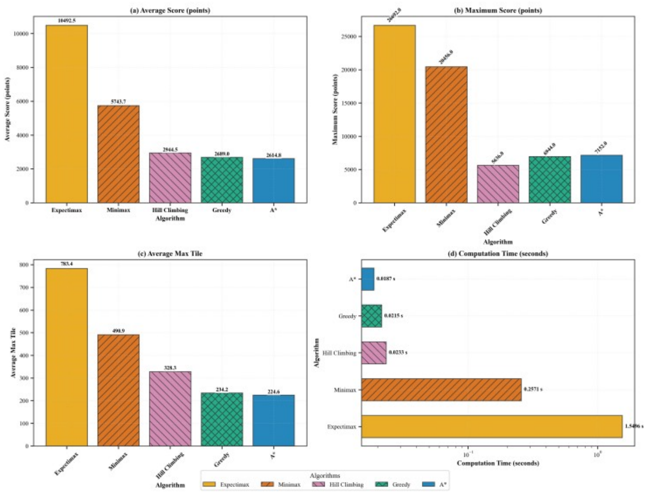


Fig. 4: Algorithm Score Comparison

Expectimax achieved the highest overall performance, scoring significantly higher average (10492.52) and maximum scores (26692). Demonstrates superior capability in tile management, generating the highest average max tile (783.36). Notably, Expectimax incurs the highest computational cost (1.5496 s per decision) due to its probabilistic and computationally demanding nature.

Minimax exhibits a balanced performance with a respectable average score (5743.68) and notably high maximum score (20456). Delivers a solid trade-off between efficiency and performance (average computational time: 0.2571 s), due largely to Alpha-Beta pruning and memoization optimizations.

Greedy Algorithm achieves moderate scores (average: 2689.00, max: 6944) with a relatively low average max tile (234.24). Exhibits highly efficient computational speed (0.0215 s), making it suitable for scenarios demanding fast, real-time decisions at the cost of overall strategic depth.

A Algorithm* produces comparable results to Greedy in terms of average score (2614.80) and max tile (224.64), albeit slightly lower. Notably efficient computation time (0.0187 s), reflecting its strength in balancing immediate heuristic evaluations without extensive lookahead.

Hill Climbing performs slightly better than Greedy in average score (2944.48) and average max tile (328.32). Quick computation times (0.0233 s), demonstrating improved performance due to randomized restart strategies, balancing exploration with efficiency.

The performance analysis clearly demonstrates a strategic and computational trade-off among algorithms:

- Expectimax is ideal for applications where achieving the highest possible scores outweighs computational concerns.
- Minimax offers a well-rounded choice when balancing computational constraints with high-level gameplay is essential.
- Greedy and A* are favorable for low-latency scenarios requiring immediate responses, though sacrificing longer-term strategic depth.
- Hill Climbing, enhanced with random restart mechanisms, provides a viable alternative for scenarios seeking moderate performance improvements over purely greedy approaches, without significant computational overhead.

The heuristic functions and optimization strategies play a pivotal role in determining these outcomes, clearly influencing each algorithm's specific strengths and weaknesses.

V. Discussion

The performance evaluation reveals significant insights into the capabilities, limitations, and inherent trade-offs of each algorithm when applied to the 2048 puzzle. Key observations and underlying reasons for their performance differences are thoroughly discussed below:

A. Expectimax: Strategic Excellence with Computational Cost

Expectimax consistently outperformed other algorithms by a substantial margin in terms of both average and maximum achieved scores. Its remarkable performance can primarily be attributed to its probabilistic consideration of future states through Monte Carlo simulations. This approach allows Expectimax to anticipate potential outcomes, optimizing moves more effectively under uncertainty. However, these benefits come at a considerable computational cost. The intensive calculations required for probabilistic expectation estimations, Monte Carlo rollouts, and recursive evaluations result in the highest computation time among the tested algorithms (average 1.5496 seconds per decision).

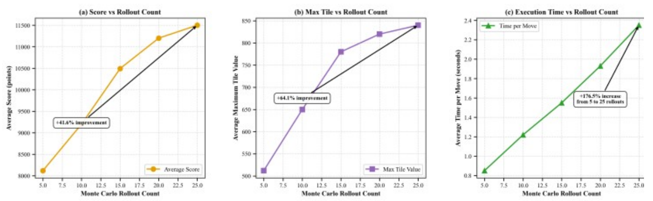


Fig. 5: Monte Carlo Rollout Effectiveness on Expectimax

B. Minimax: Effective Balance between Performance and Efficiency

Minimax demonstrates commendable performance, showcasing strong strategic decision-making capabilities while maintaining relatively low computational overhead. Alpha-beta pruning effectively reduces unnecessary state evaluations, significantly increasing computational efficiency compared to Expectimax. Furthermore, memoization (LRU caching) enhances speed by avoiding redundant calculations. Its heuristic evaluations—such as monotonicity, smoothness, and clustering—contribute significantly to its higher maximum score capability (20456). Minimax stands out as a robust and balanced solution, ideal for environments requiring consistent strategic decisions without prohibitively high computational costs.

C. Greedy Algorithm: Speed at the Expense of Strategic Depth

The Greedy algorithm offers exceptional speed due to its simple, immediate heuristic evaluation strategy, evident from its computation time of only 0.0215 seconds. However, the trade-off is clear: by disregarding future game states entirely, Greedy achieves significantly lower average (2689.00) and maximum scores (6944) than Expectimax and Minimax. While suitable for real-time decision-making scenarios where immediate response is prioritized, its limited foresight hampers overall effectiveness in long-term strategic planning.

D. A Algorithm: Immediate Efficiency with Limited Depth*

The A* algorithm closely mirrors the computational simplicity of Greedy, achieving rapid decision-making (average time 0.0187 s) but with marginally lower average performance (2614.80). Despite incorporating multiple heuristic evaluations—including monotonicity, smoothness, clustering, and corner tile placement—its decisions are confined to immediate heuristic benefits rather than strategic foresight. This immediate focus significantly restricts the algorithm's ability to achieve higher long-term scores. Nevertheless, A* remains viable for low-latency applications where slight performance compromises are acceptable.

E. A Hill Climbing Algorithm: Improved Local Search through Randomization

Hill Climbing, augmented by a randomized restart strategy, improves upon the immediate heuristic selection approach of Greedy and A*. While maintaining low computational demands (0.0233 s), its random restart mechanism partially mitigates the tendency to stagnate in local optima. This technique facilitates exploration of alternative promising states, modestly enhancing its overall average (2944.48) and maximum (5636) scores. Nevertheless, without deeper lookahead capabilities or probabilistic evaluations, Hill Climbing still underperforms compared to Minimax and Expectimax in strategic depth.

- The success of each algorithm relies heavily on the careful selection and weighting of heuristic functions and optimization techniques:
- Monotonicity and Smoothness substantially improved long-term gameplay, particularly evident in Minimax and Expectimax results.
- Tile Clustering and Merge Potential proved crucial in enabling higher-value merges and significantly higher scores.
- Monte Carlo Simulations and Memoization substantially boosted Expectimax's accuracy at the expense of speed.
- Alpha-Beta Pruning and Move Ordering greatly optimized Minimax's computational efficiency without compromising decision quality.
- Random Restarts effectively enhanced the Hill Climbing algorithm's robustness by encouraging exploration.

Considering the observed trade-offs:

- Expectimax is ideal for scenarios prioritizing maximum scoring capability without significant computational constraints.
- Minimax offers a superior balance for typical use cases, combining high scores with moderate computational requirements.
- Greedy and A* are preferred in real-time or embedded applications requiring extremely low latency, with clear recognition of their limited strategic foresight.
- Hill Climbing serves as an effective intermediate solution, providing modestly improved results over Greedy and A* due to random restarts, suitable for applications demanding minimal computation with moderate scoring performance.

VI. Future Work.

While the implemented algorithms and optimization techniques have demonstrated significant improvement in automated 2048 gameplay, several areas remain ripe for future research to further enhance algorithmic performance and efficiency. Below, we outline promising research directions:

- Adaptive Heuristic Approaches: Implementing methods such as reinforcement learning or statistical learning techniques to dynamically adjust heuristic weights based on game states, thus enhancing decision-making flexibility.
- Parallel Computing Techniques: Integrating parallel or distributed computing methods could improve computational efficiency, especially for computationally intensive algorithms like Expectimax and Minimax.
- Machine Learning Integration: Leveraging artificial neural networks, genetic algorithms, or fuzzy logic techniques to create hybrid intelligent agents capable of higher strategic foresight and real-time adaptability.
- Advanced Search Techniques: Applying metaheuristic search methods (e.g., simulated annealing, tabu search) for improved handling of local optima, particularly enhancing algorithms like Hill Climbing.

The proposed future research directions provide a roadmap for systematically advancing the capabilities of automated gameplay algorithms for the 2048 puzzle. Pursuing these directions promises substantial improvements in algorithmic accuracy, strategic depth, and computational efficiency.

VII. Conclusion

In this study, we developed and rigorously evaluated multiple artificial intelligence algorithms—Minimax, Greedy, A*, Expectimax, and Hill Climbing—for automating gameplay in the popular puzzle game 2048. Each algorithm was carefully enhanced with heuristic evaluation functions, including monotonicity, smoothness, clustering, merge potential, and strategic tile placement, alongside optimization methods such as Monte Carlo simulations, memoization, alpha-beta pruning, randomized restarts, and move ordering.

Performance analysis across 100 test simulations revealed distinct trade-offs among these algorithms. Expectimax achieved the highest performance, clearly excelling in terms of average and maximum scores through probabilistic modeling but at the cost of significantly higher computational complexity. Minimax demonstrated a robust and balanced approach, achieving competitive performance due to efficient optimizations, making it particularly suitable for moderate resource scenarios.

Conversely, Greedy and A* algorithms provided computationally efficient yet strategically limited gameplay, ideal for real-time or computationally constrained environments. Hill Climbing, augmented by randomized restarts, presented a practical compromise, achieving modest performance improvements with minimal additional computational costs.

This work highlights the critical role that advanced heuristics and targeted optimizations play in enhancing algorithmic effectiveness in complex decision-making environments. Future research avenues, such as parallelization, adaptive heuristic tuning, and integration of advanced machine learning techniques, offer promising pathways for further improvements, ultimately advancing the capabilities of automated intelligent systems in puzzle-solving applications.

References

[1] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Upper Saddle River, NJ, USA: Pearson, 2020.

[2] B. Coppin, Artificial Intelligence Illuminated, Sudbury, MA, USA: Jones and Bartlett Publishers, 2004.

[3] M. Negnevitsky, Artificial Intelligence: A Guide to Intelligent Systems, 2nd ed. Harlow, England: Addison-Wesley, 2002.

[4] V. Nabiyev, Yapay Zeka, Seçkin Yayıncılık, 2003.

[5] MIT OpenCourseWare, "6.034 Artificial Intelligence," Massachusetts Institute of Technology. [Online]. Available: http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-034Spring-2005/CourseHome/index.htm. [Accessed: Apr. 1, 2025].

[6] TOBB ETÜ, "YAP 441/BİL 541 Yapay Zeka Dersi 2025 Bahar Uygulama Planı," [Online]. Available: https://piazza.com/etu.edu.tr/winter2025/yap441bil541. [Accessed: Apr. 1, 2025].

[7] G. Cirulli, "2048 Game," [Online]. Available: https://play2048.co. [Accessed: Apr. 1, 2025].

[8] Y. Akay, "Artificial Intelligence Algorithms for 2048: Intermediate Report," Unpublished manuscript, TOBB University of Economics and Technology, Mar. 2025.

[9] M. Szubert and W. Jaśkowski, "On the Application of Reinforcement Learning to 2048," in Proc. of the IEEE CEC, 2014, pp. 1–8.

[10] K. Wu and D. Thaker, "2048 AI using Expectimax," Stanford University CS221 Project, 2014. [Online]. Available: https://cs.stanford.edu/people/karpathy/2048/

[11] F. S. Melo, "Monte Carlo Methods in Artificial Intelligence," Instituto Superior Técnico, Lisbon, Lecture Notes, 2016.

[12] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in Proc. of the IEEE Int. Conf. on Neural Networks, Perth, WA, Australia, 1995, pp. 1942–1948.

[13] A. Juliani et al., "Unity: A General Platform for Intelligent Agents," arXiv preprint arXiv:1809.02627, 2018.