Match the tiles

Master in Data Science and Engineering
Artificial Intelligence

Helena Costa
Nº 202302812

Lara Sá Neves
Nº 2020195946

Abstract **This project applies artificial intelligence techniques to develop a Solitaire Puzzle Solver project for efficiently solving the Match the Tiles'game. We compare uninformed and informed search methods, heuristics, and metaheuristics. Results highlight optimal and suboptimal approaches across different metrics (time, memory, and number of moves). Additionally, two additional levels with larger grid sizes are introduced to provide deeper insights into the method's performance.**

Index Terms: Artificial Intelligence, Heuristics, Metaheuristics, Search Methods, Solitaire Games, Time, Memory, Comparison

I. Introduction

Artificial intelligence is a field of computer science focused on creating systems capable of autonomously learning, reasoning, and making decisions. In this work, we are going to apply AI in the Solitaire Puzzle Solver project which involves creating and implementing an interface that can solve the game "Match the Tiles" efficiently. It is a single-player game played on a board with pieces. The goal is to achieve a goal piece following specific rules of movement.

II. Objectives

The primary objectives of the project include:

1) Developing an intuitive and user-friendly interface for interacting with the game
2) Implementing various search algorithms and heuristic functions to find optimal or near-optimal solutions
3) Allowing users to select different levels of the puzzle and search methods for experimentation and analysis
4) Provide insights into the strengths and weaknesses of various search methods and heuristic functions
5) Evaluating and comparing the performance of different search methods and metaheuristics based on computational resources utilized and execution time

III. Problem formulation

State Representation: In this game, the state of the game (S) is represented by several components: the position of the player's piece, which is an array of size N representing the movements chosen; the position of the goal's piece and the positions of any obstacles on the game board.

Initial State: The initial state of the game is determined by the specific level selected by the player. It defines: the starting positions of the player's piece, the goal's piece, and obstacles.

Objective State: The objective state is achieved when the player successfully navigates their piece to reach the position of the goal piece. At this point, the game is considered solved.

Operators: Operators are rules that govern how the player can transition from one state to another, and they define the allowable movements that the player can take (moving the player's piece up, down, left, or right), subject to any constraints imposed by obstacles and boundaries on the game board.
Since the main goal of the game is to reach a solution using the minimum number of moves possible, all the operators have the same cost associated.

IV. Implementation

To initiate our game development process, we began by defining a basic framework within which a human player could engage, including a comprehensive in-game menu providing clear instructions. We defined the game's rules, including piece movement, win conditions, and a user-friendly ending screen with options to replay, exit, and choose a new difficulty level.

Fig.1 - Initial screen

Next, we created a random AI opponent as a baseline for comparison. To add variety and engage players, we implemented multiple levels with different grid sizes and obstacle configurations.

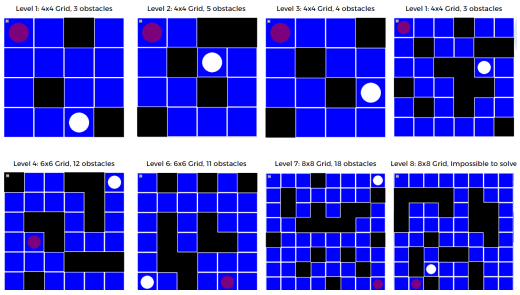

Fig.2 - Levels and corresponding grids

We then focused on improving the game's aesthetics, making it visually appealing and enjoyable. Following this, we implemented various search algorithms (described later), including a simplistic Breadth-First Search (BFS) option as a hint during gameplay to showcase problem-solving approaches if players need help. Interestingly, we also incorporated an "unsolvable" level to observe how the algorithms would react in such scenarios, under challenging conditions, enriching our understanding of their capabilities and limitations and which we aim to further explore in the future.

A. Code specifications

All the code developed in the scope of this project was implemented in Python using some of its available libraries, namely:
Pygame: It handles tasks such as window creation, event handling, and rendering graphics for the execution of the grid of the game.
Pygame_menu: This library is an extension of Pygame and it's used here to create the menu interface for selecting game options like levels and search methods, with intuitive interactions.

The main components and functions within the code are the following:

Menu Functions:
draw_level_selection_menu(): This function displays a menu for selecting the game level and search method (if applicable). It allows users to choose from various levels of difficulty and different search algorithms for the computer solver.
show_game_rules(): Displays the game rules, providing players with instructions on how to play and navigate through the levels.

Game Logic Functions:
draw_grid(): Draws the grid on the Pygame window, creating the visual layout for the game.
draw_pieces(): Draws the player piece, goal piece, and obstacles on the grid.
generate_random_positions(): Generates random positions for the player, goal, and obstacles based on the selected game level.
computer_random_move(): Implements a random move for the computer player.
computer_bfs_move(), computer_dfs_move(), computer_greedy_move(), computer_astar_move(), computer_weighted_astar_move(): Implement different search algorithms for the computer player, such as Breadth-First Search, Depth-First Search, Greedy Search, A* Search, and Weighted A* Search, and several others.
main(): Orchestrates the game flow, handling player inputs, executing game logic, updating the display, and managing the end of the game.

Main Pygame Loop:
run_game(): Initiates the main Pygame loop, which runs the game until the user exits. It handles the

flow of the game, including menu display, game execution, and termination.

V. Methodology

In order to solve the Sliding Puzzles, we implemented both uninformed and informed AI search methods, different heuristics, and also two metaheuristics.
These methods were compared on three different levels:
• quality: the quality of each method was measured by the number of steps taken to reach the puzzle solution. A method is considered to be optimal if it allows solving every puzzle in the minimum number of steps possible;
• time: the higher the computational time, the more expensive the method;
• memory: the higher the memory consumption, the more expensive the method.

For each of the implemented methods, the already-visited states were saved and avoided, to reduce the algorithm's computational cost.

a. Uninformed Search Methods:

Uninformed search methods offer a general approach to problem-solving by exploring all possible solutions systematically. Unlike informed methods, they don't rely on any specific knowledge about the current state. These methods include:

Breadth-First Search (BFS): Explores all nodes at the same level (depth) before moving to the next level. It's guaranteed to find the shortest path if one exists so it can be considered to be optimal (that's the reason why it was chosen to be the one providing hints). BFS is expected to be memory intensive due to exploring all possibilities at each level.

Depth-First Search (DFS): Against BFS, it explores one path as deeply as possible until it reaches a goal or hits a dead end. Since it is usually faster than BFS in many cases but it might get stuck in loops or explore irrelevant branches, we are interested in observing how it behaves in complex situations.

Depth-Limited Search (DLS): Similar to DFS but with a predefined depth limit (e.g., 5 or 8 in our case), it prevents getting stuck in infinite loops even though it might miss the optimal solution if the limit is too low. It is useful for exploring different search depths and understanding their impact on finding the goal.

Iterative Deepening Search (IDS): Combines the advantages of DFS and BFS, and performs multiple DFS searches, gradually increasing the depth limit in each iteration. IDS guarantees to find the shortest path eventually, similar to BFS, but with less memory usage.
Can be used to compare efficiency against BFS and analyze behavior with different depth limitations.

Uniform Cost Search (UCS): Prioritizes exploring nodes with the lowest cost so far, regardless of their depth, so it's guaranteed to find the path with the minimum total cost.
Can be slower than BFS in some situations, especially if many nodes have the same cost.

b. Informed Search Methods

i. Heuristics

A* with **Manhattan Distance**: Uses the Manhattan distance between the current position and the goal as the heuristic, a common choice for grid-based problems.
A* with **Max Distance**: Likely uses the maximum distance achievable in a single move as the heuristic, potentially prioritizing moves with greater immediate progress.

ii. Informed Search Methods

Greedy Search: Leverages the heuristic function to estimate the remaining cost to the goal from each state, and always explores the node with the lowest estimated cost first.
May find suboptimal solutions if the heuristic is inaccurate or doesn't consider all relevant information. We expect this search to be quick.

A* algorithm: Similar to Greedy Search but combines the estimated cost with the actual cost incurred so far, by using both the heuristic to guide exploration and the actual cost to avoid getting misled by inaccurate estimates. Generally considered the most efficient informed search

method when the heuristic is admissible (always underestimates the true cost).

Weighted A* algorithm: Extends A* by allowing a weight (we applied weight equal to four) to be applied to the heuristic function and balancing the influence of the estimated cost and the actual cost.

c. Metaheuristics

Hill Climbing: Iteratively explores the search space, moving towards "better" states based on a defined evaluation function. "Better" is typically defined as states with higher values in the evaluation function, which should guide the search towards the optimal solution.
Can get trapped in local optima: non-optimal solutions that appear superior to surrounding states.

Simulated Annealing: Inspired by the physical process of cooling metal, it starts with a high "temperature" that allows exploring both "better" and "worse" states (like exploring a larger search space). Gradually reduces the temperature over time, favoring exploration of "better" states with increasing probability (like focusing on the optimal solution). This approach helps escape local optima and potentially find the global optimum - the absolute best solution.

VI. Results Analysis and Discussion

In this section we proceed to analyze the methods and the levels implemented. First, we compare the number of moves, memory required (in seconds), and time needed for each method to solve level 1 to level 10.
After that, we decided to further explore how our model would behave and if the conclusions would be kept the same if the size of the grid changed drastically.
All the values of the tables correspond to mean values after running the corresponding game 10 times. Note that each section has a different color scheme so that it is visually easier to detect each aggregate analysis.

a. From level 1 to level 7 - Search Methods

Here we can see that all the tested methods can find the solution with an optimal number of moves. The only exception is Depth-First, which needs more moves in level 7. This behavior is expected since the level of difficulty is not very high.

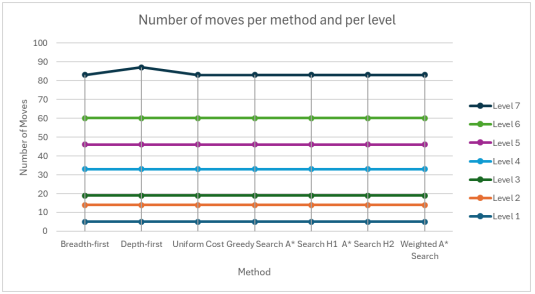

Fig.3 - Number of moves comparison (level 1 to 7)

In terms of memory needed (in bytes), Breadth-First and Iterative deepening tend to need more memory. Random AI is used as the baseline and is indeed the worst (except for the first 2 levels that are easy).
We can see that Depth Limited 5 uses slightly less memory than Depth Limited 8, as expected since it doesn't go so deep. Greedy Search, A*, and Weighted A* are the lightest, and their strategy of exploring the node with the lowest estimated cost first is successful.
Max Distance heuristic for A* search requires slightly more memory than Manhattan Distance and requires the same time, so we will prefer the second one.

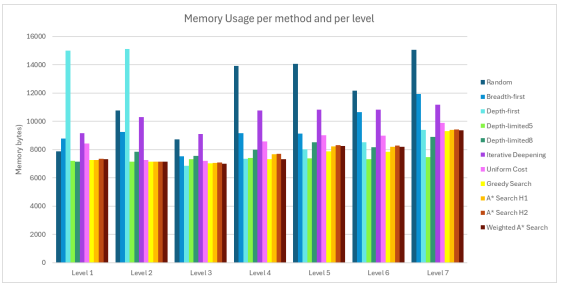

Fig.4 - Memory comparison (levels 1 to 7)

When it comes to the time needed to arrive at the solution, we see that it increases linearly with the difficulty of the game. Breath-first and depth-limited are the best ones and informed search methods have similar performances, with no differences across different heuristics.
To improve further our understanding, more conclusions are derived in the harder following levels.

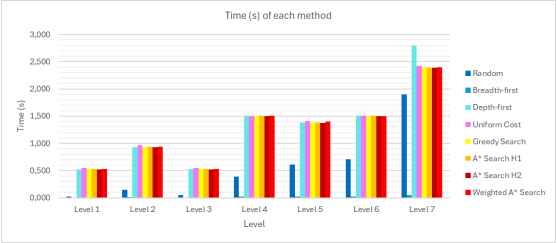

Fig.5 - Time comparison (levels 1 to 7)

In summary, these are the mentioned conclusions from these comparisons:

	Best Method	Worst Method
Moves	All others	Depth First
Memory	Depth Limited 5	Iterative Deepening
Time	Breadth First	Depth First

Table 1 - Global comparison

b. From level 1 to level 7 - Metaheuristics

Analyzing both metaheuristics implemented, Simulated Annealing (SA) and Hill Climbing (HC), we see that SA needs much more time than our previous worst and best methods (table 1). HC also doesn't seem to be a better choice.

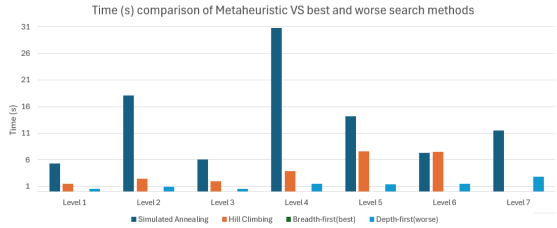

Fig.6 - Metaheuristics's time comparison

When comparing the number of moves needed, neither SA nor HC becomes a better choice. However, HC is way closer to the uninformed search methods, even though it becomes slightly worse as difficulty increases. SA shouldn't be recommended.

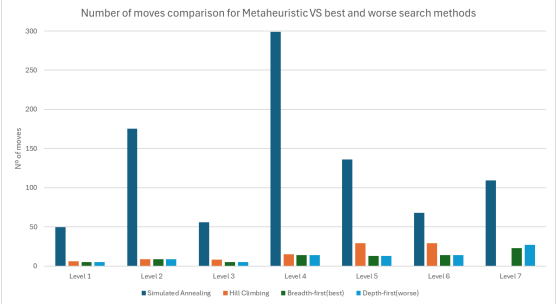

Fig.7 - Metaheuristics's moves comparison

Finally, in terms of memory, both of them become a great surprise, and it turns out they have intermediate performance between the previous best and worst candidates, as difficulty increases, so they can become acceptable solutions in terms of memory needs (only).

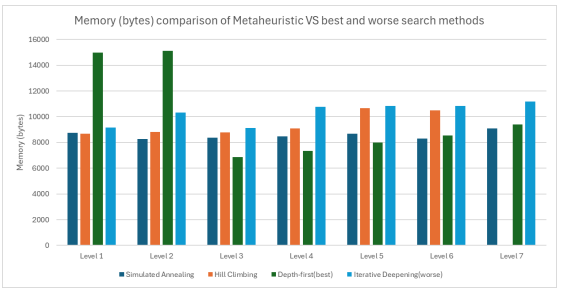

Fig.8 - Metaheuristics's memory comparison

c. Harder levels

We recognize that the current range of levels (1 to 7) may not provide sufficient insight into the progressive impact of various methods over time. This is particularly true as certain anomalies may surface without a clear or immediate explanation. To address this, we have introduced two additional levels (9 and 10) featuring larger grid sizes of 100x100 and 150x150, respectively. We kept the same units so that the comparison between levels is easier.

When it comes to metaheuristics, SA was never able to solve the problem within an hour (when the worst method evaluated was able to do it in 35 minutes), so we claim that SA can't solve these levels of the game. For simplicity, we include the HC metaheuristic in the plots of the informed and uninformed search methods.

Also, random AI was not computed for these levels because it was extremely slow and values couldn't be obtained within acceptable limits.

These are the conclusions:

In terms of time, Breath-First and Depth-First are the worst choices for both level 9 and level 10. Max Distance heuristic performs slightly faster than the Manhattan Distance. Just like detected previously, in terms of time, HC is worse than generally all search methods.

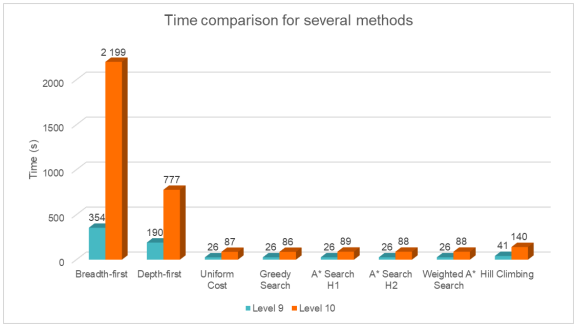

Fig.9 - Time comparison for harder levels

In terms of the number of moves needed, again we see that Depth-First is the exception and needs more steps than the other methods, just like in Fig.3 Hill Climbing needs slightly more as well, as we were expecting. These observations confirm the previous experiments.

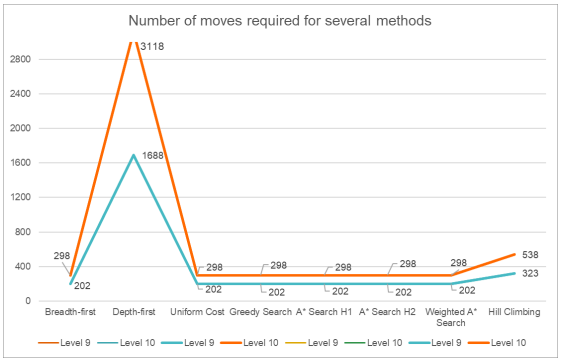

Fig.10 - Comparison of moves for harder levels

When it comes to memory, differences are not so drastic and HC shows its potential for memory limitations. With the increase of the size of the board the computer doesn't seem to me very hurted and the tendencies are the same as in levels 1 to 7. Depth-First becomes a worse choice.

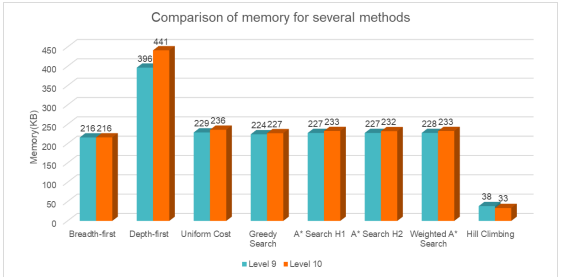

Fig.11 - Memory comparison for harder levels

VII. Conclusions

In conclusion, our project focused on applying artificial intelligence techniques to develop an efficient solver for the Match the Tiles game. We compared various search algorithms, heuristics, and metaheuristics across different levels of difficulty, aiming to understand their performance in terms of time, memory usage, and the number of moves required to solve the game.

In our exploration of uninformed search methods, we found that Breadth-First Search consistently provided optimal solutions across different levels. Depth-First Search, while generally efficient, showed limitations in more complex scenarios, often requiring more moves to reach the solution. Iterative Deepening Search offered a compromise between memory usage and solution optimality, showcasing its versatility in varying depth limitations.

Analyzing informed search methods, A* with Manhattan Distance emerged as a reliable choice, with efficient solutions and acceptable memory requirements. Greedy Search, A*, Weighted A* provided balanced approaches.

When we compared metaheuristics, both Simulated Annealing and Hill Climbing showed mixed results. While Simulated Annealing struggled to find solutions within reasonable time frames for more challenging levels, Hill Climbing demonstrated moderate performance with slight trade-offs in terms of time and solution optimality.

Our analysis extended to harder levels with larger grid sizes, revealing insights into the scalability of different methods. Despite encountering computational challenges, particularly with Simulated Annealing, our findings remained consistent, reinforcing the effectiveness of certain methods while highlighting limitations in others.

In summary, our project contributes with insights into the application of artificial intelligence techniques based on the desired balance between solution optimality, computational resources, and level complexity.

IX. References

[1] L. Reis, “Introduction to Search,” 2023.
[2] L. Reis, “Solving Search Problems,” 2023
[3] Match the Tiles - Sliding Game. Available in https://play.google.com/store/apps/details?id=net.bohush.match.tiles.color.puzzle&hl=pt_PT&gl=US
[4] ChatGPT, https://openai.com/blog/chatgpt, 2024.