PROJECT 1 IA

LUCAS FARIA – 202207540 ALEXANDRE LOPES - 202207015 PEDRO BORGES - 202207552

GAME SPECIFICATION

- **"Wood Block Puzzle"** is a strategic single-player game that tests spatial reasoning and pattern recognition. The goal is to fit different block shapes into a grid, filling lines/columns without leaving gaps.
- **Game Board:** Rectangular grid with adjusting dimensions (e.g., 5x5, 10x10).
- **Blocks:** A set of differently shaped and sized pieces, each formed by smaller square units arranged in various patterns.
- **Objective:** The player must place all blocks on the grid in such a way that every square in a line or column are filled. The game is won when all the pieces are played.
- **Scoring:** The player's score is determined by the time taken to complete the puzzle, lines/columns filled, with n points awarded per block. Bonus points are awarded when several lines/columns are filled at the same time.

SIMILAR GAMES

- **"Wood Block"** available at the Google Play Store.
- The game is divided into level and classic modes.
- The **level mode** consists in a pre-filled board where the goal is to complete it using the pieces provided.
- The **classical mode** consists in an empty board where the goal is to make the greatest number of points until it becomes impossible to make a move because none of the given pieces fit in the board.
- We are mostly following the level mode in our work.
- https://play.google.com/store/apps/details?id=com.block.puzzle.free.wood&hl=en

- **"Wood Blocks"** available at Crazy Games.
- The game only has a classic mode
- The **classical mode** is like the one in "Wood Block".
- The only goal is to score the greatest number of points by completing rows and columns with pieces of different shapes until it becomes impossible to insert a piece into the board.
- https://www.crazygames.com/game/wood-blocks

- **"Wood Block Journey"** available at Crazy Games.
- The game is divided into level and classic modes.
- The **level mode (or journey mode)** consists in a pre-filled board where the goal is to complete it using the pieces provided. It is very similar to the "Wood Block" one.
- The **classical mode is essentially the same as in "Wood Block" and "Wood Blocks"**.
- The biggest difference in the game is that it divides the board into sub-boards. If we complete a sub-board, that sub-board is cleaned, and we receive points. This game mechanic is implemented together with the normal row/column completion to create new ways of scoring points with bonuses multipliers.
- https://www.crazygames.com/game/wood-block-journey

FORMULATION OF THE PROBLEM AS A SEARCH PROBLEM

- **States:**
  - **Board Matrix:** B[N,M] filled with 0 or 1. 0 represents empty square and 1 represents occupied.
  - **Current Selection of pieces:** L[3], represents the pieces that can be played.
  - **Queue with the next pieces:** queue Q with all the other pieces.
- **Initial State:**
  - **B[N,M]** = {0}, empty matrix
  - **L** = [P1, P2, P3], where P1,P2 and P3 are the first 3 pieces.
  - **Q** = {P4, P5, P6, ...}
- **Goal State:**
  - **B[N,M]** = _, it can have pieces
  - L = [], **Q** = {}, all the pieces were played
- **Operators:**
  - **Name:** Move(Piece, Position).
  - **Preconditions:** The board must have space for the piece in the position selected.
  - **Effects:** Piece added to the board, removed from the current selection list and a new piece, popped from the queue, is inserted into the selection list.
  - **Cost:** Moves that bring more lines/columns closer to completion have a higher cost. Moves that complete lines/columns have a very high cost. The player should choose the move with the highest cost.
- **Heuristic functions:**
  - **Closer to complete:** Evaluates the piece/move by the improvement it gives to a line/column completion.
  - **Occupied Space:** Evaluates the occupied space in the board after a move and removal of complete lines/columns.
  - **Number of Points:** Evaluates the points that a move gives.
  - **Number of Pieces to Play:** Evaluates the number of pieces that the player can still play and favor states with less pieces (closer to the goal state).

IMPLEMENTATION

- **Programing Language and Tools:** Python with PyGame and Numpy using Vscode, work shared through GitHub.
- **Game Features:** The game includes menus for configuration and mode selection. There are two main modes: Ai and Human. The AI mode consists in the AI algorithm solving the puzzle and returning the solution and set of stats (time, points, visited states, …). In the Human mode, the player can control the pieces and get hints from the AI. The game also supports reading file configurations and
- **Data Structures:** Python Heap (Min-Heap), Python Set (Hash Table), Python Deque (Queue), Np-Arrays (Array).
- **Algorithms:** BFS, DFS, Iterative Deepening, Uniform Cost, Greedy, A*, A* weighted
- **GitHub Repository:** https://github.com/AlexL534/IA---Wood-Block

ALGORITHMS

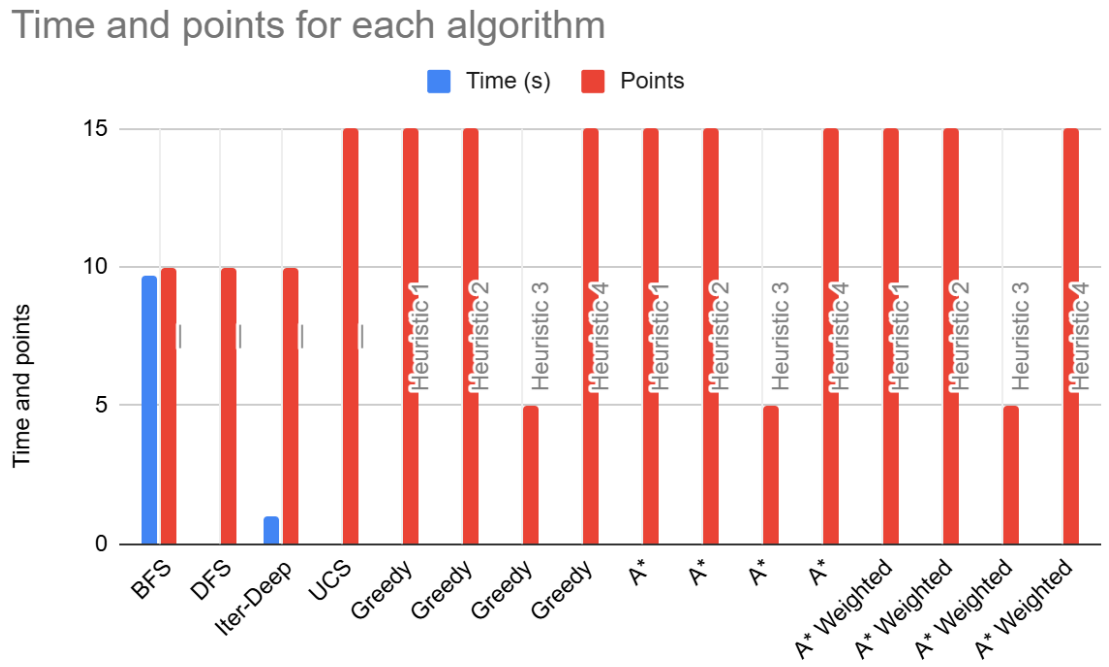
- **Greedy:** In our implementation, we use a **Set** to store the visited states, preventing revisits, and a **Heap** to manage unvisited states. The **Heap** keeps the best state (based on the provided heuristic) at the top, enabling quick retrieval at each iteration.
- **A*:** Our **A*** implementation builds on the *Greedy* algorithm, but the heuristic also accounts for the cost of the path leading to the current state.
- **A* Weighted:** Uses the same structure as the **A*** algorithm but adds a weight to the heuristic to improve the results. We use the size of the game (size of the board and number of pieces) as weight to balance the heuristic accordingly to the game dimensions/complexity.
- **BFS:** Our **BFS** implementation uses a **Set** for visited states, a **queue** for unvisited states, and a **list** to store potential goal states. At the end, the best state (the one with the highest number of points) is chosen from the list.
- **DFS:** In our **DFS** implementation, we use a **Stack** to explore the game states deeply, adding child states to the stack as it progresses. A **Set** tracks visited states to avoid revisits. The algorithm backtracks when it reaches a dead-end, continuing the search until a goal state is found.
- **Uniform Cost Search:** In our **UCS** implementation, we use a **Set** to track visited states and a **Heap** (priority queue) to manage child states. Since our goal is to maximize points rather than minimize cost, we store states in the Heap with their cost as the negative of their points. This ensures that the state with the highest score is prioritized, effectively transforming **UCS** into a best-first search that favors maximizing rewards.
- **Iterative Deepening:** Related to **DFS** algorithm. Explores the possibilities using a **Depth Limited Search** (DFS with limited depth) starting with a small depth limit and incrementing it at each iteration.

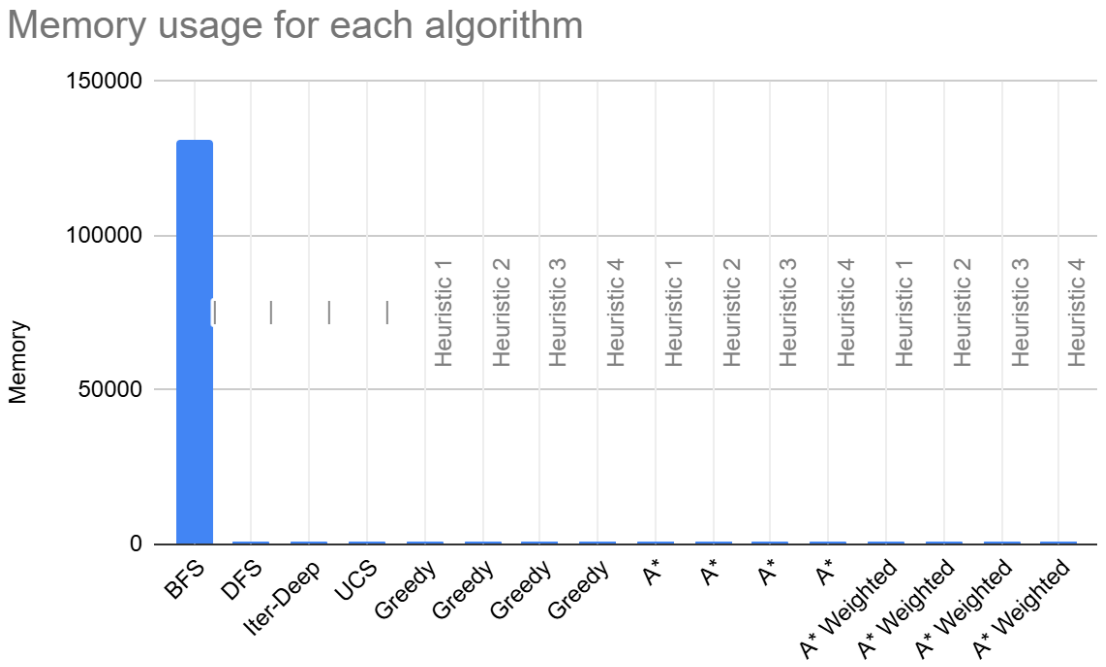
HEURISTICS

- The game uses 4 heuristics for the heuristic search algorithms. These heuristics are based on the ones presented previously : **Closer to Completion**, **Occupied Space**, **Number of Points**, **Number of Pieces to Play**.
- All the heuristics aim to achieve the best balance between the number of points earned and execution time.
- **Heuristic 1:** Sum of the number of points (negative) and the space left to complete a line/column. Gives more weight to the points proportional to the size of the game (board size).
- **Heuristic 2:** Sum of the number of points (negative) and occupied space. Gives more weight to the points proportional to the size of the game (board size).
- **Heuristic 3:** Sum of the number of points (negative) and the space left to complete a line/column. Similar weights given to both.
- **Heuristic 4:** Sum of the number of points(negative) and the number of pieces remaining to complete the puzzle. Similar weights given to both.

EXPERIMENTAL RESULTS

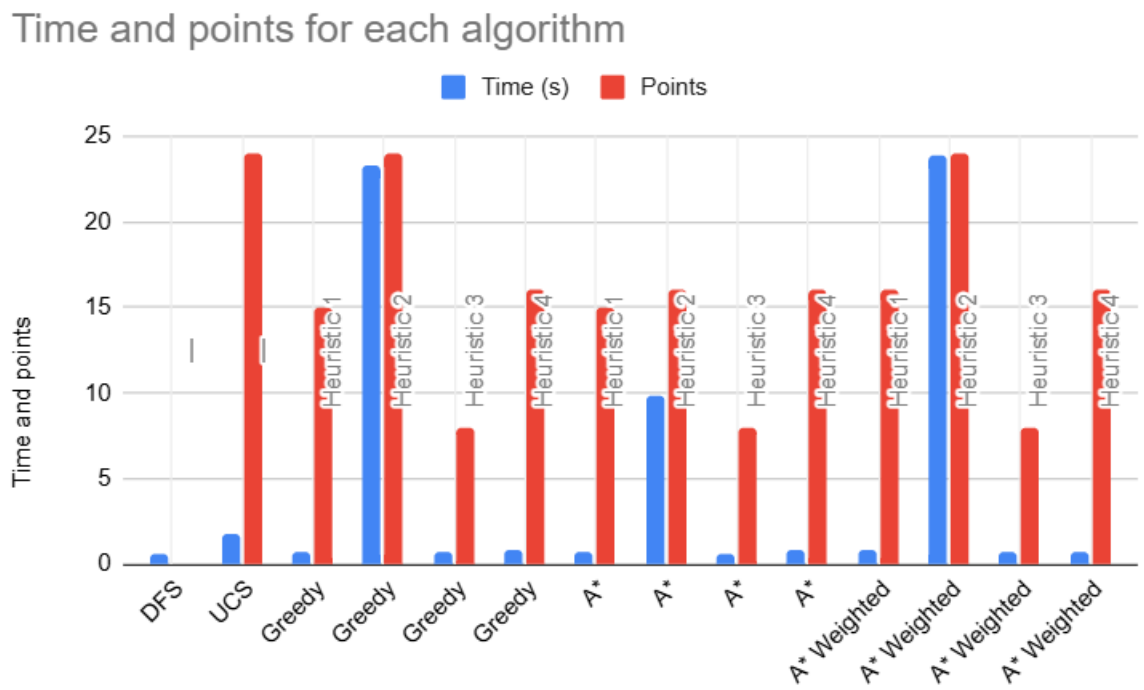
- **For a 5x5 board with 4 pieces**

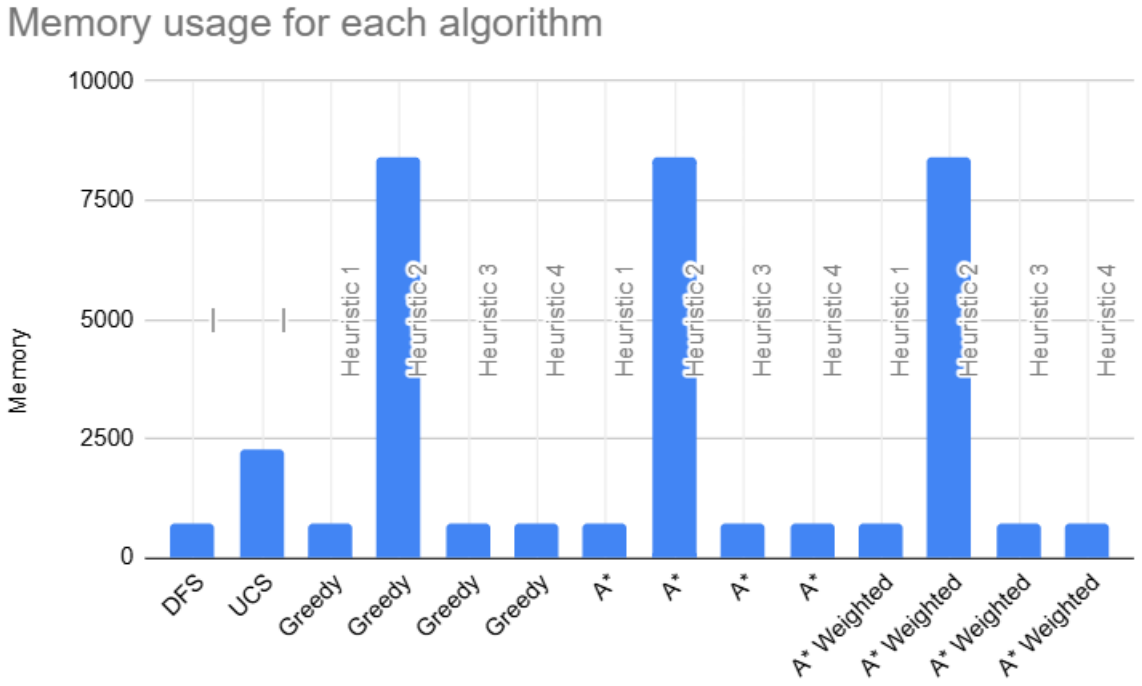




EXPERIMENTAL RESULTS

- **For a 8x8 board with 9 pieces**





CONCLUSIONS

- Wood Block is a game that generates a vast number of states due to the numerous possible positions for each piece. This characteristic negatively impacted the performance of algorithms that explore a large portion of possible states, such as **Breadth-First Search (BFS)** and **Iterative-Deepening**.
- The **A*** algorithm did not provide as much improvement over the greedy approach as anticipated. Significant enhancements were only observed when using Heuristic 2. Similarly, the **Weighted A*** algorithm yielded minimal to no improvement over standard **A*** results.
- The findings also revealed that heuristic-based algorithms utilizing more memory typically produced better results (i.e., higher scores). This is because these algorithms explore more states and evaluate a greater number of potential paths to the goal.
- In conclusion, this project demonstrated that the choice of AI algorithm significantly influences the outcomes. In our game, greedy algorithms achieved the best balance between points scored, runtime, and memory usage, whereas uninformed search methods generally delivered the poorest results.

REFERENCES

- Python docs: **https://docs.python.org/3/**
- Numpy docs: **https://numpy.org/doc/**
- PyGame docs: **https://www.pygame.org/news**
- Wood Block: https://play.google.com/store/apps/details?id=com.block.puzzle.free.wood&hl=en
- Wood Blocks: https://www.crazygames.com/game/wood-blocks
- Wood Block Journey: https://www.crazygames.com/game/wood-block-journey
- The algorithms were based on the ones presented in the course slides
- Git Repository: https://github.com/AlexL534/IA---Wood-Block