TenPair

Artificial Intelligence - Assignment 1
2020/21

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Group 33
- Davide Castro, up201806512
- Rui Pinto, up201806441
- Tiago Gomes, up201806658

Specification

TenPair is a game with a grid filled with a bunch of numbers and the goal is to eliminate matching pairs of digits.

By saying matching pairs of digits we mean digits that are the same or that sum to 10. Those pairs can be eliminated if they are in the same column with no other digits between them, or if they have no digits between them when scanning the grid from top to bottom, left to right.

You can also perform Deals which will force all the remaining digits to be pushed to the end of the board, by adding new rows to it.
To win the game you must clear the entire board by eliminating all the abovementioned matching pairs.

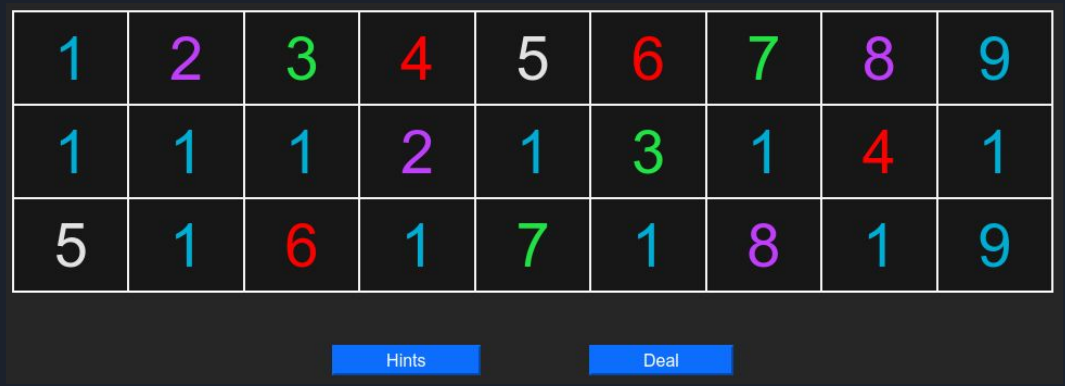


Figure 1: Game Visualization

Formulation of the problem

State Representation: The board is represented by a matrix with 9 columns and n rows; the empty cells are represented by a 0.

Initial State: Each cell has a random integer number between 1 and 9 (including).

E.g.: [[1,2,3,4,5,6,7,8,9],
[1,1,1,2,1,3,1,4,1],
[5,1,6,1,7,1,8,1,9]]

Objective test: Empty board (matrix n x 9; every cell has a 0). E.g:
[[0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0]]

Operators:

applyMove(board, validMove);

- **Preconditions**: the board has at least one valid move;
- **Effects:** makes a move that respects the rules of the game making the start and end cells empty;
- **Cost**: 1.

deal(board);

- **Preconditions**: the board is not empty;
- **Effects**: forces all the remaining digits to be pushed to the end of the board, by adding new rows to it;
- **Cost**: 1.

Implementation

The project was developed in **JavaScript**, using a **HTML/CSS** interface through the browser for the problem's visualization. As for the development environment, the group uses **Visual Studio Code.**

Implemented **search algorithms**:

- Depth-First Search
- Breadth-First Search
- Iterative Deepening Search
- Greedy
- A *

File structure:

- **src** directory, containing all the source code of the project;
- **resources** directory, to contain possible initial boards for input;
- **docs** directory, with the project's experiments documents

Data structures:

- class Node, used to generate the search tree
- 2D arrays (matrix), to store the board's state
- priority queue, for the greedy and A* algorithms, ordering the nodes by their value

Regarding the **uninformed search algorithms** (DFS, BFS and IDS), because a 'deal' can be performed at any time in the game, the search tree would grow exponentially, and so making the search time and space complexities eventually unachievable. Facing that problem, the group decided to limit the number of 'deals' that can be performed in an uninformed search.

For the A* algorithm, we considered that the cost of all moves is 1, including the 'deal' move, being the evaluation function the sum of the heuristic function value with the node's depth in the tree.

Approach

We tried several heuristic functions to solve this problem, and decided to give to the user the option to choose which one is used in the greedy or A* algorithms.

Also, during the development, the group developed two functions (‘heuristic’ 1 and 3) that ended up making up a new algorithm mixed with the greedy algorithm, that achieved better results, both in performance and number of moves, than the tradicional greedy algorithm with heuristic function.

- **Heuristic 2** evaluates the board based on the number of moves and empty cells: gives priority to the moves where the number of valid moves is minimum and where the number of empty cells is maximum.
- **Heuristic 4** evaluates the board by examining the numbers inside every non-empty cell and calculating the number of possible matches between them without looking at the valid moves, and adding that to heuristic value, also considering a necessary ‘deal’ move and an additional move for each value that is “isolated” (cannot be matched). For example, if there are three 3s and six 7s in the board, the value will be 6 because there are three 3-7 matches, one 7-7 match, and an isolated 7 that also makes a ‘deal’ move necessary.

Approach (new algorithm)

New mixed algorithm (heuristic 1 and 3):

Estimates the number of moves and deals that are needed to reach the final state. For each move in the tree, three actions can be performed:

- The first one is to evaluate all valid moves recursively for a given state and estimate the number of moves left to solve the board plus the ones already taken to get to that state.
- The second one is the same as the previous one, but additionally we perform a deal and repeat the process.
- The last one is to immediately make a deal, and then analyze all the valid moves. We then select the node that gives us the less amount of moves to get to the solution.

It works like a new algorithm, because it has a part where a node is looked up according to the aforementioned criteria and another part where the number of moves left is deducted by the **number of occupied cells / 2**. As soon as a solution is found, the algorithm will converge to that defined path.

Similar to **heuristic 1**, **heuristic 3** doesn't have into consideration the number of moves already taken by the node lookup.

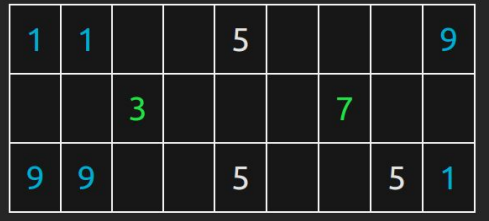
Experimental results

Results (represented in milliseconds), for each heuristic using different informed algorithms and boards:

Board 1

	1	2	3	4
Greedy	60	3200	60	2300
A*	56	INF	56	INF

Board 2

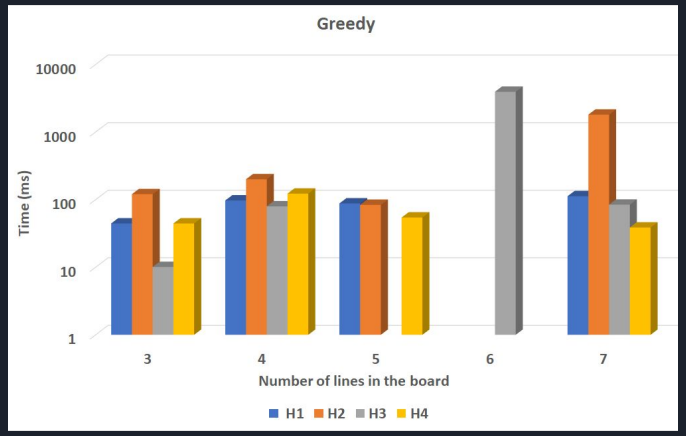


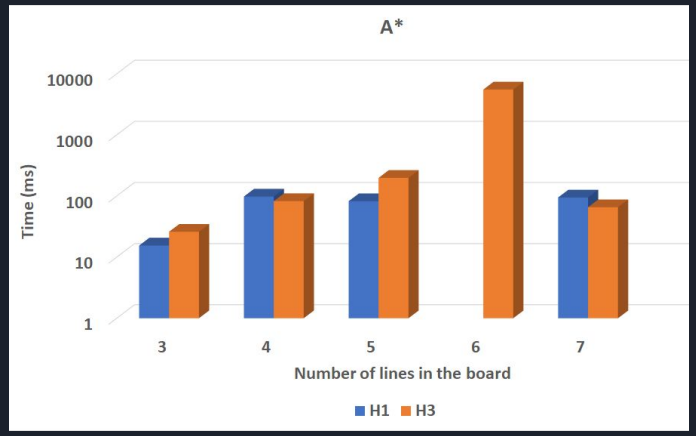
	1	2	3	4
Greedy	15	10	10	8
A*	12	65	20	11

Board 3

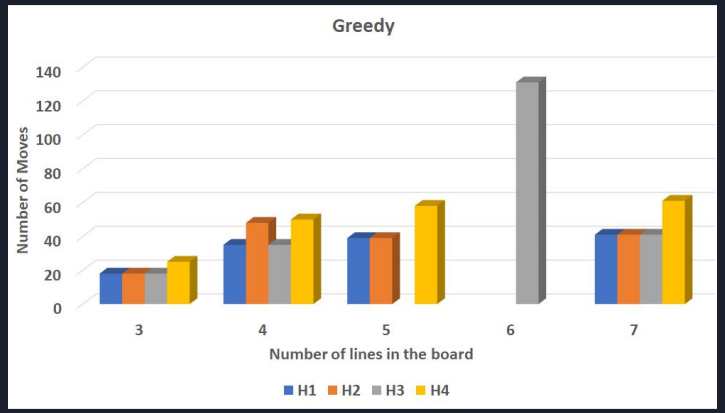
	1	2	3	4
Greedy	55	INF	48	422
A*	49	INF	18000	INF

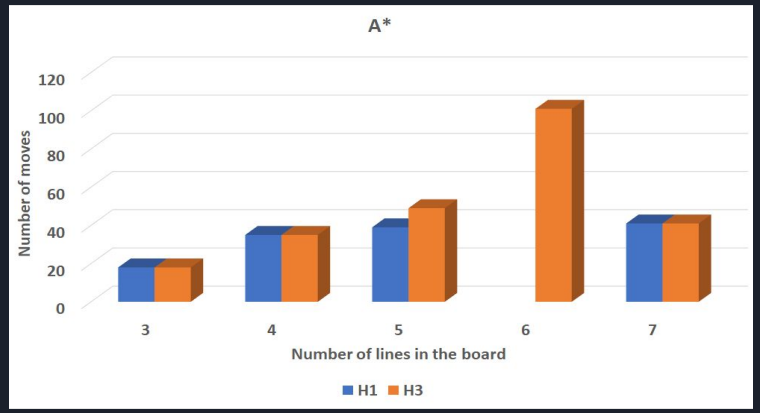
Time Performance:





Number of moves:





Experimental results (cont.)

As we can observe in the charts, the most efficient search algorithm to solve our problem is the greedy with approaches 1 and 3, and heuristic function 4 behind also with good results compared to the second.

Depth-First Search (Board 1)		
Max deals	**Time (ms)**	**Space (nodes)**
1	31	191
2	40	932
3	109	4036
4	1028	16726
5	7688	68028

The DFS, BFS and Iterative Deepening algorithms could not find a solution to the problem in a timely manner and reasonable memory usage, given that with the large quantity of possible moves they need to analyse because of the 'deal' move, the tree would grow exponentially. We concluded that, out of all uninformed search algorithms, only the DFS algorithm would find a solution but with a limit on 'deal' moves.

As for the informed search algorithms, the group could achieve good results all around with the greedy algorithm using the 4 approaches. The A* star algorithm, however, showed poor results when in comparison, not being able to achieve reasonable times with the normal heuristics (2 and 4). The group thinks that that is due to the relatively poor ability of the heuristics to correctly predict the total number of moves remaining to end the game while still being admissible, specially in the beginning of the search.

Conclusions

The development of this project helped us getting a more in-depth understanding of some of the contents of this course, namely the heuristic search methods. Although the project has some aspects that made it more difficult to implement, we managed to overcome most of the difficulties. Thereby, we consider that the assignment was successfully completed and the project goals were achieved, although with some limitations due to the complexity of the game and the problems that come with it.

References

During the research stage, the group found some helpful books and resources on the web that may help us during the development of the project, including articles referring the game ‘Numberama’, similar to ‘TenPair’

- Implementation of a priority queue in javascript: https://www.geeksforgeeks.org/implementation-priority-queue-javascript/
- Website where we can play the game https://www.logicgamesonline.com/tenpair/
- Root Beer (New Zealand) - Numberama game strategies http://www.rootbeer.co.nz/numbers-game-numberama-strategy/
- S. Russel and P. Norvig, *Artificial Intelligence : A Modern Approach*, *Third Edition*
- James Clune, *Heuristic Evaluation Functions for General Game Playing* http://ggp.stanford.edu/readings/cluneplayer.pdf