

# Piano mover puzzle

A 10X10 board room with an entrance of size 2 on one side and an exit of size 2 on the opposite side is crowded with furniture represented by tiles of size 1, 2, and 3 cells of different shapes.

The player has to move a piano, represented by a 2X2 tile, by sliding it (horizontally or vertically) into empty space through the room from the entrance to the exit. Besides sliding the piano, the player can slide furniture too.

You start by constructing an initial configuration by placing random furniture in random places into the room, depending on a density parameter  $[0,1]$  that represents the ratio between occupied cell/100.

The project has been divided into two packages:

- **GameModel**: it contains the file used for model the problem;
- **Heuristics**: it contains the file used for the heuristics' implementation.

The following file is present inside the first package:

**\_\_init\_\_.py**: it contains the piano mover puzzle game, where the various operations necessary for the correct execution of the game are carried out.

The following classes are implemented inside the file:

- **PianoMoverRepresentation**: it is used for having a “Snapshot” of the piano mover game representation. Thus, it gives information about the board, the piano and the objects in all the different states;
- **PianoMoverState**: it is used for having a “Snapshot” of the piano mover game state;
- **Game**
- **PianoMoverGame**: it contains the various operations necessary for the correct execution of the game. The following methods are implemented inside it:
  - **solution**: method that performs the check on the board after each move, returning 'true' if there is a solution and 'false' otherwise;
  - **neighbors**: method that performs the check on the board after each move, returning the possible moves that the piano could do.

The following file is present inside the second package:

**\_\_init\_\_.py**: it represents the file where the heuristics are implemented.

The following classes are implemented inside the file:

- Heuristics
- PianoMoverHeur: it contains the two heuristics that are used by the game.  
They are separated into the following methods:
  - manhattan
  - euclidean

The two heuristics are described better below.

**main.py** is the last file that represents the main class in which the moves made during the game are managed. Here is called the heuristics implemented as well. Moreover, the board is represented by a 10X10 matrix and initialized to value 0 for each cell. Random objects are put inside the board as well, represented by value greater than 1.

The picture below shows a possible initial board configuration.

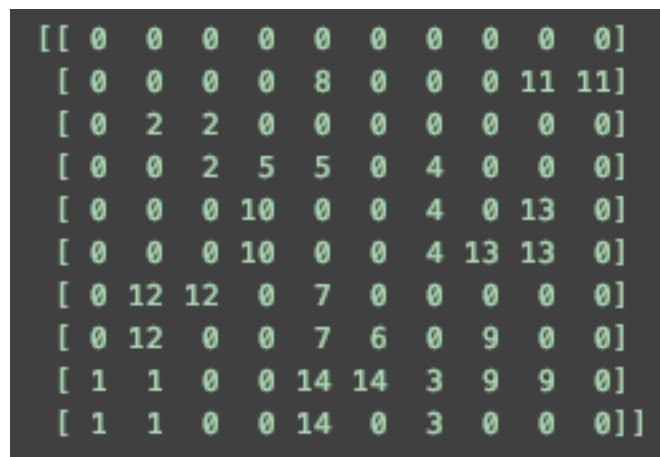


Figure 1. Possible initial board configuration

As we can see, the piano is initialized in the first four cells, starting from the left and bottom board's side.

## Heuristics function

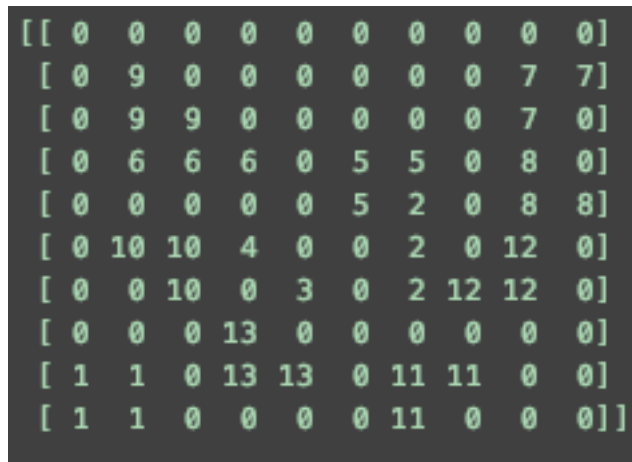
Most heuristics are linear weighted sums of various aspects of a given game state. In this sense, the heuristic shell is a static weighted summation of each individual aspect of a given board state. The base heuristic is given by the formula  $w * a$ . Each  $a$  is a static representation a different aspect of the board state. Examples of aspects include number of enemy pieces. The set of  $w$  represent the weights of each board aspect.

I took inspiration from this formula in order to realize the two heuristics that are used for the project.

## Manhattan heuristic

Inspired by the previous formula, I calculate the number of objects that are above the piano, so the ones that the piano might meet during its route. This sum is considered as the “ $a$ ” in the formula above. It is labelled as  $k$  and the exit is added as well. As “ $w$ ”, the manhattan distance between the top-left corner of the piano and the left side of the exit is calculated ( $m$ ). Thus, the final formula is  $m * (k + 1)$ .

The figure below shows the initial board configuration during a run with the manhattan heuristic.



[	0	0	0	0	0	0	0	0	0]
[	0	9	0	0	0	0	0	7	7]
[	0	9	9	0	0	0	0	7	0]
[	0	6	6	6	0	5	5	0	8]
[	0	0	0	0	0	5	2	0	8]
[	0	10	10	4	0	0	2	0	12]
[	0	0	10	0	3	0	2	12	12]
[	0	0	0	13	0	0	0	0	0]
[	1	1	0	13	13	0	11	11	0]
[	1	1	0	0	0	0	11	0	0]

Figure 2. Initial board configuration with the manhattan heuristic

The figure below, instead, shows the final board configuration with the time needed for all the configuration and the memory space used.

```
Level of the node: 45
[[ 0 0 0 0 0 0 7 7 1 1]
 [ 0 9 0 0 0 0 7 0 1 1]
 [ 0 9 9 0 0 5 5 0 0 0]
 [ 6 6 6 0 4 5 8 0 0 0]
 [10 10 0 0 0 0 8 8 0 0]
 [ 0 10 0 0 0 3 2 0 0 0]
 [ 0 0 0 0 0 0 2 0 0 0]
 [ 0 0 0 0 0 0 2 11 11 0]
 [ 0 0 0 0 13 0 0 11 12 0]
 [ 0 0 0 0 13 13 0 12 12 0]]

Level of the node: 46
Cost of the solution: 144

The file name is:
/Users/Luca/Documents/Università/Magistrale/Secondo_anno/Intelligenza_Artificiale/Repository/Homework1/main.py

The time needed is:
--- 88.36355590820312 seconds ---

The memory used is:
svmem(total=4294967296, available=1094995968, percent=74.5, used=3423633408, free=16547840, active=1120194560, inactive=1078448128, wired=1224990720)
```

Figure 3. Final board configuration with performance results

## Euclidean heuristic

Inspired by the previous formula, I calculate the number of objects that are above the piano, so the ones that the piano might meet during its route. This sum is considered as the “a” in the formula above. It is labelled as  $k$  and the exit is added as well. As “w”, the euclidean distance between the top-left corner of the piano and the left side of the exit is calculated (m). Thus, the final formula is  $m * (k + 1)$ .

The figure below shows the initial board configuration during a run with the euclidean heuristic.

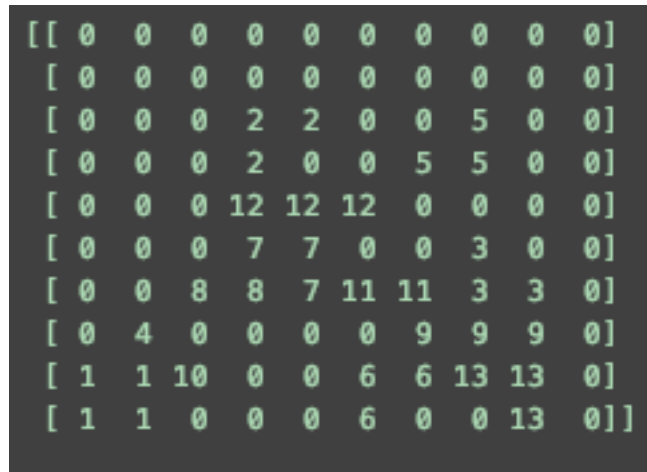


Figure 4. Initial board configuration with the euclidean heuristic

The figure below, instead, shows the final board configuration with the time needed for all the configuration and the memory space used.

```

Level of the node: 28
[[ 0 0 0 0 0 0 0 0 0 1]
 [ 0 0 0 0 0 0 0 0 0 1]
 [ 0 2 2 0 0 0 0 5 0 0]
 [ 0 2 0 0 0 0 5 5 0 0]
 [12 12 12 0 0 0 0 0 0 0]
 [ 0 7 7 0 0 0 0 3 0 0]
 [ 0 0 7 0 0 11 11 3 3 0]
 [ 4 8 8 0 0 0 9 9 9 0]
 [ 0 0 0 0 0 6 6 13 13 0]
 [ 0 0 0 0 10 6 0 0 13 0]]

Level of the node: 29
Cost of the solution: 93

The file name is:
/Users/Luca/Documents/Università/Magistrale/Secondo_anno/Intelligenza_Artificiale/Repository/Homework1/main.py

The time needed is:
--- 19.592330932617188 seconds ---

The memory used is:
svmem(total=4294967296, available=1156849664, percent=73.1, used=3480641536, free=32718848, active=1139609600, inactive=1124130816, wired=1216901120)

```

Figure 5. Final board configuration with performance results