TenPair

Heuristic Search Methods for One Player Solitaire Games

Artificial Intelligence - 1st Assignment Checkpoint
MIEIC 2020/2021

Group: Class 8, G40
Diana Freitas - up201806230
Diogo Samuel Fernandes - up201806250
Hugo Guimarães - up201806490

Teacher: Luís Paulo Reis

TenPair - the game

TenPair is a logic one player game in which the player must clear all the digits from the board by **eliminating pairs of digits that are the same or that sum to 10**. The fewer the number of moves, the better the score.

A pair can be eliminated if one of the following conditions verifies:

- both digits are in the same column or row with no other digits between them;
- the right end of a row pairs with the left end of the following row;

At any time, specially when there are no matches left, a **Deal** option is available. The Deal option, which does no count as a move, adds all the remaining digits of the board, in order, to the end.

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

Fig. 1 - Initial board with 3 types of pairs

	2	3	4	5	6	7		
				1	3	1	4	
5	1	6	1	7	1	8		

Fig. 2 - A board with no matches left

	2	3	4	5	6	7		
				1	3	1	4	
5	1	6	1	7	1	8		
2	3	4	5	6	7	1	3	1
4	5	1	6	1	7	1	8	

Fig. 3 - After a Deal on the board of Fig. 2

Search Problem Formulation

State Representation

- matrix, **which we represent as a list**, in which each cell may be a number from 1 to 9 or empty (None):
  $\forall$ element $\in$ matrix , element $\in [1,9] \cup \{None\}$
  (We decided to use a list because it facilitates comparing the last element of a row with the first of the following row)
- number of rows
- number of columns
- number of moves
- number of deals *(bfs & dfs)*
- previous state
- heuristic

```
class Game:
  def __init__(self, moves, dealVal, rows, cols, matrix, previousState=None):
    self.moves = moves # number of pairs made
    self.dealVal = dealVal # number of deals
    self.cols = cols # number of columns
    self.rows = rows # number of rows
    self.matrix = matrix # game matrix as a list
    self.previousState = previousState
    self.heuristic = 0 # heuristic evaluation of the state
```

Search Problem Formulation

Initial State

```
gameState = [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]
cols = 9
rows = 3
moves = 0
game = Game(moves, 0, rows, cols, gameState)
```

We decided to limit the number of deals to 1 in the breadth first search and depth first search algorithms, because, otherwise, the number of states would grow too quickly and increase the cost of evaluating the board.

Objective Test

To end the game, the board must be empty:
$\forall$ element $\in$ matrix, element == None

```
def objectiveTest(self):
  if self.matrix == [None]*len(self.matrix):
    return True
  return False
```

X / Y	0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8	9
1	1			2	1	3	1	4	1
2	5	1	6	1	7	1	8	1	

Search Problem Formulation

Operators - names, preconditions, effects and cost

Names	Preconditions	Effects	Cost
Remove pair of cells B[Y1][X1], B[Y2][X2] Has we represented the matrix as a list, we need to calculate the column and row of each cell with index I: Y1 = I1 // 9 Y2 = I2 // 9 X1 = I1 % 9 X2 = I2 % 9	The elements of the pair are the same number or sum 10: (B[Y1][X1] == B[Y2][X2] ∨ B[Y1][X1] + B[Y2][X2] == 10) ∧ (The elements are in the same column in consecutive positions: (X1 == X2 ∧ *Y1 < Y2** ∧ Y1 + 1 == Y2) ∨ The elements are in the same row in consecutive positions: (Y1 == Y2 ∧ *X1 < X2** ∧ X1 + 1 == X2) ∨ One element is the last of a row , the other is the first of the next row: (*Y1 < Y2** ∧ Y1 + 1 == Y2 ∧ M1 = B[Y3][X1] | X3 > X1 ∧ M2 = B[Y3][X2] | X3 < X2 ∧ ∀ el ∈ M1 ⟶ el == None ∧ ∀ el ∈ M2 ⟶ el == None))	B[Y1][X1] == None B[Y2][X2] == None	1
Deal	This precondition is only applied to bfs and dfs algorithms dealValue < 1 (no deal has been made yet)	All the remaining digits are added to the end of the board, in order, ignoring empty spaces.	0

* To remove symmetric comparisons

Search Problem Formulation

Heuristics

For the Greedy and A* algorithms we tested the following heuristic:

H1 = Maximum number of pairs on the board

```
def heuristic(matrix):
  return len([el for el in matrix if el != None])/2
```

For the A* algorithm, we minimize the sum of the path already carried out , which is the number of pairs already made, with the minimum expected until the solution, which is obtained by applying H1 heuristic.

Data Structures

Algorithms	Data Structures
DFS, BFS, Iterative Deepening	Dequeue, Set
A*, Greedy	PriorityQueue, Set

Use Data Structures:

- **Set** - Used to store the visited nodes.
- **Deque** - (double ended queue) Used to Expand Nodes by the order they are added to the queue. Allows O(1) pop and append.
- **Priority Queue** - Used to sort nodes the nodes to be expanded based on a heuristic.

Work developed & File Structure

Programming Language: Python

Development Environment: Visual Studio Code

References:

- To implement the interface: https://stackoverflow.com/questions/26213549/switching-between-frames-in-tkinter-menu

File Structure:

- algorithms
  - aStar.py
  - breadthFirstSearch.py
  - depthFirstSearch.py
  - greedySearch.py
  - iterativeDeepening.py *(still incomplete)*
- core
  - **game.py**:
  - Game class that represents a state and includes the objective test, the functions that verify the preconditions and apply the operators ("deal" and "make pair"). Also, it includes the functions to print a state and a sequence of states.
- interface *(in development)*
  - main.py
- **main.py**: to run the game