universidade de aveiro

IA-Rush Hour

LEI – IA 2022/2023

Inês Baptista – 98384 (65%)

Catarina Costa – 103696 (35%)

Implementation and Optimizations

- Optimized cursor
- Use of heap queue – python heap library - to store unexplored nodes (this orders the open nodes by the respective heuristic)
- Minimization of python classes – nodes are represented by tuples

- student.py
  - This code communicates with the server. It receives the current level configuration and the viewer event updates. These updates are contained in a dictionary with the current position of the cursor, the selected car and the current grid in a string format.
  - In each level, it calls a search function (imports module from search.py) to calculate the level solution.
  - It detects crazy cars also. To handle them, we adopted the strategy of correcting the move performed by the crazy car. To avoid unnecessary movements, the crazy car is only corrected if its movement is not beneficial to solve the puzzle. Our solution checks for crazy cars every time there is an update to the current state.
- my_common.py
  - This code was provided by a professor, and it contains a Map class that represents the grid. It also contains methods so that we can manipulate the grid – like moving the cars – and get information of the grid – examples are the current piece that the cursor is holding and the coordinates of all the cars in the grid. We optimized some of the operations in this module to
- search.py
  - This code contains the class for the Search Domain where we define a Heap Queue for storing the unexplored nodes and a dictionary to store the visited nodes. Each node represents a configuration of the game. The nodes are represented by tuples and store the information – a map object with the grid cost, heuristic, depth and an array with the all the states that lead to the solution. The search function has the search algorithm. In each cycle of the search algorithm, a node is open and the program generates all the possible moves for every car. Then we add restrictions to which nodes can be added to the unexplored nodes array for them to be explored next.

The Search Algorithm

For this project we used the perspective of the Best First Search Algorithm.

Best-first search is a graph traversal and search algorithm that explores the nodes of a graph in an order determined by a heuristic function. The heuristic function is used to evaluate the "promising-ness" of a given node, and the algorithm always selects the most promising node for expansion.

The algorithm works by starting at the root node or the starting node of the graph, and enqueuing it into a heap queue. The queue is ordered based on the heuristic function, so the most promising node is always at the top of the queue.

The algorithm then dequeues a node from the queue and adds all its unvisited neighbors to the queue. This process is repeated until the queue is empty or the solution is found.

.

Heuristics

To **speed** the search process, we ordered the queue by the heuristic value.

- DistanceToExit: the distance between the car A and the exit wall
- blockingCars: the list of all the cars that are blocking the car A

The metrics above were used to calculate the heuristic. The final value was calculated by:

distanceToExit + blockingCars

Conclusion

We are satisfied with the performance of the agent – the agent is able to find a good solution in a small amount of time (he is able to solve all the puzzles in 30 seconds without the connection to the game engine). However, the solution found by the agent is not the best solution possible, this is due to the algorithm that is used to calculate the solution.

Our search algorithm, is the Best First Search algorithm, which means that the node with the lowest heuristic value is always chosen for expansion. We would be able to have a more intelligent agent if we included the cost in the ordering of the unexplored nodes but when we tried ordering by the sum of the heuristic and the cost, the program got very slow.

We calculated the cost using the Manhattan distance. Which is the distance to parent node that is given by the steps that the cursor needs to make to get to the new node/configuration.

Github link: https://github.com/detiuaveiro/ia-tpg-rush-hour-98384_103696