Sokoban

Desenvolvimento de um Agente autónomo

Inteligência Artificial
2020-2021

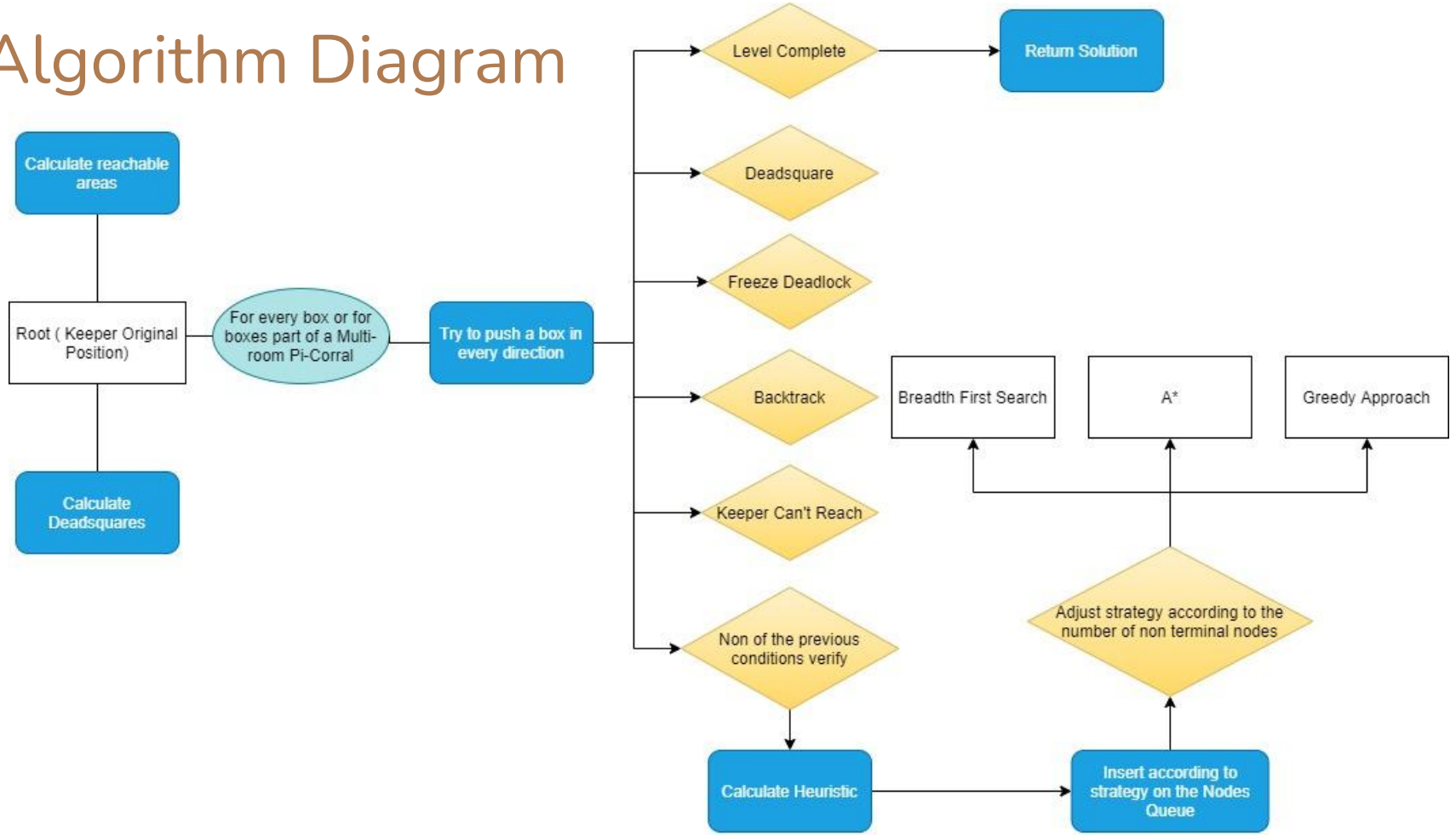
Grupo: Daniel Gomes (93015)
Mário Silva (93430)

universidade de aveiro

Algorithm Structure

Node - represents a possible push of a Box.
Tree Search - tree structure where a root node expands new nodes, calculates every possible action for the current state.
Search Functions - mostly used the BFS strategy in this algorithms since the maps are relatively small, the BFS is more efficient and gives the shortest path.
Simple Deadlock- positions in which a box can't be at otherwise it won't be able to move (deadsquares).
Freeze Deadlock- This function checks if moving a box to a certain position gets "frozen" (immoveable). If it has boxes surrounding it, it has to checked if the other boxes are frozen too after this move, if all the frozen boxes are on goals then it's not considered a deadlock, otherwise the push is on a deadlock state.
Hash Table (Backtracking) - dictionary structure that stores the hashes of all the boxes as a key and a list of the keeper's position as the value. If in two states the hash of the boxes are equal and the keepers' are different but are reachable from one to the other keeper's position, the state is the same and can be pruned, if it can't reach the other keeper's position, the keeper position is just appended to the list.
Pi-corral Pruning - A corral area consists in the positions that the keeper can not reach, which can include boxes (corral-boxes). The keeper's reachable positions are calculated at each state to find them using an initially calculated reachable positions of the map and see if they are now unreachable. Then we check if the boxes on the edges of the corral only have moves to the inside of the corral or if it doesn't have, check if there are surrounding boxes that may be blocking it, this way it joins different corrals and checks for the existence of multiroom Pi-corrals, if it is a Pi-corral, the pruning itself may occur if at least one box that is in a pi-corral position is not on a goal square, or if at least one goal square in the PI-Corral does not contain a box.

Algorithm Diagram



Search Strategies

Greedy assignment

In order to do the heuristic base calculation, we looked forward to the Greedy Approach. This method consists on calculating the distance of every goal to every box and, after that, assigning to every goal the first available box with with the shortest distance. The distance of a box to a goal is done using the Manhattan distance and in order to check if a box is available, in every assignment we must be sure that no box is assigned to more than one goal. Finally, the value of the heuristic for the push is the sum every distances to the assigned boxes to its goals.

Strategy alternation

Since our algorithm works fast for levels that have small amount of possible pushes, we change our heuristic calculation strategy based on the number of non terminal nodes. Until a certain number of non terminal nodes, it's used the depth (number of pushes) as the comparitision heuristic between two nodes, then, the heuristic calculated with previous function plus a certain percentage of the depth, and finally the same as the previous one but an even smaller percentage of the depth.

Optimizations

Initially, in order to find faster ways to append nodes to the list of open nodes we tried to use an Min Heap and maintain it structure instead of adding to a list and sorting it. However, we came across another solution: the **Bisect** module, which gives better solutions in less time. With this module we do one-by-one insertions on the right place of the list using the comparisitions of the current strategy, this way, we keep our list of nodes always sorted and don't have to sort the whole list each iteration.

To check certain positions on the initial map we simply do bitwise operations.

The map class is not used on the creation of new states, instead we use a set of the boxes. The choose of this data structure is pretty simple due to the fact the searching in Sets is a O(1) operation. Also if we used the map class instead of Sets, we would need to do "deep copy" copies of objects of these class, that are a little bit more expensive computationally than write operations in these sets.

During the calculation of each possible push, the hash of the boxes for the backtrack is reused to compare with the hash of the storages, this way the comparison is faster and allows to immediately detect if the level has been solved.

Another optimisation that we decide to do was instead of only verifying if the keeper could reach the position to do a certain push of a box, after the creation of a Node Object was to do this verification before the creation of the Node. This increases a little bit of the speed of our algorithm since we do not need to create an unnecessary object.

We also tried to use as much as possible local variables to reduce variables' lookup times.

Results and Conclusion

After the implementation of our Algorithm Structure mentioned before, and all the optimisations done we manage to do 150 consecutive levels out of the original 155 levels, and over 4 levels out of additional 11 levels . Unfortunately we were not able to complete all of them.
Besides this we achieved pretty good solutions in small amounts of time in most of the levels, and there were ideas that we had in mind that were not implemented in the due of time, but, as future work, we would do:

- A function that calculates the exact order on which the goals must be filled to prune a lot of cases that cause a deadlock, this could be also made by using a graph (Maximum Bipartite Matching) and assigning the boxes to storages and see if a move causes any box or goal without a possible assignment.
- A function that detects Symmetric levels. This would be important to get the solution in a short amount of most of the additional levels since great part of them are symmetric. For example, if we a level is symmetric by 90 degrees, then we could divide the map into 4 sub-areas and solve only one of them, since the solution would be the same to the other sub-areas.

We used some fonts as research to complement our work in this assignment: http://sokobano.de/wiki/index.php?title=Main_Page and discussed ideias with:

Bruno Bastos (93302) and Leandro Silva (93446).