Sokoban Report

Caillau, Arthur caillau@kth.se
Coots, Alden coots@kth.se
Andrn, Joakim joaandr@kth.se
Reina, Luis lreina@kth.se

2012 - Oct -10

1 Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.
[1]

2 Introduction

An agent was designed for solving the game Sokoban. The goal of the game is to move the player through a maze in order to push boxes onto goals. This problem is one often studied within the field of artificial intelligence due to its high complexity. The problem is also similar to that of a robot moving boxes within a warehouse, thus the solutions to the problem have possible real life applications. The high complexity of the problem is caused by the enormous search tree of possible player actions. Each turn the player has up to 4 different moves, if the solution consists of N player moves, the size of the search tree will be in the order of 4^N. For larger maps this quickly becomes unsolvable within reasonable time. The number of maps possible to solve can be increased by reducing the search tree and employing efficient search algorithms.

3 Problem formulation

A Sokoban puzzle consists of a map with static walls and goal squares. The map also gives the initial positions of the boxes and the player. The goal is to push all the boxes onto the goals. The player can move, up, down, left and right and can only push one box at the time. The solution to the problem is a string of player movements that accomplishes the goal. The task was given to build an agent able to solve any given Sokoban puzzle within 60 seconds. A set of practice maps were given to demonstrate the complexity of the maps and also give opportunity for training. Next follows the methods employed to solve the problem.

4 Methods

4.1 A* (A star) search algorithm

To solve any search problem of this magnitude, the choice of a search algorithm is important. The A* search algorithm provides an efficient way of finding the shortest path between two points on the board. The A*

[1] TODO .

algorithm is an improvement of the breadth first search algorithm in the way that it prioritizes squares that lie closer to the goal according to a certain heuristic function. While doing it this it also keeps a record of the distance traveled so that it can return to the start if the path proves to be inefficient. The heuristic function is in this case simply the distance disregarding walls and other objects to the goal. This will lead to varying efficiency of the algorithm due to the vastly different layouts of the maps. On some maps disregarding walls will lead to very bad approximations and thus high time complexity. But most of the time the search should work in polynomial time with respect to the distance to the goal.

4.2 Modifying search for boxes and players

The possible movements are different for the player and the boxes, therefor the search algorithm have to handle them differently.

4.3 Matching box to goals

The Sokoban puzzle requires more than simply finding a path between to points. The puzzle is dynamic and moving one box will open up paths for other boxes.

4.4 Reducing the search tree

The methods above are enough to solve smaller problems and would be able to solve any Sokoban puzzle given infinite time. But since the time is limited some optimizations are required to solve larger problems. One way of optimizing is to reduce the size of the search tree. This can be done in many different ways, here are some methods that were considered:

1. Remove identical subtrees by saving old states.
2. Disallowing movements of boxes into dead ends such as corners, dead locks and walls without goals.
3. Handle tunnels as single cells.
4. Identifying goal areas and use predefined paths to store the boxes in them.
5. Use predefined recursive functions to move blocks out of the way.

5 Implementation

The agent was implemented using Java. The map was given to the agent in the form of strings containing the map with all the initial positions of player and boxes. To handle the problem more efficiently a data structure was created for the map where all the cells, boxes and actions were represented by different classes. This made it possible to quickly manipulate the map without worrying about illegal moves or losing track of any elements. This structure was essential in implementing an A* search that would be able to handle both players and boxes. Since the box have to be pushed by the player, certain moves that are available to to the player, are not available to the box. With a fully operational search function it is possible to start matching boxes to goals.

6 Analysis of experimental results

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

[2]

7 Reflection

In our first meeting we decided to first focus on creating an efficient search algorithm and using that algorithm to solve a relaxed problem where we simply tried to match each box to a goal without implementing methods to push boxes out of the way of other boxes. Arthur got the task to i mprove his search algorithm from home work 1 with A* search. The rest of us constructed functions to handle the problem of moving a box instead of a player. At the milestone we tried to put it all together but failed to solve any maps since the interface between our functions were not functioning properly and our relaxed problem was too relaxed. After failing at the milestone everyone looked through the code by themselves and after meeting up we decided to integrate the possible box moves into the search function. We also looked into adding multiple new features to reduce the search tree.

During the entire work process we created test functions to assure that our methods yielded the expected results. We also used a git repository to synchronize our work during the process.

If we could do the project again we would probably increase the number of shorter meetings and work more on the interface between our functions. We would also have used more unit tests to detect problems earlier. Strategy wise we probably would have started off with first creating the search function again

[3] How did you plan to solve the problem? How did you plan the work in the group? What other methods did you try out? What did work/ did not work? Why? How did you end up with the current approach? How did you measure the performance and success during the process? How would you solve it if you were asked to do it again given what you know now?

References

[1] Computer Science Department. *Sokoban strategies* , University of Alberta, CA , 2004. `http://webdocs.cs.ualberta.ca/~games/Sokoban/program.html`

[2] TODO .
[3] TODO