Transportation With Artificial Intelligence

Moldovan Horia-Andrei

January 2020

1 Introduction

Transportation is a well spread domain across the whole world. Each day we strive to make our life as easy as possible and not to waste precious time stuck in traffic, on a wrong route or in other circumstances. The world of Artificial Intelligence is growing bigger and bigger with every passed day. So why not try to find the most efficient solutions to different transportation problems by making use of it. In the following I will be presenting a very small piece of the whole world of AI interacting with transportation, but the final goal is to growing the users interest in this domain by showing a small part of what AI is capable of.

2 Bibliographic Study/Research

The exact idea I chose for my project is composed of smaller projects which simulate different problems which have to do with transportation. Firstly I represented the way a taxi would move through the city in order to move people around. After that I wanted to create some type of movement with a bit more constrains, not to let the objects move that freely, so I moved to the idea of the Hanoi Tower, where the movement of the disk must follow some requirements. So what is exactly the Tower of Hanoi? The Tower of Hanoi (also called the Tower of Brahma or Lucas' Tower and sometimes pluralized as Towers) is a mathematical game or puzzle. It consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.

2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.

3. No larger disk may be placed on top of a smaller disk.

The minimal number of moves required to solve a Tower of Hanoi puzzle is 2n 1, where n is the number of disks. With 3 disks, the puzzle can be solved in 7 moves. Besides the theoretical ideas a bit of programming research has been made in order to properly know how to use the PDDL language. There are a lot of problem definition domain language examples, some of them which even won big prizes so they were a truly helpful way to let me familiarise more with this programming language. When things got complicated, Fast Downward web page was also an useful helping hand. Not to mention the lecture slides we were presented at the Artificial Intelligence course at the university and all the guidance and the answered questions from the laboratory assistants and sometimes some help even from my classmates.

3 Analysis and Design

The actual idea was to make a program which makes use of the artificial intelligence program Fast Downward, by giving him some PDDL files where he will be told to give the most optimal solution to the tower of Hanoi problem. I first started implementing the game with only 3 disks. Then I observed that it can easily find solutions for 10 disks or even more. Thenceforth I started thinking of ways in order to make it more difficult. I thought of adding to the scene two robots which will have the task of moving the disks themselves. The solution improved exponential with the growth of the number of disks. It was working fine but the robots had no interaction between them so I started thinking of a way they could work together in order to achieve some task, still referring to the tower of Hanoi problem. The solution I found was to add more constrains to the robots so I made it that one of the two robots can only work with the left and middle rods and the other one only with the middle and the right rod. So this way if they were for example to move a disk from the left rod to the right rod in the most optimal way, they had to work together in order to achieve the task. In order to make this possible a lot of variables and action had to be declared because the complicated the task gets, the more specifications you have to create in order to create a way to tell the program what you want from him. So you have to properly specify what it has to do and how he has to do it. In order to do that, each individual object has to have his own predicates, which work like instantiating the object, and predicates where he interacts with other objects. The actions are the place where the state of the objects is verified and changed in order to achieve some functionality. Besides the Tower of Hanoi programs I also wanted to do something which is more applicable to the real life, even if it's easier to implement so I decided to implement a taxi simulation, where cars can move persons from a location to another. The hardest part in writing PDDL files are the proper choose of the predicates. If you have already found the way you want to instantiate the objects and where to place them in relation with the other objects then the other tasks become more simple. The same approach works here too. After the role of each object, like the taxi going from a location to another, being initially placed somewhere, the person being in or out of a take and at a specific location, the consistence of the fuel of the car and some others tasks were decided, all that is to be done is to put them together.

4 Implementation

While implementing the programs, the usual model of planning domain definition language was used. So minimum two pddl files must be created: one domain file and at least one problem file. The problem file must contain the used objects, the initial state of the world and the final state which is the goal state we want to reach. The domain file contains the predicates, which are booleans specifying the state of the objects. It also contains actions, which are the only way to change the state of the world.. The actions are composed of parameters, preconditions and effects. They work in the following way: the state of the objects given as parameters is checked to be true, with the help of the predicates, and if none of it is proven to be false, the action happens by changing the state of the objects in the effect component, again by making use of the predicates. So well defined predicates could simplify your work a lot. While implementing the original version of the Tower of Hanoi two kind of objects were defined: Disk and Rod. Predicates were created in order to tell if the disk or the rod have no object on top of them, if one object is on top op another and if one object is smaller than another. The initial state is the one where all the 3 disk are on top on another in ascending way on the left rod and the goal state would be the same stacking of the disks but on the right placed rod. In order to achieve this, the action move was created where we make sure that the rules of Hanoi are not broken and move one disk from a rod to another. Furthermore, 2 robots were added to the implementation of the planning problem with the task of replacing the move action and use their help in order to change the location of the disks. So two new objects have been added: robot one and robot two. The added actions which have to do with the added robots are help us to tell if a disk is carried by a robot, if a robot is carrying a specified disc and if a robot has no disk in hand. The action move was replaced with the actions pick and drop. The same conditions and effects as by the move action were split in two and composed with the new created predicates the two new actions were created. Last modification to the Tower of Hanoi problem was to make the robots cooperate and in order to do that the robots were split into left robot and right robot, because now they are able to place disks only on the left or mid rod, respectively mid or right rod. In order to do this and to still respect the rules of the Hanoi Tower a lot of predicates and actions were needed. The pick and drop actions were replaced with pick right, pick mid, pick left respectively drop right, drop mid, drop left. The take action was also necessary. This one does nothing more than passing a disk from a robot to another. The initial and goal state stays the same in all of the modifications. The last problem I implemented had nothing to do with the Hanoi Tower but is also related to the same field, namely transportation. It does nothing more than moving people from a place to another with a taxi. The objects of this program are the taxis, the people, the locations and the possible status of the fuel. With the help of predicates we set the taxi or the people to a specified location, we specify if a person is in a taxi or not, if the fuel must be filled, and if the locations are connected. The actions are very simple to understand. We have the get in action which moves a person in a taxi if they are both at the same location, the get out action, which moves the person outside of the taxi and actualises his location, the move action, which moves the location of the taxi from the first to the second location if the taxi has enough fuel and if the locations are connected. Lastly is the fill fuel action, which does nothing more than filling the tank of the taxi if it remains without fuel.

5 Testing and Validation

All the testing and the validation of the programs were doing by repeatedly changing the effect in the problem file and observing if the output coincide with what we said and if the path to the specified output is the correct one. When this was not true, debug was necessarily in order to find where the code breaks. The only way to do this was by finding which actions were not performed or not correctly and taking a look at their preconditions. By doing so, each bug could be fixed with more(4 hours) or less work.

6 Further improvements

Between the Hanoi modified problems and the taxi problem, the taxi one is the closest one to the real world. One of the possible problems in my application is the fact that when the taxi car runs out of fuel, it automatically uses the action fill fuel, like the taxi driver had some recipients filled with fuel in his trunk. In real life this scenario is not very reliable because the way it should work should be that the taxi driver anticipates the fact that he will remain without fuel and he should start looking for a gas station. So another type of object should be added, namely the gas stations. Moreover, in my application the taxi car consumes one quarter of fuel on each street, fact that is again not very applicable in real life, where different streets have different lengths. So another object that must be added is the distance. So a lot of computations should be added in order to bring the app as close as possible to the real world.