Artificial Intelligence – Final Project

Search Algorithms

Antonio Viñuela Pérez - 100383461  
Fernando Pérez Lozano - 100383462

Index

1. Introduction
   1. The Concept
   2. The Task
2. Basic Problem
   1. Task description
   2. Code description
3. Advanced Problem
   1. Task description
   2. Code description
4. Conclusions
5. Personal comments

Introduction

**The Concept**

In this project our objective is to put into practice the knowledge we’ve acquired during the first weeks of the course on **search algorithms**. The implementation of the project has been carried out using **Python 2.7**. We’ve been provided with some libraries and packages such as *pydot* and *pygame* for the graphical interface implementation, and the library *SIMPLE-AI* for the implementation of the search algorithms.

**The Task**

The scenario we’ve worked with in this project is that of a pizza deliverer who has to please a set of customers choosing the best path depending on the criteria (the algorithm that is applied). Therefore, in the map we have:

* **A start position** from which the deliverer starts working and to where he/she has to go back to give the task for concluded.
* **A deliverer** that will be moving around loading pizzas on the pizza shops and unloading them on the customer’s places.
* **Several hungry customers** that have made an order of up to 3 pizzas per customer.
* **Several pizza shops** that will provide the deliverer with the pizzas to be delivered.
* **Several buildings** that represents parts of the city the deliverer must get around in order to get to the customers (meaning that buildings are obstacles).

Basic Problem

In this part we had to design and implement the problem representation in the form of states and actions, together with a sensible heuristic function. This heuristic didn’t have to consider the cost that could be added from the pizzas that were being carried by the deliverer, but only the distance to the start position at each state. The possible scenarios had some limitations:

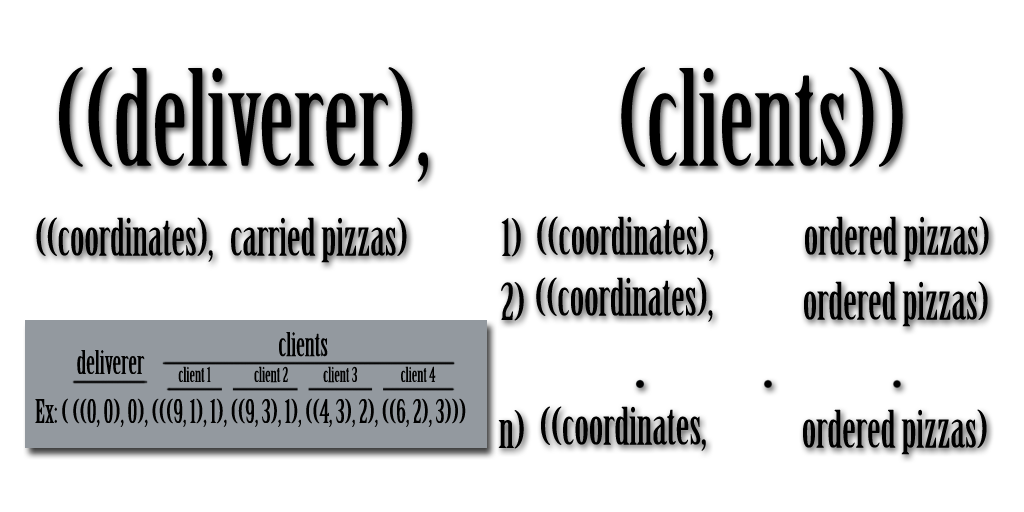
* Only one restaurant in the map.
* Only one client who can make up to two orders.
* The deliverer can only carry up to 2 pizzas simultaneously.
* The deliverer can only run through street tiles, delivery locations and restaurants.
* All actions have a cost on 1.

**State Design**

In search algorithms, every state is an immutable entity that will be visited as a node on the search tree the algorithm is generating. This is why we had to implement a state that was immutable, and for that we used the **tuple** entity on python, which is, in the end, a sequence that cannot be modified.

We observed that the only information we needed to keep track of the state of the simulation was:

* The coordinates of the deliverer.
* The number of pizzas the deliverer is carrying.
* The coordinates of every customer.
* The number of pizzas each customer has ordered.

Of course, this data had to be presented in an accessible way, so we generated a tuple that looks as follows:

In the end is just a tuple that stores other 2 tuples. The first one stores the info of the deliverer in a tuple that holds 2 entities. These entities are a tuple of coordinates (x, y), and an integer with the amount of loaded pizzas. The second tuple stores the information of the clients in *‘n’* other tuples. Each of this client tuples store a tuple of coordinates (x, y) and an integer with the amount of ordered pizzas.

**Code Description**

For the implementation of the simulation, we had to complete 5 functions. We’re briefly going to describe their functionality below. For further information, we’d like to refer you to the code, where the comments may cover the subject more extensively.

Every function receives as a common argument the instance of the gameProblem class that is being used.

* action(self, state):

This function receives a state and returns a list with all the possible actions that can be carried out. These actions are *‘North’, ‘South’, ‘East’, ‘West’, ‘Load’ and ‘Unload’.*

For those that implied movement, we checked that the “new” position was inside the dimensions of the map, and that it was free of obstacles, which meant that the tile was not a *‘building’*.

For the *‘Load’*  operation, we checked that the deliverer was in the same position as a pizza shop, and that the maximum load capacity hadn’t been achieved yet.

Similarly, for the *’Unload’* operation, we first checked that the deliverer had at least one pizza to deliver, and then we checked that the customer that was in the same position had at least one order left to be delivered.

* result(self, state, action):

This function receives a state and an action to be performed and returns the resulting state of applying the function.

For each action, it just generates a new state tuple preserving the data that hasn’t been modified and including the new updated data.

* is\_goal(self, state):

As its name suggests, it receives a state and checks if it’s equivalent to the defined final state. This final state has been defined as the state in which the deliverer finds itself in the start position without carrying any pizzas, and where all the customers have been satisfied (no orders left on any of them).

Since we defined the tuple representing this final state in the setup function, we just compared it with the state and returned the outcome.

* cost(self, state, action, state2):

asdasdasdasdasdasdasdasdasdasdasd

* heuristic(self, state):

asdasdasdasdasdasdasdasdasdasdasd

* setup(self):

This method is in charge of creating the initial state

* printState(self, state):
* getPendingRequests(self, state):