|  |
| --- |
| TITLE FOR PAPER 2 |

## Team

Team B:

* P4048819 - Daniel Boden
* P4261930 - Dean Duff
* P4048801 - Jo Humphreys

### Focus

A comparative analysis of two or more inference mechanisms, which are utilised to solve a “Car Park” puzzle, noting their efficiencies and shortcomings as well as comparing the results of the application of each algorithm to a variety of puzzles, within a wide range of sizes and complexities. From the aforementioned comparison, it may be possible to determine the ideal algorithm of the two for the puzzle format.

#### Inference Mechanisms:

* A\* Search - Provided by agent-domain (<http://s573859921.websitehome.co.uk/pub/clj/tools/Astar-search(2a).clj)>
* Planner Algorithm – Provided by agent-domain (<http://s573859921.websitehome.co.uk/pub/clj/tools/planner(1a).clj>)
* Additional standard Breadth-first and Depth-first search algorithms, time permitting, to support further analysis.

EXPLAIN WHY WE ARE USING THESE MECHANISMS HERE (WE ALREADY KNOW A\* IS IDEAL FOR THIS PROBLEM)

#### Performance Metrics:

* Time taken to find a solution that leads to the goal state.
* Steps involved in solution found (Efficiency).
* Possible RAM usage for each mechanism (Time permitting).

### Experimental Brief

The experiment consists of a slider type puzzle with the theme of a crowded car park, from which a vehicle must exit. However, the crowded nature of the car park has led to several other vehicles blocking the exit. These vehicles most be moved out of the way to allow the player vehicle to leave, but they are in turn obstructed themselves. Thus, the player must rearrange the car park with a number of constraints in mind in order to leave.

#### Netlogo:

* Rendering the puzzle board and its pieces in a manner that makes the position and direction of each car clear.
* Representing the various states that will be reached via each inference mechanism.
* Receipt of metrics recorded by the Clojure code, and displayed to the user, as well as output via BehaviorSpace that can then be passed into graphs for clear representation.

#### Clojure:

* Obtain the world state from the NetLogo model and store it in a format that can be easily modified.
  + Example data format: ((isa 2 car) (colour 2 yellow) (x 2 (3 4)) (y 2 (6)))
* Define the rules of the puzzle
  + Horizontal vehicles can only move left or right.
  + Vertical vehicles can only move up or down.
  + Vehicles cannot pass through one another.
* Record and return available metrics from the process of calculating a solution.

### Key features of experimental scenario

* The player vehicle has to leave via the exit.
  + Only one such exit exists.
* Vehicles will face one of the four cardinal directions on the board.
  + However, these vehicles remain locked in their given orientation. They can only move back and forth, and never sideways. In addition, the vehicles can never turn.
  + Vehicles also obstruct one another – no vehicle may pass through another like a ghost – further restricting each vehicle’s range of movement at any one time.
  + As such, the player vehicle will always line up with and face the exit.
* Vehicles can vary by length, based on the board’s grid, to a minimum of 2 grid squares, but not width (which remains at a size of 1 grid square).

#### Extending the problem:

* Most implementations of this puzzle begin with a board with no static obstacles. However, the introduction of such obstacles could potentially be implemented.
* Variable board sizes can also be considered, as a means to testing each algorithm against larger and smaller boards and comparing the results.
* Variation of car layouts within a specific sized board, which not only includes new positions but also variation in vehicle density, can also be considered.

### Inference Mechanisms

LOOK AT THIS AS A GROUP

* Create operators for the planner.
* Create tuples which describe our world states.

Describe briefly how each algorithm will be used.

### Workload

* Building the NetLogo representation of the world.
* Write Paper.
* Gather results from the metrics for each experiment.
* Implement A\* algorithm Clojure.
* Implement Planner algorithm in Clojure.