**CAB 320 – Assignment 1 Report**

**Nicole Barritt – 10157182 Mackenzie Wilson – 10155856**

Introduction

Planning agents are a frequently used concept in the artificial intelligence domain, being used for everything from resource allocations to activity scheduling (ngoc thanh nguyen).There are many questions surrounding the successful creation of an agent. Such as how the agent will observe the world, how the agent affects the world and what environment it is in. The purpose of this experiment is to successfully construct a planning agent and to optimize its functionality.

Experiment

The purpose of this planning agent is to assemble parts on a workbench, based on a given goal. The planning agent must select parts in the correct order, and then stack them to form the parts that are defined within the goal. The planning agent has to function in a few different scenarios. The factors that will change are if the parts can be rotated and if the agent can apply filtering to its actions. Within this experiment, filtering will be defined as checking to see if created part is contained within the goal state.

The first scenario is that the agent will not be allowed to rotate or apply filtering to the parts. The second scenario also does not allow rotation, but allows filtering. The third scenario is that rotation is allowed, but filtering is not. The fourth scenario is that rotation and filtering are both allowed. In addition to this, in the fourth scenario the agent will have a heuristic that it can use. A heuristic is a calculated value that can be used to estimate how far away a state is from a goal.

Results

The implementation of the agent included creating and solving the problem for each scenario. After implementation of the agent, the agent underwent testing to find the optimal search algorithm for each scenario. The search algorithm describes the methodology of how the agent will search through and act upon the actions that are available to it. Some common metrics that are used to measure the optimality of a search algorithm are the time and memory it takes to find a goal. The following table shows the results from the optimizing of the agent. The results for the first and second scenarios are from using the planning agent on the same assembly problem. The results for the third and fourth scenarios are from using the planning agent on the same assembly problem.

Table 1:

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario | Search Algorithm | Time to complete (s) | States created |
| 1 | Depth First Graph Search | 0.414 | 53 |
| 2 | Depth First Graph Search | 0.0264 | 2 |
| 3 | Depth First Graph Search |  |  |
| 4 | A\* Graph Search |  |  |

Discussion

Overall, the implementation of the agent was successful. Some initial areas of concern included representing the different actions and the method of iterating through all possible actions. The final implementation of the agent uses actions of different lengths to represent a part that is rotated and the joining of 2 parts together. After testing different methods of iterating through the actions, it was decided that *loops* would be used instead of the *itertools* class. The *itertools* classed caused a noticeable increase in run time.

For the first scenario, when no rotation or filtering is allowed, it was found that a depth first graph search was the most effective. A graph helps to prevent loops in the assembly path, by eliminating repeated assemblies from being built. This makes the agent more efficient by having less states created and taking less time to explore them all.

The second scenario also found that a depth first graph search was the most efficient, for the same reasons discussed above. Additionally, on the same problem, the second scenario was a whole magnitude faster than the first scenario. Scenario two also explores less actions, as it does not consider assembling parts that do not appear in the goal state. The improvements in efficiency was expected, due to the filtering that has been implemented.

Assembly Problem 3:

* Would make sense to need a graph here, because you do not want to continually rotate things as this can put you into a loop, and a graph will check to make sure that each state is one that you have not explored

Assembly Problem 4:

Conclusion