## Scenario and Default operations

Our scenario is an agent in a maze of rooms. Rooms are connected by doors and agents can open, close, lock, unlock the doors using different keys which can be found in the world. Agents can move around the maze by entering different rooms until they reach the end state.

## Goal State Size

### Overview of the test

In this experiment we are evaluating the efficiency, performance and results of both the ops-search and planner in respect to changes in quantity and complexity of the goal state. Before performing the analysis we suspect that the larger the goal state size and/or the more complicated the goal state, the more in-efficient both the ops-search and planner will be.

### Plan of how the analysis will be performed

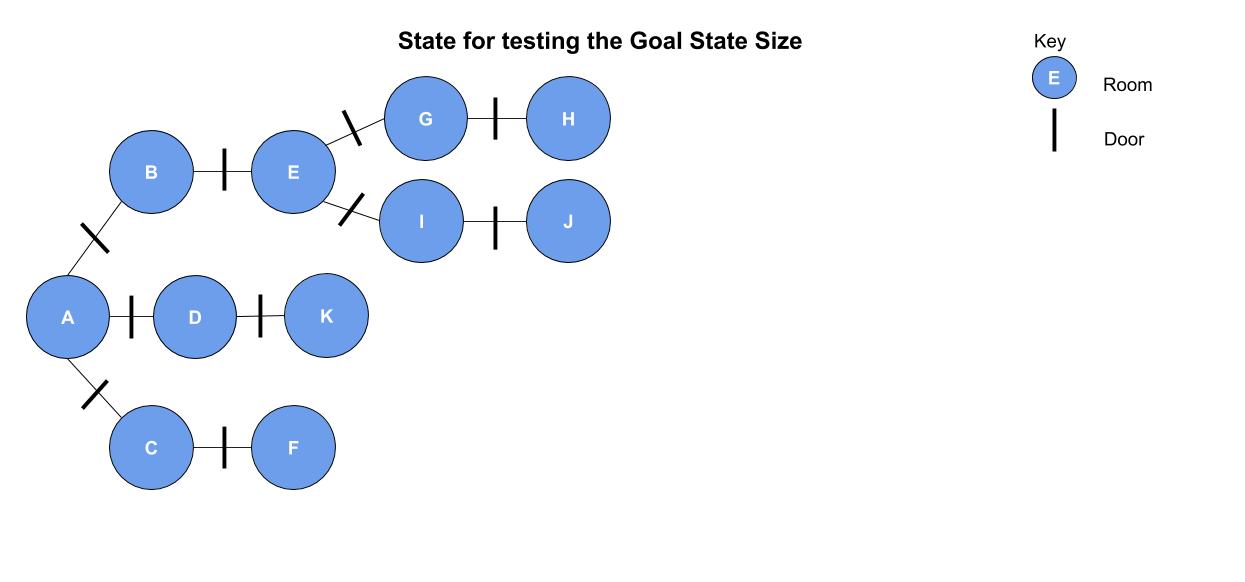
In the test I will be keeping the “world” state and operator functions the same but changing the quantity and complexity of the goal states that both ops-search and planner need to accomplish.

Independent variable - The size of the goal state

Dependant variable(s) -

1. The time taken to complete the execution of the function
2. Evaluation of the stack track outputted by the function
3. Final result of the function call
4. Memory consumption of the object

### Diagram of the State



### Graphs and Results

#### Overview of tests

I will run a series of tests for both ops-search and planner starting from basic operations and changes to the state to more complicated and advanced scenarios.

##### Test One (Basic Move) -

In the first test I will be moving the agent from “Room A” to “Room D” with all doors in the scenario being unlocked and open. The agent will only need to move from one room to the next.





##### Test Two (Basic Move with Close operation) -

In the second test I will be moving the agent from “Room A” to “Room D” with all doors in the scenario being unlocked and open. The agent will only need to move from one room to the next. The agent will then close the door “A-D”.





##### Test Three (Basic Move, Close and Lock operations) -

Next, I will be moving the agent from “Room A” to “Room D” with all doors in the scenario being unlocked and open. The agent will only need to move from one room to the next. The agent will then close the door “A-D” and lock it behind themselves. In this test there is a total of three operations being specified in the goal operations.





##### Test Four (Basic Move with Close operation) -

In the next test I will be moving the agent from “A” to “D”, closing the door, locking the door and dropping a key the agent is holding. In this test there are a total of four operations being specified in the goal states.





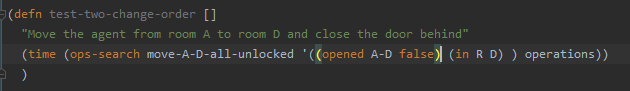
##### Test Five (Pickup, open, move, close lock, drop) -

Test five was executed and failed to output an answer after leaving it for over an hour.

***From the results above, I noticed that as more goal states are specified the longer ops-search takes to run and complete the task. I decided to test to see if the order of the goal states affected the time or ability to achieve a correct result***

##### 

##### Test Two, Changing the order of goal states

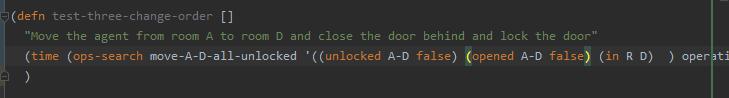


The goal states (open door and moving the agent) have been switched over to see if it affects the system achieving a result.





##### Test Three, Changing the order of the goal states



The goal states have been rearranged in a different order from the first test.





***The order of the states did not affect the results.***

***I then decided to test going to deeper depths and increasing the goal state size as well.***

***I introduced a new world state to try and make the efficiency of ops-search more efficient to speed up the process of the tests for the next set.***

##### Test Six (Increase depth distance with two operators)

The agent will open the door A-D and also open the room C-F. This is a depth of two.

 (76 seconds)



##### Test Seven (Increase depth distance, with three operators)

(198 secs)



### Analysis and Explanation of results

### From my analysis and experiments, I have concluded that the total time taken for ops-search to complete the operation is not directly dependant upon the number of goal states specified but is in fact relational to the number of intermediate operations it must complete.

### In tests one to five, where the agent engaged with two adjacent rooms. We can see that the more goal states that are added the longer the process takes to complete. In test five, it becomes almost impossible to solve as we left the process running for one hour and it was still incomplete.

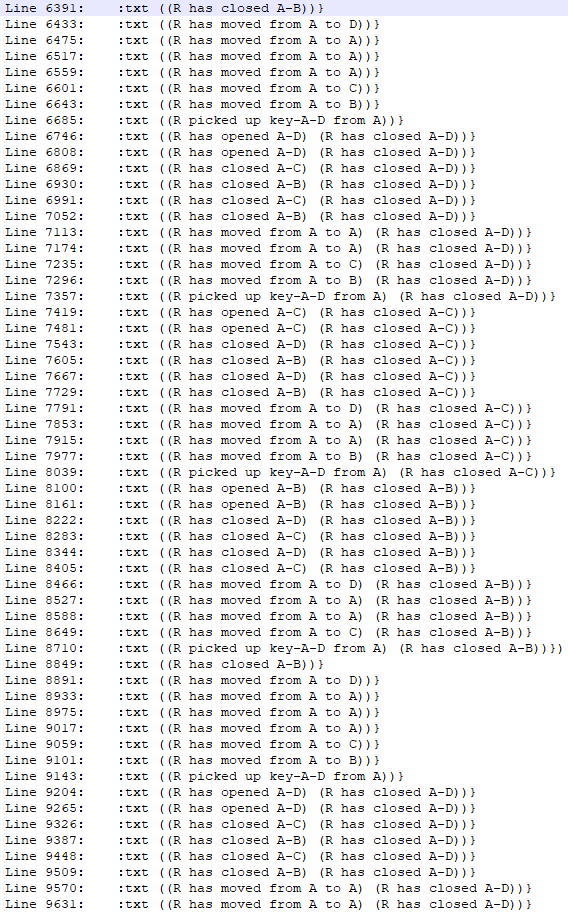
### However, in tests six and seven, we attempted to make the agent go to deeper depths and perform operations in rooms which where not directly next to each other. From this we could see that even though we specified less goal states, it took longer than before as more intermediate operations (operations that needed to happen in order to complete the overall objective) needed to be complete.

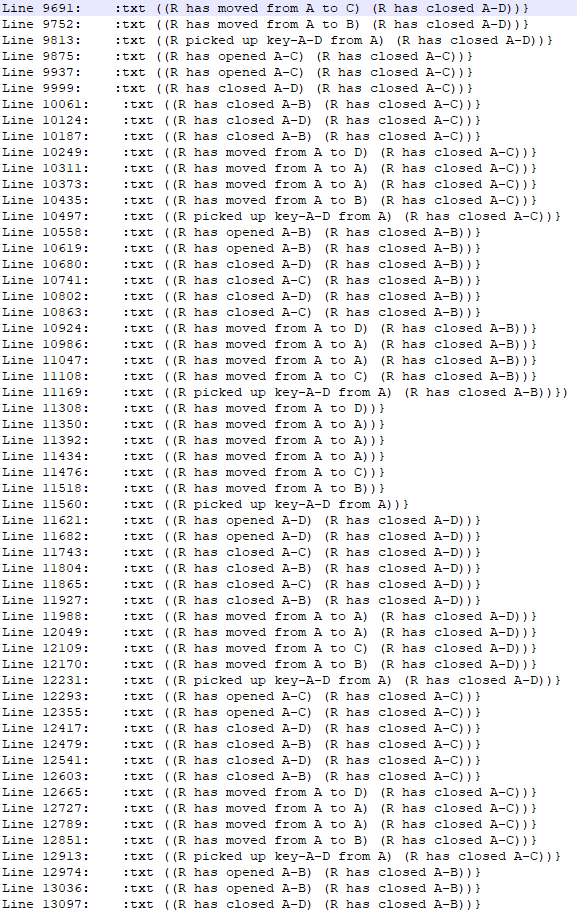
### ***The time complexity of a breadth first search for ops-search is 0(V+E) which is the number of vertices plus edges it must visit.***

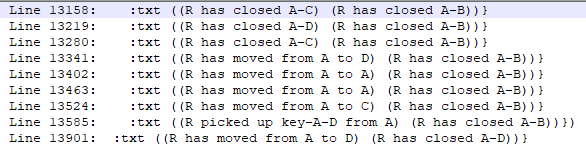
### Ops-search is a type of breadth first search, so it will start at a specified node and added all adjacent nodes to a queue and visit each of them. Ops-search will then look at the state it’s been given and compare the *:pre* of each operation for an action with the state. If the *:pre* operation is valid (the state and the *:pre* match) then the *:add* and *:del* operations for that action will be applied and the *:txt* operator printed out. If the goal state then contains the new state, the program will terminate and give an answer. Otherwise ops-search will continue applying and checking other operators for different nodes until it finds a solution or exhausts all options. By definition, the more operators it must perform the more nodes it will likely visit and the more matches it will have to apply inorder to get the final goal state.

As we can see from a selection of *:txt* outputs below, the ops-search will check every operator and apply the ones where the *:pre* matches the state regardless of if it is ‘logical’ to perform that operation. For example, in test-two, there is no need to try and pick up a key as the specified goal state is to move to a room where all the doors are already open. By performing these actions the time complexity of the request is greatly increased. Additionally, the ops-search engine ‘repeats’ operations that have already been tested which leads to unnecessary checks being performed and applied.

### 



****

****