# Mini-Project 2: Block World
# CS7637

Josh Adams
jadams334@gatech.edu

***Abstract*—**Mini-Project 2 Block World is based on an environment of blocks. The goal is to move the blocks around from the initial configuration to the goal configuration, while maintaining certain constraints, in as few as moves as possible.

## 1 AGENT DESIGN

The agent uses both generate and test as well as means-end analysis, this coupled with a priority queue allows the agent to find the minimum number of moves necessary to reach the end goal. The Pseudocode is provided below and further explanation after.

```
"""
Pseudocode: BlockWorldAgent.solve

def solve(self, initial_arrangement, goal_arrangement):
    Initialize 'Goal_State' using goal_arrangement
    Initialize 'Initial_State' using initial_arrangement
    Initialize 'States' Priority Queue to maintain the flow to lower delta states
    Initialize 'Visited_State' set, to track what states have been visited
    Add 'Initial_State' to the 'States' Priority Queue
    Solved:= False

    While the problem is not solved and the States Priority Queue is not empty:
        Current_State:= Get lowest delta state from 'States' Priority Queue
        Next_States:= Generate next possible states using Current_State
        Iterate over the returned Next_States:
            Add each 'temp_state' to the 'States' Priority Queue
            If 'temp_state' == 'Goal_State':
                Solved:= True
                return the variable within the 'temp_state' which tracks the moves used to reach that state

def generate_next_states(some_state):
    returned_states:= set()
    For 'temp_block' in some_state.moveable_blocks:
        Find 'all_locations' where 'temp_block' could move
        For 'location' in 'all_locations':
            If 'location' non moveable and 'temp_block' should be on top:
                Create New State with 'temp_block' at new location
                Add the new state to 'returned_states'
        If no valid moves were made and 'temp_block' is not currently located on the table:
            Create New State with 'temp_block' on the table
            Add the new state to 'returned_states'
    return returned_states


Pseudocode: State

class State:
    def __init__(all needed parameters):
        Set Current State
        Set Goal State
        Calculate current states delta
        Find blocks which can be moved
        Find blocks which cannot be moved

"""
```

The agent uses generate-test by taking all possible moves the agent could make and processes those to find their corresponding new states. Then, means-end analysis is used by comparing the deltas of the new states to determine which of

those possible new states reduce the differences between that state and the goal state. Figure 1 shows an example state in the block environment along with how the agent calculates the delta between the current state and the goal state.


***Figure 1*** — Example state in the block environment provided in the assignment instructions. Source.

I went through many variations of calculating delta, finally coming back to the most simplistic, which is shown in Figure 1. To calculate the delta in an environment, the agent iterates of each block in the current state and compares to the same block in the goal state. If they are essentially different then the delta is increased by one. If you look at the location of the "A" in the current state is located on the table and the "A" in the goal state is located on top of the "B" block, this increases the delta by one. The agent does this for each block. Once a block is in its final position, that block is not able to be moved any further.

## 2 AGENT RESULTS

### 2.1 Performance

The performance of the agent is what I would consider just average. The agent can solve any of the problems with any combination of blocks in less than a second. In my opinion the reason is due to the finite number of blocks being only twenty-six as well as a finite number of possible moves for each block being at most twenty-six. The agent does not explore all twenty-six moves, which also cuts down on the exploration. Given larger number of blocks the agent would slow down dramatically because the agents search space grows exponentially as

possible moves increase. I have not come across any case where the agent struggles.

### 2.2 Efficiency

The agent is moderately efficient for the given search space of the block world. I would expect considerable slow downs in performance with much larger number of blocks. Currently the max is twenty-six and can solve those in less than a second. The reason I am skeptical of the agent's efficiency is due to one of the not so 'clever' ways of finding new states. As the search space is increased it will not change the number of generated states the agent would return. It will increase the time to get those state because the number of moves possible the agent finds will increase. The agent will search through each of those possible moves, ultimately finding the optimal move.

## 3 AGENT COMPARED TO A HUMAN

Comparing the agent to a human in terms of performance, there really is no comparison. Given any problem the agent would finish faster than a human would, typically many orders of magnitude faster but this also depends on the complexity of the problem. When comparing the methods and agent and human use to solve, there are similarities and distinct differences. Figure 2 shows an example of a state in the block environment.


***Figure 2*** — Example image provided in the assignment instructions. Source.

A human would see that "D" should be on the table. They would then see that "C" needs to be above "D" but is currently under "B", so they would move "B" to the table. Then "C" on top of "D", then "B" on top of "C" and then "A" on top of "B". This would solve the problem, but the agent would solve it a little differently. The agent first would see that, only two blocks are able to be moved, "A" and "D". Block "A" could be moved above "D" or "D" could be moved on top of "A" or to the table. After finding the possible moves, it would then see if any of those moves bring it closer to the end goal by calculating the differences between the next state and the goal state. If any of those moves bring it closer, then it adds those to a priority queue. Ultimately solving the problem in the lowest number of moves, but also very quickly. Both the human and the agent determine the best move to make based on if the next state is getting closer to the end goal. Where they are different is that the agent keeps a queue of moves and will calculate differences for moves which a human may not think is sensible.

## 4 REFERENCES

1. Joyner, David A. "Mini-Project 2: Block World (Spring 2021)." *OMS CS7637*, lucylabs.gatech.edu/kbai/spring-2021/mini-project-2/.