CMPT 417 - Individual Project

Multi-Agent Path Finding

Fitzpatrick Laddaran

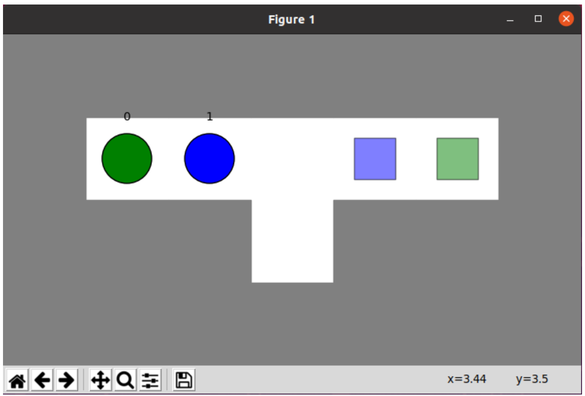
July 03 2022

1 Space-Time A*

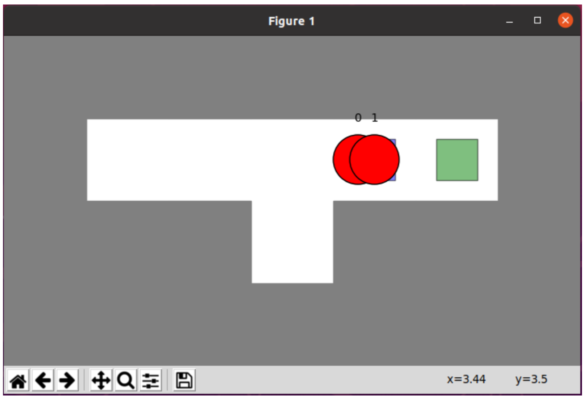
1.1 Searching in Space-Time Domain

These are the results after modifying `single_agent_planner.py` to support temporal constraints, pertaining to Section 1.1 of the lab.

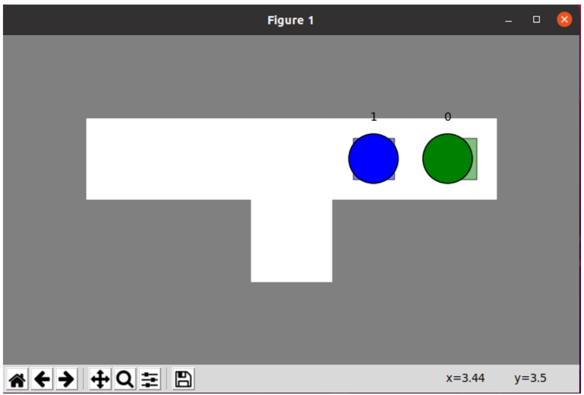
Figure 1 presents similar animation results.[1]



(a) Start condition.



(b) Agents collide.

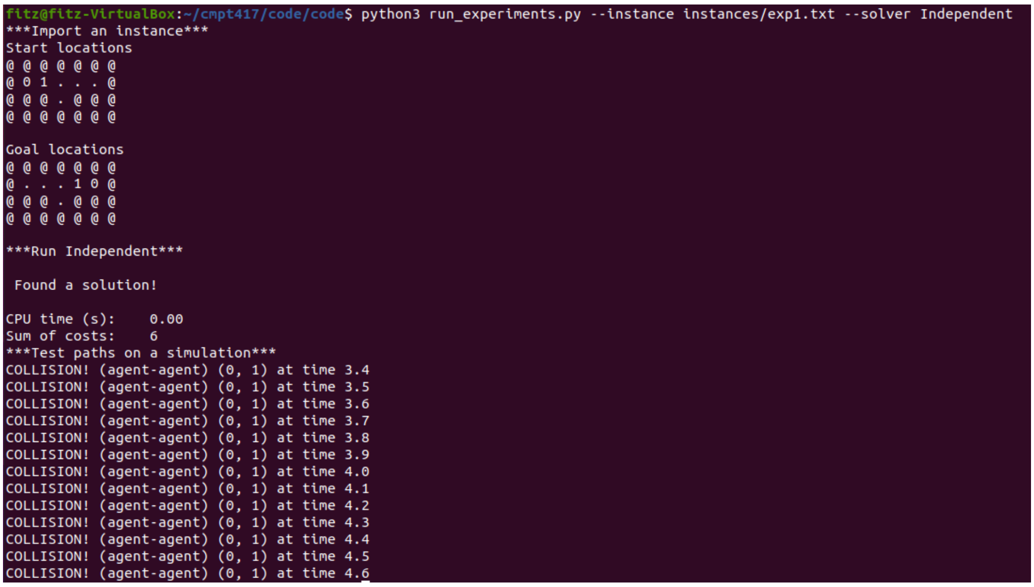


(c) End state.

Figure 1: Animation after modifying `single_agent_planner.py`.

[1] https://tex.stackexchange.com/questions/148438/putting-two-images-beside-each-other

Figure 2 presents the terminal output:



```
fitz@fitz-VirtualBox:~/cmpt417/code/code$ python3 run_experiments.py --instance instances/exp1.txt --solver Independent
***Import an instance***
Start locations
@ @ @ @ @ @ @
@ 0 1 . . . @
@ @ @ . @ @ @
@ @ @ @ @ @ @

Goal locations
@ @ @ @ @ @ @
@ . . . 1 0 @
@ @ @ . @ @ @
@ @ @ @ @ @ @

***Run Independent***

 Found a solution!

CPU time (s):    0.00
Sum of costs:    6
***Test paths on a simulation***
COLLISION! (agent-agent) (0, 1) at time 3.4
COLLISION! (agent-agent) (0, 1) at time 3.5
COLLISION! (agent-agent) (0, 1) at time 3.6
COLLISION! (agent-agent) (0, 1) at time 3.7
COLLISION! (agent-agent) (0, 1) at time 3.8
COLLISION! (agent-agent) (0, 1) at time 3.9
COLLISION! (agent-agent) (0, 1) at time 4.0
COLLISION! (agent-agent) (0, 1) at time 4.1
COLLISION! (agent-agent) (0, 1) at time 4.2
COLLISION! (agent-agent) (0, 1) at time 4.3
COLLISION! (agent-agent) (0, 1) at time 4.4
COLLISION! (agent-agent) (0, 1) at time 4.5
COLLISION! (agent-agent) (0, 1) at time 4.6
```

Figure 2: Terminal output after modifying `single_agent_planner.py`.

1.2 Handling Vertex Constraints

These are the results after modifying `single_agent_planner.py` to handle vertex constraints, pertaining to Section 1.2 of the lab.

Using the example constraint provided in the lab, Agent 0 stops before (1,5) at time step 4. Figure 3 presents the animation where Agent 0 stops.

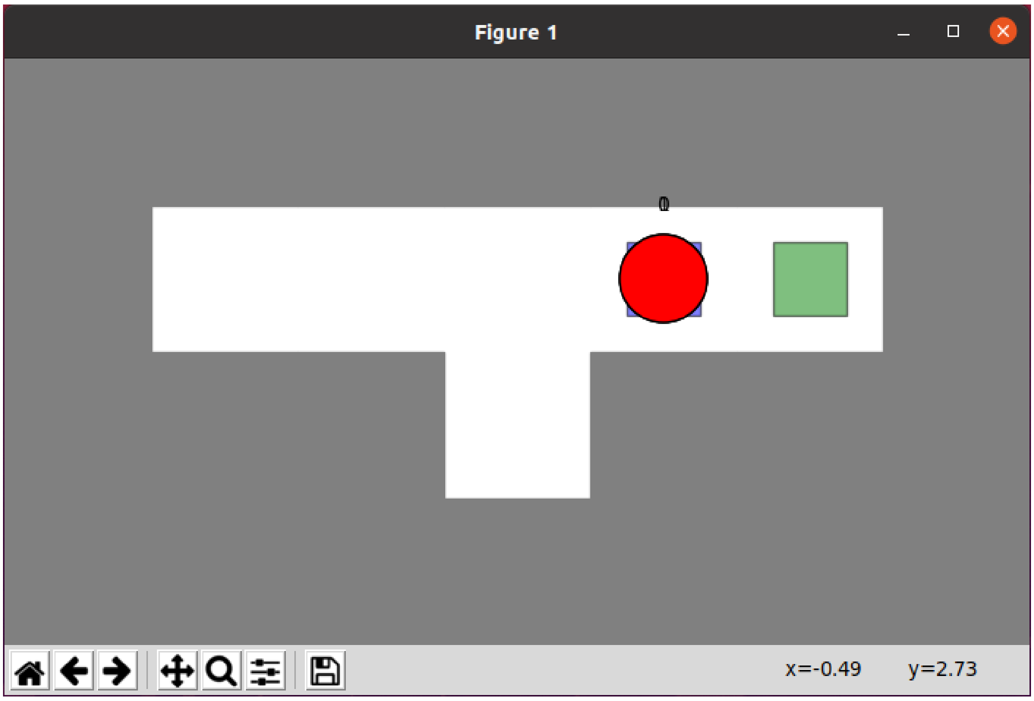


Figure 3: Agent 0 at time step 4 after adding vertex constraints.

Figure 4 shows the terminal output:

```
***Import an instance***
Start locations
@ @ @ @ @ @ @
@ 0 1 . . . @
@ @ @ . @ @ @
@ @ @ @ @ @ @

Goal locations
@ @ @ @ @ @ @
@ . . . 1 0 @
@ @ @ . @ @ @
@ @ @ @ @ @ @

***Run Prioritized***

 Found a solution!

CPU time (s):    0.00
Sum of costs:    7
[[(1, 1), (1, 2), (1, 3), (1, 4), (1, 4), (1, 5)], [(1, 2), (1, 3), (1, 4)]]
***Test paths on a simulation***
COLLISION! (agent-agent) (0, 1) at time 3.4
COLLISION! (agent-agent) (0, 1) at time 3.5
COLLISION! (agent-agent) (0, 1) at time 3.6
COLLISION! (agent-agent) (0, 1) at time 3.7
COLLISION! (agent-agent) (0, 1) at time 3.8
COLLISION! (agent-agent) (0, 1) at time 3.9
COLLISION! (agent-agent) (0, 1) at time 4.0
COLLISION! (agent-agent) (0, 1) at time 4.1
COLLISION! (agent-agent) (0, 1) at time 4.2
COLLISION! (agent-agent) (0, 1) at time 4.3
COLLISION! (agent-agent) (0, 1) at time 4.4
COLLISION! (agent-agent) (0, 1) at time 4.5
COLLISION! (agent-agent) (0, 1) at time 4.6
COLLISION! (agent-agent) (0, 1) at time 4.7
COLLISION! (agent-agent) (0, 1) at time 4.8
COLLISION! (agent-agent) (0, 1) at time 4.9
COLLISION! (agent-agent) (0, 1) at time 5.0
COLLISION! (agent-agent) (0, 1) at time 5.1
COLLISION! (agent-agent) (0, 1) at time 5.2
COLLISION! (agent-agent) (0, 1) at time 5.3
COLLISION! (agent-agent) (0, 1) at time 5.4
COLLISION! (agent-agent) (0, 1) at time 5.5
COLLISION! (agent-agent) (0, 1) at time 5.6
```

Figure 4: Terminal output after modifying `single_agent_planner.py` to handle vertex constraints.

1.3 Handing Goal Constraints

These are the results after modifying `single_agent_planner.py` to handle goal constraints, pertaining to Section 1.4 of the lab.

With the original goal condition and the constraint where Agent 0 is prohibited from being at its goal location at time step 10, the animation ends prior to time step 10. Therefore, I have modified the goal conditions such that:

- First check if there are constraints.
- If there are constraints, check if a solution is found. If found, only return the path once the current time step matches the time step of the constraint with the latest time step.
- If there are no constraints, check if a solution is found. If found, return the path.

The following shows the pseudo-code of the modifications:

```
if (number of constraints) > 0:
    if (current location = goal location & UNLOCK): return path
    if (current time step + 1 = latest constraint time step): UNLOCK
else:
    if (current location = goal location): return path
```

In the code implementation, variables `lock` and `n_constraints` have been created to keep track of the locking mechanism and number of constraints (respectively) as shown in the pseudo-code. Note that the constraints table have been ordered[2] starting from the constraints with the earliest time step to the latest time step; therefore, variable `latest constraint time step` is easily specified because it is at the end of the table.

In the condition where constraints exist, the second if-statement indicates `current time step + 1`. The `+ 1` stems from the fact that the path should be returned as soon as the constraint with the latest time step is met. Without it, an extra iteration of the overarching for-loop is conducted because the lock is only set to `UNLOCK` once the time step matches the time step of the constraint with the latest time step. Increasing the value would cause the animation to terminate prior to the goal constraint being met (at least visually), and having negative values would cause the animation to run longer than it needs to.

1.4 Designing Constraints

These are the results after modifying `single_agent_planner.py` to find collision-free paths with a minimal sum of path lengths, pertaining to Section 1.5 of the lab.

The set of constraints used to create a collision-free path between the two agents is as follows:

1. Agent 1 cannot be at (1,2) at time step 2.
2. Agent 1 cannot be at (1,3) at time step 2.
3. Agent 1 cannot be at (1,4) at time step 2.

This forces Agent 1 to navigate towards (2,3) at time step 2, letting Agent 0 pass through and avoiding any collisions. Figure 5 shows the animated solution where Agent 1 navigates towards (2,3):

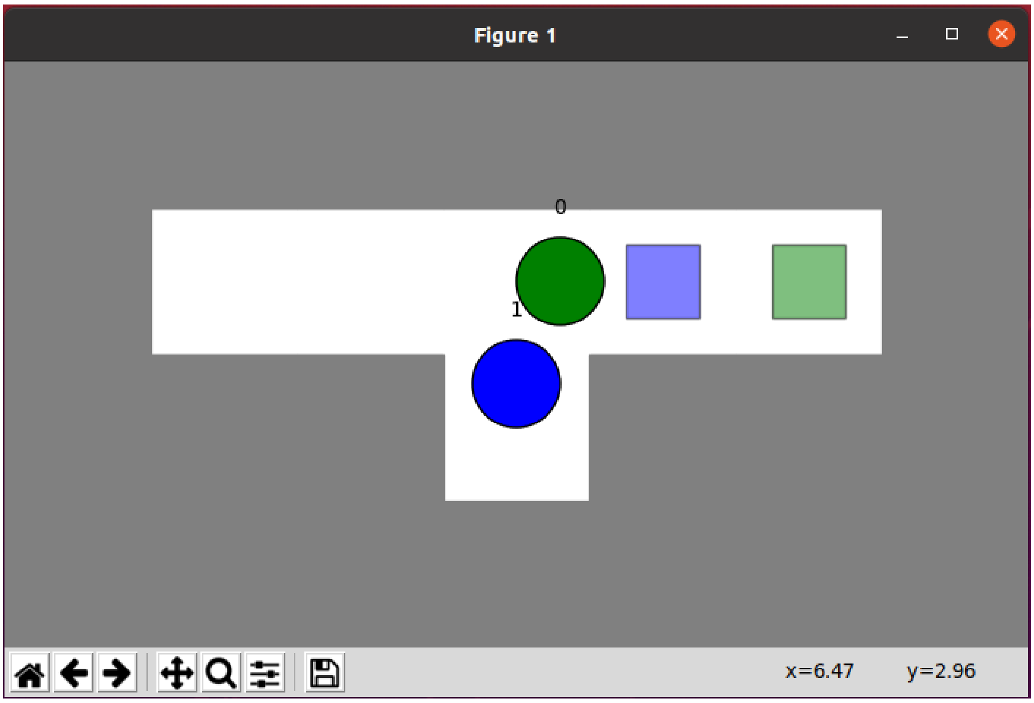
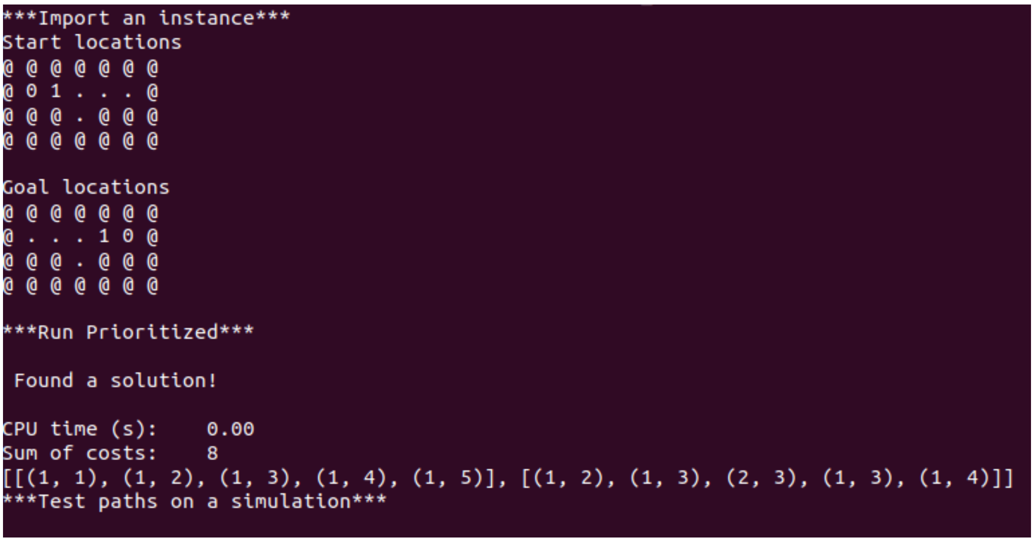


Figure 5: Agent 1 navigating towards (2,3).

[2]https://stackoverflow.com/questions/72899/how-do-i-sort-a-list-of-dictionaries-by-a-value-of-the-dictionary

Figure 6 shows the terminal output:



```
***Import an instance***
Start locations
@ @ @ @ @ @ @
@ 0 1 . . . @
@ @ @ . @ @ @
@ @ @ @ @ @ @

Goal locations
@ @ @ @ @ @ @
@ . . . 1 0 @
@ @ @ . @ @ @
@ @ @ @ @ @ @

***Run Prioritized***

 Found a solution!

CPU time (s):    0.00
Sum of costs:    8
[[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)], [(1, 2), (1, 3), (2, 3), (1, 3), (1, 4)]]
***Test paths on a simulation***
```

Figure 6: Terminal output after designing collision-free constraints.

.

From the terminal output, the (minimal) sum of path lengths is 8.

2 Prioritized Planning

2.1 Addressing Failures

These are the results after modifying `prioritized.py` to handle edge, vertex and goal constraints, pertaining to Section 2.4 of the lab. Note that the constraints are implemented according to the given instructions in Section 2.1 to Section 2.3.

Unlike in `exp2_1.txt` and `exp2_2.txt`, my solver does not terminate properly when running the experiment on instance `exp2_3.txt`. In fact, it reports that a solution is found when a collision between two agents occurs. When running the experiment, Agent 1 just waits on (1,3) as shown in Figure 7.

To fix this issue, I modified the code written (for Section 2.3) to handle the constraints to have an upper-bound. This upper-bound is strictly as follows:

```
if maximum time step < counter: return 'No Solution'
```

where `counter` is defined as the number of constraints created for other agents once a higher priority agent is at its goal location.

The upper-bound prevents the situation where a lower priority agent i will wait out the constraints where a higher priority agent k is at its goal location, and the only path for i to its goal location is through k's goal location. Note however, that the implementation is simplified. The upper-bound condition is set to return no solution despite a non-collision-free path being found.

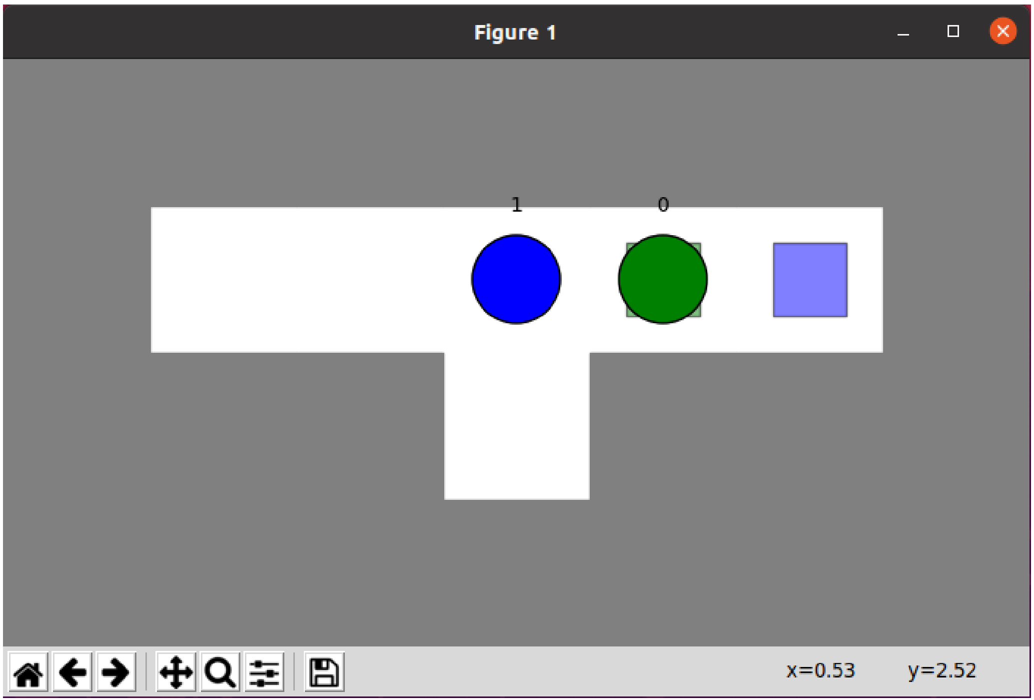
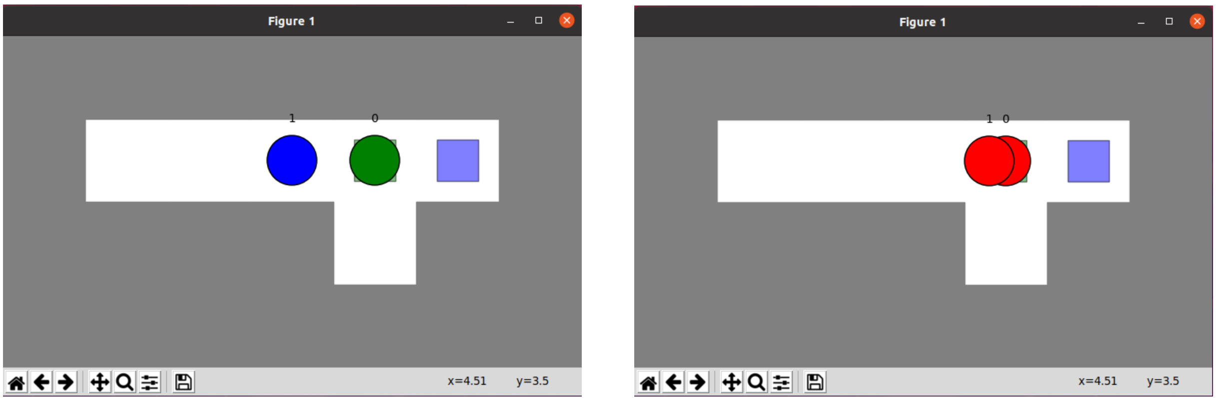


Figure 7: Agent 1 waits on (1,3).

2.2 Incompleteness and Suboptimality

This section discusses instances that fulfills the requirements as mentioned in Section 2.5 of the lab.

File `exp2_4.txt` is an instance for which prioritized planning does not find an optimal or non-optimal collision-free solution for a given ordering of agents. Without the upper-bound condition mentioned in Section 2.1 of the report, Agent 1 will simply wait out goal constraints because Agent 0 has a higher priority than Agent 1. This is seen in Figure 8a. In Figure 8b, we see the collision between the two agents when Agent 1 attempts to reach its goal location.



(a) Agent 1 waits.

(b) Agents collide.

Figure 8: Animations for instance `exp2_4.txt`.

Building on-top of `exp2_4.txt`, file `exp2_5.txt` is an instance where the ordering of the agents have been reversed. In this situation, both agents reach their goal as seen in Figure 9. This shows that there is an ordering of agents in which a collision-free solution is found despite having an ordering of agents in which no collision-free solution (based on `exp2_4.txt`) is found.

In conclusion, both instances provide the same environment; however, `exp2_4.txt` shows that there is no collision-free solution given some ordering. This satisfies the first requirement. `exp2_5.txt` reverses the priorities between the agents presented in `exp2_4.txt`, showing that there is a solution for a different ordering. This

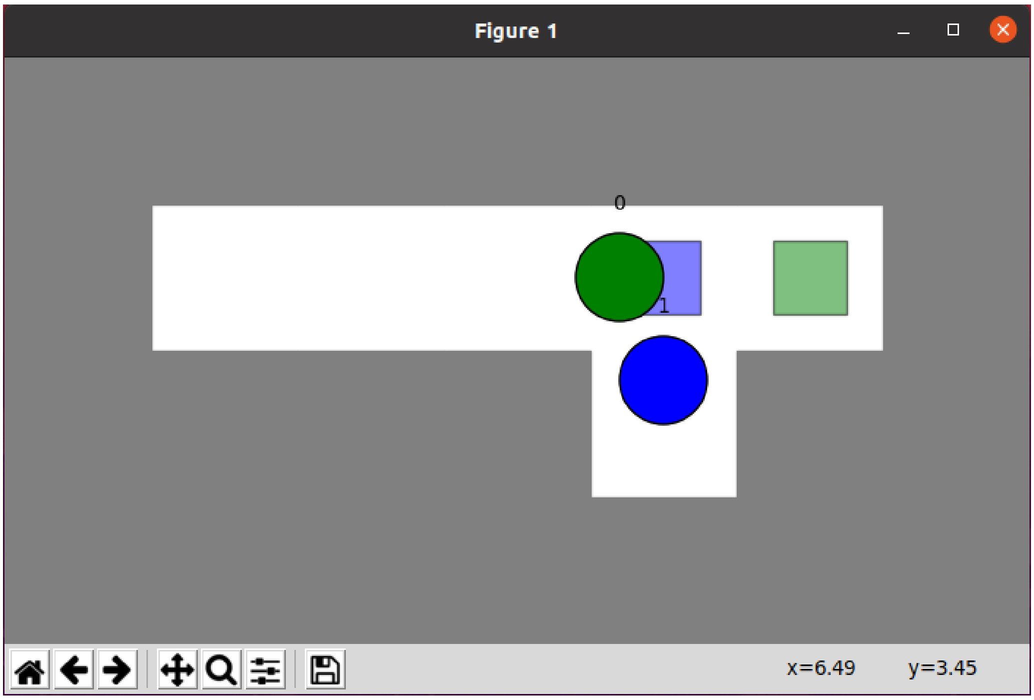


Figure 9: A solution is found in `exp2_5.txt`.

satisfies the bonus requirement. Both of these instances prove that Prioritized Planning is incomplete and suboptimal.

3 Conflict-Based Search

3.1 Implementing High Level Search

These are the results after modifying `cbs.py` to detect collisions, convert collisions to constraints, and to conduct a high-level search. This section pertains to Section 3.3 of the lab. Note that collision detection is implemented according to the instructions in Section 3.1, and the conversion of collisions to constraints is implemented according to the instructions in Section 3.2.

The minimum sum of costs that I receive (based on `results.csv`) compared to `min-sum-of-cost.csv` shows that my Conflict-Based search algorithm works as intended. The costs match for every instance.

Figure 10 is the (truncated) transcript of running `cbs.py` on `exp2_1.txt` with print statements on the nodes expanded.

3.2 Custom Instances

This section discusses the custom instances made for `cbs.py`, pertaining to Section 3 of the lab.

File `exp3_1.txt` is made to assess whether the correct information (for both Section 3.1 and Section 3.2) is returned when an edge collision occurs between two agents.

```
fitz@fitz-VirtualBox:~/cmpt417/code/code$ python3 run_experiments.py --instance instances/exp2_1.txt --solver CBS
***Import an instance***
Start locations
@ @ @ @ @ @ @
@ 0 1 . . . @
@ @ @ . @ @ @
@ @ @ @ @ @ @

Goal locations
@ @ @ @ @ @ @
@ . . . 1 0 @
@ @ @ . @ @ @
@ @ @ @ @ @ @

***Run CBS***
Expand node 0
Expand node 1
Expand node 2
Expand node 3
Expand node 6
Expand node 10
Expand node 11
Expand node 12
Expand node 13
Expand node 16
Expand node 20
Expand node 21
Expand node 22
Expand node 26
Expand node 27
Expand node 28

 Found a solution!

CPU time (s):    0.00
Sum of costs:    6
Expanded nodes:  16
Generated nodes: 31
***Test paths on a simulation***
```

Figure 10: Transcript of running Conflict-Based Search on `exp2_1.txt`.

4 Conflict-Based Search with Disjoint Splitting

4.1 Adjusting the High-Level Search

These are the results after modifying `cbs.py` to perform disjoint splitting, pertaining to Section 4.3 of the lab.

Based on my results, normal splitting expands 15 nodes while disjoint splitting consistently expands 9 nodes on `exp4.txt` (which nearly matches the results indicated in the lab). The run-time and number of expanded nodes on the `test_*` instances have also decreased (by observation).

5 Additional Notes

I have added libraries in `cbs.py`, namely: `import copy`. This was used to deep copy some objects.