*Armin Bazarjani*
*Csci360 Project 2*

# Task 1:

*Question 1:*
At time step 4 agent 0 is at node 11 and at time step 1 agent 1 stays at node 9.

Agent 0 path: 8 9 10 11 11 12
Agent 1 path: 9 9 10 11

*Question 2:*
I changed the goal state checker with a simple modification to allow our path finding algorithm to run for more than 10 timesteps (I made mine run for 13). Inside of the if statement to check for the goal state I added another checker that checks curr's timestep:

```
if (curr->location == goal_location && curr->timestep > 12)
```

Agent 0 path: 8 9 10 11 12 12 12 12 12 12 11 12 12 12
Agent 1 path: 10 11 11 11 11 11 11 11 11 11 11 11 11

As you can see from the paths at time step 10 Agent 0 moves from its goal state at node 12 and goes to node 11 for a single time step before moving back to node 12.

*Question 3:*
Constraints:
C1: Prevent agent 1 from moving to node 11 at t = 2 < 1, 11, -1, 2 >
C2: Prevent agent 1 from moving to node 9 at t = 2 < 1, 9, -1, 2 >
C3: Prevent agent 1 from staying at node 10 at t = 2 < 1, 10, -1, 2 >

Paths:
Agent 0: 8 9 10 11 12
Agent 1: 9 10 17 10 11
Sum of cost: 8

# Task 2:

*Question 1:*
Vertex Constraints: agent of lower priority can't be at the same location as an agent of higher priority at the same time that the agent of higher priority is there.

Edge Constraints: agent of lower priority can't move from x -> at time t when an agent of higher priority is moving from y -> x at the same time. This prevents an agent of lower priority from back tracking onto the higher priority agent.

The path:
a0: 8 9 10 11 12

a1: 9 10 11
Sum of cost: 6

As one can see agent 1 avoids agent 0 until it reaches its goal destination. This problem will be addressed in question 2.

*Question 2:*

I decided to make a constraint after the agent with priority reached its final timestep where I made the same vertex and edge constraints that I made before except I encoded them with a negative timestep. I then changed AStarPlanner to see if the timestep on a constraint is negative and the constraint applies to the agent, then it will fail the current path:

```
                        if(t < 0){
                            if(agent_id == ai && next_location == x){
                                fail_constraint = true;
                            }
                        }
```

I also added another constraint to the goal_test() for AStarPlanner:

```
if(curr->location == goal_location && curr->timestep > largest_timestep_PP){
                path = make_path(curr);
                break;
            }
```

And to make sure that largest_timestep_PP only increments iff our next_location is at the goal node:

```
    int largest_timestep_PP = 0;
    for(auto constraint : constraints){
        int ai = get<0>(constraint);
        int x = get<1>(constraint);
        int y = get<2>(constraint);
        int t = get<3>(constraint);

        if(ai == agent_id  && t > largest_timestep_PP && (goal_location == x ||
goal_location == y)){
            largest_timestep_PP = t;
        }
    }
```

Paths:

Exp2_1:
Agent 0: 8 9 10 11 12
Agent 1: 9 10 17 10 11
Sum of cost: 8

Exp2_2:
Agent 0: 9 10 11
Agent 1: 8 9 10 17 18 19 12
Sum of cost: 8

*Question 3:*

I had to change a few things around in order to reach the desired exit constraint. At first it wasn't terminating because my prioritized planner wasn't actually prioritizing properly. Meaning, I always treated agent 0 as having the highest priority. After this my program would go into an infinite loop where agent_0 got stuck in a location. In order to bypass this I set an exit constraint saying that if our next time interval is greater than a some arbitrary value then we should return an empty path, meaning no path could be found. I choose 1000 arbitrarily, I only did this because if a path couldn't be found after 1000 time steps then it is safe to assume there is no path.

```
                // if our next time step is too large, we are in an infinite loop and
should exit without finding a path
                if(next_t > 3000){
                    Path empty_path;
                    return empty_path;
                }
```

# Task 3:

*Question 1:*

Sum of Costs:

9,8,9,9,17,9,13,14,11,12,73,74,64,80,71,74,77,61,50,69,75,63,77,62,84,59,78,51,62,71,53,69,81,60,71,60,77,76,70,74,69,83,66,68,74,67,61,68,71,59

Milliseconds:

1.49,0.73,1.18,1.36,16.69,3.08,77.11,80.09,2.06,70.81,1.96,43.58,6.62,76.83,7.44,39.89,25.93,5.61,1.62,1.10,13.50,13.56,13.84,0.78,6.51,11.41,122.64,4.48,3.43,66.161.60,3.14,12.54,6.40,8.86,3.94,2.79,24.43,22.92,1.35,5.54,32.76,5.58,2.11,4.99,1.49,17.36,1010.14,15.87,12.39

*Question 2:*

Sum of Costs:

9,-1,-1,-1,17,-1,-1,-1,11,12,-1,-1,66,89,-1,78,82,68,55,69,78,73,79,75,92,63,82,52,70,79,57,71,98,64,76,74,82,79,70,79,76,85,68,74,76,75,69,70,85,71

Milliseconds:

0.39,7.86,44.15,0.087,0.27,103.18,19.42,9.22,0.55,0.45,188.31,92.61,5.80,20.88,436.1,12.11,17.05,7.10,3.77,5.57,12.75,13.71,12.60,12.09,14.65,8.35,17.94,3.02,7.78,13.99,4.94,6.70,24.50,6.24,8.34,15.16,18.25,11.23,8.43,6.95,11.54,7.98,11.61,8.40,6.80,8.60,17.28,8.56,19.00,16.10