Challenge Problem 1 Report

Andrew Swanson, Amy Heerten, Evan Boiko

For the first challenge problem of the class our group implemented python code that strategically moves robots in a simulated warehouse space from a randomly generated starting position in a grid to a goal location while following the prespecified constraints of the problem. The problem was approached by breaking it into four main components: the environment, the robots, the simulation process, and the visualization.

The environment is modeled as a grid that enforces spatial rules, tracks which cells are occupied, and manages updates as robots move. Robots are represented through an inheritance structure, with a generic base that defines common behaviors like movement and distance calculation, and specialized types that override collision and occupancy constraints. This separation ensures that adding new robot types or changing rules can be done with minimal disruption to the rest of the system.

The simulation proceeds in discrete timesteps. At each step, active robots propose moves toward their assigned goals using a greedy strategy that reduces their Manhattan distance. The environment evaluates these proposals, resolves conflicts based on distance-to-goal priorities, and applies random tie-breaking when necessary. Special handling allows two neighboring robots to swap positions directly, preventing unnecessary delays. Once a robot reaches its goal, it is removed from the grid to open space for others.

The visualization animates the process by showing the positions of robots, their goals, and their progress over time. A new timestep in the virtual environment is shown to the user every 3 seconds, simulating their movement. The system runs until all robots reach their targets, ensuring a complete solution.

The biggest challenge that our team faced was the implementation of the rule that allows robots of any kind to switch positions with each other if they are adjacent. Many of our initial simulations would solve if robots never needed to switch places with each other, but as soon as this was required the simulation would continue to run forever without updating the positions of the robots that needed to swap with each other. This bug happened because we thought that the swapping subroutine was only checking to see if the swap would be beneficial for both robots but in reality it was checking if the switch was allowed based on the other rules for robots occupying the same space. So, if for example a quadcopter wanted to swap with another quadcopter, the first quadcopter would check if it was allowed to move into the second quadcopter's spot before either quadcopter actually moved, preventing the operation from ever being carried out.

The switch part of the challenge problem had a few problem solving steps to it. At first the code for the switch was only being called to check once.This resulted in the robots not being switched as efficiently as possible. The update was to call to check every robot on the board that could switch. Then the switch would happen if it was beneficial. Then the update checks again to see if there should be another switch. The time step is over when the final switch check is completed. Another issue with the switch code was that we were creating "deadlocks" to check if the robot was allowed to switch. Often the robots were not allowed to switch due to the rules about occupying the same space. This resulted in a loop of infinite timesteps and the robots stuck in the same positions. The debugging process involved adding a counter to loops that would stop the program after so many times. The hard stop allowed us to review the timesteps and analyze what was causing the deadlock. Then the next step to debugging was to step through the code's different functions and analyze exactly what was being called and when it was being called.

Analyzing the timesteps revealed another inefficiency. The program was ending the timestep when one robot entered the goal state.This was an issue because multiple robots should have been able to enter the goal state in the same turn. This was causing more timesteps to be generated than necessary. The debugging solution was to end the timestep after all robots had done their moves then to generate a list of all the robots that made it to the goal state.

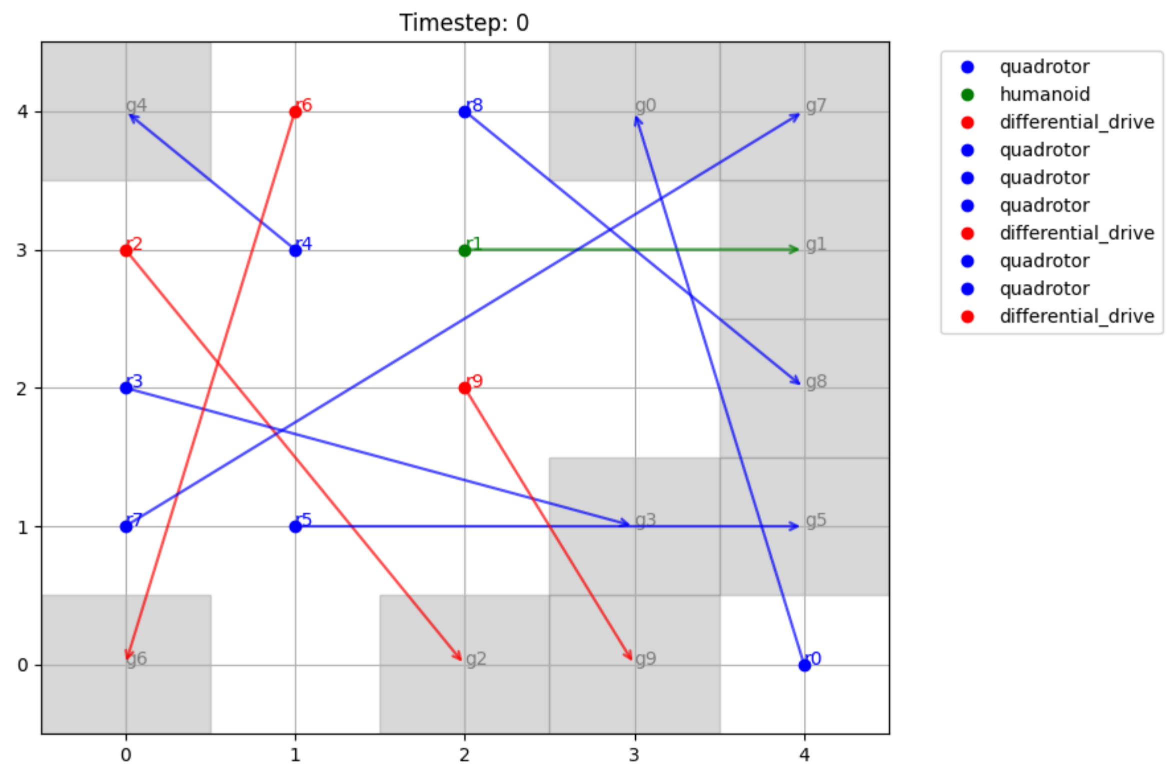

Figure One: Generate the robots and goals

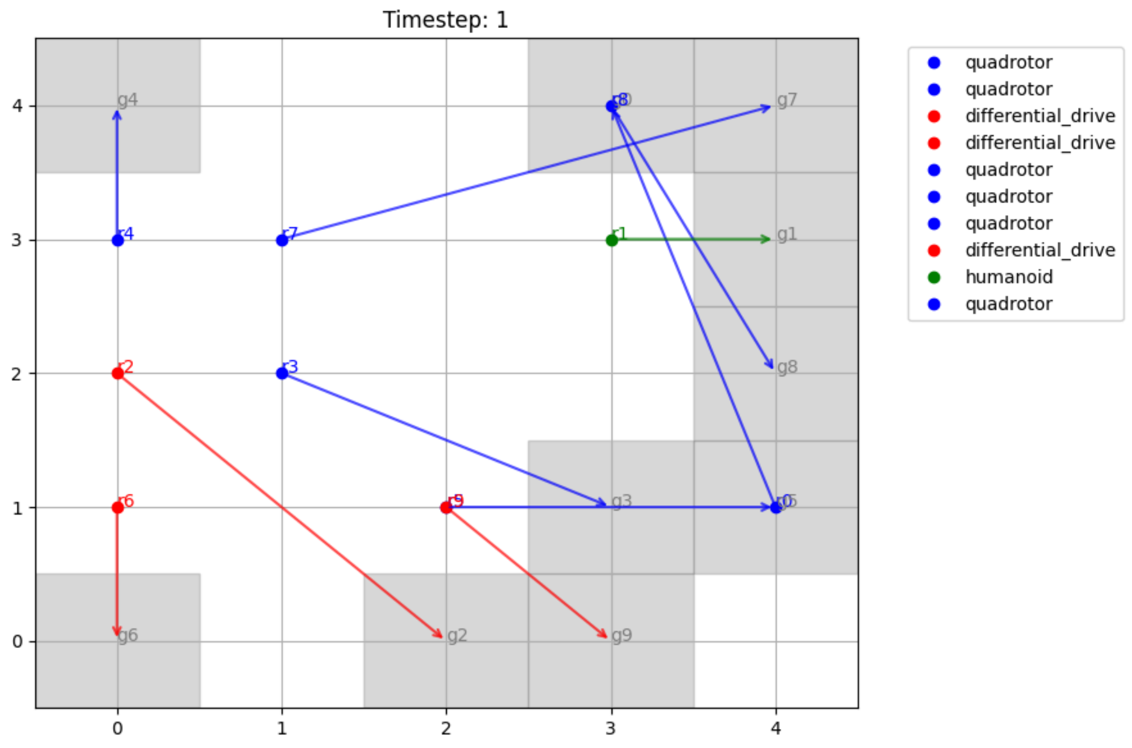

Figure Two: Robots move one step closer to goal

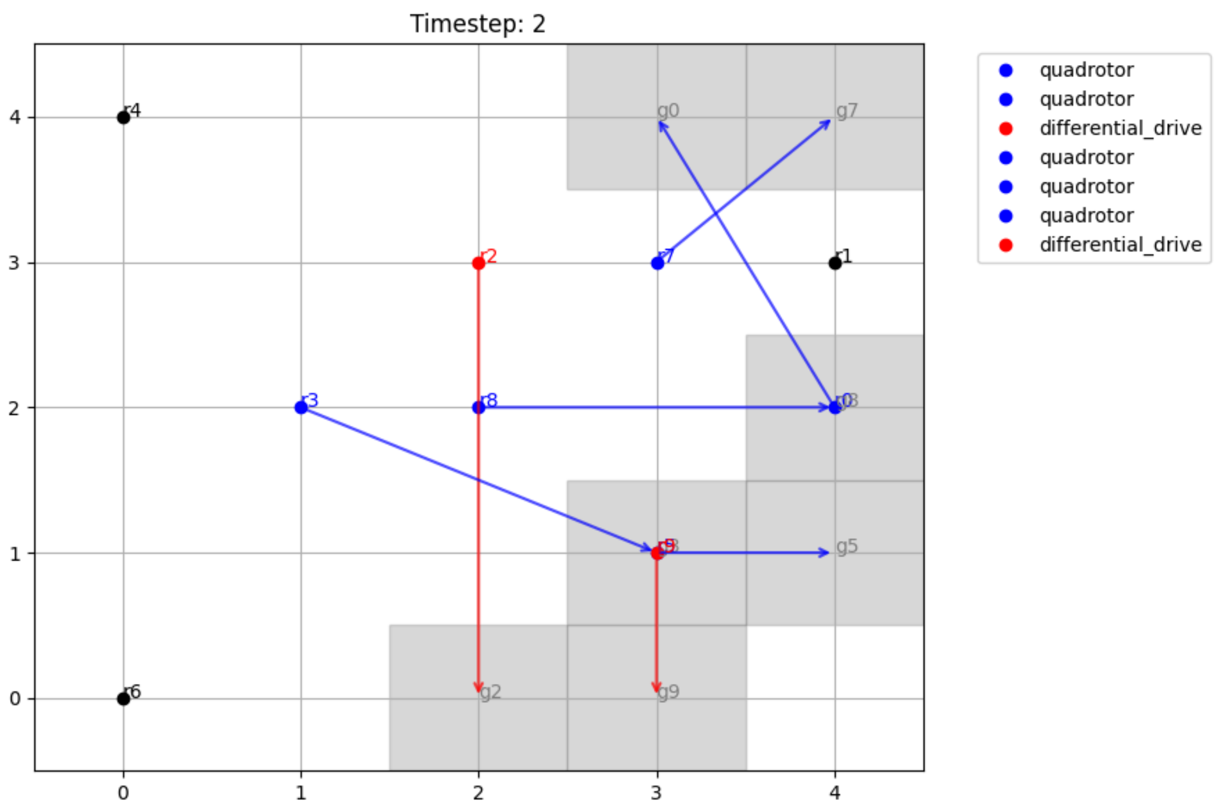

Figure Three: Robots have entered goal state on the same time step

Citations

Anthropic. (2024). *Claude 3.5 Sonnet* (June 20 version) [Large language model]. In GitHub Copilot, Microsoft