**Final Report**

The basis for the pathfinding was completed by Leon using an A\* search algorithm. This algorithm was run for every possible starting position with every goal, meaning that any start position simply has to query the database for the path it needs. This reduces the time required as running the A\* algorithm at runtime would timeout the system. Using this path (a list of positions that the agent will occupy), a set of direction is calculated and used in the actual agent to change positions during runtime. Note that during the running of the A\* algorithm collision detection is not used, and each is treated as a single agent system. Colin took the role of writing the collision detection that would act at runtime.

When creating the collision detection and avoidance, an iterative approach was used. Each agent is checked, and new instructions are checked for a collision. This is done by checking if two agents will be on the same tile at the end of this turn or if they swap positions. If either of those is the case, the agent will then attempt to avoid the collision. To be able to do this the agent must know the location of the other agents as well. This is done by having the agent query the database for the other agents’ path. This allows the agent to keep its simulations in lock step with the real actions and keep an accurate view of what is going on. The agent keeps track of positions, initial positions, and directional steps in 3 dictionaries labeled locations, paths and initials. This allows the agent to lookup by name instead of having to iterate through.

To avoid the collision, the agent will check what available actions are left once the action they were originally going to take is removed. They then check the Manhattan distance from the goal and chose the option with the lowest, essentially picking the second-best option for that location. The database is then queried to get the ideal path from that new starting point to the goal, updating the rest of the path to the goal. This allows the agent to move out of the way or pick a different path if their current action will result in a collision.

To maintain continuity over multiple runs a check is put in place to make sure no actions carry over. This is done in two ways. The first is by checking if all of the actions are complete. This will then restart the process to search for the new action set that is needed and is used when the process completes successfully. The second checks if the current game state is part of the set of actions that were given to the agent, if it is not then the next test has begun and it must again query the database. This second option is only used in the case of failure such as a deadlock but maintains that subsequent tests will not fail if the first does.