CS-171 Wumpus World Final AI Report

Team name **MelOna**

Member #1 (name/id) **Timothy Kim / 25625138** **Member #2 (name/id or N/A)** **N/A**

I. In about 1/2 page of text, describe what you did to make your Final AI agent "smart."
There are multiple strategies I used to make my Final AI agent "smart". My AI agent remembers known nodes through a map whether these nodes are visited or deduced from the positioning of potential nodes. Based on the known and potential nodes, it chooses the next node to move to, prioritizing non-visited nodes first. The agent stores the moves needed to get to a destination using a queue (step_queue) that is filled after the next node to go to is found. When there are no neighboring nodes that are safe unvisited nodes, it chooses the closest safe unvisited node in the model. The agent uses Uniform Cost Search to find a path to the destination node.
If there are no potential nodes that are safe, then the agent decides to go home and charts a path back home at (0,0). After it charts a path (vector of coordinate pairs), the step_queue is filled with moves to get to the destination. The agent only updates its maps and finds a new node to explore once the step_queue is empty.
My agent eliminates potential dangers based on what it knows. For example, if the agent is on a warning node (let's say breeze), then it should rightfully believe that the nodes next to it may be dangerous. However, if the agent knows all but one of the neighboring nodes of this warning node, then it can conclude that the unknown node does contain a pit. I solved this problem by checking each warning node in my known map to see if the agent knew all but one of its neighbors. If the agent did find such a node, the agent would clear the other nodes of that specific warning. I also modified my agent to account for nodes that are touching the side of the model bound or in the corner. For these coordinate nodes, the agent only needs to know 2 nodes for coordinates touching the side and 1 node for coordinates in the corner to deduce a value for the unknown neighboring node.
My agent shoots the arrow as soon as it senses a stench. If it perceives a scream, the agent will know that the Wumpus is dead and that all stench values in all maps should be removed. Additionally, the agent does not store any future stench or Wumpus values in any of the maps. It only needs to deal with breezes and pits.

II. In about 1/4 page of text, describe problems you encountered and how you solved them.
One of the first problems I had to solve was how I was going to choose the next node to explore. At first, my agent implemented a local search algorithm rather than a more decisive algorithm. It fared slightly better than the random AI. It could not go through known nodes, so when there was a "fork" between two nodes, it could not explore the other node. I solved this by giving the agent instructions to fill the step_queue with moves towards the nearest unexplored node when there were no more neighboring unexplored nodes. If there were no more unexplored nodes, the agent would fill the step_queue with moves towards (0,0).
Storing data was bit a problem. How can the agent store breeze, stench, pit and Wumpus in the map? I first started to store a coordinate node's value in a map as an array, storing ones in each index of the array starting with index 0 being a breeze, 1 being a stench, 2 being a pit, 3 being a Wumpus. However, as suspected, looking through the map was highly inefficient. To easily interpret stored values for each coordinate node in the known/potential maps, I decided to express each percept as binary converted into an integer. Breeze is 1, stench is 2, pit is 4, and Wumpus is 8. That way if any of these percepts are combined, I can easily decipher which percept is present or not.

III. In about 1/4 page of text, provide suggestions for improving this project.
It's a bit hard to provide any more suggestions to a project that's been improved over many years. I will suggest some improvements that could help students understand how to better apply the algorithms and theorems learned in class.
While trying to implement the breadth first search algorithm in trying to get from one coordinate node to another, I had a hard time trying to find the actual path between the root node and the destination node. I believe that once we learn a way to trace the optimal path between two nodes with one algorithm, we can apply that same coordinate node searching algorithm to all the other search algorithms that we learned.

For improvements on my project submission, I believe there are some major issues that affect my score. First, my agent eliminates certain nodes by deductive reasoning. However, the agent only goes through the maps once. Therefore, other nodes that could be modified using the previously modified nodes are not modified. A new fix could implement a recursive function that continues to deduce node values.