3547 Assignment 2

This assignment is to build a planning module for your agent that will allow it to get back to the lower left-hand corner with the gold in as few steps as possible once it's found it.

We'll use a slightly improved version of your naïve agent and let it wander around until it occasionally finds itself in the room with the gold.

We know once the gold has been grabbed the agent could retrace its steps back to (1,1) but it may have wandered a lot before it found the gold and every action costs a point. A more direct route is desirable.

We can describe the structure of the grid using a graph with nodes for squares and arcs/edges to denote which squares are connected to which neighbour squares. If we only include nodes we know to be safe for our agent, we can use a graph search algorithm to find the shortest safe path from the gold room back to (1,1). (More precisely, it's the shortest to the best of the agent's knowledge. There could be a shorter path back, but the agent won't know whether it's safe to traverse if it hasn't explored it).

The agent knows any square it's visited previously is safe (because it didn't die when it moved there previously). In the next assignment we'll give our agent the ability to reason about how likely an unvisited square is to be safe, which will make it more likely to survive long enough to find the gold, and possibly notice an even quicker route home occasionally.

Instructions

1. Create a copy of the NaïveAgent called BeelineAgent (because it will make a "beeline" for home). Remove the Grab and Climb actions from the list of random actions and add code to the agent so it only does a Grab when it first senses a glitter and a Climb if it is in (1,1) and has the gold. You'll need to add state to the agent so that it knows it's holding the gold when it has it. Change the main loop of the program to use your new agent.
2. Add state to your agent so it keeps track of where it's been and which squares it considers to be safe. This can be done either by adding mutable variables to its class, or functional-style by creating a new altered agent on each state change (like the Scala example code does by returning a new copy of the environment on each move).
3. You can use the Python NetworkX library to build a graph of safe locations once the agent has grabbed the gold. You should use a directed graph with the arcs flowing from the gold room toward (1,1) and either use one of NetworkX's built-in shortest path algorithms or write your own if you'd prefer. If you use A* you can use either Euclidean distance or Manhattan distance as an admissible heuristic. Alternatively, you can write your own search (breadth-first would work well here). If you want to get fancy, take into account the number of turns the agent must make along each path when choosing between them (but given the shape and size of the grid it'll rarely make a difference and then only of a couple of points at most).
4. The agent should build the graph and run the search when it first grabs the gold, then follow the path the search generates (the "escape plan") to finish the game as quickly as possible.