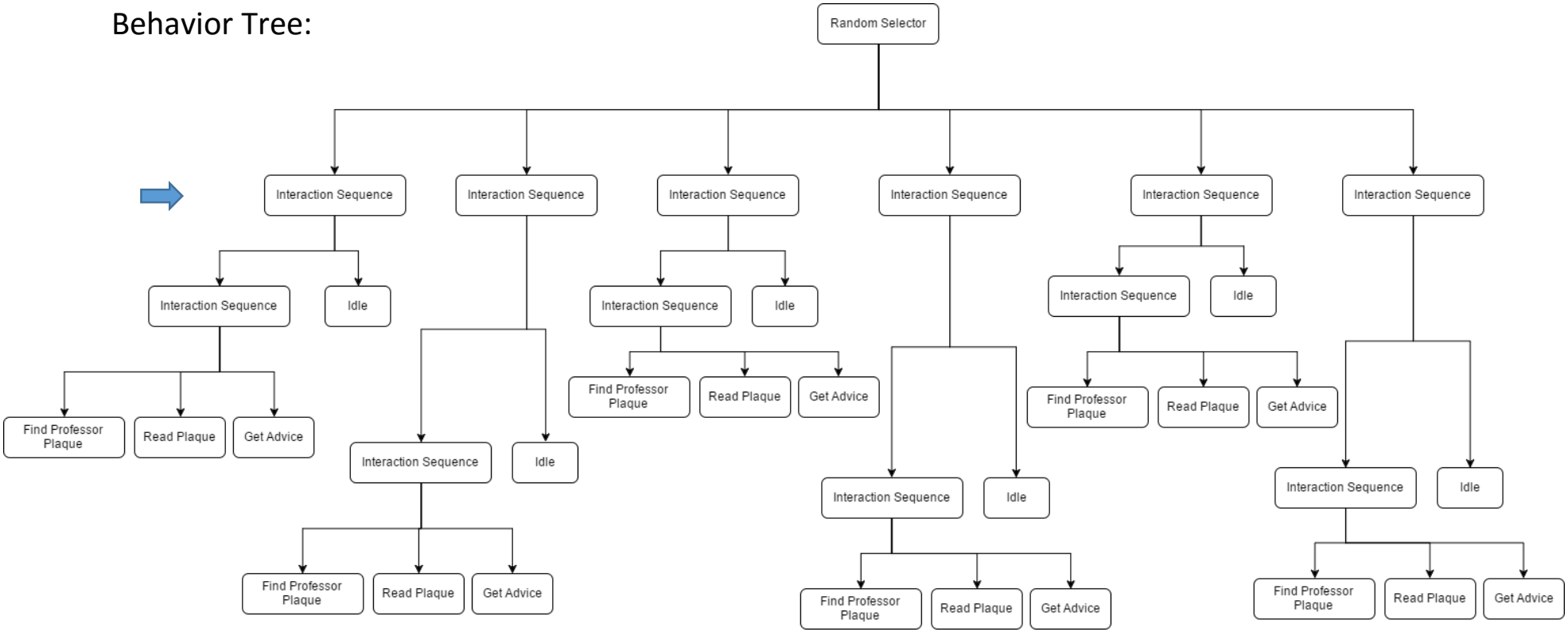
Behavior Tree:



Explanation:

The way the tree has been organized is in a way that each of the interaction sequences on the second level of the tree, the level marked with a blue arrow, represent one professor. Such implies that the children of each of those interaction sequences also belong to the professor their parent does. Given that, the random selector randomly selects one of its children and starts processing it.

Brief explanation on how behavior nodes work in my tree:

All behavior nodes have a function called initialize and a function called process. Initialize is called once, and behaves differently based on the type of node, while process is called on every tick of the game and it also behaves differently based on the type of node.

On any composite nodes, initialize attempts to initialize the node's children. For selectors, it initializes them all, and for sequence nodes, it initializes them one after the other, given the previous succeeds.

On leaf nodes, the initialize function simply initiates the action that the leaf node is supposed to carry. For example, for the "find professor plaque" leaf, the actual request to find a path to the plaque is sent in the node's initialize function.

For the process function, since these are called every frame, the process of composite nodes is essentially passed down the tree to a call to process the child in question. For the process function of leaves however, they are each tweaked differently such that the leaf returns running if it's busy doing something such as calculating a path, moving to some target, or agent being paused briefly to simulate some behavior. Once that is done, the result of the leaf is submitted as either SUCCESS or FAIL and sent back up to the parent.

Interaction Sequence: composite node that is very similar to a normal sequence node except that it sends the initialize command to a child once the previous child has succeeded.

Idle: leaf node that in 50% of the time it returns success without doing anything, i.e. skipped, and in the other 50% of the time it sends a path finding request to the agent to a random location on the level and returns "RUNNING" while the agent is not at its random target. Once the agent reaches the random idle location, idle will return "SUCCESS".

Find Professor Plaque: leaf node that makes the agent send a path finding request to the path request manager, for a path from the current location of the agent, to the professor's plaque that the node represents. This node directly skips if the professor it represents is in the memory of the agent.

Read Plaque: this node sends the agent to a very short delay once it's in front of the plaque, to simulate the behavior of reading. However, it sends back FAIL as result if the professor it represents is not the target of the agent. Whether this node is reached by actually pathfinding to the plaque, or skipping the path finding to the plaque because the agent knows which professor it represents,

this node has the same effect. It sends back SUCCESS as result if the professor is the same as the target of the agent, and returns RUNNING during that short period of delay that it has.

Get Advice: if this node is reached from the parent interaction sequence, it means that the professor was the target of the agent. Given that, it sends a path finding request from the agent's current location to the professor's location inside his office. Once the destination is reached, it pauses for a brief second to simulate the process of actually receiving advice from the professor, it sets the target professor of agent to another random professor, different from the previous target, and the node succeeds.

Preventing head to head collisions in cooperative pathfinding:

Although the implementation of pathfinding in my application is not perfect and has many arguable flaws, the idea was to prevent head to head collisions by attempting to reserve a cell at time t as well as time t + 1, whenever an agent reserves a cell, at any given time, t. If the number of students is decreased and targets given to the students in a way to simulate such a collision is a smaller scale, it can be seen that this indeed works as one of the agents will always attempt to path-find around the other agent before such a collision takes place. However, due to "lerping" an agents position through its path's waypoints, it seems like this is not always effective, and more specifically in cases where there could be diagonal collision, the agents just move past each other because the cross nodes are actually open but the "lerping "makes it look like a collision took place. The effectiveness of this strategy could also be noted with the actual reservation windowing frame. It seems like such a strategy is more effective with smaller reservation windows so that more A* re-pathing calculations are made throughout the life of a path to be taken by an agent.