Exercise 6: Teaching your Computer to Kill

Overview

In this assignment, you'll add one final feature to the tanks game: enemy AI. You'll implement a version of behavior trees based loosely on Chris Hecker's liner notes for Spore (good read, but optional).

Note: Since the game controllers turned out to be a pain, this version of the code is configured to use keyboard control for the player:

- **Arrow** keys: movement
- **Space**: shoot
- **Escape**: use powerup

Starting out

Start by reading over the code in the Assert/CodeAI/Behavior Tree folder. We've built the base class and rough hierarchy for the behavior tree system, but haven't implemented most of the methods.

The current system has stubs for most of the classes. You'll fill in the stubs to build out the system.

Implementing GroupDeciders

The GroupDecider class chooses between child nodes by calling their Decide, Activate, Deactivate, and Tick methods. For this, you just need to fill in the SelectChild and Tick methods. SelectChild chooses what child to run during a given Tick, while the Tick method calls SelectChild, and then Activates, Deactivates, and Ticks children accordingly.

SelectChild is written to allow different policies for selecting children, but we'll only be using the PrioritizedList policy for this assignment. So that's all you have to fill in. The basic idea of the prioritized list policy is to find the first child whose Decide function returns true. However, there are a couple of subtleties:

- If a child was selected on the previous frame, you always assume it wants to continue running; you don't call its Decide function again. It will be assumed to want to continue to run until some other child is selected or until its Tick method returns false
- If no child wants to run, then SelectChild should return null.

The Tick method just needs to call SelectChild to find out what child is selected for this tick, and call its Tick method. However, again, there are some subtleties:

- If this tick's selected child is different from last tick's selected child, you need to deactivate the one from last tick and then activate the one from this tick
- If the selected child's Tick method returns false, it doesn't want to run anymore and so it needs to be deselected.
- If there is no selected child (SelectChild returns null), then the GroupDecider's Tick method needs to return false to tell its parent it doesn't want to run anymore.

Note: it's important that you keep the child that you've selected inside the field called selected, since the debugging code assumes it will be stored there.

Implementing behavior nodes

Now we can implement the behaviors to control the tank. Fill in the appropriate methods of the following classes:

- MoveTowardPlayer
  This behavior should drive in a straight line toward the player. You can call the tank's MoveTowards(point) method to steer and move toward a given point.

- Flee
  This implements running away. Write this class so that when it is activated, it chooses a random point to drive to (the goal), and then drives toward it in a straight line. One subtlety:

  - Since it's going to drive in a straight line, you need to be sure to pick a random point that it can reach with a straight-line path.

- PathToPlayer
  This implements driving toward the player when they're not in sight (if they're in sight, we use MoveTowardPlayer instead). When the behavior starts, it should call Waypoint.FindPath to compute a path to the player. Then it should drive toward the first waypoint in the path, removing the waypoints as the tank reaches them. Tick should return false when there are no waypoints left.

  Note: we have included an OnDrawGizmos method to visualize the path in the field currentPath. So you want to use that field to hold the path you. The "gizmos" button in the unity editor will toggle the display of the visualization.

Improving the path planner

The path planner we gave you just does breadth-first search; that is, it ignores the length of edges. So it doesn't come up with least-cost paths. Moreover, there aren't that many waypoints. So the AI chooses pretty bad paths. Fix this by:

- Adding some more waypoints to the level
- Rewriting the path planner (found in Assets/Code/AI/Waypoint.cs) to find true shortest paths. You should do this using the A* algorithm. To do this, you'll need to have a priority queue implementation. You should feel free to grab a priority queue implementation from off the net if you like. But you must write your own A* implementation.