Assignment 3 Report

Che Chang
The University of Utah
che.chang@utah.edu

ABSTRACT
This is the write-up for assignment 3, which covers my implementation of decision trees and behavior trees.

KEYWORDS
Behavior trees, decision trees, Game AI decision making

INTRODUCTION
In this write-up, we will walk through my implementation of decision trees and behavior trees, and also introduce the DT nodes and BT nodes I implemented. We will also see how these nodes could be applied to our boids.

IMPLEMENTATION
Decision Tree
My decision tree implementation is a binary decision tree, each tree node could go either the TRUE path or FALSE path, and we traverse the decision tree recursively.

Types of Decision Tree Nodes
There are various types of tree nodes, but in our implementation we limited the type of nodes to the following:

Decision Nodes
These kinds of nodes have no actions bound to it, their only job is to encapsulate a condition check and connect to TRUE path or FALSE path. I implemented the following decision node:

1. Float decision nodes: these kinds of nodes encapsulate a value check (e.g. `value >= 20.0f`).

Action Nodes
These kinds of nodes have actions bound to it, so they can be connected to decision nodes' TRUE path or FALSE path. They can also be implemented in a way that combines decision nodes and actions together. Please see the following implementations:

1. Bool action nodes: these kinds of nodes encapsulate a boolean check, if the check evaluates to TRUE, then it cycles back to the node itself and continues running the action. If the check evaluates to FALSE then traverse to the node connected to FALSE.

2. Timeout action nodes: these kinds of nodes have member timers. Before the timer counts down to zero, the node cycles to itself and keeps executing the action. After time out the node then traverses to whatever node connected to it.

Behavior Tree
My behavior tree implementation is some basic nodes that allows a sequence of actions to be performed, and these nodes are traversed iteratively.

Types of Behavior Tree Nodes
The Unreal Engine AI system has a ton of behavior tree node implementation and a black board structure aiding the inter-node or inter-tree communication. However, here we have the most basic node implementations and no black board structure, all the information needed is hardcoded into each node data structure. Please see the following:

Task Nodes
These kinds of nodes represent the action performed in a behavior tree, and unlike my decision tree implementation using `std::function` to store actions, the actions are predefined as we code a task node, so essentially a user would need to pick from a predefined set of task nodes to form a behavior tree (e.g. BTPathFind, BTEnemyNear, BTPatrol, etc.)

Sequence Nodes
These kinds of nodes encapsulate a list of child nodes, could be decorators or simply tasks. A sequence node runs its children, if one child fails, the whole sequence fails.

Selector Nodes
These kinds of nodes, just like sequence nodes, also encapsulate a list of child nodes. However, a selector node runs its children, and even if one of the child fails, the selector node would continue running the next child, and the selector node fails only if all children failed.

Decorator Nodes
These kinds of nodes describe how its child should be run, and it only has one child. For example, I implemented a kind of decorator node called “BTUntilFail” which would continuously run its child until the child fails.

THE GAME
Screenshot of the Game Map

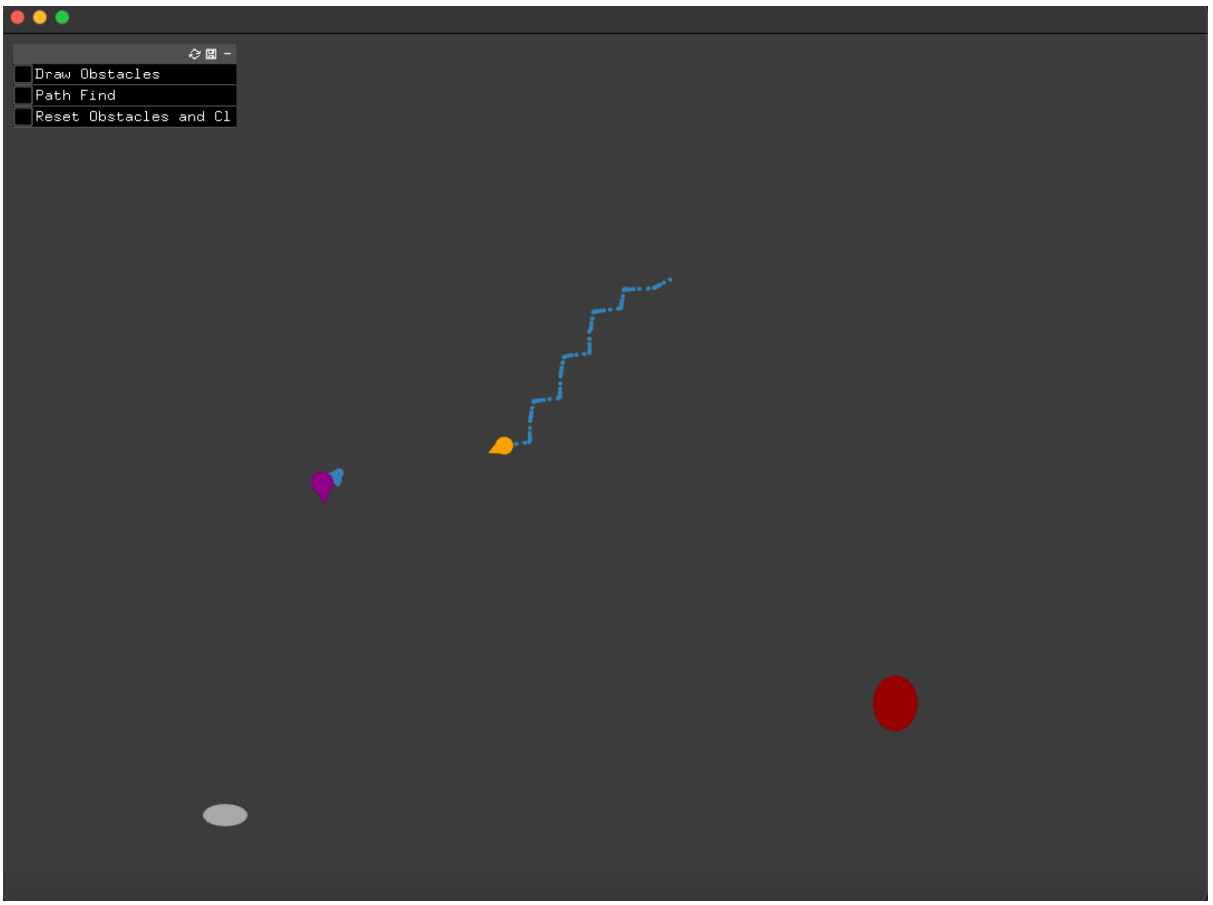

Figure 1: Game map (Purple boid is the monster, orange boid is the character, red eclipse is the shelter, and the gray eclipse is the flag)

Decision Tree Diagram

The orange character boid is controlled by the decision tree, and the following is the decision tree diagram:

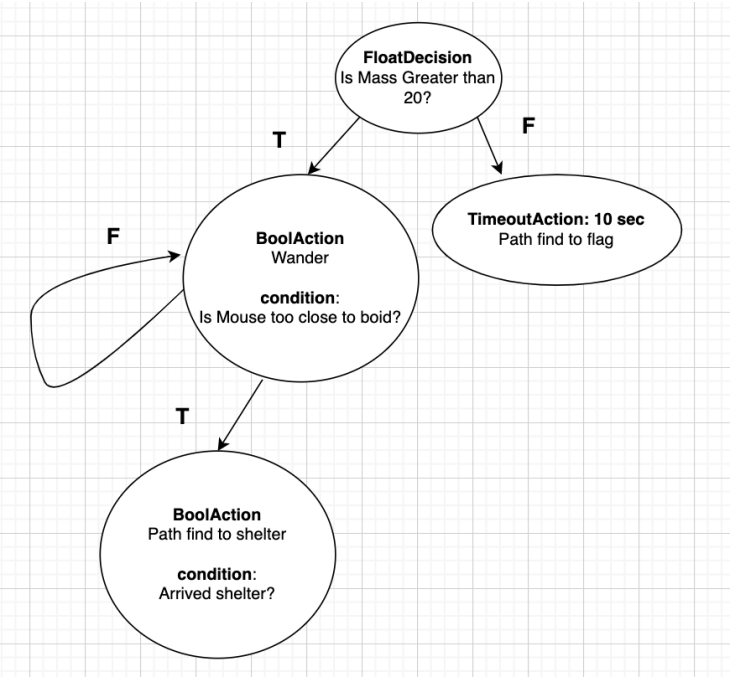

Figure 2: Decision tree, controls the character

Behavior Tree Diagram

The purple monster boid is controlled by the behavior tree, following is the behavior tree diagram:

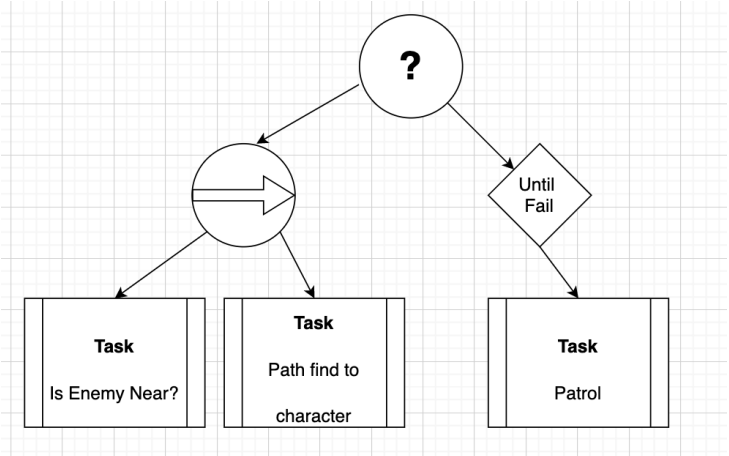

Figure 3: Behavior tree, controls the character

FUTURE WORK

I didn't manage to finish the action manager, so for now I have to invoke the tree traversal methods during Update manually, so one task is executed in one frame. Having an action manager would give me more flexibility and shift the responsibility of scheduling tasks to the manager. I could also prioritize tasks if I have an action manager doing the job of scheduling tasks.

I also didn't finish Goal Oriented Action Planning as my current implementation isn't sufficient enough for me to move on to that part.