Process Letter

Assignment 3 - Decision Making

The final assignment of the Game AI class was a great way to explore and implement decision-making algorithms and combine them with my previous work in this course. Although I had already heard about concepts like Decision Trees and Behavior Trees beforehand, implementing them in a game environment along with the movement algorithms gave me deep insights into how they work with NPCs and agents in-game.

It led me to think about these algorithms in a practical way and make some interesting decisions based on how I wanted to design and integrate them into my existing project. The most challenging part of this assignment was implementing Goal Oriented Action Planning, and although I am not fully satisfied with my final results in that section, it was still a great learning curve to implement GOAP as a concept.

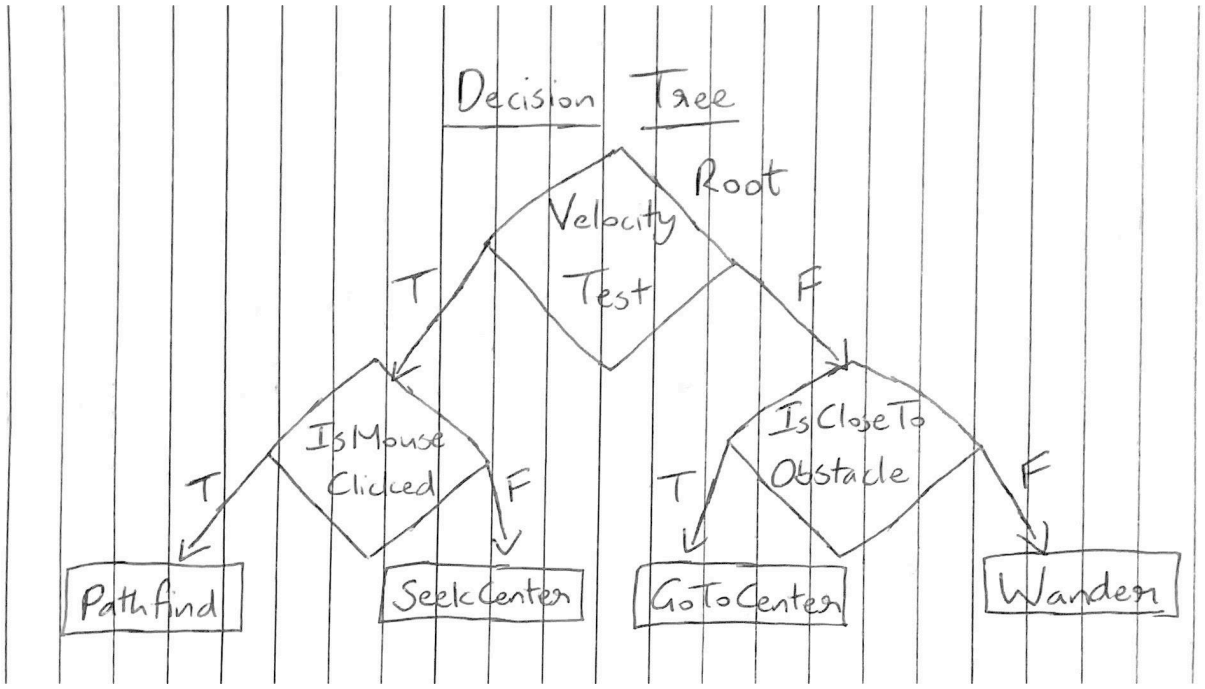
Decision Trees

Writing the base code structure for the decision tree was pretty straightforward. However, there were moments during this section where I had to really think about the design of the decision tree itself, and what behavior I wanted to implement.

My decision tree first checks for the velocity of the player/boid character. If the velocity is greater than a threshold value, it goes into another decision node called isMouseClicked, which basically checks for a mouse click on the screen. This is because I wanted to use the same pathfinding behavior as the previous assignment, so if the decision tree detects a click on the screen, the character path finds to that specific point using the tile graph. If there is no mouse click, then it simply seeks the center of the screen.

On the other hand, if the velocity check returns false, I proceed to a decision node called isCloseToObstacle, which calculates the distance between the

character and the nearest obstacle, and returns true if they are too close (again specified by a threshold value). In that case, the boid simply changes direction and goes to the center of the screen, to avoid bumping into obstacles. Otherwise, it is free to wander about.



For the abstraction scheme, the main variables I decided to use for my decision and behavior trees were:

1. The velocity of the character/boid - This is for the initial velocity check that determines which decision node to go to next.
2. The locations of all the obstacles in the scene - Data on the obstacles is important because we want to check for distances between the character and obstacles at all times, to avoid bumping into them.
3. Mouse Click - I am not sure how to frame this as an abstraction scheme/parameterization yet, because it is technically not a state of environment, but it is something that the decision tree takes into consideration when outputting actions.

Talking about actions, one problem I had when implementing the decision tree was the class structure and where to put the function to actually implement my action nodes. Since all the actions were related to movement algorithms or

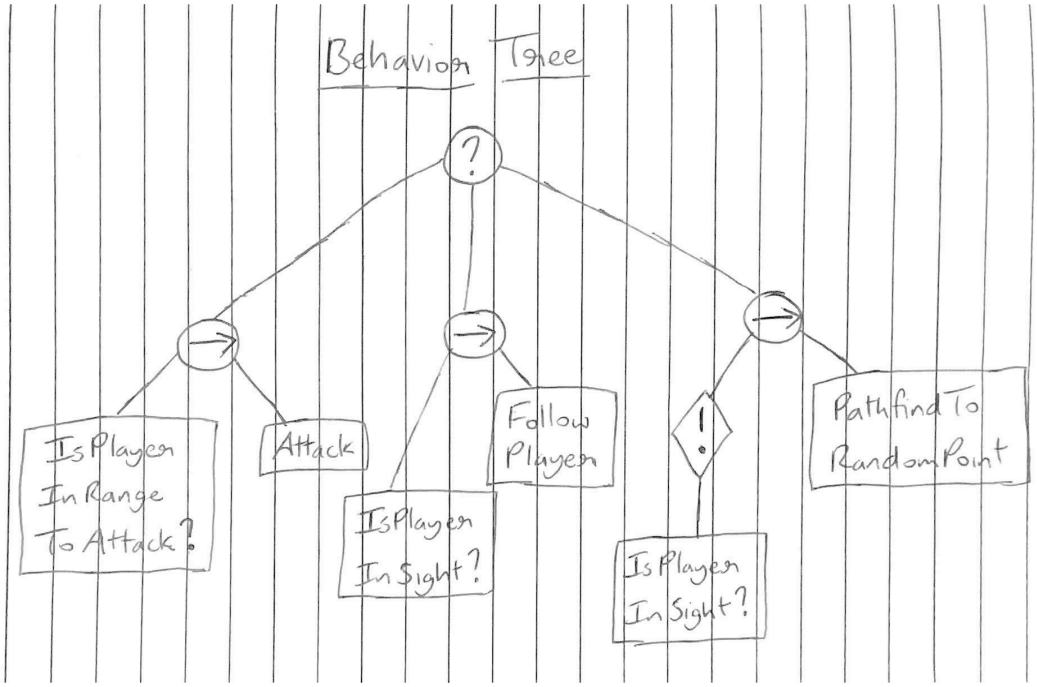
pathfinding (stuff I had done in the previous assignments), I was a little confused as to how to structure my code with the various classes I added.

Initially, I could not access the movement and pathfinding algorithms from the action classes I defined. So, I had to find a workaround and implement them in the update() function itself, by keeping track of which node the decision tree currently is at. This is a problem I encountered due to messy class structure, and is something I would like to fix later on to refine my code.

Behavior Trees

Moving on to the behavior trees involved many of the same concepts as the decision tree, but I wanted to implement a more thorough and complete architecture for my monster character. Again, writing the code for the various types of nodes was pretty simple. But, the harder part of this section was developing re-entrant behavior and keeping track of where the tree is currently at.

Although I did not explicitly define a 'Tick' as discussed in class, I found another way of storing node names and using them to implement the respective decision and action tasks. This was a really nice discovery, as I was able to update the names accordingly, which made it much simpler and easier to understand for me as a developer.



The behavior tree for the monster character starts off with a root node which is a selector, and then branches out into three separate sequencer nodes. The first sequence contains two tasks: first, a condition task that checks if the player character is within range to attack. Second, an action task simply attacks the player, destroys it, and resets the positions to the starting ones.

The second sequencer is very similar in terms of structure. First, it checks if the player is in sight. Then, it outputs an action that moves toward/follows the player's character. Finally, a third sequencer uses an inverter node (a decorator) on the same condition task of the player in sight. Then, it picks a random point on the screen and performs pathfinding to there, which is the default behavior of the monster.

Goal-Oriented Action Planning

As mentioned earlier, this was the most challenging section of the assignment, where I found it difficult to differentiate between the implementation of Goal-Oriented Action Planning, compared to Decision and Behavior Trees. Specifically, I was confused about how to represent goals as a sequence of actions, instead of just an action itself. Goals and actions seem pretty interchangeable to me, but this is something I have to research more about.

For my player character, I defined goals like avoiding the monster, avoiding collisions, and/or reaching a particular destination in the environment. The corresponding actions to these goals were also defined, such as pathfinding and evading the monster.

In terms of the abstraction scheme for Goal-Oriented Action Planning, I had to consider variables that are beyond just the state of the player character. Since one of the goals was to avoid the monster, it was essential to take in the monster’s state (position, velocity, orientation) to determine what actions to take at each point. This was different from the decision tree implementation where I only focused on the state of the player and environment.

In terms of the final results, I believe both the characters interact pretty effectively. However, I still think more intelligent behavior can be

implemented with the GOAP of the player character to make it seamless. Qualitatively, I don’t see much difference in how the monster's behavior looks compared to the player (although they are being controlled by different algorithms). This may be because of the blurry boundaries I discussed earlier with respect to actions and goals. But, it’s definitely something I look forward to exploring more.