Homework 4 Reflection

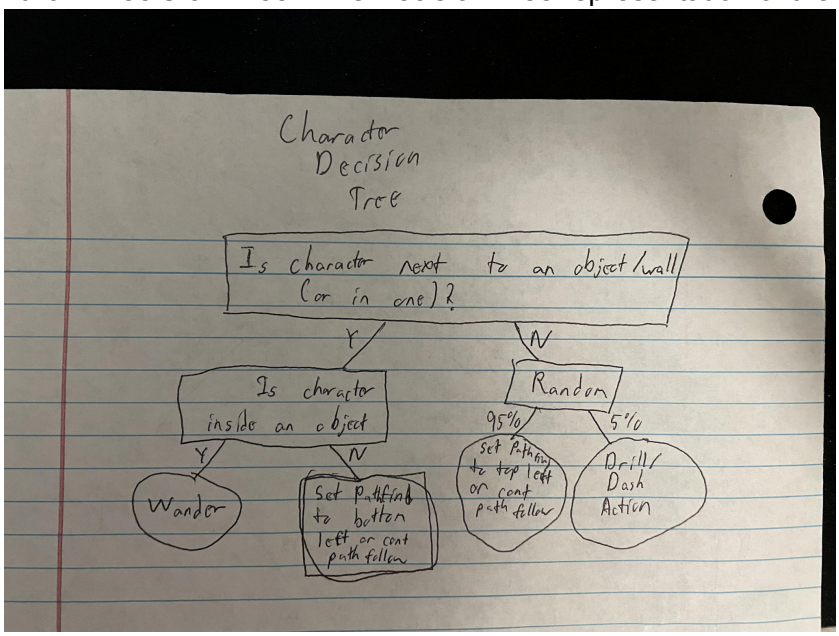
Starting off with the implementation of a Decision Tree and the nodes that existed within it, I largely used the class textbook for the implementation of the tree nodes. The DecisionTreeNode looks exactly like it does in the textbook however, my Decision class (the DecisionTreeNodes that make a Decision based off of some data) doesn't use the testValue() function that it proposed (because I have no clue how to have it just say "any type" as a return value. Probably some void pointer shenanigans but that's dumb) and instead just had those classes hold information that was needed inside the class and check those conditions in each makeDecision() call. The different action types were very confusing to me as just returning the action itself seemed weird. I looked at how the book created an ActionManager system which is something I would definitely look into for implementing DecisionTrees in games that I create, however it was more than what I was willing to do for this homework assignment and thus the Actions themselves are merely named differently just for me to check their types in my main function loop and do the desired action based on those names. As for what the contents of my decision tree were for my character (See Part 1 in Appendix), my first decision was to ask if my character was next to (or inside) an object/wall in the play space. If yes, I have another decision to ask if my character is inside an object/wall or not. If yes, then a simple wander action is performed, if not then it sets a pathfinding position to the bottom left of the screen. If the first decision of being next to a wall is no, then there's a random decision (skewed, not 50/50) where 95% of the time it will set a pathfinding position to the top left of the screen, the other 5% it will perform a Drill/Dash action. The idea for this decision tree is that the character can't really see in the play space and wants to put their hand against the wall to get to where they are going, and if they aren't next to a wall they panic and dash/drill around. The way I implemented checking if the character was next to an object or inside an object was through checking the vertex at the quantized position of the character and those around it if it they are valid or not, instead of looping through the whole list of objects and checking for intersects as the information is basically already there in my graph representation of the world. The dash/drill action can put them inside objects, which is a behavior that I'm ok with, and they will wander around until they can find a way out. This is a simple movement based decision tree that is able to show some kind of thought process in a setting as the one I described with semi-reasonable actions performed based on what's happening. As I had mentioned before, since I don't have an action manager, I did these actions each frame with some code located in main.cpp. This was ok for things like the wander and drill/dash action as those could be reapplied each frame with no problem but I had to come up with a solution from pathfinding as I didn't want a new path to be calculated every frame. What I ended up with was that if the character already had a set path to follow, then no matter what path following action came up, it would just follow the current path, then once that path was finished, or if the path following was interrupted by another action, then I would allow for a new path to be constructed for following. When testing out the runtime actions of the decision tree, I had to do some fiddling with the dash velocity value as well as the random percent chance in which pathfinding happened most of the time while the drill/dash action happened only some of the time. I also had to add more obstacles to my play space in order to see the wander inside object actions happen more frequently as you can see as I had

increased the object amount even in between screenshots below. The only thing that I may want to change is that sometimes my pathfinding will just have the character sit in the top left corner for a long time before it dashes so implementing a time out feature for these actions may prove useful.

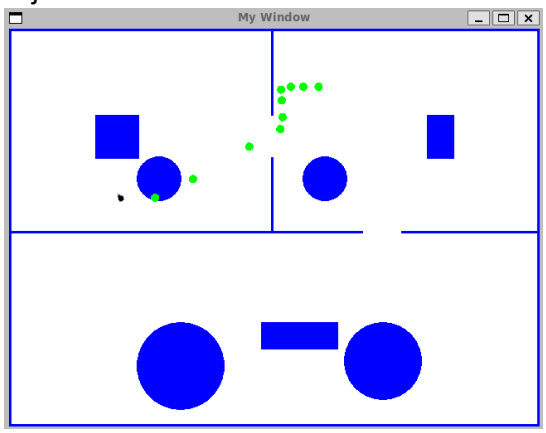
For my Behavior Tree, I similarly did my implementation through the help of the class textbook. My three composite nodes that I chose to utilize were Selector, Sequence, and Random Selector nodes. As opposed to Decision Trees and how they return actions, I thought the idea of each "Task" just running with a boolean return type was very intuitive. All information would be placed within these nodes and the run function is just recursively called on the children of CompositeTasks, with specific functionality differing per node. However, I do see the use in the delayed actions or actions that need to take place over a period of time and that being handled with an action manager and that a similar kind of system could be implemented into Behavior trees however, this recursive run() system just made a lot of sense to me. As for my monster, I wanted it to somehow also interact with the character and how it is able to go into objects so my behavior tree was constructed as follows (See Part 2 Appendix): At the root, there is a selector that has three children, two sequences and a random selector. The first sequence has two Tasks, one that checks if the character is inside an object and if so, the next task makes the monster make a beeline towards the player, charging through all obstacles in its way to reach the player, almost demonstrating that those are a part of the monster's domain in a way. The second sequence checks if the monster is in an object and similarly to the character, the monster will then take the wander task and wander until they can get out of the object. Finally, the random selector chooses between three different pathfinder tasks: one for the top right of the screen, bottom right of the screen, and the player's position at that time. Again, the pathfinding in this tree is handled similarly to the decision tree in which a new path is only found once the previous path has been found or if the path following is interrupted by another task action. The order I chose to place these tasks reflect what I thought should be the highest priority, the highest priority being the action that will allow the monster to at least make a lot of progress towards the player in order to collide with them and "eat" them. The "eating" is a check that is found in main and merely resets the play space. I was able to get my desired functionality very quickly after implementing the initial version of the code, most changes were done to the play space (again similar to the decision tree with adding additional obstacles for the character and monster to move around and through). In the end, the beeline task is a behavior that I really enjoy and makes a lot of sense for the purpose of the monster in my own design. I think that possibly making it easier for the monster to move towards the player would make the process sped up and probably better for viewing as there are times where both the player and the monster are pathfinding to opposite sides of the play space. Something that probably could've helped that would be if the positions that I chose would cross if they go from one side to the other so that it would be possible to incidentally collide as the general idea is that the character and monster (though less so) are trying their best to navigate a dark play space and thus an incidental collision is something that isn't out of the question. Either that or if I wished to have some more intention on the monster's side is to include other methods in which the monster would be able to approach the player, either through different/easier conditions or through more aggressive methods for approaching the player. This would merely just make this kind of demonstration easier to follow or see these kinds of changes happen in.

Appendix

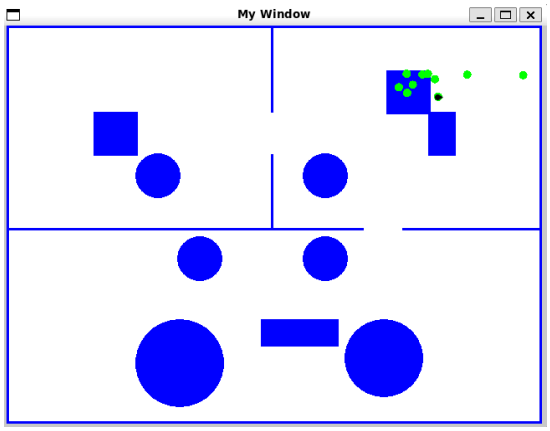
Part 1: Decision Tree - The Decision Tree representation of the character's decisions.



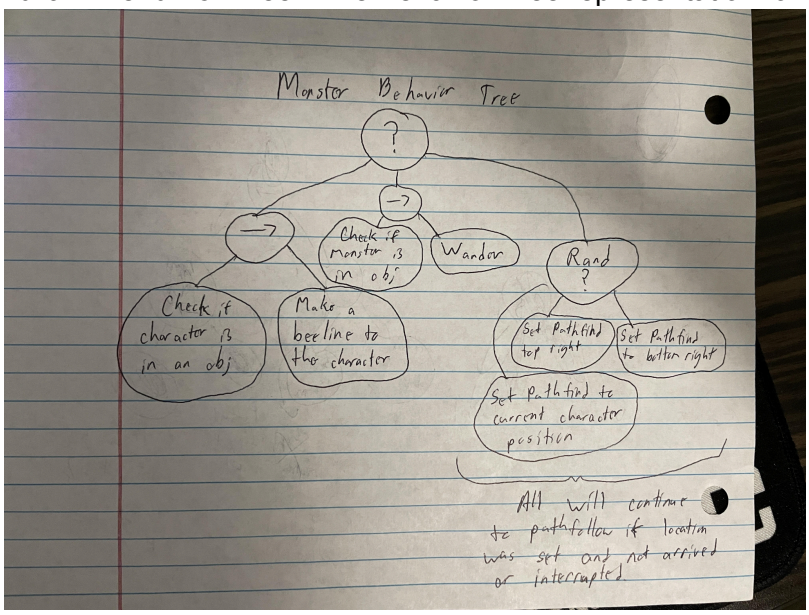
This shows how the dash action works as the boid character travels through one of the circle objects.



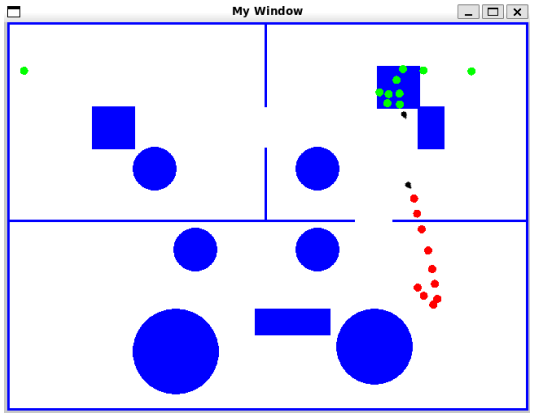
This shows how the character boid wanders if it is inside an object.



Part 2: Behavior Tree - The Behavior Tree representation for the Monster's behaviors.



This demonstrates the character (green bread crumbs) being inside and object and the monster (red bread crumbs) chasing after them, through a barrier



This demonstrates both boids merely pathfinding to different parts of the play space.