**BASE TASK IMPLEMENTATION**

For the this type of simulation, the best algorithm for the agent’s pathfinding may be Dijkstra or A\*, My only reasoning for aiming towards A\* at the moment is that in the base code, all the variables to the spawned food are nicely packed in an accessible public array. So I thought I may as well let the agents access that, which means its pathing Heuristics are more accurate.

So the pathing algorithm I’ll be going with for agents to find food will be A\*, and when it calculates the A\* path, it’ll pre-emptively grab the nearest available food as the goal node.

I’ve set up the agents to have access to the level generator for pathfinding.

Implemented A\* code functionality based on week 4’s content.

I’ve gone with the assumption the agent can traverse through the following grid types:

Open, Forest, Swamp and Water(they swim I guess :P) and not Wall.

Currently will be running tests simulating with a single agent.

After feeling stupid and wondering why the agent was not moving, only a temporary fvector variable claiming to be the current position was being set in the Agent’s Tick, only after I called SetActorLocation with that Fvector did agent movement occur… :D

Another issue I found was that after constructing the A\* path for an agent, I added the GridNodes beginning from the goal node to the start node. So I reversed the agents reading of the paths to read from the end of the array.

The agents now hopefully reaches its first food target, once the agent reaches the food target, it “consumes” food and recalculates its next valid food path.

After some tests, I found an error somewhere in my A\* path generation code, I accidentally returned all the nodes before the goal node. I fixed this by just while looping through each node instead of each Node’s parent.

Although my Agent is now getting the correct array of path nodes, it always ended up getting very close to the node with the food, but due to a rounding error when converting an actors position to a grid position (the agent couldn’t “eat” the node with the food).

That was fun, I had an infinite-locked while loop and memory-creep that used up all my Ram and then OS’s drive’s space. I found out I didn’t clean out the GridNodes’s “Parent” Variable after doing a A\* path creation, so when the next call to that A\* path generation was attempted, there must have been a infinite recursion between two nodes thinking their each-others parents and infinitely adding those nodes to the agents path causing me to run out of memory…

Ran into another issue where the agents would not set its actor position even though its “Current Position” variable was set to the “Target Position” after breaching the tolerance. When this occurs, it needed to set the actors position to that newly set “Current Position” or else my code lower down would get the position of the agent before the that snapping to the “Target Position”.

After fixing all these issues, a single agent can now traverse the entire map and successfully eat all available food assuming it survives the travel.

I’ve just discovered the “GetTravelCost()” function for the grid node, this being likely an additional heuristic needed for calculating which node is best to go to next, I’ve now added that functionality in when calculating the A\* path, and based on the values it gives, the agent should now be prioritising on grid nodes that are “open”.

I am now going to reactivate all 5 agents again and see what needs to change to allow for collision avoidance and investigate possible scenarios where more than one agent is going for the same food.

As a anticipatory measure, before an agent reaches its food node, each time an agent snaps to a target pos (when hitting a target node on its path towards food), it will now check that the food at the final node is still un-eaten. If its been eaten, it calculates a new path to another uneaten food.

Implemented 3x3 search around each node with the A\* path generation, which essentially allows for the paths to be diagonal (which is technically faster for agents reaching their targets).

Now to investigate possible collision issues with agents!

The method I am implementing to avoid agent collision is that when a path is generated, it tags all the grid nodes that are in use by the agent. And when the agents path elements are removed, it blanks out that tag from the grid nodes, allowing them to be used again for pathing.

This method may not be the most real time effective but will guarantee avoidance of agents and will technically be less computationally expensive than constantly checking if an agent is in the way.

This method seemed to have worked but I found a bug where my agents; after eating a food, couldn’t find a path to the next food, even when there was a clearly visible path available. This was due to my food finding method not accounting for food’s grid node already being in use by an agent, so it tried to find a path to an already claimed food and failed.

I’ve also set up events which will trigger all agents to recalculate their paths if its needed, EG; if an agents dies/consumes food and also when new food is generated.

Refactored food spawning so that eaten food gets cleaned up and removed from the map. Food also now respawns after all food is eaten.

Attempted a more experimental method of allowing agents to still generate paths towards food nodes that other agents were also going to. However, during this process of implementation, I broke some of my A\* path generation code and had wrapped all my experimental code in a hash define (“ENABLE\_PATH\_DISTURBANCE”) in the process of finding where I broke it. Turns out I had accidentally passed the parameters for my “GetAccessibleNodes” function backwards….

After running a few tests, I ran into a crash where my AFood class was garbage data, I found out this was because each time I destroyed my food actor, I didn’t null the “ObjectAtLocation” on the grid node that had said food.

As a test, I’ve implemented Dijkstra’s method of pathfinding and wrapped it in a define. This is possibly less effective in terms of creating the fastest path for an agent, however, it does allow for me to not have to do any distance calculations with all the food, which may be more efficient the more food there is on the map. So I think I’ll stick with Dijkstra!

I’ve removed the offsets off all models so its easier to tell where everything is!

I’ve now discovered that due to the paths being able to go diagonal, there was still possible collisions that could occur, for example when two agents crossed paths diagonally in a 2x2 (since neither are technically touching each other’s path grid nodes), so due to its complexity in fixing that, I’ve disabled diagonal pathing, however, if you’re interested to see it “work”, I’ve wrapped it in a define you can re-enable.

**EXTENDED TASK IMPLEMENTATION**

Duplicated the blueprints for “BP\_Agent”, “BP\_Food” and all the materials for food. I also added a little cube shape on top of both agent types to dictate visually which food type its going for.

created new variable for AAgent; (“AgentType”) , which can either be “Carnivore” or “Herbivore”. Created new variable for AFood; (“FoodType”), which can be “MEAT” or “FRUIT”.

Agents with the type “CARNIVORE” can only consume “MEAT”.  
Agents with the type “HERBIVORE” can only consume “FRUIT”.

On a random note, I fixed a rare random stack overflow crash with agents recursively disturbing each other’s paths. I first changed having an array of agent pointers stored on the grid node to only store a single pointer to one (since only one agent will be using a grid node at a time).

An additional change I made is that when a agent’s path has been disturbed, instead of recalculating the path on the same frame, I set a bool on the agent to recalculate on the next tick (so that the agent who is doing the disturbance can finishing its pathing).

These changes then allowed me to track down why the recursion occurred. It was because after an agent generated its new path and added itself to the grid nodes, it notified the other agent that was using the grid node to recalculate their path, but due to new agent not being assigned to the grid node yet, the old agent assign themselves back onto the path again, which would of notified the original new agent to recalculate, thereby causing a recursion and leading into a stack overflow crash...oh god that hurt my head.

But if this crash for some odd reason decides to appear again, it is very likely due to bad disturbance code, which can be disabled via “ENABLE\_PATH\_DISTURBANCE”.

Fixed a bug where agents who were standing still could be collided with (since they had no path). I fixed this by setting the gridnode variable “IdleObjectAtLocation” to the agent each time they failed to calculate a path. Then within the path calculation, in the “GetAccessibleNodes” function, It will avoid nodes that have an agents idling there.

Fixed yet another bug where if two agents were equal distance away from the same target food node, they would both get stuck, I fixed this by making it so if the distance of the new agent creating the path to the food node is less than **OR EQUAL TO** the distance of the previous agent to the same food node, the new agent’s will always win.

After running more tests, that fix only made it this issue occur more frequently, however, I believe I’ve fixed it properly now. I added a Boolean flag in the GridNode “WasJustClaimed”, which when a agent sets up its path, it flags that value as true. So when the situation occurs where two agents continuously think they are closer then one another, In the “GetAccessibleNodes” function, if the difference between their two distances is under a set tolerance and the node “WasJustClaimed”, it will disallow the “closer agent’ to go for the node. Bug solved 😊

That is it, Assignment complete 😊

Or so I thought…

I had accidentally wiped my entire assignment and had to start over…

So I have re-written everything but the only major difference is that there are no defines for swapping the pathfinding functionality. Which means pathfinding is strictly Dijkstra and Non-Diagonal.

I had also slightly changed the way agents get their paths disturbed, which should have eliminated any recursion issues my previous iteration may of had.

Ok, Now I am finished 😊