FIT3094 Brief - Assignment 2

Jessica Nguyen

Overview

The solution includes three different types of agents deriving from a parent GOAPAgent class. Each agent class has their own set of GOAPActions with their preconditions and effects set in each. The actions for each agent are then planned by the GOAPPlanner which takes into account each action's ActionCost, the action's current and goal state and the agent's values/inventory.

GOAPPlanner

The GOAPPlanner class works out an order of actions based on all actions that are available and their action costs, the current state of the world and the goal state. It does this by getting all available actions, defining a start node and searching for a goal node. If goal nodes were found, it would then find the cheapest goal available by calculating the running costs, conditions and comparing between each goal node that was found - which in the case of the assignment was the closest/nearest. Once it finds the nearest node, it would then work backwards until it reaches the start node and builds the queue of actions from the result. The GOAPPlanner also had functions for creating an action subset, checking if conditions exist within the world state and generating a new state based on the current state of the world and updating the effects of the changes.

GOAPAction

Parent Class

The GOAPAction parent class mainly acts as a base for the child actions detailed below. It is used to define variables and functions that all actions will need to have using virtual functions that the child could then override. The main features of the parent class involve the following functions: AddPrecondition, RemovePrecondition, AddEffect and RemoveEffect - which allows for adding and removing effects and preconditions for each action called and set in its respective GOAPAgent class.

Collect Materials (Wood/Stone/Both/Food)

The solution has four classes derived from the GOAPAction parent class for collecting materials - CollectWoodAction, CollectStoneAction, CollectBuildMatAction and EatFoodAction. Each of these have their own overridden functions from the parent class. The action cost is set to 2 for all action classes other than the EatFoodAction so that the food action can be prioritised. The FoodAction differs from the others in the sense that instead of setting resources, it resets the

health of the actor upon collision through an OnOverlap method in the FoodActor's cpp which checks for the agent type and sets the health accordingly.

- Check Procedure Precondition which checks if any of the materials that are being searched for is within range of the agent using a collision channel and sphere overlap with a specified radius. For every overlap that is detected, it would cast to ensure that the target is the right class type and if there are more than one, it would search to find the nearest target in the list and set it as the target.
- Perform Action is the tick function of the actions. It is called every tick by checking the current time compared to the target time and if conditions are met, it would gather the materials and update the values for the affected actors then return true until a condition returns false which indicates it is done or if an actor can no longer allow resources to be gathered. In this case, it would deduct the amount of resources from the target and increment one to the agent at every tick that it is valid until either the target is destroyed or the agent runs out of space.
- Requires In Range checks to ensure that the target is right next to the agent that needs to access it.
- Reset as described by its name, resets the action back to its default settings so that it could be used again in a clean slate.

Drop off Materials (Wood/Stone/Build)

The solution has four classes derived from the GOAPAction parent class for collecting materials - DropOffWoodAction, DropOffStoneAction and BuildAction. Each of these have their own overridden functions from the parent class. The action cost is set to 2 so that the food action can be prioritised. The way it is programmed is pretty similar to the collected materials. The main difference is that in CheckProcedurePreconditions, its target is a singular actor in the world so it doesn't need to find the nearest one. However, instead it gets the actor and checks if it is the valid type before setting it to target.

GOAPActor

Parent Class

The GOAPActor parent class holds a list of states for state machine initialisation. It holds a set of available actions and the queue of actions that the agent is planning to use. It has functions for the state machine such as OnIdleEnter, OnIdleUpdate and OnIdleExit which dictates what the agent should be doing while in or transitioning to or from each state. For example, OnMoveUpdate will move the agent towards the target by calculating the distance from the current agent to the target location, normalising it so that it is a unit vector and move based on move speed, delta time and direction.
The GOAPActor class also holds virtual functions to be overridden in each child agent class which gets the world state, creates a goal state, handles plans not being generated or if it was aborted midway.

BuilderAgent

The builder agent is instantiated with a root component and a visible component and uses constructorhelpers to set a static mesh and material which is then scaled smaller to suit the size of the world. It also starts with no wood and stone and maximum health. It has the actions EatFood, CollectBuildMat and Build. These are all set within the agent's BeginPlay, specifying its preconditions and effects and then adding them to the list of available actions. The timer is also set here where every 2 seconds, the agent would lose 1 health specified in the DecreaseHealth function. In the GetWorldState, it adds the state name and condition to the world state. In CreateGoalState, it adds the goal state depending on whether the agent is hungry, has a full inventory or still can collect resources which allows replanning of the sequence of actions based on the new goal state.

StonemasonAgent

The stone mason agent is instantiated with a root component and a visible component and uses constructorhelpers to set a static mesh and material which is then scaled smaller to suit the size of the world. It also starts with no stone and maximum health. It works the same as the builder agent in terms of setting the preconditions and effects and handling goal state and world state. However, its actions differ in the sense that instead of CollectBuildMat and Build, it has CollectStone and DropOffStone which work relatively the same way.

WoodcutterAgent

The woodcutter agent is instantiated with a root component and a visible component and uses constructorhelpers to set a static mesh and material which is then scaled smaller to suit the size of the world. It also starts with no wood and maximum health. It works the same as the builder agent in terms of setting the preconditions and effects and handling goal state and world state. However, its actions differ in the sense that instead of CollectBuildMat and Build, it has CollectWood and DropOffWood which work relatively the same way.

Known Bugs

A known bug is that when running the code, once any of the agents hit full resource inventory amounts and want to move onto the next action, it returns with an exception error on GetActorLocation() and the Root Component of the specific agent. I am unsure why this error occurs as I have personally looked through every single GetActorLocation() call and compared all GOAP classes with the week 6 and 7 lab tasks. Furthermore, it is certain that the actor's Root Component is causing it as the root component was renamed to test and the error spat out the renamed changes to the actor's root component name. Due to this, testing of the project was hindered which affected the progression (spent two days trying to figure it out).

Problems Encountered

See above. It is unsure if the other sections of code are working as intended due to not being able to run through without error. Possible future problems could include plans not being

generated properly or resources not being destroyed but cannot be tested as they do not move to the next action.