Lim Zong-Yi MID:28696328

FIT3094 A2 Documentation

GOAP

I used Goal Oriented Action Planning (GOAP) for this assignment as it was the best way for my agents to decide what actions are available to them, which of them will be useful at the current time and plan out the action based on their current world state and goal state.

I created the GOAP Action class first as this will make use of Polymorphism to create my subset of actions that are used by the actors in the scene. I created virtual functions for the GOAP Action to check preconditions, if the action is done, perform the action, check if the actor using the action needs to be in range of the actor that is being acted on and reset all used variables in the action. Non-virtual functions are to add or remove preconditions and effects of the actions that the action needs to perform or has an effect on the actor(s).

Secondly, I created the GOAP Planner class. Here is where all actions available, the current world state and the desired goal state of the actor is worked out to create a valid sequence of actions to achieve the goal. In this class, I created the Plan function which is where the actor inputs the parameters to create the sequence of actions to perform. This is helped by private functions, BuildGraphRecursively, CreateActionSubset, CheckConditions and PopulateNewState, for the Plan to go through a recursive loop finding the best action to take to reach its goal state.

Lastly, I created the GOAP Actor class. This actor would be the parent class to all the actors that are needed for the scene through Polymorphism. This class would contain the StateMachine, the list of available actions, the queue of current actions that it needs to perform in order to achieve its goal and universal variables such as move speed of all the actors. The virtual functions that all child classes have to override are the GetWorldState, CreateGoalState and BeginPlay functions. The StateMachine contains three states, Idle, Move and Action. It will begin at Idle to create a Plan in the GOAP Planner class using the current world state, the goal state and the available actions. The Move State will be invoked when the current action is not in range of the actor and therefore a Move algorithm is in place. When the actor is in range of the action, the state is changed to the Action State to attempt to perform the action. If the action is successful do the next action in the CurrentAction list, else go back to the Idle state.

Children of Actors and Actions

I wanted to create the actors with their individual actions one by one to test if their own interactions with the world is achieved. Later, I would implement shared actions such as getting food or getting tools. During creation of actors and actions, my sequence of operation was to create an actor, its individual actions that it can use, initialise the actions in BeginPlay, add to world states and finally add goal states.

Creating Wood Cutter, Stone Mason, Builder and Tool Crafters

Creating the actors needed in the scene, I created new C++ classes with the inheritance of the GOAPActor. Firstly, I added a static mesh component that is used as the visible object in the scene. This holds the mesh and is attached to the root component of the actor. Secondly, I added the current Health, the Max Health and the Min Health variables. The Min Health would be the threshold before the actors need to plan the action of getting food. I created the function of Decreasing Health with a World Time Manager so that the actor will lose health per two seconds, and if the Health is 0 then the actor will be destroyed. The constructor will assign the static mesh instance to an object and set all variables to its starting value.

A Tool Durability variable is added to allow Wood Cutters and Stone Mason to use the tool while collecting resources

Creating Collecting/Dropping/Getting/Crafting Actions

These interactions are Polymorphed from the GOAPAction. They have a list of the respective resources to check if there are any known resource points that the actor can go to or the respective place to drop the resources, i.e Village Centre or the Building , a boolean to check if the right amount of resources has been collected/dropped/crafted by their respective actors, the TargetTime when the actor can do the action and the Timer to increment the TargetTime.

A CheckProceduralPrecondition function checks if there is a stored list of pointers to the resource or a pointer to the drop off point. If there isn't any, create a collision sphere to find any actors in the vicinity. Then, check if found actors are of that resource/drop off point and add it to the list of pointers/pointer. If there is a list of pointers, check to find the nearest resource in that list and set it as the Target of the actor and return true to signify there is a target for that action. If it is only a pointer, set that as the Target of the actor and return true to signify there is a target for that action.

The action will most likely need the actor to be in range of the acted on and therefore the RequireInRange would return true to tell the GOAPActor to change its state to the Move State. When the action is there, the PerformAction function will run through. First, it will check if the actor doing the action has enough health to perform the action. Then, depending on the action, it will check the resource to see if there are any resources there. It will cast the actor and the Target of the actor to decrease or increase the amount of resources. If the goal number is met, the boolean to check if the correct amount is in the actors inventory turns true and will notify that the action is done in the IsActionDone function. Before each action is performed, a reset function is invoked to reset all variables, such as SetInRange, Target and resource pointer(s). The TargetTime is set to the next target time that the action can be performed.

In actions for getting Food and Tools, instead of adding resources/tools to the inventory, I changed the value of the Health and Tool Durability of the actor. This is called by a Heal or ResetToolDurability function in the actor’s class to set the actor’s Health to the Max Health and the Tool Durability to be Max Tool Durability. For the case of Food, the actor will move to and overlap the collision component of the FoodActor. When this happens, the Food Actor will check if the actor overlapping is a GOAPActor and the Target of that actor is the Food Actor. If this is true, then destroy the Food Actor.

Assigning actions to Actors in BeginPlay

To assign the actions to the actor, I initialised the action by creating a new instance of the action and stored it as a pointer. Then, I added the Precondition for the action to be usable and the Effect that the action will have on the actor. I made sure that I had the Key of the Precondition and the Effect the same as it will be cross checked in the GOAP Planner class. Lastly, I added the action to the list of available actions the actor has so that it can be used in the Plan to get the goal.

I added all the individual actions to their respective actors first and then the common actions to check if the actor’s individual actions are performed when the scene is played.

Adding to World and Goal States

The world state is the current state that the actor is current in. It would have the state of the amount of resources the actor currently have, the health of the actor and, depending on the actor, whether it has a tool to work with.

For the World States, I got the World State from the GOAP Actor and added it to a local variable for easy access. Then, I added a Key and Value as a state as it was a TMap of String and Bool. The Key is to be the same as the Precondition and Effect of the action so that the Plan could check if the World State value satisfies the Precondition’s value of the action and the action can change the World State value using the Effect’s Key.

The Goal State checks what the actor needs at the current state it is in. First would be checking if the health is valid, then, depending on the actor, whether it has a tool to use, lastly the individual actions that it needs to perform (Collecting/Dropping/Crafting). Only one Goal State is allowed to be returned to the GOAP Actor for it to be planned as this could cause conflict of goals that the actor needs to perform and therefore cause a No Plan Found error.

Problems and Bugs

A problem encountered when coding the Preconditions, Effects, World State and Goal States is that the Key has to be the same throughout the class. If any discrepancy is made, the whole Plan function will not work and will create a No Plan Found Error. This took me 3 hours looking through the GOAP Planner, specifically the BuildGraphRecursive function, to find any problems in the code.

Another problem was when coding the check on the amount of resources collected/dropped/crafted with the amount needed. If the actor needs to get food in the middle of the action, the counter resets when the actor returns, therefore adding another amount needed to the already filled inventory of the actor. To solve this, I got rid of the check to amount needed and checked if the actor has the maximum number to collect/drop/craft and used a boolean to return true when the condition is met.

A known bug in the system is when the number of resources is depleted in a resource point and the actor of that resource point is destroyed. The acting actor does not reset its resource list after the first CheckProceduralPrecondition loop and therefore will be able to move to the destroyed resource actor's location and obtain resources there. I have solved this issue with the GetFoodAction as it resets the Food list and checks for food in the vicinity per action call.

Another known bug is when the actor is dropping resources at the drop off point and it needs to get food. After it gets food, the number of resources is not valid to the Goal State and therefore it invokes the action to collect more resources until the maximum number to collect before returning to drop the resources at the drop off point.

A small bug is noticed when the Tool Crafter drops the tool in the Village Centre, the actors needing the tool will take at more than a second to get the tool and start moving to collect the resources.