ReGoapExample:

GoalState:

1. <“collectedResource” + ResourceName, true>
2. <“reconcilePosition”, true>

There are two unfulfilled conditions in the root node, and there are two actions that can satisfy one condition each:

|  |  |
| --- | --- |
| AddResourceToBanckAction | GenericGoToAction |
| * Effects: <“collectedResource” + ResourceName, true> * Preconditions: <“isAtBankPosition”, bankPosition>; <“hasResource” + ResourceName, true> | * Effects: <“isAtPosition”, objectivePosition> and <“reconcilePosition”, true>, and this only happens if there had been only 1 condition in goal state and that’s <“reconcilePosition”, true>, and <“isAtPosition”, objectivePosition> is the starting position. * Preconditions: None |

Note, unlike AddResourceToBankAction, there are two setting options for GenericGoToAction. The first setting simply checks if given some stackData if the goal state has the key “isAtPosition”, then the first setting would be <“objectivePosition”, “isAtPosition”.Value>. Meanwhile, the second setting accommodates for a goal state that has a single condition which is <“reconcilePosition”, true>. And the resulting setting would be <“reconcilePosition”, true> and <”objectivePosition”, startPosition>

The GenericGoToAction would not be possible as the next node from the root, as it accounts that the goal state must only have one unfulfilled condition which is <“reconcilePosition”, true> and it will apply two effects:

1. <“isAtPosition”, the agent’s starting position prior planning>
2. <“reconcilePosition”, true>

Fortunately, there are no preconditions, so more than likely this type of GenericGoTo action relying on this setting is meant to finish the example off, in which the agent before performing the plan always begins at some starting position. This go to action doesn’t really act as a go to, but rather a way for planning to regressively plan its way back to the planning position. This action doesn’t really do anything under this setting. If go to starting position, then it’s already there and thus the action succeeds and pushes the next one in the plan.

Remember these things:

1. Initialize the settings based on the goal state and other factors (i.e. state copy at that instance of parent node)
2. The conditions applied by preconditions and effects to the goal state act as another virtual world/blackboard same as the virtual states, only this time you’re trying to wipe the board clean with applying effects and preconditions.
3. During Run Time, the settings are what is relied on in order to run the action, think of it as a private memory after some evaluation of the planning states.
4. Effects and preconditions are applied to the same goal state, and there could be remaining applied effects that don’t cancel out with the goal state. Also, effects and preconditions are only applied if there are no conflicts, and if CheckProceduralCondition(stackData) passes

Here’s how Patrol Goal can play out when responding to a patrol order:

GoalState of the Root Node:

<“investigatedPoint” + patrolPoint.ReturnName(), true>

<“reconcilePosition”, true>

1. Investigate action:
   1. Effects: Investigate the Point, so <“investigatedPoint” + patrolPoint.ReturnName(), true>
   2. Preconditions: <“isAtPosition”, PatrolPointPosition>
2. GoToAction:
   1. Effects: <“isAtPosition”, PatrolPointPosition>
   2. Preconditions none
3. GoToAction:

At this point, the is only one condition left unfulfilled which is <“reconcilePosition”, true>, so GoToAction will just go to the starting position. We set the following effects

* 1. Effects: <“isAtPosition”, Agent’s startingPosition>; <“reconcilePosition”, true>

1. GoToAction:

This seems redundant, but it does guarantee that we start at the starting position if need be.

* 1. Effects: <“isAtPosition”, Agent’s startingPosition>