Goals and Actions
Table of Contents

About the Planner

The Planning process to calculate a Valid chain of actions (plan) for the current goal of highest priority takes into account two world states: the current world state [a deep copy made prior to the planning process] and the goal world state. Any modifications to these states DO NOT affect the environment, nor do they affect the agent's world state [working memory].

When considering Valid actions to chain to the subsequent plan, the planner will copy all the properties that exist in the goal state, and the current state a deep copy of the actual Agent's world state [working memory] prior planning. The planner's objective is to calculate the lowest cost plan possible in which the goal state has no conflicts with the current state, in other words, the goal state becomes empty. Or if the goal wasn't empty, but had no conflicts, then the current world state already satisfies a goal's properties.

With a given Goal state, the planner looks for valid actions in which they have effects that satisfy a property of the Goal state without conflicting any other Goal State property, and preconditions that do NOT conflict with Goal State properties [keys that are the same, but different values]. Lastly, the action's Context Preconditions must be true, in the case of ReGoap, the method CheckProceduralConditions() must be true which includes some information that needs to be considered, not picked up by an Agent's sensory system.

These effects and preconditions are then applied to a copy of the Goal state for a resulting Action State Node to inherit. The Actions State Node has three properties, a total cost, f, a heuristic cost, h, and a cost of g:

$$f = g + h;$$

The g represents the sum of action costs up to this Action State Node from all preceding Action State Nodes and their Action costs. Meanwhile the h represents the number of unsatisfied properties in the Goal State, so f is the total sum of these properties g and h. Remember that the planner favors the Action State Nodes of lower cost to chain together a plan of least cost to satisfy and validate the current possible and highest priority goal state.

Actions with a higher cost will be least considered before other actions of lesser cost are considered. But no matter the cost, if a lower priced action doesn't lead to a satisfied goal state, then the Actions of higher costs may still be considered. For example, if the planner is to satisfy

the goal to KillEnemy with the single property/symbol TargetIsDead set to true, and two actions can fulfill it AttackEnemy and AttackEnemyFromCover, it will depend on each action's cost and resulting heuristic. If the AttackEnemy action has a cost of 5.0 and the AttackEnemyFromCover has a cost of 1.0 with the precondition <IsAtCover, INode>, then the AttackEnemyFromCover will be considered first, but if no Valid Cove is nearby, then the actions to validate AttackEnemyFromCover subsequent's goal state would fail. The AttackEnemy action will then be considered.

These costs of actions are more so to help designers to have the Agent favor certain actions over others. The lesser an action costs, the more the Agent favors that action and will try to calculate a plan including that action if appropriate to the chain via preconditions and effects. This can effect the order of the plan too, as actions of lower cost if in a validated plan would be higher up the chain, or the latest actions for the Agent to perform, as the Planner does **A* regressive searching**. As Jeff Orkin states, "We apply a cost to actions to force A* to consider more specific actions before general ones", so the more general actions are of a higher cost, and the more specific are of lesser costs.

Dynamic Problem Solving

Dynamic Behavior arises out of re-planning while taking account of knowledge gained through previous failures. As the A.I. discovers obstacles that invalidate his plan, he can record his knowledge in working memory and take it into consideration when re-planning to find alternate solutions to the KillEnemy goal, for example.

Goals & Scenarios

Note: The key to advance AI behavior is the ability to facilitate the layering of behaviors. Also, some Goals may not be as broad [as open ended] as others. This does not mean however a goal is not to be open ended, for example, in F.E.A.R. the Dodge Goal can be satisfied with either DodgeShuffle or DodgeRoll actions. We do want to separate certain behaviors as goals, because in ReGoap, when another goal not currently running has a sudden higher priority, for example, an Agent is suddenly in Player's LOS (line of sight) while running KillPlayerGoal, then if the Agent values its life it will dodge from Player's aim, so DodgeGoal will become the new current goal. Developers "never have to manually specify the transitions between these behaviors. The A.I. figure out the dependencies themselves at run-time based on the goal state and the preconditions and effects of actions.

KillPlayer Goal: The Goal has only a single property in its goal state: <"PlayerDead", true>. The Goal should only be possible if the Agent "seePlayer" is true which is a key inside the Agent's memory. Not necessarily just see the Player, but if the Agent's squad is engaged, then the Agent is capable of ambushing the player in order to kill the player. **Note: Don't want too many Agents attacking the Player at the same time! Otherwise it can feel overwhelming and unfair**.

Scenarios Concerning Cover Nodes

- [Scenario] **<u>Shooting at Player</u>**: Standard scenario in which the Agent sees the Player and starts attacking in its current position, and this is satisfied by the following plan:

  AttackAction
- **<u>Shoot at Player from Cover [already in cover]</u>**: If the Agent is at a CoverNode, then Agent will attack from Cover.

  AttackFromCoverAction
- **<u>Shoot at Player from Cover [not in cover]</u>**: If the Agent is not in cover, and it sees the Player, but the Agent is not in Player's LOS, and there is a nearby Valid cover, then the Agent will navigate to the nearest unlocked Valid CoverNode. Afterwards it will proceed to attack from Cover.

  ReserveCoverAction -> GoToAction -> EnterCoverAction -> AttackFromCoverAction
- **<u>Shoot at Player while Going to Cover</u>**: If the Agent is not in cover, and it sees the Player while being in Player's LOS, and there is a nearby Valid cover, then the Agent will shoot at the player while navigating to the nearest valid Cover. **Note: This could have a combination of Dodge and or DodgeToCover if the Agent's dodge is not on cooldown**.

  ReserveCoverAction->AttackToAction->EnterCoverAction

Scenarios Concerning Ambush Nodes

Remember: AmbushNodes serve two purposes:

- [Scenario] Shoot at Player from a Valid Ambush Node [sees Player]: If an Agent is at a Valid Ambush Node, it can proceed to attack if "seePlayer" is true:

  AttackFromAmbush
- At Valid Ambush Node and don't see Player: The Agent will proceed to wait, by default [3 secs, 20 secs], for "seePlayer" to be true. It will by default wait for 3 seconds before considering to stop waiting [AmbushNode becomes invalidated], and the action to wait will finish after 20 seconds if Player hasn't been seen yet.

  ~~WaitAction~~ -> AttackFromAmbushAction
- Near a Valid Ambush Node [Within Radius] and don't see Player]: This scenario would have to take higher priority than a scenario involving to get to cover, as ambushing the Player [without having seen the player] is more important.

  ReserveAmbushAction -> GoToAction -> ~~WaitAction~~ -> AttackFromAmbushAction
- Cover is unavailable and a Valid AmbushNode is nearby [See Player]: Agent will go to hide at a Valid Ambush Node while shooting at the Player, if necessary:

ReserveAmbushAction -> AttackToAmbushAction

- [Scenario]

GetToCover Goal: This Goal may seem redundant to have a GetToCover Goal when already the KillEnemyGoal is capable of having the Agent navigate to Cover in order to kill the Player. Well, what if we just want the Agent to navigate to Cover and not Attack. For example, this can be dependent on Orders given, or the status of the Agent like Health. Behaviors can be dependent on Agent's aggression, but would have to quantify this somehow… **Note: Don't want too many Agents attacking the Player at the same time! Otherwise it can feel overwhelming and unfair**. Also, this will permit the scenario in which the Agent is already at Cover when the Kill PlayerGoal becomes the current priority Goal. Additionally, what if a Goal like KillEnemy were to fail, because we ticked goals to be BlackListed on Failure, would need another Goal to be the current, and that is where this Goal can come in.

GetToCover Goal is a Order responding goal to Squad Behaviors like "GetToCover" Behavior of the same name. An Agent will prioritize going to the CoverNode specified by some Order given to it via its memory. This Goal will not have the effect that the Agent will Attack the Player while trying to navigate to some CoverNode. This goal can fail if the Agent detects that it is in the Player's LOS if the Agent sees the Player. Though the goal failed, the Agent can still get to its ordered CoverNode through the Dodge Goal and, or the KillEnemy Goal which the Agent can dodge in the Dodge Goal and if still not in cover and in Player POS, the Agent proceeds to enact KillPlayer Goal and Attack while heading to cover. DodgingToCover action could even happen if the KillPlayer Goal is active.

This implementation allows the Player to be only engaged with one enemy upon the acting of the Squad Behavior GetToCover Behavior, but the moment an Agent understands that it's under threat (i.e. being in PLayer LOS), then the Player as consequence of their decision, would have to deal with more than one Agent Attacking Him/Her if choose to ignore Agent suppressing fire.

Agents in cover will always look in the same direction of the Cover Node's forward direction. And if an Agent were to shoot from cover, then they either remain in the CoverNode if not invalidated, and they see the player. Another alternative is if they don't see the Player and yet they're in Valid Cover, then the Agent will Use the Smart Object in the Cover Node playing the CoverStep animation to step out of cover and shoot if they see the Player.

- [Scenario] **Squad is engaged with the Player and the Agent is given direct orders to take cover**.

  ReserveCover-> GoTo -> EnterCover

-

Ambush Goal: Squad is engaged, the Agent doesn't see the Player, but the Agent detects a Valid Ambush Goal nearby. This should have a higher priority than getting to cover. This is not an order responding goal. The goal may seem redundant, but it will enforce that the Agent will wait at a Valid Ambush Node if it doesn't the Player!

- [Scenario] Agent enters a Valid Ambush Node if it doesn't see the player
  ReserveAmbushAction -> GoTo -> WaitAction
- Agent goes to Valid Ambush Node if no other cover is nearby.
  ReserveAmbushAction -> GoTo

AdvanceCoverGoal: A goal responding order to the SquadBehavior AdvanceCover Behavior in which Agents in Vaid Cover will advance up to Valid Cover closer to the Player.

- [Scenario]

Dodge Goal: Will be possible when the Agent is within Player's LOS (line of sight). Goal becomes highest priority if the Agent is capable of dodging again and is in Player's LOS.

- [Scenario] Dodge is refreshed and available and Agent is in LOS of the Player:
  DodgeAction (Direction dependent on a direction that is not in the direction of the Player's Pivot, and in a direction that facilitates the space).

Actions

With a cost per action, the Agent can be forced to prefer one action over another. With the cost metric, the A* search planner can be guided toward the lowest cost sequence of actions to satisfy some goal.

Note: **Can worry about ammo and accuracy of the Agents attacking depending on action later**…

~~[Important], periodically, the Planner will recalculate a new plan, even if it's for the same current Goal of the Agent, as the Agent will have to adapt to its environment dynamically, as conditions could change while executing a plan. This is better than having to wait for a Goal to fail to reevaluate a new plan.~~

[Important] we layer goals, so that a current goal, such as KillPlayer, would NOT fail and can be reused immediately once another goal finishes.

[Important] Rather than tick the BlackList Goal on failure option, if a GOal like KillEnemy were to fail, for example, the Player looks at the Agent's direction when it has passive actions to only Attack when in cover, we would need the Agent to reevaluate a new plan for KillPlayer that involves attacking the Player while trying to get to cover.

Action/Run()	Cost	GetSettings()	Effects	Preconditions	Procedural Condition

Attack With a Gun, Agent will shoot at the Player's current position if visible	5	The Agent <"seePlayer", true>, then set to the settings the Target	<"playerDead", true>	If melee weapon, then Agent needs to be within range of Player	Gun has Ammo or Agent has melee weapon and is within range
AttackFrom Cover Agent will either use the SmartObject of the CoverNode to play the CoverStep Animation, or the Agent can see the Agent from their cover and decides to shoot while standing at the Node instead.	3	Check if Agent sees the Player, and if it does, set to settings the Target, otherwise if Cover is still Valid, have the Agent use the CoverNode's Smart Object to play the CoverStep Animation, so Agent faces a specified direction and steps in another specified direction	<"playerDead", true>	<"inCover", true>	Agent has either reserved cover, or there is Valid Unreserved Cover within radius
AttackToCover or AttackTo If Agent is within LOS of the Player, it will Attack while navigating to cover, rather than Attack from cover. Keep shooting if see the	2	Check if within Player's LOS. Set the Player as the Target if so, and similar to GoTo, set the Objective Position if there is an "isAtPosition" in the goal state.	<"playerDead", true> <"isAtPosition", CoverNode position>	None	The Agent is detected in the Player's LOS.

Player and stop either Player is no longer visible and have navigated to the Cover, or Cover has been invalidated					
GoToAction Agent just goes to some specified Position. // **This action will fail or be overridden if the Agent is in Player LOS**.	5	Check in the goal state for a "isAtPosition" key and if the goal state only has one property and it's the "reconcilePosition" key, then the Goto Action makes sure that the Agent navigates back to its original position where it started planning.	<"isAtPosition", some position>	None	If there's an objective position set to the Action's settings.
EnterCover Agent enters Cover. This is more as a placeholder to make sure we specify the correct cover for Agent to navigate too.	1	Agent either is occupying its reserved cover, or not occupying its reserved cover, or hasn't reserved cover yet. Get the nearest CoverNode to this Agent that's unreserved, if Agent has not already reserved cover	<"inCover", true>	**If already reserved cover** <"isAtPosition, CoverNode position> **If not reserved cover**: <"reservedCover, CoverNode> **If the Agent is already in cover, but the Cover doesn't match the Cover the**	Is there valid nearby cover that has not been reserved yet?

				Agent is told to got to, then set to <“exitCover”, currentCover>	
Exit Node Agent exits its Valid Node. Reasons can include the Advance Cover Goal where there is cover closer to the Player than the cover the Agent currently occupies, but is still Valid.	5	Check the Goal state for the key “exitNode” and set the associated value to the settings	<“exitNode”, INode>	**None**	There is a Node that Agent needs to exit found within the Goal state
~~**ReserveCover**~~ ~~The Agent will reserve the CoverNode, so that other Agents are locked from it when planning or trying to navigate to it. Whatever Agent reserves the Node first gets it~~ **This can be**	~~1~~	~~Check the goal state for the key “reservedCover” and set the associated CoverNode to the settings~~	~~<“reservedCover”, true>~~	~~**None**~~	~~If there is Cover that needs to be reserved in settings.~~

a generic action that just reserves any type of Node as long as they inherit INode.					
AttackFrom Ambush Agent will attack the Player from some ambush Node. Waiting is just another by product of running this Action, don't explicitly need to tell the Agent to wait through some Action.		If the Agent sees the Player, then attack from the Ambush Node If the Agent doesn't see the Player, but the Ambush it occupies is valid, then the Agent will wait until player is visible If Agent is near a Valid Ambush Node and hasn't reserved one yet, then the Agent will Navigate to it	<"playerDead", true>	~~If don't see Player:~~ ~~<"wait", true>~~ If not at AmbushNode : <"atAmbush", true>	Are there valid AmbushNodes unreserved that Agent is within Radius of? Or has the Agent already reserved an Ambush Node
~~**Wait**~~ ~~Agent will wait from minimum seconds to maximum seconds, specified from some Ambush Node it occupies~~					

EnterAmbush		Check in the goal state for the key “atAmbush”, otherwise, don’t bother with action	<“atAmbush”, true>	**If already reserved ambush** <“isAtPosition, AmbushNode position> **If not reserved ambush**: <“reservedAMbush, AmbushNode>	Are there valid AmbushNodes unreserved that Agent is within Radius of? Or has the Agent already reserved an Ambush Node
ReserveNode Agent will reserve a Node of any type, so other Agents won’t occupy or use the node besides this Agent. The types of nodes include CoverNode and AmbushNode, as long as they are children of the interface INode		Check in the goal state for any key “reservedNode”, and get the associated Node	<“reservedNode, true>	**None**	Are there any Nodes to reserve from the goal state?
Dodge Agent will dodge out of Player’s LOS **[Important] Dodging Related**	1	Goal state has key “dodge” If the Dodge functionality is no longer on cooldown (check	<“dodge”, true>	**None**	The Agent has a destination it is able to dodge towards. And Dodge

Actions are separate from the other Goals to make implementation easier. Also, a Dodge related goal can override a current goal, but not fail it, so that overridden goal can presume if still a priority!!		from Agent's memory if "canDodge" is true), then the Agent can Dodge. **If Agent has reserved Cover, then ignore this Action** Look for the nearest valid CoverNode if the Agent hasn't reserved a CoverNode. But if no valid CoverNodes are nearby, then dodge to the nearest valid Ambush Node **But if there are no valid AmbushNodes nearby, then dodge to some nearby Agent. Note that if no other Agent is alive, then there should be presumably a Valid Ambush or Cover**			Mechanic is refreshed
DodgeToCover Agent will dodge to Nearby Cover **[Important]**	1	Goal state has key "dodge" If the Dodge functionality is no longer on cooldown (check	<"dodge", true>	**None**	Same as Getettings

Dodging Related Actions are separate from the other Goals to make implementation easier. Also, a Dodge related goal can override a current goal, but not fail it, so that overridden goal can presume if still a priority!!		from Agent's memory if "canDodge" is true), then the Agent can Dodge. If the Agent has reserved cover, and it's not yet at Cover, and the distance from the reserved cover to Agent is greater than the distance Dodge animation can traverse, then			

Goals:

A goal should be some open-ended objective that an Agent tries to satisfy its properties/conditions. A goal by principle should be broad and open-ended.

- KillPlayerGoal:
  - This goal should be open ended in which Agents try to kill the Player from some Ambush Node, Cover Node, or an Ambush Node from out in the open. Also Agent should be able to move while shooting to some Node if see the player.
- GetToCoverGoal:
  - This goal is necessary for Squad Coordinator to give a specific order to an Agent, through a chosen Squad Behavior, to get to cover while some other agent provides suppressive fire of some form.
  - Or if the Agent needs to get into cover right away, like being on low health, and is within dodging distance to the Cover.
  - This could override KillPlayerGoal, as the Agent being on low health prioritizes their safety, rather than being aggressive on the Player.
- ~~ShootToCoverGoal~~:

        - Unnecessary, as GetToCover is broad about Agents getting to cover.
    -

Actions: All actions cost are defined on a scale of [1-5]. 5 being the more general actions, and 1 being actions that are more specific. Actions are specific performances that Agents can commit in order to satisfy a goal. The lower the cost of an Action, the more likely the Agent will perform it near the end of a subsequent plan

- **AttackAction**: Agent has either a loaded weapon or a melee weapon and Player is within distance for melee. This is a standard general action regarding attacks.
- **AttackFromCoverAction**: Agent will attack from Cover Node.
- AttackFromAmbushAction: Agent will attack Agent from an Ambush Node.
- ~~AttackFromNodeAction: We want to differentiate from the Cover Node and AMbush Node when debugging, so make these separate~~
- **AttackToNodeAction**: Agent attacks and moves at the same time to some Node, unless the Node becomes invalidated, then the Action fails, but if it gets to the Node, then action succeeds. This Node can include CoverNodes and AMbushNode, as long as the Node is derived from the INode interface.
- **GoToAction**: This is a generic GoTo action, given some transform.position or Vector3 position, it will tell the Agent through its NavMeshAgent component to navigate there by setting the destination.
- **AttackFromAmbush**: An Agent will attack from an AmbushNode
- ~~ReserveCoverAction:~~
- ~~ReserveAmbushAction:~~
- ReserveNodeAction: Agent will reserve some Node.Will consider the type of Node being reserved.
- EnterNode: Agent will enter a node, and set the type of Node being entered to the goal state for reserve.