Table of Contents

Defining Individual Agent Behaviors

Description: In this documentation, individual Agent behaviors are abstractly defined, in order to better plan logistics of how certain behaviors will be implemented into the ReGoap architecture. Individual Behaviors consist of some goal and a subsequent plan of Actions. Each individual Behavior will then go through isolated testing, before integration to other behavior sets.

Note: Some individual Behaviors will share the same names as Squad Behaviors, so for naming conventions, Squad Behaviors will be named followed by the Key Behavior. For example, GetToCover is the individual Agent Behavior, meanwhile “GetToCoverBehvior” is the Squad Behavior, similar to the Squad Behavior “PatrolBehavior”.

Template

Dates:
Description:
Goal:
Actions:
Relevant Nodes:
RelevantSensors:
Summary:
Results from Testing:
Known Issues:
Considerations:

Shooting

Dates: [8/22/2023 - present]

Description: Agent is able to shoot at the Player if detected in their FOV (field of view) defined in the Player Sensor (Scripts.Sensors). The Agent should be able to maintain line of sight, and rotate its sprite body, separate from Movement. In this case, the agent's capsule with the NaveMeshAgent component. This will allow the player to rotate and look at a certain direction or target like the player, while moving in another direction. Movement speed should be somewhat inhibited if not moving related to the direction the agent is looking at.
Goal:
Actions:
Relevant Nodes:
RelevantSensors:
Summary:
Results from Testing:
Known Issues: Need to rotate the agent to look at and maintain line of sight of the player, even if the player and, or the Agent were to move. Unless the player is too fast, maintaining line of sight is impossible (i.ePlayer activates Slow motion abilities).
Considerations:

GetToCover

Dates: [8/24/2023 - present]
Description: Agent is able to navigate to Valid Cover if within vicinity, to avoid attacks from the Player. The Agent will exit cover if the Player is too close for comfort relative to the CoverNode the Agent occupies. If the Agent were to exit cover, due to Player aggression, then the Agent should try to navigate to another Valid Cover or some other defined Node. As long as the Agent is capable of exciting Cover and fleeing from the invalidated cover to a different location.
Goal:

- **GetToCoverGoal**:
    - Goal State:
        - goal.Set("inCover", true);
    - IsGoalPossible(): Only possible if the key condition "nearbyCover" in Agent memory and the associated INode<string, object> is not null. Need to get the IReGoapAgentComponent in Awake(). Further, remember that all agents are organized into squads, even if their squad only comprises one agent. It would make sense for an agent to take cover if its associated squad is engaged in combat.
    - GetPriority(): Default Priority of 10. [***Note: Could later have a Higher Priority based on an Order from some Patrol Behavior***].
    - [**Reworked**] IsGoalPossible(): The goal is only possible if the agent's associated squad is engaged with the Player. The rule of engagement means if the squad has directly encountered the Player, including last known/suspected whereabouts that haven't been invalidated due to search. This way, other agents that haven't

encountered the Player though they are aware of a reported last known location, won't bother to take Cover.

    - The IsGoalPossible() is open to change, such as permitting Agents, even if not engaged, to take cover. For example, if the player's whereabouts are unknown or not, an Agent of extremely low health will occupy Cover with adjusted Cover parameters.

Actions:

[IMPORTANT]: Remember to keep in mind the Cost of the Actions, as a valid plan could be made, but the Actions may NOT be ordered as intended. Costs can be dynamic as well.

- **ReserveCover**: Ticks the bool property "reserved" of the nearby CoverNode instance to true and sends an instance of itself to the CoverNode.
- **GoTo**: Go to a listed objective position from Action's settings, set within the planner inside the ReGoapNode class when called GetSettings().
- **EnterCover**: Placeholder action if developers want to implement a EnterCover Animation, thus calling the Animation State in the FSM.

Relevant Nodes:

- **CoverNode**: Has the following responsibilities:
    - Detect the Player if within its FOV and BoundryRadius.
    - Upon Detection, the CoverNode becomes Valid, and it updates to the CoverNodeManager that it is Valid.
    - While Valid, check whether the Player is still detected within FOV and BoundryRadius, otherwise, the Node becomes invalidated.
    - While Valid, check if the Player has entered the ThreatRadius, and if the Player has, the Node becomes Invalidated.
    - Hold a float property of the Radius for nearby Agents to be within to even consider using the CoverNode.
    - Hold a bool property called reserved that is set to false by default. If an Agent considers to use the CoverNode, then they must reserve the Node by ticking its reserved property and passing an instance of itself to the Node.
    - Holds a IReGoapAgent<string, object> property that holds a reference to an instance of the Agent. Property is called "currentAgent"
    - Reservations between Agents can be changed based on the Unique property of ReGoapAgents - **their names**.
    - Check whether the Agent has died while reserved, if so then untick "reserved" and set "currentAgent" to false.
    - Reactivate itself if no longer reserved, and reactivation can occur after some delay, so other Agents do not immediately use it. Note: the reserved property can also act as an occupied property too.
    - CoverNodes can be locked, so other Agents will not reserve it, if already reserved.

    - Only instances CoverNodes are unlocked is if the Agent that reserved it dies, <"alive", false>, or the CoverNode becomes invalidated while reserved and the Agent's Cover Sensor calls to exit it which consequently unlocks it, or the Agent calls to exit cover through some Action.

- **CoverNodeManager**: The CoverManager is solely responsible for updating only valid CoverNodes to the Global BlackBoard inside the Game Scene. Valid CoverNode's again being nodes that detect the Player.

RelevantSensors:

- **CoverSensor**: Detects nearby Valid Cover not reserved by other Agents, and what Valid cover the Agent currently reserves if any.
    - Try and get a valid list of cover nodes, that isn't empty, from the Global BlackBoard.
    - If the list isn't empty, then it will iterate through the list, searching for a CoverNode not reserved by any other Agent, and that the Agent is within the CoverNode's radius
    - After finding a valid nearby unreserved CoverNode, it will add Node to a list of CoverNodes deemed usable: List<INode<string, object>>.
    - After iterating all valid CoverNodes, if the usableCover list isn't empty, then it shall set to the Agent's memory: <"nearbyCover" INode<string, object>>
    - CoverSensor also detects whether the agent has reserved a Valid CoverNode. [Whether the Agent occupies it, based on its location compared to the transform.location of the cover is done by other systems or scripts like Goals]. The sensor will then proceed to write <"atCover", INode<string, object>>.
    - Cover Sensor will remove its written conditions if no nearby cover is found or no longer at a nearby Cover's position. Also if the nearby cover had been invalidated.
- PlayerSensor:
    - The Player Sensor has two Field of Views (FOVs):
        - Vision FOV: If the Player is detected by this FOV while within the defining respective radius, then the Agent sees the Player. This FOV is blocked by obstructions, including walls.
        - Extended FOV: When the Agent loses sight of the Player, then this extended FOV will activate for a set period of time and is NOT obstructed by anything. The purpose of this vision is to detect the Player's location if they move behind an obstruction. This prevents the Agents from looking unintelligent if they can't recognize that the player is behind some obstruction after losing sight. This Extended FOV is only active for a small duration of time, so that it wouldn't seem like Agents could see through walls (though they could). Additionally, the small duration would allow the Player to escape.

        - The Extended FOV tries to detect the Player, and if it does, it will write to the Global BlackBoard the estimated location of the Player (Even though it's exact). The key word being estimated because it will only be affirmed if Agents physically see the Player with the regular FOV.
        - The latest Agent that lost sight of the player will update to the Global BlackBoard, the latest estimated location of the Player.
        - If the latest Agent that lost sight of the Player, doesn't detect the Player even with the extended FOV, then it would be as if the Player had disappeared. This could cause another Behavior to be warranted if Agents lost complete knowledge of Player's whereabouts.

Implementation Summary:

An Agent will have a CoverSensor Script that will try to get a list of valid cover nodes from the Global BlackBoard, that have been uploaded by the CoverNodeManager in the Game Scene. The ***Cover Sensor*** will continue to try to get a valid list, and upon retrieving one, it will then check if it's within that Cover Nodes specified Radius. As long as the Cover Node hasn't been reserved yet, then the Cover Sensor will write to memory <"nearbyCover" INode<string, object>>. This will continue to be written, unless the Cover has become invalidated, so it's no longer in the list of valid cover, or the CoverNode has been reserved by another Agent. Agent will then choose a new CoverNode if applicable, otherwise, it will remove its previous written memory condition.

If a nearby [Valid] CoverNode is detected, then the goal ***GetToCover*** will be possible, and the Agent will construct a plan to navigate onto the Cover's position. The Plan will consist of the ***ReserveCoverAction***, the ***GoToAction***, and the ***EnterCoverAction***. This plan will fail if the CoverNode becomes **invalidated**, or another Agent has occupied or **reserved** it. Note that the plan should fail immediately if the ***ReserveCoverAction*** were to fail, but if it were to succeed, then any other Agent that tried to plan to go to the Node will fail too. [***Note: Rather than a bool property, Agent should probably pass some HashCode that requires verification***]

Agent will also pass a reference to its instance to the CoverNode, so that the CoverNode can untick the booking reservation if the Agent were to die, thus the CoverNode may become available again, if valid.

Results from Testing:

Known Issues:

Considerations:

Exit Cover

Dates:[8/25/2023 - present]
Description: Exit from CoverNode, if the CoverNode becomes invalidated. How this could happen is that the Player is within ThreatRadius, thus invalidating the CoverNode, or the Player is no longer in FOV and BoundryRadius which also invalidates the CoverNode. If CoverNode has been Invalidated, then the Agent should exit from the CoverNode through calculating some new Goal.

[IMPORTANT] ***Don't have to specifically write an Exit Cover Behavior, since this can be the byproduct of other goals chaining off from one another***.

For Example, If the CoverNode becomes invalidated, what if it finds another nearby CoverNode to occupy.

ShootToCover

Dates:
Description: Agent is able to navigate to Valid Cover while shooting at the Player, to increase odds of survivability. Very similar implementation wise to GetToCover, but this time, the Agent can shoot. How this behavior can be warranted is still debatable, perhaps the Agent is of lower health, so they are more desperate to waste ammo for preservation. Another possibility is that the agent's other cover was invalidated, so it must navigate to another cover, but the Player is too close for comfort. Essentially, the Agent is trying to maintain distance from the Player. Or it could be a response to some order, or dependent on Agent's personal aggression based on Threat Level rating.

Goal: ShootToCover Goal

- This goal permits the Agent to shoot at the Player if within Agent's FOV while navigating to the nearest Valid and unreserved CoverNode.
    - GoalState:
        - goal.Set("tryKillPlayer, true);
        - goal.Set("inCover", true);
    - IsGoalPossible(): Goal should only be possible if the Agent can see the player and detects nearby cover.
    - GetPriority():

Actions:
Relevant Nodes:
RelevantSensors:
Summary:
Results from Testing:
Known Issues:
Considerations:

AdvanceCover

Dates:
Description: Agent while in Valid Cover, will Advance to new cover closer to the Player. Does Not necessarily have to be near the Player, as long as another Agent in the GameScene sees the player, or Agents lose line of sight of the player, but they suspect the Player to be around a certain area. For example, Agents lose line of sight of Player due to the Player navigate behind some Wall Obstruction or corner.
Goal:
Actions:
Relevant Nodes:
RelevantSensors:
Summary:
Results from Testing:
Known Issues:
Considerations:

Retreat

Dates:
Description: Agent will retreat from a location to a new location, away from the player.
Goal:
Actions:
Relevant Nodes:
RelevantSensors:
Summary:
Results from Testing:
Known Issues:
Considerations:

Dodge

Dates:
Description:
Goal:
Actions:
Relevant Nodes:
RelevantSensors:
Summary:
Results from Testing:
Known Issues:
Considerations:

DodgeToCover

Dates:
Description:
Goal:
Actions:
Relevant Nodes:
RelevantSensors:
Summary:
Results from Testing:
Known Issues:
Considerations:

RunToCover

Dates:
Description:
Goal:
Actions:
Relevant Nodes:
RelevantSensors:
Summary:
Results from Testing:

Known Issues:
Considerations:

RunShootToCover

Dates:
Description:
Goal:
Actions:
Relevant Nodes:
RelevantSensors:
Summary:
Results from Testing:
Known Issues:
Considerations:
