Table of Contents

Defining Squad Behaviors Documentation

Note: Behaviors should either inherit the Behavior class which is inside the directory Scripts >> SquadBehaviors >> Behaviors, or define a class that inherits the interface IBehavior. Other related interfaces to the Squad Behavior System (SBS) are found in the Core folder; Scripts >> SquadBehaviors >> Core.

SquadBehavior (SB) Name & Priority: State the name of the behavior and the priority. The priority of the Behavior dictates how soon the Behavior is checked if it's warranted for a given squad. Squad Behaviors are sorted in the Start method of the SquadBehaviorSystem (SBS) class inside the directory Assets >> Scripts >> SquadBehaviors, after calling the RefreshBehaviorSet() function. RefreshBehaviorSet() is overridden in the derived custom class SquadCoordinator, also in the same directory.

Squad Behavior (SB) Description: Here define an overall abstract description of the behavior a given squad is to accomplish. This serves as the foundation to further define the logistics of the behavior, such as how the behavior will be checked if it's warranted, how to determine whether the behavior finished or failed, and what orders are to be given and how agents will respond to their assigned orders.

IsWarranted(): **Note, only one behavior at a time is given to a squad**. This is a function call to check whether the behavior is warranted to be assigned/given to the squad.

IsFinished(): This is a function call to determine whether the behavior is completed which can be simply whether the agents completed their orders. Upon finishing the behavior, any orders given inside agents' memory in the squad are removed.

IsFailed(): This is a function call to determine whether the behavior failed, for example, this behavior may have a time limit that if exceeded, then the behavior failed. Same with IsFinished(), any remaining orders in Agent's memory are removed.

GiveOrders(): **Note, only one order at a time can be given to an agent**. This is a function call to give orders to the agents in a given squad. Orders are simply classes with a private Dictionary

similar to ReGoapState (ReGoap >> ReGoap >> Core) in which key-value pairs are set and retrieved from the Order. When defining Orders, they should either inherit the Order class (Scripts >> SquadBehavior >> Orders) or implement the IOrder interface (Scripts >> SquadBehavior >> Core). WHen giving Orders, construct the Order and set necessary key-value pairs to the Order and set the Order to the agents memory as:

squad.Members()[i].GetMemory().GetWorldState().Set("order", IOrder someOrder);

This allows for the agents to detect in their memory any orders and respond with an appropriate goal. Orders are removed if finished either by the assigned Behavior class or the OrdersSensor (Scripts >> SquadBehaviors >> Sensors)

This function can be very complex code-wise and messy, so as coherent as possible try to define assignments appropriate for the behavior. Additionally, the call to GiveOrders() inside the SquadCoordinator class derived from SquadBehaviorSystem, calls every Update() if the Behavior isn't finished and hasn't failed.

Orders: Define the Orders relevant for assignment to agent by the assigned Behavior.

Responding Goals: Here, define the goals that agents will possess in order to respond to their orders and fulfill them.

Set some abstract conditions that are to be fulfilled. Note that some goals may only become possible if their respective Order is detected inside the agent's memory, and such goals may also have a dynamic priority that increases to a bigger priority from less notable priority as an effect. Also, it's best practice to keep conditions that need to be fulfilled constant, rather than updating the conditions based on the Order given.

- **Goal1**: Here write what order the goal is to fulfill.
    - Condition1: < "condition", object value>
    - Condition2: < "condition", object value>
    - …,
- **Goal2**:
    - …,

Actions: Here define abstractly the sequence of actions that complete each goal

Nodes: Relevant nodes that may be needed to complete behaviors or determine whether a behavior is warranted, finished, and, or failed.

Issues: Report known issues after implementing the behavior with a timestamp

- Some Issue <TimeStamp (Month/Date/Year):

Other Important Notes

- The Squad Manager manages what agents are alive and assigns them to a squad based on proximity and if the max squad size hasn't been exceeded. The Squad Manager is a component with public fields to choose the number of squads to manage, and the fixed size for every squad. Any available squads and living agents associated with some squad are updated to the Global BlackBoard in the Game Scene.
- The Squad Behavior System will try to retrieve a squad list from the Global Blackboard, otherwise it would not bother assigning behaviors to squads as none are available, and hence no agents are given any orders to respond accordingly.
- Performance may be an issue due to the number of orders being given per second and the complexity of checks to give orders for each defined Behavior.
- Global BlackBoard
-

TEMPLATE

SquadBehavior (SB) Name & Priority:
Squad Behavior (SB) Description:
IsWarranted():
IsFinished():
IsFailed():
GiveOrders():

Orders:
Responding Goals:

- **Goal1**:
  - Condition1: < "condition", object value>
  - Condition2: < "condition", object value>
  - …,
- **Goal2**:
  - …,

Actions:
Nodes:
Issues:

Patrol Behavior

SquadBehavior (SB) <Name, Priority>: <"patrolBehavior", 1>
Squad Behavior (SB) Description: The Patrol Behavior has a squad navigate throughout an arena/map by investigating patrol points laid at junctions of corridors throughout the map. The squad will first patrol to the nearest patrol point. If the patrol point, the squad navigates, to is at a junction or splits off to other corridors leading to other ***adjacent*** patrol points, then the team will split off into subdivisions [if they can] to investigate the adjacent patrol points.

This phenomenon will occur again if a division were to navigate to another junction patrol point, but only if they have the resources [available agents in their sub-division] to do so. The purpose of splitting divisions into smaller subdivisions, to the point that there are even sub-teams of single agents, is to cover as much ground as possible with a squad. Theoretically, if the agents were to navigate all corridors in the map, they will eventually encounter the player, so other Squad Behaviors will become warranted.

IsWarranted(): This Behavior is warranted if all patrol points are NOT ticked as investigated. Patrol Point Nodes can be ticked as investigated or non-investigated. Check the Nodes sub-section to understand Patrol Points further.

IsFinished(): The Patrol Behavior is finished once all PatrolPoints have been investigated.

IsFailed(): The Player has been spotted by the Squad, or the Squad has been wiped out [KIA] somehow without the player being detected. May update this to also account if the player has fired a weapon, so a new behavior will be warranted to investigate, making this Behavior fail.

GiveOrders():

At the first iteration of Giving Orders for the Patrol Behavior, the squad of agents will navigate to the nearest patrol point that is non-investigated. There will be an appointed team leader and the team leader will be given the Patrol Order that contains the nearest non-investigated patrol point in reference to the squad [Actually the team leader]. Other Agents will follow each other to mimic some Conga Line phenomenon.

The second iteration of giving orders, if the first nearest patrol point had been investigated, then search through its adjacent patrol point list for non-investigated adjacent patrol points, and split the squad if necessary to navigate and investigate these points. Rinse and repeat.

Orders:

- Patrol Order: Agent will navigate to a given Patrol Point to Investigate it.
    - < "patrolPoint", PatrolPoint pointOfInterest>
- Follow Order: Agent will follow another agent described in its given order.
    - < "followAgent", IReGoapAgent<string, object>>

Responding Goals:

- **Patrol Goal**: Responds to the Patrol Order. Results with the Agent navigating to the patrol point of interest specified in the Patrol Order, then the agent will investigate it when nearby.
    - <“investigatedPatrolPoint”, true>
- **Follow Goal**: Responds to the Follow Order. Results with the Agent following another agent specified within the Follow Order given.
    - <“following”, true>

Actions:

- **Investigate Action**: Action that investigates the Patrol Point, simply tick the PatrolPoints.investigated to true.
- **GoTo Action**: Tells Agent to navigate to some objective/target position. Can simply call the Nav Mesh Agent component of the Agent and set the destination for NavMeshAgent.destination.

Nodes:

- The Patrol Point Node is simply a class with a name and a bool called investigated, and if the Patrol point is investigated, then this bool instance is ticked as true. Additionally, each patrol point has a list of adjacent patrol points that developers can modify and add patrol points in the Inspector window.

Issues:

- [8/22/2023 - current]The Patrol behavior is only warranted if all Patrol Points in the Game Scene are non-investigated, so only one squad can do the patrolling behavior at a time, unless all squads are assigned Patrol Behavior immediately before any squad could investigate a point. The reason for this implementation of IsWarranted() is that when patrol points are reset to non-investigated, there are common phenomena where the squad DOES NOT patrol the nearest non-investigated point even if they are right on top of it. This may be due to the fact that Patrol Points are resetted sequentially, rather than instantaneous at the same time.
- [8/22/2023 - current] Agents may navigate to the same patrol point and start colliding into each other, competing against each other for the same target position. Possible Solution, add a stopping distance from target and within range, the GoToAction will succeed and push the next action in the resulting plan.
- [8?22/2023 - current] When Agents are order to follow one another, they have an orbiting behavior to try to avoid collisions, and this is especially apparent when the agents spawn in an enclosed space.

GetToCover Behavior

SquadBehavior (SB) Name & Priority:

Squad Behavior (SB) Description:

IsWarranted():

IsFinished():

IsFailed():

GiveOrders():

Orders:

Responding Goals:

- **Goal1**:
  - Condition1: < “condition”, object value>
  - Condition2: < “condition”, object value>
  - …,
- **Goal2**:
  - …,

Actions:

Nodes:

Issues:

AdvanceCover Behavior

SquadBehavior (SB) Name & Priority:

Squad Behavior (SB) Description:

IsWarranted():

IsFinished():

IsFailed():

GiveOrders():

Orders:

Responding Goals:

- **Goal1**:
    - Condition1: < “condition”, object value>
    - Condition2: < “condition”, object value>
    - …,
- **Goal2**:
    - …,

Actions:

Nodes:

Issues:

Retreat Behavior

SquadBehavior (SB) Name & Priority:

Squad Behavior (SB) Description:

IsWarranted():

IsFinished():

IsFailed():

GiveOrders():

Orders:

Responding Goals:

- **Goal1**:
    - Condition1: < “condition”, object value>
    - Condition2: < “condition”, object value>
    - …,
- **Goal2**:
    - …,

Actions:

Nodes:

Issues:

ReGroup Behavior

SquadBehavior (SB) Name & Priority:

Squad Behavior (SB) Description:

IsWarranted():

IsFinished():

IsFailed():

GiveOrders():

Orders:

Responding Goals:

- **Goal1**:
  - Condition1: < “condition”, object value>
  - Condition2: < “condition”, object value>
  - …,
- **Goal2**:
  - …,

Actions:

Nodes:

Issues:
