Patrolling Example

Squad Behavior System (SBS):

The Squad Behavior System is a manager and coordinator of how squads of agents are to behave, and at any time, a squad is to have zero or one squad related behavior.

This system upon initialization (Awake()) will create a new list of IReGoapAgents called "squad", and a list of List<IReGoapAgent<T, W>> called "squads". The system will be responsible for administering squad behavior per order to each squad created in the Game Scene. Every agent within the game scene upon awakening (Awake() in ReGoapAgent) will send a reference to their IReGoapAgent component to the SBS, so that the SBS can add them to a Dictionary of living agents, also the SBS will have access to the world state of each agent's IReGoapMemory component.

After all living agents are added to the dictionary "Alive", upon Start(), the

- Dictionary of alive agents with an associated value of their respective squad. Followed by a list of squads.
- Inside the SBS, when Awake() runs, the list of squads and dictionary of agent-squad pairs are initialized.
- For every agent upon Start(), all agents will send a reference of themselves for the SBS to place within the Dictionary of alive agents. Each agent will be added to a squad, based on their proximity to other agents. Agents not contained in the *aliveAgents* dictionary, and not within any proximity of agents in any squad in the dictionary are then added to their own squad in the dictionary. Otherwise, an agent is added to a squad where it's in proximity to another agent.
- **Merging squads**: Squads that are not full and whose sum of agents equals or are less than the set maxSquadSize will merge. But, agents of separate squads must be within proximity of each other too!
- There is a struct called ***squad*** that keeps track of a list of agents inside the squad, and a placeholder for the squad behavior. If the SBS were to give a squad an order, it would call the squad struct's GiveOrders(string behavior) method and pass a behavior represented by a string. Depending on the passed behavior, the squad struct will retrieve the memory world states of each agent and decide what orders to set for each agent based on the retrieved memory states. This is done through the **OrdersSensor**.
- What specific orders are handled by the squad struct!
- Additionally, the squad struct has an instance that keeps track of how long it's been since the behavior was last assigned. If this time were to surpass some threshold and squad behavior conditions weren't met for each agent in the squad, then the squad behavior fails. Otherwise, a new squad behavior is determined for a squad and orders are given.

Squad behaviors could also fail based on other conditions too before reaching the threshold, such as a whole squad being wiped out!

- There are defined goals added for the agents that can accomplish any respective order given by the SBS. Orders are given by the SBS by setting to the agent's memory <"order", string SomeOrder>, and if a squad behavior goal were to detect this in memory for IsValid() inside the planner, then the priority of the goal will be calculated (Note: this priority calculation considers the agent's Threat Level meter).
- When a squad behavior goal is finished, then the last action will remove from the memory <"order", string SomeOrder> to signal that the agent has followed orders.
- The SBS upon every Update() will check each squad from the list of squads, and iterate through each squads' list of agents checking if the <"order", string SomeOrder> condition has been removed from their memory. If all agent's don't have this condition in memory before the time limit, then the squad behavior is a success.
- SBS will then proceed to give a new squad behavior to a squad either they finished or failed their goal (unless completely wiped out). The given squad behavior depends on threat level.
- When removing an agent when deceased, look for its associated squad inside the aliveAgents dictionary, and inside the associated squad struct, remove that agent from the list. Check if the squad's agent count is still more than zero, otherwise, remove not only the agent from the aliveAgents dictionary and its associated squad value, but the squad from the squads list as well! This is done inside Update() too.
- SBS handles what NavMesh areas to consider (by default agents consider only walkable areas) when traversing with the Unity AI Navigation system. SBS on Update() keeps track of where the player is, and if an agent were to see the player, the overall squad Threat Level will increase and some squad behaviors won't even be considered.
- If the player were to shoot in some corridor then the preemptively baked area within the corridor with Area Type Danger Zone Corridor 1 (example) will be considered by all agents in the squad, and Threat Level meter of the SBS will increase based on player's actions!

PatrolPoint:

- A patrol point is simply a GameObject with a child capsule and the component class PatrolPoint. The class PatrolPoint has a name and a bool flag to determine whether it has been investigated or not. Agent's action "Investigate" will tick this flag to true if at the Patrol point position.
- PatrolPoint also has references to other PatrolPoints, because a PatrolPoint can be set at junctions or cross roads, so squads patrol may have to split up to cover more ground. Handling which PatrolPoint nodes Agents in a squad are to go to is handled by the *squad struct*.

PatrolPointsManager:

- Manages all PatrolPoints within the GameScene, and if all Patrol points were to be investigated, then after some set delay, all patrol points will be ticked as not investigated. Agent's sensors reference this Manager in order to set all PatrolPoints into memory.

Sensors:

- **PatrolPointSensor**: Update to memory all patrol points and a stack of Patrol Points that have been investigated, and can be expanded on.
- **PositionSensor**: Updates to memory the position of the agent.
- **OrdersSensor**: Update to the memory orders given to the agent by the SBS to accomplish a squad behavior. Can remove orders and associated conditions, as the value of an order is the key of an associated condition. Orders will be removed if…
- ThreatLevelSensor: Updates to memory the threat level the agent is experiencing. For example if the agent hasn't seen the player yet and, or no agent has seen the player, then the Threat Level is zero. Meanwhile, if the agent sees the player, then the Threat Level increases, and will increase drastically if the agent were within line of sight of the player and if the player were to shoot. By default at the start of the game this Threat Level is zero, and could either increase or decrease on a meter 0-100 of type float.
- PlayerSensor: Updates to memory the last known player location, either sensed by this agent or other agents. The SBS will collect the data and send it to all other agents!
- HealthSensor: Update to the memory the health status of the agent. 1-100.

Squad Behavior:

- Patrol all goals seen by agents in the Game Scene!
- Failed if agents couldn't patrol all goals within 2 minutes.

Goals:

- **PatrolGoal**: Given some PatrolPoint by the SBS have the agent go and investigate the area. SBS sets to memory <"order", "Patrol"> and <"PatrolPatrol", PatrolPoint patrol_point>, then in the planner, when checking IsValid() goal, the method will search in memory for a condition key "order", then will check the associated value "Patrol". The goal state would then be <"investigatedPatrolPoint", true>, and the agent must head to the given PatrolPoint.
- FollowGoal: Given some IReGoapAgent to follow by the SBS, the agent will follow the given agent. SBS will be set to agent's memory <"order", "Follow"> and <"FollowAgent", IReGoapAgent>, and similar to PatrolGoal, FollowGoal is only valid if these conditions are written in memory. These orders and associated conditions are added and removed only through an Order Sensor. The squad behavior Patrol succeeds as long as all Patrol Points have been investigated. All orders are removed if a squad behavior fails or succeeds!

Actions:

- **GoToAction**: specify a location or Transform.Position for the NavMeshAgent component to go to. This can be used to follow an agent or go to some generic location like the position of some patrol point.