Unity Project Documentation

The Unity Project

Unity version: 2020.3.9

You will be provided with a simple Unity level, this level will contain two teams of AI agents, the flags, which start in the friendly base, some health kits and power ups which can be collected, they will respawn after 5 seconds, as well as a base for the AI agents to drop the flag to earn points. The project includes a code framework which implements various methods and properties that allow an AI agent to move to a location within the AI's visual range, detect and collect objects, randomly wander the level and attack opponents. These properties and methods are detailed below and this information is duplicated in the provided Unity project at the top of the file you will be editing.

There are three members in each team, if a team member dies they will respawn after 5 seconds. The objective of the game is for each team to attempt take the opponents' flag from their base and drop it in its own home base. If any team member, friendly or enemy, dies they will drop the flag. While the enemy flag is within the area of the friendly base (not while being carried, the flag must be dropped in the base) the score for that team will increment. The score for both teams will be indicated on the screen. There is no winning condition so the game will continue until the user stops it, this is to aid debugging.

The only script file you need to edit is `AI.cs` which contains a framework for your AI code. There are several other files providing supporting code which you are not encouraged to edit unless absolutely necessary. `AI.cs` has access to the AI agents' actions, senses and data through three member variables called `_agentData` which is of type `AgentData`, `_agentActions` which is of type `AgentActions`, `_agentSenses` which is of type `Sensing` and `_agentInventory` which is of type `InventoryController`. All of these are script components and the scripts are viewable in the folder AI Support under the Scripts folder.

Do not change the basic gameplay, but you can annotate the map and all the objects in it if you want to use that information for tactical purposes or make other small changes for the use of your AI. You may also place triggers if required by your AI.

The Code API

The script variables include the following methods, properties and public variables, you may look at the source code and Unity inspector for more details:

Predefined constants for Unity names and tags
Use these to access objects in your scripts

Unity Tags	
`public static class Tags`	
`public const string BlueTeam = "Blue Team";`	The tag assigned to blue team members.
`public const string RedTeam = "Red Team";`	The tag assigned to red team members.

`public const string Collectable = "Collectable";`	The tag assigned to collectable items (health kit and power up).
`public const string Flag = "Flag";`	The tag assigned to the flags, blue or red.

Unity GameObject names	
`public static class Names`	
`public const string PowerUp = "Power Up";`	Power up name
`public const string HealthKit = "Health Kit";`	Health kit name.
`public const string BlueFlag = "Blue Flag";`	The blue teams flag.
`public const string RedFlag = "Red Flag";`	The red teams flag.
`public const string BlueTeamMember1 = "Blue Team Member 1";`	Blue team member 1.
`public const string BlueTeamMember2 = "Blue Team Member 2";`	Blue team member 2.
`public const string BlueTeamMember3 = "Blue Team Member 3";`	Blue team member 3.
`public const string RedTeamMember1 = "Red Team Member 1";`	Red team member 1.
`public const string RedTeamMember2 = "Red Team Member 2";`	Red team member 2.
`public const string RedTeamMember3 = "Red Team Member 3";`	Red team member 3.

`_agentData` properties and public variables	
`public string AgentName`	The individual name of this agent. This is the same name as the GameObject name.
`public bool IsAlive`	Check if the agent is alive, returns `true` if agents alive, `false` otherwise.
`public bool IsPoweredUp`	Have we powered up, `true` if we're powered up, `false` otherwise.
`public int CurrentHitPoints`	Our current hit points as an integer
`public bool HasFriendlyFlag`	True if we have collected our own flag
`public bool HasEnemyFlag`	True if we have collected the enemy flag

`_agentActions` methods	
`public bool MoveTo(GameObject target)`	Move towards a target object. Takes a `GameObject` representing the target object as a parameter. Returns `true` if the location is on the navmesh, `false` otherwise.

`public bool MoveTo(Vector3 target)`	Move towards a target location. Takes a `Vector3` representing the destination as a parameter. Returns `true` if the location is on the navmesh, `false` otherwise.
`public bool MoveToRandomLocation()`	Move to a random location on the level, returns `true` if the location is on the navmesh, `false` otherwise.
`public void CollectItem(GameObject item)`	Pick up an item from the level which is within reach of the agent and put it in the inventory. Takes a `GameObject` representing the item as a parameter.
`public void DropItem(GameObject item)`	Drop an inventory item or the flag at the agents' location. Takes a `GameObject` representing the item as a parameter.
`public void UseItem(GameObject item)`	Use an item in the inventory (currently only health kit or power up). Takes a `GameObject` representing the item to use as a parameter.
`public void AttackEnemy(GameObject enemy)`	Attack the enemy if they are close enough.). Takes a `GameObject` representing the enemy as a parameter.
`public void Flee(GameObject enemy)`	Move in the opposite direction to the enemy. Takes a `GameObject` representing the enemy as a parameter.

`_agentSenses` properties and methods	
`public List<GameObject> GetObjectsInViewByTag(string tag)`	Return a list of objects with the same tag. Takes a `string` representing the Unity tag. Returns `null` if no objects with the specified tag are in view.
`public GameObject GetObjectInViewByName(string name)`	Returns a specific `GameObject` by name in view range. Takes a `string` representing the objects Unity name as a parameter. Returns `null` if named object is not in view.
`public List<GameObject> GetObjectsInView()`	Returns a list of objects within view range. Returns `null` if no objects are in view.
`public List<GameObject> GetCollectablesInView()`	Returns a list of objects with the tag `Collectable` within view range.

	Returns `null` if no collectable objects are in view.
`public List<GameObject> GetFriendliesInView()`	Returns a list of friendly team AI agents within view range. Returns `null` if no friendlies are in view.
`public List<GameObject> GetEnemiesInView()`	Returns a list of enemy team AI agents within view range. Returns `null` if no enemies are in view.
`public GameObject GetNearestEnemyInView()`	Returns the nearest enemy AI in view to the agent. Returns `null` if no enemies are in view.
`public bool IsItemInReach(GameObject item)`	Checks if we are close enough to a specific collectible item to pick it up), returns `true` if the object is close enough, `false` otherwise.
`public bool IsInAttackRange(GameObject target)`	Check if we're with attacking range of the target), returns `true` if the target is in range, `false` otherwise.
`public Vector3 GetVectorToObject(GameObject target)`	Return a normalised vector pointing to the target
`public Vector3 GetVectorToObject(Vector3 target)`	Return a normalised vector pointing to the target vector
`public Vector3 GetFleeVectorFromTarget(GameObject target)`	Return a normalised vector pointing away from the target
`public Vector3 GetFleeVectorFromTarget (Vector3 target)`	Return a normalised vector pointing away from the target vector

`_agentInventory` properties and methods	
`public bool AddItem(GameObject item)`	Adds an item to the inventory if there's enough room (max capacity is 'Constants.InventorySize'), returns `true` if the item has been successfully added to the inventory, `false` otherwise.
`public GameObject GetItem(string itemName)`	Retrieves an item from the inventory as a GameObject, returns `null` if the item is not in the inventory.
`public bool HasItem(string itemName)`	Checks if an item is stored in the inventory, returns `true` if the item is in the inventory, `false` otherwise.

You can use the game objects name to access a `GameObject` from the sensing system. Thereafter all methods require the `GameObject` as a parameter.