**Designer Document**

**Player:**

NetworkPlayer:

This helps units identify their owner player and its client status. Ensure that this is on any player prefab.

NetworkHandler:

This is used to abstract server and command functions, and is a required component of a NetworkPlayer so should also be placed on any player prefab.

**Resources:**

Mod\_Resource:

A scriptable object that holds data for a resource, this makes it easy to create new resource types for any designer that works with this modular system. It also has UI data such as the icon and name of the resource.

Mod\_ResourceManager:

This manages all of the players’ resource data across the network and locally, to ensure that the server knows how much resources each player has (since all actions are performed on the server) and all clients know their own resources so that their UI can display them.

Mod\_ResourceCollection:

A script to blace on any resource pile (an object that will allow a worker to mine resources from it), it holds the data for how many resources the pile has and how much is replenished over time.

**Selection:**

Selector:

Handles the player’s ability to select units, using single/multi select variants. It also holds the data for all selectable objects currently in the scene.

ISelectable:

Interface for selectable objects, holds functions for OnHover, OnSelect, OnDeselect, SetHover, SetSelect, SetDeselect. This allows for different effects to happen based on the namesake of the function through the Selector script.

SelectableCallback:

Exposes unity events for the different ISelectable functions, is handy for designers to add their own effects to occur.

DefaultSelectable:

A default implementation of the ISelectable interface is simply for ease of use as most ISelectable implementations will have the same basic implementation.

**UI:**

SelectorUI:

Displays the selection rectangle when the player is attempting a drag select to select multiple units.

Mod\_ResourceUI:

Displays the current list of resources in the game with each player’s specific amount that they currently have. Works in tangent with Mod\_resourceManager to receive updates for when to update the UI for the resources.

ActionUIManager:

Manages the UIs for current action, queued actions and possible actions of the first selected unit that is currently selected. It will also determine when to turn off the action UI canvas when there is no currently selected unit, and will update the UI when the Selector tells it that there has been a change.

DisplayCounterUI:

A UI script that will display the text and image of some sort of counter, for example is utilised in the Health and Mod\_ResourceCollection scripts to display the amount of life and resources left, respectively. Will change the fill amount of the image based on the value passed when updating the UI.

UIDisplay:

Is used to display an image and text GameObject, essentially instantiates the prefab and gets the image and text components. It will then display the instantiated prefab as a child of the input parent GameObject.

**Units:**

AIManager:

This works as a more efficient way to have the server update AIAgent’s, it means that only a certain percentage of the total AIAgent’s in the scene will be updated every predetermined amount of time.

This is required to update AIAgents, but can be set to just update them every update.

AIAgent:

The brains of the AI, all units will require this so that they can process and acquire AIActions. The framework for self-deciding AI is there; it just needs to be implemented in the AIActions themselves. AIAgent receives AIActions from clients passed to the server and updates them server side.

AIAction:

Utilising these can allow an AIAgent to perform actions such as: move, attack, build etc. To make new subsets of AIActions you can combine already existing ones easily enough in the Actions.cs script, or make completely new ones. After a new implementation is made, you will need a complimentary AIAction ScriptableObject class to manage the new Action.

UnitHandler:

Specifically used for player controlled AIAgents, the UnitHandler handles the distribution of AIActions to AIAgents by the player respective to their button presses. If an AIAgent won’t be controlled by the player then this is not necessary.

DefaultUnitHandler:

Handles all default AIActions that can be performed by all other UnitHandlers in the scene. It also allows for the ability to clear all actions on currently selected AIAgents. The DefaultUnitHandler allows units to draw from the same ‘move’ AIAction or something similar to decrease the need of re-entering the same data into fields.