**How Turrets Select Targets:**

1. **Changes to TurretManager:**

Inside the TurretManager script there is an array of integers called “classification”.

This array holds the values that the user will be given at the start of the round:

e.g, “All code with Speed = 3, Attack = 4 must be stopped” => classification holds {3 , 4}

This array is public and can be accessed from anywhere using TurretManager.instance.classification.

Values of this array are returned by calling the getClassification(int index) function within the TurretManager script, where index corresponds to the cell of the array to retrieve.

1. **Changes to Enemy:**

Enemy now has a public integer array called “properties”. This will store the properties of an enemy such as “Speed”, “Damage”, “Language (Java, C etc..)”, “Size (MB, GB etc..)”. These are ultimately represented as integers. When an Enemy GameObject is created (Start() function), two things can happen:

* The enemy properties array is filled randomly (within a range). This corresponds to creating a GameObject that will not be shot by turrets. This has not been implemented yet.
* The enemy properties array is filled with the conents of the classification array stored in TurretManager. This corresponds to creating a GameObject that matches the “Enemy” specification given to the user at the start of the round. If the user has correctly programmed their towers by following the specification, then their turrets WILL shoot this enemy.

1. **Changes to Turret:**

Each Turret now has a public array of integers called “userVariables”.

This array holds the values that the user will “program” into the turret during the coding phase.

e.g: through drop boxes, the user specifies that a turret template should shoot “IF Speed = 3 AND Damage = 4” => userVariables holds {3, 4}.

Turrets no longer populate their “enemies” array with GameObjects tagged as “Enemy”. Instead, turrets search for all GameObjects tagged as “code”.

Previously, a turret would check if the enemy within its range is the nearest enemy. An additional condition has been added to this check. The new function “checkIfEnemy(int[] enemyProperties)” is called, which takes the “properties” array from the current enemy being examined. This function checks, cell by cell, the contents of the “userVariables” array against the contents of the “properties” array held by the current enemy. If the contents of these two arrays are found to differ, then a 0 is returned (“The properties of this enemy do not match what the user told the tower to shoot at”). Otherwise, a 1 is returned (“The properties of this enemy DO match what the user told the tower to shoot at!”). A return value of 1 stores the enemy as the nearest enemy to shoot at, just like how it worked previously.

1. **What will need to changed / considered in the future:**

* The userVariables array is currently public, and edited within Unity for the purposes of testing. In the future, once the tower programming feature has been implemented, the contents of userVariables will need to be collected from the coding interface, and should differ between towers on the users taskbar (so different towers can target different enemies)
* The classification stored in TurretManager should perhaps be moved to its own class, but it should still be a Singleton class. The contents of classification need to be able to change between rounds, which ties in with whatever method we are going to use to control the flow of the rounds.
* This allows turrets to target different enemies. It may make the program look non-functional: towers might not be shooting when you expect them to. To make sure a tower will shoot an enemy, change the contents of classification in TurretManager to two integers of your choosing (set to 3, 4 currently), and ALSO set each turret prefabs User Variables to match those integers. This is done within the Unity editor, not within a script.