Mission 15: Searching and Sorting IV

Start date: 22 October 2017
Due: 25 October 2017, 23:59

Readings:

- None

IMPORTANT WARNING:

For this mission, we will be using an external program to aid in the visualisation of our solution. As a result, there are some additional constraints applied to the solutions for this mission. In this mission, we do not permit the declaration of any other functions other than the one already provided by the template solution.

Background

Members of the Source Academy generally favor peace over conflict. Consequently, the bulk of the Academy's lessons using the power of the Source for peaceful purposes - from the Arts of Runes and Curves, to the use of Sounds for entertainment and emergency communications.

Unfortunately, conflicts are a very real thing, and they do exist outside the Academy. As such, even if the Academy is going through peaceful times now, that does not mean that the peace should be taken for granted. With most of its cadets adept at the basic survival drills now, Grandmaster Hartin believes that it is time to equip its cadets with skills to deal with hostile beings and dangerous situations, that otherwise cannot be easily resolved through peaceful methods.

This mission has **four** tasks. Please remember to press "*Save*" button before you submit this mission.

Task 1:

In this simulation, you will be need to demonstrate the ability the neutralise all hostiles in record time. In other words, your hostiles neutralisation plans cannot afford any wasted movement! To accomplish this, it is your belief that the most efficient method would involve making a single, full spin counter-clockwise, while shooting at enemies as they enter into your crosshair.

We represent an enemy an an array of length 2, containing an id as its first element, and an angle as its second element. The angle represents some measure of counter-clockwise spin that you need to turn, from the original direction, to put the enemy into your crosshair. Intuitively, if `enemyB` requires you to turn more from the starting direction so that you are facing it, compared to `enemyA`, than `angleB` should be greater than `angleA`. You should **not** assume that the angle is provided in any standard measurements (e.g. radians, degrees, etc). You can obtain the angle of an enemy using the `getPlanarAngle` method, by passing it an enemy as argument.

Before diving deep into the process of sorting enemies by their angle, Grandmaster Hartin advises you to start with a simpler `compareEnemy` function. The `compareEnemy` function takes in `enemyA` and `enemyB` as arguments, and returns either:
`-1` if `getPlanarAngle(enemyA) < getPlanarAngle(enemyB)`
`0` if `getPlanarAngle(enemyA) === getPlanarAngle(enemyB)`
`1` if `getPlanarAngle(enemyA) > getPlanarAngle(enemyB)`

```
Example:
var enemyA = [1, 0.5];
var enemyB = [2, 0.9];

compareEnemy(enemyA, enemyB); // returns -1
compareEnemy(enemyB, enemyA); // returns 1
```

For this task, complete the template `compareEnemy` function.

Task 2:

With the compareEnemy function completed, you are now ready to implement your sorting algorithm. For this task, you strongly believe that `quicksort` will give you the best performance. In order to implement `quicksort`, you will need to, first, define a `partition` function.

Since we want to partition the array of enemies by our own `compareEnemy` functions, defined previously, we will need to make a small modification to the original `partition` function. Hence, define a `partition` function that takes in an array of comparable values (such as enemies), the start index `low`, end index `high`, and a comparator function. In this case, our comparator function is our `compareEnemy` function that was defined previously.

The `partition` function should return the new index of the pivot element after partitioning. The `partition` function also mutates the array in place, to produce the partitioned array. The choice of the pivot is left to you. Hence, you should also clearly explain and justify your choice of pivot for your `partition` function.

```
Example (with last element as pivot, on the entire array):
var enemiesA = [[1, 9], [2, 1], [3, 4], [4, 6]];

// Last element is [4, 6] (our pivot)
var pivotIndexA = partition(enemiesA, 0, 3, compareEnemy);

// Valid values are [[2, 1], [3, 4], [4, 6], [1, 9]]
// or [[3, 4], [2, 1], [4, 6], [1, 9]]
enemiesA;
```

```
// Pivot Index is 2, and Pivot value is [4, 6]
pivotIndexA;

Example (with last element as pivot, ignore last 2 elements):
var enemiesA = [[3, 4], [2, 1], [4, 6], [1, 9]];

// Last element is [2, 1] (our pivot)
var pivotIndexA = partition(enemiesA, 0, 1, compareEnemy);

// Only valid value is [[2, 1], [3, 4], [4, 6], [1, 9]]
enemiesA;

// Pivot Index is 0, and Pivot value is [2, 1]
pivotIndexA;
```

For this task, complete the template `partition` function.

Task 3:

At last, you are ready to define your `quicksort` function. Your `quicksort` function should take in 4 arguments – an array of comparable elements (such as the array of enemies), the start index `low`, the end index `high`, and a comparator function to pass to the `partition` function (such as our `compareEnemy` function). The `quicksort` function should mutate the array into a sorted one.

```
Example (using compareEnemy):
var enemiesA = [[1, 9], [2, 1], [3, 4], [4, 6]];

// [[2, 1], [3, 4], [4, 6], [1, 9]];
quicksort(enemiesA, 0, array_length(enemiesA) - 1, compareEnemy);
```

For this task, complete the template `quicksort` function.

Task 4:

Note: This task is not graded.

Note: The simulator runs on UnityScript instead of the Source, which has some additional language features that you might not have seen yet. You will not need to deal with these constructs for this mission, so they can be safely ignored.

After implementing the algorithm, it is time to test it out in the simulation room. To do that, download either the Simulation Program for Mac or the Simulation Program for Windows.

The mouse can be used to change the direction of the character's view. The `Tab` key is used to teleport inside and outside the spaceship. Use the `W, A, S, D` or the `Up, Down, Left, Right` keys to navigate around the room.

Enter the inside of the spaceship and look for a red button somewhere. When you have found the red button, click on it to enter the simulation. Copy and paste the entirety of this task (including function headers) into the panel. Next, click the `Submit` button, followed by the `up` arrow button located at the bottom of the screen, to see the simulation in action.

If you want to check the result of doing `addTargets(...)` in the Academy, you can call `getTargets()` in the interpreter.

Submission

Submit your mission on the Source Academy.

IMPORTANT: Make sure that everything for your programs to work is on the left hand side (Editor) and not in the Side Content! This is because only the programs in the Editor are preserved when your Avenger grades your submission.