Programming test Report

()

January 29, 2021

1. Refactor

1. Describe what the goal of the code is: When we instance Runner, we can let instance Runner has several Entities. The entity here stores the basic parameters: HP, speed, and Armor a runner can have. In the program, we can instance Runner once, by switching its Entity or randomize a stateModtoAll to have various type of runner.
2. Describe potential issues that you see with that implementation:
   (a) To use method GetVal we need to instantiate its class in begin. The calss only help to compute value in somewhere.
   (b) functions in Class StatModHandler are helper functions, we don't have to instance it to access member.
   (c) Class StatModHandler AddStatMod always instantiate a new StatMod and add into old list it. When using GetValue, we need to iterate all items in the list to compute the final the result.
   (d) The contents of AppliedStatMods is accessed by foreach too many times. In the default, list contains 3 states, however as AddRandomStatToAll is called, method always add new state into list without checking. The worst case for compute the GetValue is O(MN)(consider size of Entity).
   (e) Switch doesn't have default, it cannot handle an unexpected value.
   (f) Entity exposes many members to public.
3. Describe possible solutions on how to solve some or all of the aforementioned issues.
   (a) Use static function and variable in Randomizer/StatModHandler class, then we don't have to instance in begin of program or through instance to use method.
   (b) Use the Dictionary to store AppliedStatMods and when add StatMode, check whether idnet is exist then apply operation to reduce GetValue Computing time.
   (c) Change AppliedStatMods to dictionary type, remove the StatMode by idnet key.

2. The Ticker

Test Scripts: TickerTest, ExampleClass
The Log message of Coroutine always comes after the update. In the Unity order of execution, the Coroutine executes after Update. In the TickerTest Class, Coroutine loop we adding the waiting x-seconds for the timer variable, the adding won't complete at once, when finishing one iteration, it waits for x-second and return the main thread by yield return to Unity, then after x-second, it continues to execute the next iteration in the loop.

3. Mover

Using the interface to refactor the Mover, You can introduce new pathfinding method without having to change the context. Suppose each move method using different pathfinding strategy. The strategy needs to be initialized and be called in Fixedupdate to parse the deltaMovment back.

Basically, fly and walk can be done by feeding different ignoredLayer params(If they using the same pathfinding strategy, then we don't need to implement another script). Teleporting path finding strategy is quite different from others, it requires Coroutine from Monobehavioiur with implemented interface. To minimize the changing of old code , all move strategies inherit Monobehaviour, when we need to switch to Teleport method, we can through addComponent to instance it, then during the fixedUpdate we don't have to change code, we can still call the Interface method: UpdateMovment and it will have different behavior to others.