Sample 4 readme

Author: Song Jian

Code Creation Time: March 2018 - April 2018

This sample is written in C# based on Unity3D.

This sample is the whole project of a tower-defense game. Aiming at refactoring (this is only the second time I tried to rewrite the project, and the third is still better but I do not with to post it up because of the incompleteness) the aforementioned TD game, the gameplay experience would be mostly the same as the last one, but several programming details are tremendously improved.

- Each scripts are contained in a specific folder and the folder indicated its aim, such as **EnemyBuff** folder contains several common enemy debuffs while debuffs delimated by perks are still writen in the perk file.
- Each perk is now seperate files instead of a wagonload of if clauses and function invocations. So are the properties and abilities. This means the encapsulation is much better, and a collaborator shall not need to edit some base classes to build a new perk.
- Classes initialization is divided into self-initialization (common operations) and interactive-initialization (mostly event registrations). In fact, these classes are mostly singletons (only one object of such a class shall exist spontaneously). This has been improved (in the rebuild third version), where an explicit self and interactive intialization order is defined in a **GameMode** class (I learn this from Unreal Engine and move it here, and it proves correct) class, and singletons are accessed by static readonly properties instead of public fields.
- **EnemyBuff** is now a solitude class rather than function invocations. Three important properties of this class are listed below. Here is how this class works: `startBuff()` is called when such an `EnemyBuff` (debuff actually) is attached to an `Enemy` and therefore becomes active. If the buff is not an instant one, a timer will be intialized and `inBuff()` is called every frame (with `Update()`) and eventually `quitBuff()` is called when the timer expired. In this way, nearly all effects in a TD game can be implemented based on the one and only template. Details are elaborated in the code and examples are written in **ContinuousDamage.cs**, **DecelerateEffect.cs**, **ExplosiveEffect.cs**, and **StunEffect.cs**.

```
public delegate void InstantBuff(Enemy enemy);
public delegate void ContinuousBuff(Enemy enemy, float time);
public InstantBuff startBuff;
public ContinuousBuff inBuff;
public InstantBuff quitBuff;
```

You may also clone the whole project from github. The newly rebuild project can also be downloaded.