TJ Cioffe
GAM 655
Midterm Project
Prof. Greene

Code Requirements Documentation: Tower 2Defense

Functions

SpawnManager's SpawnEnemy() function is used to Instantiate a new Enemy. Used by 3 functions:

- SpawnRoundEnemy(): called from Update(); calls SpawnEnemy() and sees if the State Machine should change states afterwards
- SpawnEnemies(): called to spawn enemies outside the Update() loop. Starts a Coroutine to spawn multiple enemies.
- Update(): SpawnEnemy() is also called directly to spawn the Boss enemy at the end of a round, if applicable.

Loops

Enemy's BuildPath() function uses a loop to create an ordered List of Path Tiles for the enemy to follow. Depending on the enemy's proximity to the destination, the loop will run between one and a number equal to the total Path Tiles in the level times. Each loop, it finds the closest Path Tile and adds that to its follow list, while removing it from the hopper of tiles that it examines (ensuring that the Path always moves forward).

Collections

TilePath objects will find the PathManager upon Awake()-ening and add themselves to a List<TilePath> containing all Path Tiles in the level. This is then used by Enemy objects to calculate their path.

Class Inheritance

Projectile offers a virtual method, OnImpact() which can be overridden by inheriting objects to change what happens when the Projectile hits its target. This is used by ProjectileGlue to slow the enemy's speed, by ProjectileMissile to use OverlapCircleAll() to deal splash damage, and ProjectileBomb to tell the Enemy to walk in the opposite direction and explode after a period of time.

State Machine

SpawnManager utilizes a state machine to track whether it is Waiting (the pause between rounds), Spawning, or Attacking (all enemies have been spawned and are heading to the player home tower). This GameState affects what happens when the enemy count is updated (whether or not to check for the round's end), and determines whether the code in the Update() block runs on a given frame or not.

Public/Private

Enemy's EnemyAction fields (OnUpdateAction, OnDieAction, OnOverrideAction) allow enemy behavior to be easily defined via the Inspector or swapped out at runtime. OnUpdateAction and OnDieAction are only meant to be set via the Inspector and are set to Private Serializable Fields. OnOverrideAction can be set by code (for example, the EnemyBomb action that causes an enemy to self-destruct) and runs immediately upon being set (i.e., it doesn't require an initialization call), so is available as a Public property.

Comments

Code is commented for clarity of purpose and readability throughout, with Header sections used to further organize the Inspector.