Final Project Part 2

Out: Monday, November 6th 2017

Due: Monday, November 13 2017 at Noon (11/13/2017 at 12:00 PM)

This assignment will build on the work that you did last week. Use your own code as starter code.

Before you do anything else, set your project's aspect ratio to 4:3.

Introduction

This part of the final project will build on the work that you did last week. You did a decent amount of scaffolding last week, but it's time to make your game look like a proper tower defense. This week, we're going to focus on five things:

1. earning money for defeating enemies (and the associated UI)
2. spending that money on building new towers (and the associated UI)
3. having the enemies pathfind around the towers that you build
4. enemy waves (and the associated UI)
5. victory and defeat (...)

Let's get started.

Getting Money

First things first, we need some cash. As of right now, the lone tower that we have just sits there and blows enemies away as they come out of the spawn point. This is fine, but it's kind of boring. Let's add money. Here's what you need:

1. A UI element in the top left of your screen that indicates how much money a player has
2. Defeating enemies (i.e., reducing their health to zero or below) destroys them as before but also earns the player a sum of money
    a. It's basically impossible at this point to decide *how* much that sum should be, since you haven't decided on a lot of other things. Just pick a number for now.
3. Give the player some starting sum of money (so they can build a couple of towers at the beginning of the game)

Spending Money (and Adding Menus)

Our players are all dressed up with nowhere to go. We have loads of cash, but nothing to spend it on. Enter the build menus. Clicking on a cell will display one of two things:

1. If there is no tower on that cell (i.e., it's empty), a build menu will pop-up, offering the player the chance to spend their hard-earned cash to place a tower there
    a. hide this menu when a buildable cell is not selected
    b. add a button with the option to build that cell; this adds a new tower there and decreases the player's money by the cost of that tower
        i. if the player cannot afford to place a tower, gray out the button (see Button.Interactable)
    c. upon building the tower, close the menu and deselect the cell
2. If there is a tower on that cell, show an "Upgrade/Sell" menu, offering the player a chance to upgrade the tower in question or to sell it for a fraction of its original value
    a. hide this menu when an "Upgradeable" cell is not selected
    b. for now, just add a button that sells a tower, destroying it and giving the player back 90% of its value
    c. upon selling the tower, close the menu and deselect the cell

Some other things about selection and menus:

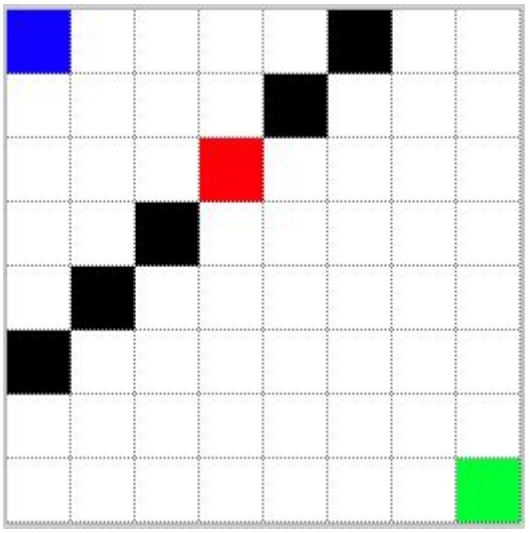
1. Clicking on the same cell twice should close whatever menu was opened and deselect that cell
2. Make sure that the player *cannot* select the cells associated with the spawn point or the base - we don't want to build towers where there are permanent structures!

After this is done, you should be able to build towers on 23 of the 25 cells in your grid. Go ahead and remove your "starting tower" since the player can now build towers.

Pathfinding Proper

You may notice that if you build a tower in the path of the enemies (which are currently just walking straight towards the base), the enemies walk either into or through the towers, depending on how you implemented their movement. Let's fix this by adding proper pathfinding. Unity has a built-in pathfinding system that you can feel free to use, or you can use whatever pathfinding algorithm you want (I'd recommend A*) and do it yourself if you want a challenge. Here's what you need:

1. Enemies need to walk in a path that does not cause them to collide with any towers
2. Adding or removing a tower should cause enemies to recalculate their path
3. Change tower construction so that players may **no longer** place a tower that prevents enemies from reaching the base (see below for an image)
    a. they also may **not** place a tower if there is an enemy on the cell where they are trying to build that tower



the player may not build in the red square (assuming that the black tiles are towers)

Enemy Waves

In a typical tower defense, enemies don't spawn infinitely, one after the other, *ad nauseum*. They tend to come in *waves*: set groups of enemies that come, one after another, until the wave is finished. After each wave is a break for the player to place/upgrade towers (using the money that they earned during the last wave) and - for fancier games - to repair features of the level or do other fiddly things. We're going to think about waves as having the following features:

- a time until the wave arrives (this will be represented by the wave icon's position along the bottom of the screen)
- the number and types of enemies that appear in that wave, and in what order they appear
- the amount of time between each enemy spawned (smaller delays for large groups; larger delays to give the player breathing room)
- how strong each of those enemies is, as indicated an arbitrary value (which increases with wave number)
    - enemy health and value (i.e., the money the player gets for defeating them) should increase with this value (in other words, later waves are stronger, but also worth more money)

There are a lot of ways to represent waves, but the cool thing about Unity is that you can make almost anything into a component, including an abstraction like a wave. The simplest way to tackle this problem is to have a component with a bunch of public variables, including an array of Prefabs into which one can drop the enemies that show up in that wave (there are fancier ways of doing this, but they're beyond the scope of this class; Google "Unity custom inspectors" for more info).

If you decide to do things editor-side (no shame in coding everything), you'll have the freedom of being able to tweak all of the characteristics of a given wave without needing to change anything within your scripts.

Displaying Wave Information

So now we have waves, but it's hard to tell if they're working correctly - we can't even tell when the next wave is *supposed* to happen, let alone whether it's starting at the right time. Let's add some UI. The information about the waves will be displayed at the bottom of the screen, each wave showing up as an icon listing the type and number of enemies in that wave. Here is what you need:

1. A container at the bottom of the screen to hold everything. It should be the width of the screen, tall enough so that it can hold four lines of text, but not tall enough that it makes it difficult to see what's going on the grid
2. Each wave should be represented by an icon with (at most) four lines of text:
    a. "Wave #Number", where #number is, well, the wave's number
    b. A line indicating how many normal enemies are in that wave
    c. A line indicating how many fast enemies are in that wave
    d. A line indicating how many strong enemies are in that wave
3. The icon's position should be proportional to how much time is left until that wave starts
    a. thus, an icon representing wave that is just about to start should be all the way in the bottom left corner of the screen
    b. feel free to have wave's start offscreen (that is, if fitting five waves on screen is unrealistic, don't force it)
4. Once a wave starts, its icon should disappear

After you've done all of this, your enemies should spawn in waves (with a little bit of break between them, hopefully ;)) and there should be a corresponding UI element indicating what waves are coming up.

Winning (and Losing)

The game is missing something - victory and defeat. Fortunately, these two things are relatively simple: victory comes when every enemy has been destroyed, and defeat comes if the player's base ever reaches 0 hp. In the case of a "tie" (that is, the last enemy reduces the player's health to 0, defeat takes precedence over victory). For both victory and defeat, the game needs to "stop." This entails:

1. The player can no longer earn/spend money
2. The player can no longer build/destroy towers
3. The player can no longer win/lose (since, presumably, they already did one of these things)
4. All towers need to stop shooting
5. All enemies need to stop moving
6. The spawner should no longer spawn new enemies
7. The base's health should no longer change

It seems like a long list, but these are all (hopefully) simple to implement. Remember that removing a component (or disabling it) stops all relevant *Update*/*OnCollisionEnter*/etc. methods.
If the player wins:

1. Stop the game as outlined above
2. Display a giant "You Won!" sort of message on the screen

If the player loses:

1. Stop the game
2. Display a giant "You Lost :(" sort of message on the screen

Don't worry about restarting this game.

Submission

As per usual, zip up your Assets and ProjectSettings folders and submit **one copy** of the final project *as your Canvas* group.