Lesson 01 – Absolute Basics

Hello and welcome to my very first tutorial for Unitale wave creation! My goal in this series will be to teach you the basics of creating waves in Unitale. We will mostly work with the wave scripts only, but later on we will learn to manipulate wave behaviour from other scripts like Monster scripts or the Encounter script. I'm not going to explain LUA scripting here, so I will assume that you already know what an integer, a boolean, a string, a variable, an argument or any such expressions mean. You should probably know these when you start coding.

Part 1 – Projectile Management

Let's start with the very basics. To hurt the player, we will have to get projectiles. Projectiles are always above the UI, the Soul and basically everything else. To create a projectile you use the following function:

```
CreateProjectile( 'sprite' , spawnx , spawny )
```

Let's go through what each of the arguments here mean.

- **'sprite'** is the path to the sprite you want to use for your projectile from the Sprites folder. For example, if your sprite is named bullet and is located directly in your Sprites folder, use 'bullet', if it's in a subfolder called Silly_bullets, use 'Silly_bullet/bullet', and so on. The apostrophes (') are important, because you want to input the path as a string.
- **spawnx** and **spawny** are the x and y coordinates relative to the center of the arena. Positive numbers are right and up, negative numbers are left and down.

Note: You can also use CreateProjectileAbs('sprite' , spawnxabs , spawnyabs) as well, and in that case, x and y coordinates are relative to the bottom left of the game screen.

Now we know how to create a projectile. However, this will only make it appear in the game, but won't do anything yet. To make it move, we will need to use the Update() function. This function is run each time your screen updates, so anything that happens within happens at the same FPS your game runs in. If it runs in 30 FPS, Update() will happen 30 times each second and so on. Usually Unitale is capped at 60 FPS, but there are instances where it can run slower or faster. That can mess with the speed of your waves if you don't take this into consideration. A later tutorial will deal with speed handling. For now, we will just assume the game runs at a constant 60 Frames Per Second.

So, as I've mentioned before, Update() will handle moving our bullet. We will need to assign the created bullet to a variable, so we can call it later. For now, we deal with a single projectile, so let's just do the following at the start of our monster script:

```
bullet = CreateProjectile( 'bullet' , 0 , Arena.height/2 )
```

Note: see Arena.height and Arena.width in the DOCUMENTATION.

This will create the projectile with the ’bullet’ sprite at the top of the Arena, and assign it to the variable ’bullet’. Now we can call the bullet. For now, we want to do something very simple, like moving it downward with a constant speed. So let’s pull up that Update() function and put the following in it:

```
bullet.Move( 0 , -1 )
```

There are three ways to move a projectile in Unitale:

- Move(x , y) will move the projectile relative to it previous location. Again, positive is right and up, negative is left and down.
- MoveTo(x , y) will move the projectile relative to the center of the Arena.
- MoveToAbs(x, y) will move the projectile relative to the bottom left corner of the game screen.

Knowing that, this function will make the projectile we stored in the ’bullet’ variable downwards 1 pixel each time Update() runs, which for now we assume is 60 times a second.

Next, we are going to learn how to remove a projectile. If the projectile is off the screen, we don’t want it to use up valuable processing time, so we will get rid of it. Since this projectile only moves downwards, there’s only one check we need to do to make that happen:

```
if bullet.absy < -15 then bullet.Remove() end
```

If the projectile’s absolute y coordinate reaches -15, it will be removed. I use -15 here so that it doesn’t just pop out of existence once it reaches the edge of the screen, but continues downward a bit longer so we can’t see it anymore when it disappears.

However, now that we removed the projectile, it will give us an error if we try to move it, since it doesn’t exist anymore. For that reason, we are going to do a check before our Move function:

```
if bullet.isactive then bullet.Move( 0 , 1 ) end
```

Now you know how how to create a projectile, move it and remove it. I have put an example LUA file to show you my version of this beside this document called LESSON_01_01.lua.

Part 2 – Spawntimers and moving several projectiles

The second thing we will do in this lesson is working with spawntimers. If you don’t want all of your projectiles to be present from the beggining of the wave, you will need to create them sometime during the wave. For this, we will create a variable called ’spawntimer’ to keep track of time during the wave, and make it equal to 0, for now. We will increment this by 1 with each Update(). Next up, we want to create the projectiles only at certain times, like each 60 updates. The simplest way to do this is – by using this spawntimer method – is to check each update if the remainder of the spawntimer divided by 60 is 0, using the following:

```
if spawntimer%60 == 0 then DO_STUFF end
```

Now, let's see what the ’DO_STUFF’ part does. Here, we want to create the projectile. Since it is possible that we will have several of these at once, we can’t store them in only one variable, we will need a table for that. Let’s create a table at the start of the script called ’bullets’.

```
bullets = {}
```

Now, we will create the projectile and then put it in the table.

```
local bullet = CreateProjectile( ’bullet’ , 0 , Arena.height/2 )
table.insert( bullets , bullet )
```

I’m using local variables for bullet, because later on we will refer to it from within the table we created, and not from the original variable we created it into. While we are at it, let’s see how we can handle several projectiles at the same time. That’s why we made this table in the first place.

To move all of our projectiles at once we will use a for cycle:

```
for i = 1, #bullets do
    local bullet = bullets[i]
    LIKE_BEFORE
end
```

Here we go through all our projectiles inside ’bullets’ one by one. You can see that bullets[i] will always represent the bullet that comes at the ’i’-th place in the table.

LIKE_BEFORE refers to the way we did things in Part 1. You do the exact same thing here, including the remove check, and the isactive check, because inside the for cycle we assigned the projectiles to the ’bullet’ local variable each time the cycle is run. You can see my example in LESSON_01_02.lua.

This concludes my first tutorial of the very basics of Unitale wave creation. We've learned how to create one or more projectiles, and also how to move one or more at the same time. In the next lesson, we will discuss how to create several projectiles at the same time, and how to manipulate projectile speed on a projectile-by-projectile basis.

Thank you for checking out this tutorial! If you have any suggestions or criticism, you can reach me on Reddit at /u/BanzayIkoyama. See you in the next one!