CODE NINJAS®

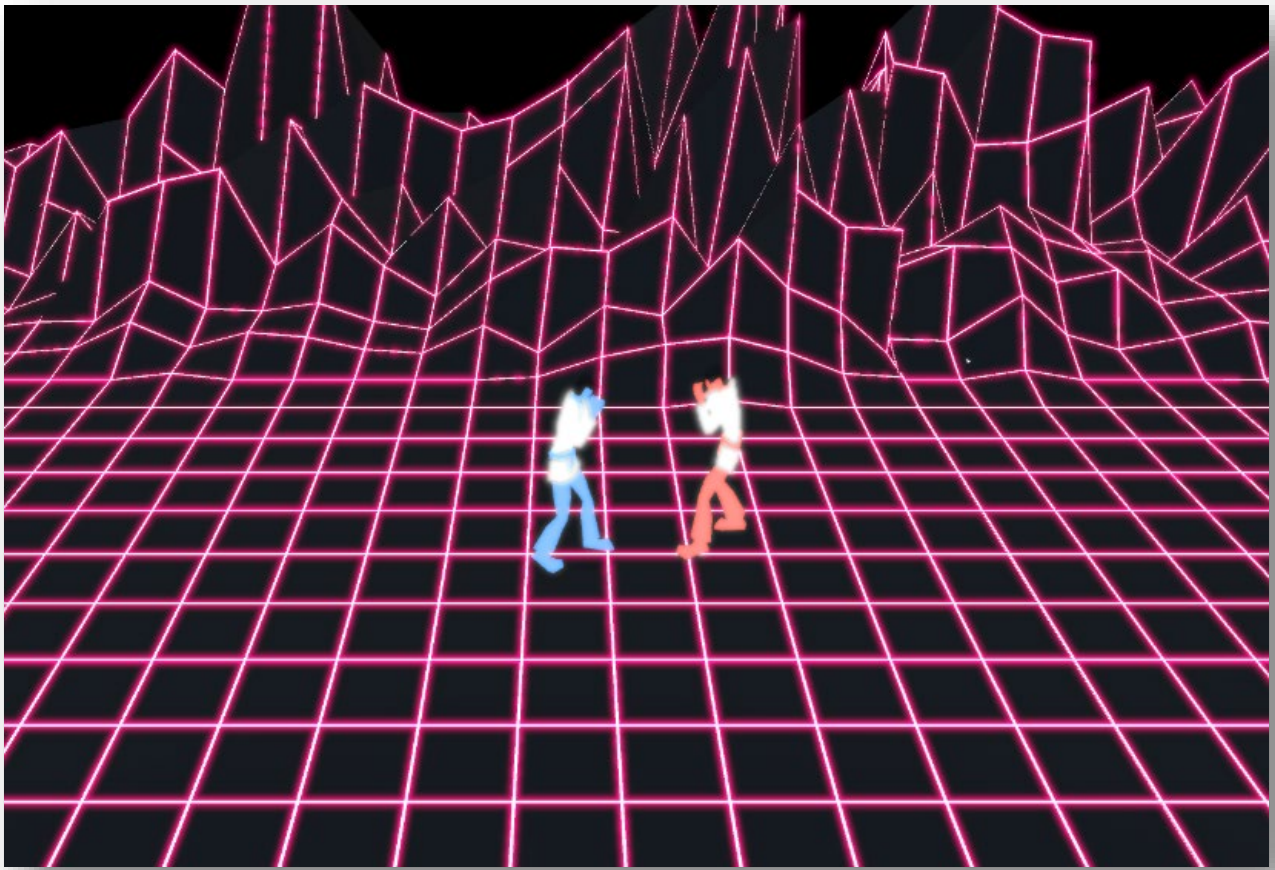
Silver Belt Ninja Guide
Activity 07: CyberFu Part 1

Activity 7

CyberFu - Part 1

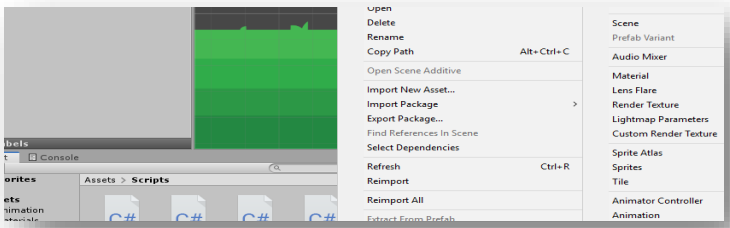
If you press **Play**, the enemy objects that are spawned in the game aren't moving! The enemies can't move or follow the player yet because there is no movement code.

New mission: Code your enemy to follow and chase the player. This is a key step in making your game because it will cause the enemies to move towards the player by calculating how far each enemy is!



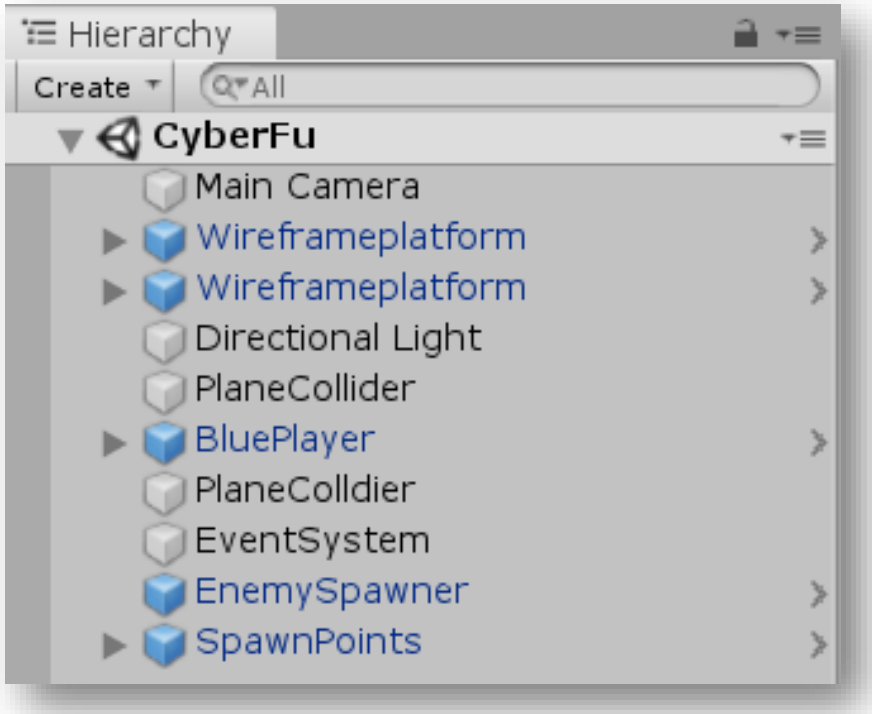
1 Start a new Unity Project and name it *YOUR INITIALS – CyberFu*. Select **3D core**.

2 Import the **Activity 07 – CyberFu Part1** starter Unity Package by going to **Assets > Import Package > Custom Package> All > Import**.

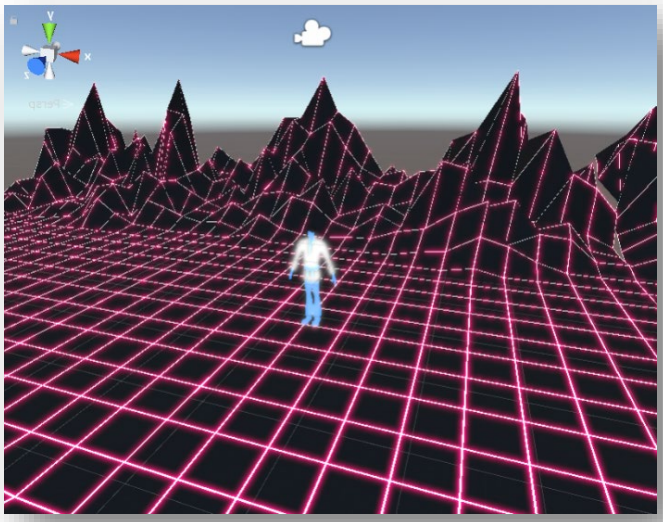


3 After it is done importing, double click the **CyberFu** scene. Located in the **Project** tab in the **Scenes** folder inside of the **Assets** folder.

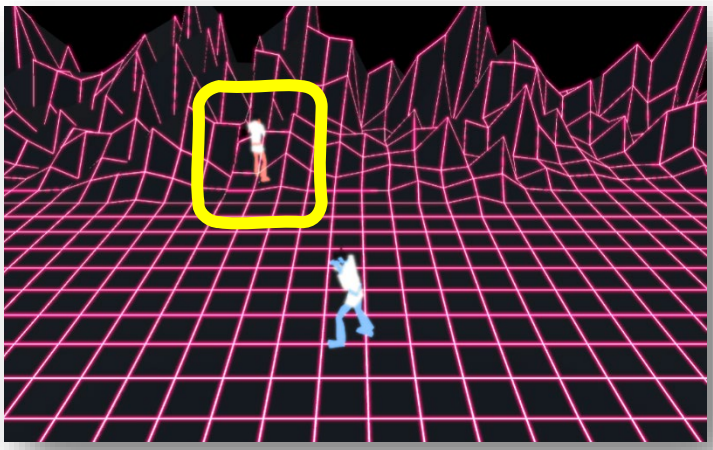
4 When all the assets have been imported into the project, open the scene "**CyberFu**" and you will see that the game is already set up with a couple of objects shown in the **hierarchy** and **scene**.



5 Before we start configuring our **enemy,** start the game to see what happens when we **play** the game as is.



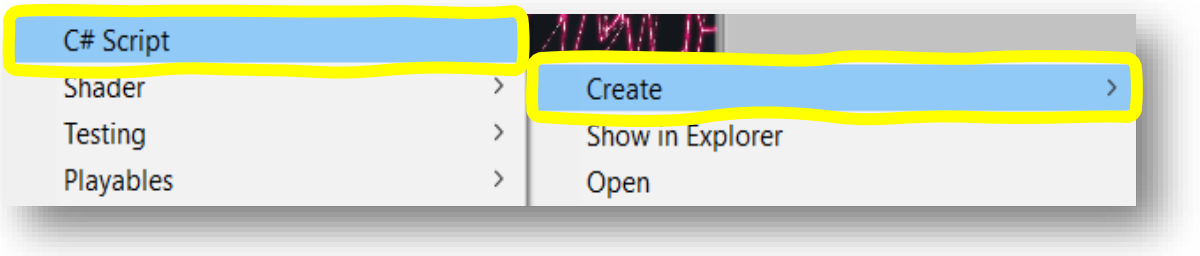
6 As you can see, when the game starts **enemies** will **spawn** at one of three **spawn points**. All the **enemies** are the same **prefab** we will use throughout the lesson. When the **enemy** is **spawned** into the game, it's not facing nor following the player.



7 We are going to have our **enemies** follow and attack the **player**, so let's go ahead and start controlling our **enemies** to do so.

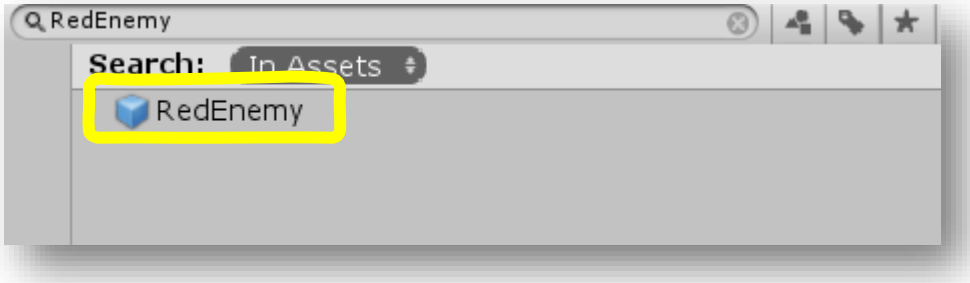
8 First, go into your **Scripts** folder.

9 Next, **right-click** an empty space. Go to "**Create**" and select "**C# Script**".

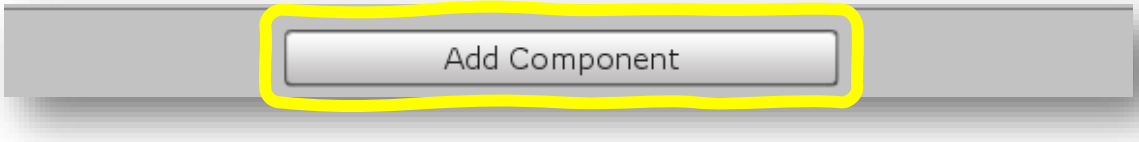


10 Rename it to "**EnemyControls**".

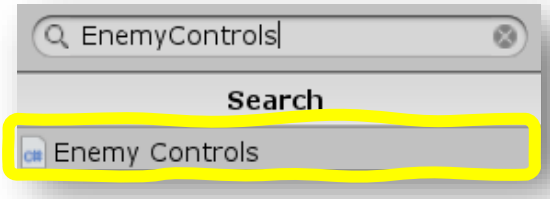
11 Next, let's find our **enemy**. Instead of searching through many folders to find the **enemy prefab**, let's use the **search bar** in the **project**. Type "**RedEnemy**" and select the **prefab** model.



12 Now, let's add the "**Enemy Controls**" **script** to the **enemy prefab**. To do that, in the "**Inspector Tab**", scroll down and click on "**Add Component**".



13 In the **search bar**, type "**EnemyControls**" and select the **component** from the **drop down menu**.



14 The **enemy controls script** is now part of the **enemy prefab**.

15 We are going to get started with **scripting** our **enemy** with `public` **variables** first.

These **variables** will be inserted between the first bracket and the `void Start` function.

16 The **first variable** will be for setting the **speed** of the **enemy**:

```
//the set speed of the enemy
public float speed = 2f;
```

17 The **second variable** will be for the **set attacking distance** for the **enemy**:

```
//the set attacking distance for the enemy
public float attackingDistance = 0.6f;
```

18 This is what the code should look like for you without comments.

```
public class EnemyControls : MonoBehaviour
{
    public float speed = 2f;
    public float attackingDistance = 0.6f;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }
}
```

19 Now, let's add **private variables** to the **enemies**. We are going to start with the **components**.

20 The **first variable** will be for the **Animator component** for the **enemy**. The **animator component** controls the **animations** for the **enemy**.

```
//the animator component controls the animations for the enemy
private Animator animatorEnemy;
```

21 The **second variable** will be for the **Rigidbody** for the **enemy**. The **rigidbody component** controls the **physics** for the **enemy**.

```
//the rigidbody component controls the physics for the enemy
private Rigidbody rigidbodyEnemy;
```

22 The **third variable** will be for position, rotation, and size of the object. This variable allows the enemy to figure out who it is targeting.

```
//this variable will help the enemy target any object with the
variable
private Transform target;
```

23 This is what your **script** should look like so far without comments.

```
public class EnemyControls : MonoBehaviour
{
    public float speed = 2f;
    public float attackingDistance = 0.6f;

    private Animator animatorEnemy;
    private Rigidbody rigidbodyEnemy;
    private Transform target;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }
}
```

24 Now, we will use a **boolean** to track if the enemy is following or not. A **boolean** is a type of variable that can be one of two values, either **true** or **false**.

25 Declare the variable above Start.

```
//If the enemy is following the target or not
private bool isFollowingTarget;
```

26 What your **script** should look like without comments:

```
public float speed = 2f;
public float attackingDistance = 0.6f;

private Animator animatorEnemy;
private Rigidbody rigidbodyEnemy;
private Transform target;

private bool isFollowingTarget;
```

27 The last two **variables** we will add will be for **attacking** the **target**.

28 The **first variable** will be for **current value** of what the current time is for **attacking** the **target**.

```
//current value of what the current time is for attacking
private float currentAttackingTime = 0f;
```

29 The **second variable** will be the **set value** of the maximum time of **attacking**.

```

//the set value of the maximum time of attacking
private float maxAttackingTime = 2;
```

30 Here is what your script should look like without comments:

```
public float speed = 2f;
public float attackingDistance = 0.6f;

private Animator animatorEnemy;
private Rigidbody rigidbodyEnemy;
private Transform target;

private bool isFollowingTarget;

private float currentAttackingTime = 0f;
private float maxAttackingTime = 2f;
```

31

Add **three variables** into the **Start function**.

The first two variables will be for the components of the **enemy** and the third will be the **gameobject** that will be the **target** for the **enemy** to **follow** and **attack**.

The first **variable** we will add is getting the **component** for the **animator**.

```
//getting the component for the animator
animatorEnemy = GetComponent<Animator>();
```

32

The **second variable** is getting the **component** for the **rigidbody**.

```
//getting the component for the rigidbody
rigidbodyEnemy = GetComponent<Rigidbody>();
```

33

The **third variable** is the **target**.

This is different from the other variables because we will set the target for the enemy to **follow** and **attack** to the player. We'll find the player using the Player tag that is only assigned to the player gameobject.

We do **"target = ..."** that way we get the position where the game object with the tag "**Player**" is located.

```
//set the target for the enemy to follow and attack to the
player by tag

target = GameObject.FindGameObjectWithTag("Player").transform;
```

34 Here is what the function should look like without comments.

```
Unity Message | 0 references
void Start()
{
    animatorEnemy = GetComponent<Animator>();
    rigidbodyEnemy = GetComponent<Rigidbody>();
    target = GameObject.FindGameObjectWithTag("Player").transform;
}
```

35 So, now that we have set up our enemy to where the **target** is **player**. Let's get our enemy to follow the player.

36 In the Update method, let's make the enemy face the player. We can do this by using the LookAt method. This method rotates a transform to face any position. In this case, we want the enemy transform to rotate towards the player's position. Add this line inside the **Update** method.

```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    transform.LookAt(target.position);
}
```

37 **Save** your script and return to Unity. Press **play**. The enemies should now be rotating towards the player. But they are still standing still....

38 In the Update method, we need to determine if the enemy should be following or attacking. We can determine this based on the distance that the player is from the enemy. We can tell the distance between two positions using **Vector3.Distance()** method. Add this line in Update.

```
isFollowingTarget = Vector3.Distance(transform.position,
target.position) >= attackingDistance;
```

39 Now use an if statement to check if the enemy is following the player. Add the if statement below the previous line in Update.

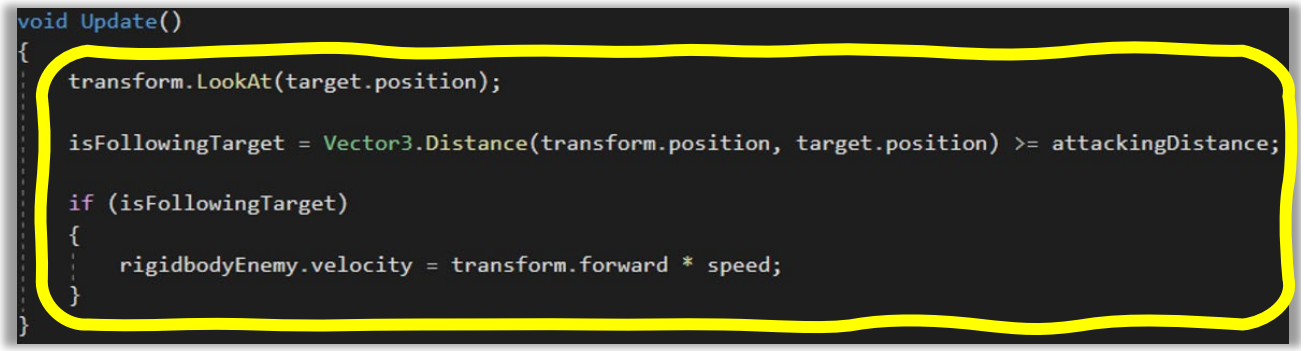
```
if (isFollowingTarget)
{

}
```

40 Inside the if statement, we are going to use the rigidbody's velocity to move the enemy towards the player. Since the enemy is already facing the player, we can use transform.forward as the direction for them to move. Add this line inside the if statement.

```
if(isFollowingTarget)
{
     rigidbodyEnemy.velocity = transform.forward * speed;
}
```

41 Here is what your script looks like without comments:



```
void Update()
{
    transform.LookAt(target.position);

    isFollowingTarget = Vector3.Distance(transform.position, target.position) >= attackingDistance;

    if (isFollowingTarget)
    {
        rigidbodyEnemy.velocity = transform.forward * speed;
    }
}
```

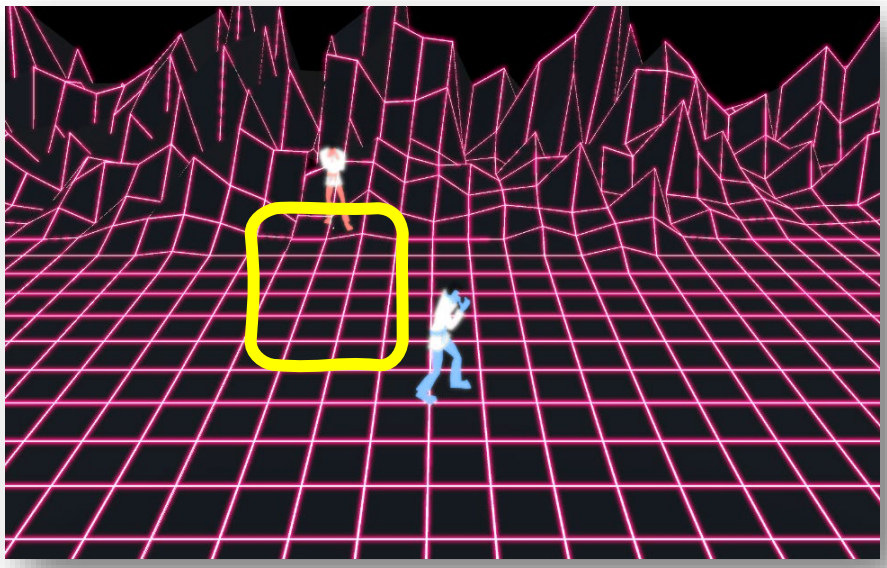
42 Before we **playest** the game, let's see if the enemy is going to attack the player by checking the **boolean** "**isFollowingTarget**".

To see isFollowingTarget while playing, let's expose the variable to the inspector. Normally, a private variable is hidden and doesn't show up in the inspector. We can make this variable show up in the inspector by using SerializeField. Place this above the declaration for isFollowingTarget.

```
[SerializeField]
private bool isFollowingTarget;
```

43 **Save** the script and **play** the game to see if it works.

44 Our enemy should now look at **the player, and move towards them until they get to attack range**.



45 Even though we got our enemy to **move** and **follow** the player, there are still some problems. The enemy does not stop walking when they are close to the player, nor do they attack. Let's update the script.

46 First thing we are going to add is an **else statement** to where the enemy's position is **less than or equal to** the **attacking distance** to attack the player. So, below the **if statement**, add:

```
//If enemy's distance is close to attack the player

else
{

}
```

47 Since Attacking will have a lot of code, it's a good idea to separate this into its own method. Create a `void Attack()` method at the bottom of our script.

48 Next, inside the **Attack method** we are going freeze the enemy's **rigidbody** to where it cannot move from its spot. We will use **Vector3.zero** and have all of the **values** of all velocity in each direction 0.

```
//Keeps the enemy from moving within its place

rigidbodyEnemy.velocity = Vector3.zero;
```

49 Your Attack method should look like this so far.

```
void Attack()
{
    rigidbodyEnemy.velocity = Vector3.zero;
}
```

50 Add a call to the attack method in the else section in the Update method.

```
else
{
     Attack();
}
```

51 **Save** your script and return to Unity. The enemy should stop as soon as it gets in attacking range of the player.

52 To have the enemy attack the player, we will use some attack animations. Before we do that, let's use the animation for walking that is already created for us.

Walking is a boolean, so we can re-use the isFollowingTarget boolean to determine if the walk animation should be playing.

53 Add this line below the else statement in Update.

```
else
{
    Attack();
}

animatorEnemy.SetBool("Walk", isFollowingTarget);
```

54 Save the script and check in Unity. The enemies should have a walking animation until they get in attack range.

55 Now let's have the enemies attack! There are already some attack animation created for our player, we just have to play one when they get in range, and the attack cooldown is ready.

56 In our Attack method, have the **current attacking time** add to the **value** of **delta time**. Delta time is the difference in time between frames, so this is like adding the number of milliseconds every time.

```
//the current attacking time add to the value of delta time

currentAttackingTime += Time.deltaTime;
```

57 Add an **if statement** to where if the **current attacking time** is **greater** than the **max value** of attacking time .

The reason why we need to add this is because if the **current attacking time** equals to **2**, we want to have the enemy do something.

```
//if the current attack time is greater than the max attack
time

if (currentAttackingTime > maxAttackingTime)
{

}
```

58 Inside the **if statement**, set the value of the **current attack time** to **0**. Every time the **current attack time** is set **2**, it will reset to the value to **current attacking time** to **0**.

```
//set the current attacking time to 0

currentAttackingTime = 0f;
```

59 Let's add an **attack** as well. This **attack animation** will only trigger once every time the **current attacking time** equals to **2**. We only want the enemy to attack once **every 2 seconds**, so instead of using **SetBool** for a boolean, which will continue to play the animation over and over, we will use **SetTrigger** to execute the animation only once.

```
//trigger the enemy's attack animation

animatorEnemy.SetTrigger("Attack1");
```

60 Here is what the script should look like:

```
void Attack()
{
    rigidbodyEnemy.velocity = Vector3.zero;

    currentAttackingTime += Time.deltaTime;

    if (currentAttackingTime > maxAttackingTime)
    {
        currentAttackingTime = 0f;
        animatorEnemy.SetTrigger("Attack1");
    }
}
```

61 **Save** the script and **play** the game to test it.

62 There are more **attack animations** for the enemy and we want the enemy to use them. Let's go back to the **enemy control script** and update it.

63 All of the attacks are named "Attack" with a number at the end. For example, "Attack1", "Attack2", "Attack3", etc. Therefore, we can get a random number to determine what attack we will use. To do this, we will use Unity's built in **Random** method.

64 We will use `Random.Range` to get the random number. The first paramater is the minimum number inclusive, and the second is the maximum number exclusive. Inclusive means that it **can** be the random number, and exclusive means it **cannot** be the random number. Therefore, if we want a random number to either be 1, 2, or 3, then we need to set the minimum to 1 (inclusive) and the maximum to 4 (exclusive).

65 Go back to the Attack method. Above the "SetTrigger" line, create a variable to store the random number. Add the line below.

```
int rand = Random.Range(1, 4);
```

66 Now, in the "SetTrigger" line, we can replace the "Attack1" string with "Attack" string plus the random number we generated. Change the line so it reads as follows.

```
AnimatorEnemy.SetTrigger("Attack" + rand);
```

67 Your script should look like this.

```
void Attack()
{
    rigidbodyEnemy.velocity = Vector3.zero;

    currentAttackingTime += Time.deltaTime;

    if (currentAttackingTime > maxAttackingTime)
    {
        currentAttackingTime = 0f;
        int rand = Random.Range(1, 4);
        animatorEnemy.SetTrigger("Attack" + rand);
    }
}
```

68 **Save** your script. Return to Unity and playtest your game! The enemies should throw out different attacks.

69 **Congratulations**, you have complete part 1 of the lesson for **CyberFu**.

In part 2, you will learn how to add **health** and **damage** to both the enemy and the player.