Player Health & Stats | Types of Enemies

Prefabs

- A prefab is a master entity of a GameObject that you can use in your game endlessly. In other words, creating a prefab is like creating a template that contains all the characteristics of something (e.g. Enemy), and that template can be used to create many copies. That saves a lot of time.

To Create a Prefab

- To create a prefab, first create a folder in the Project pane under Assets and name it Prefabs
- Select the GameObject you want to turn into a prefab from the Hierarchy pane and drag and drop it to the prefab you created
- If it turns blue, then it has been created successfully
- If you make a change to a prefab and want it to reflect on all the copies in the game (e.g. you made a change to Enemy #1 and you want all instances of Enemy #1 to be updated), you must click *Apply* for the changes to take effect across the entire scene

Exercise #21 – Player Stats

- Create a script that contains everything needed to control the player's health and lives. Call it PlayerStats and attach it to the Player gameObject

- We need the player stats to have 3 things:
  - Health
  - Lives
  - Temporary Immunity

- Variables we'll need are listed on the right:

```
public int health = 6;
public int lives = 3;

private float flickerTime = 0f;
public float flickerDuration = 0.1f;

private SpriteRenderer spriteRenderer;

public bool isImmune = false;
private float immunityTime = 0f;
public float immunityDuration = 1.5f;

public int coinsCollected = 0;
```

Exercise #21 – Player Stats (Cont.)

- In the Start() function:

```
void Start()
{
    spriteRenderer = this.gameObject.GetComponent<SpriteRenderer>();
}
```

- Create a SpriteFlicker() function

```
void SpriteFlicker()
{
    if(this.flickerTime < this.flickerDuration)
    {
        this.flickerTime = this.flickerTime + Time.deltaTime;
    }
    else if (this.flickerTime >= this.flickerDuration)
    {
    spriteRenderer.enabled = !(spriteRenderer.enabled);
    this.flickerTime = 0;
    }
}
```

Exercise #21 – Player Stats (Cont.)

- Create the TakeDamage() function

```
public void TakeDamage(int damage)
{
    if (this.isImmune == false )
    {
        this.health = this.health - damage;
        if (this.health < 0)
            this.health = 0;
        if (this.lives > 0 && this.health == 0)
        {
            FindObjectOfType<LevelManager>().RespawnPlayer();
            this.health = 6;
            this.lives--;
        }
        else if (this.lives == 0 && this.health == 0)
        {
            Debug.Log("Gameover");      // ADD Gameover splash screen later
            Destroy(this.gameObject);
        }

        Debug.Log("Player Health:" + this.health.ToString());
        Debug.Log("Player Lives:" + this.lives.ToString());
    }

    PlayHitReaction();
}
```

```
void PlayHitReaction()
{
    this.isImmune = true;
    this.immunityTime = 0f;
}
```

Exercise #21 – Player Stats (Cont.)

- In the Update() function:

```
void Update()
{
    if(this.isImmune == true)
    {
        SpriteFlicker();
        immunityTime = immunityTime + Time.deltaTime;
        if(immunityTime >= immunityDuration)
        {
            this.isImmune = false;
            this.spriteRenderer.enabled = true;
            //Debug.Log("Immunity has ended");
        }
    }
}
```

Enemies

- You will need to add enemies to your levels to make them more challenging and more interesting

- Your enemies can:
  - Walk on the ground
  - Spawn when player passes a specific point
  - Fly overhead (Sin wave)
  - Follow the player

Exercise #22 - Create A Super Class For Your Enemies

- To avoid redundancy in the code, create a script and name it EnemyController
- That script will include functions and variables that can be inherited by various types of enemies. Add those 3 variables

```
public bool isFacingRight = false;
public float maxSpeed = 3f;
public int damage = 6;
```

  - (Note that since the variables are public anyway, you don't need to give them specific values inside the code. Do it inside Unity instead)

- Also create a Flip() function that allows your enemy to flip when it hits a wall or another enemy. Use the code you have already written in your Player Controller code if that's easier for you, or use the one below:

```
public void Flip()
{
    isFacingRight = !isFacingRight;
    transform.localScale = new Vector3(-(transform.localScale.x), transform.localScale.y, transform.localScale.z);

}
```

Exercise #22 - Create A Super Class For Your Enemies (Cont.)

- Whenever the player runs into the enemy's collider (which has the "IsTrigger" property turned on), the function TakeDamage() is called from another class called PlayerStats, and the player's health receives damage

```
void OnTriggerEnter2D(Collider2D other)
{
    if (other.tag == "Player")
    {
        FindObjectOfType<PlayerStats>().TakeDamage(damage);
    }
}
```

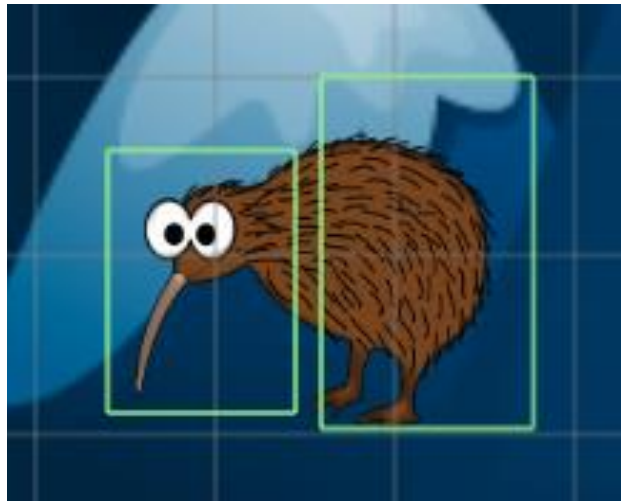
Exercise #23 - Create a normal walking enemy

- Choose an enemy sprite to be your enemy game object

- Add a Rigidbody2D Component to it and check the Z rotation freezing box so the enemy can’t get flipped over

- Add a box Collider 2D Component to the GameObject

- Add another Box Collider 2D Component to the GameObject and check the “Is Trigger” box. (The trigger will be used to check for collisions with walls or other enemies that would cause the enemy to turn around)



Exercise #23 - Create a normal walking enemy (Cont.)

- Create a script for your enemy gameObject and name it WalkingEnemy.

- Create a FixedUpdate() function and write code for movement along the x-axis

```
void FixedUpdate ()
{
    if(this.isFacingRight == true)
    {
        this.GetComponent<Rigidbody2D>().velocity =
            new Vector2(maxSpeed, this.GetComponent<Rigidbody2D>().velocity.y);
    }
    else
    {
        this.GetComponent<Rigidbody2D>().velocity =
            new Vector2(-maxSpeed, this.GetComponent<Rigidbody2D>().velocity.y);
    }
}
```

Exercise #23 - Create a normal walking enemy (Cont.)

- Create a wall that the enemy – upon colliding with – flips and starts moving in the opposite direction
- Inherit the OnTriggerEnter2D() function you had created earlier in the EnemyController superclass. However, you must override it because you'll be making some changes to the function's code. Compare the code below with the code from Slide 6
- Make sure to write the correct game object tag in the if statement

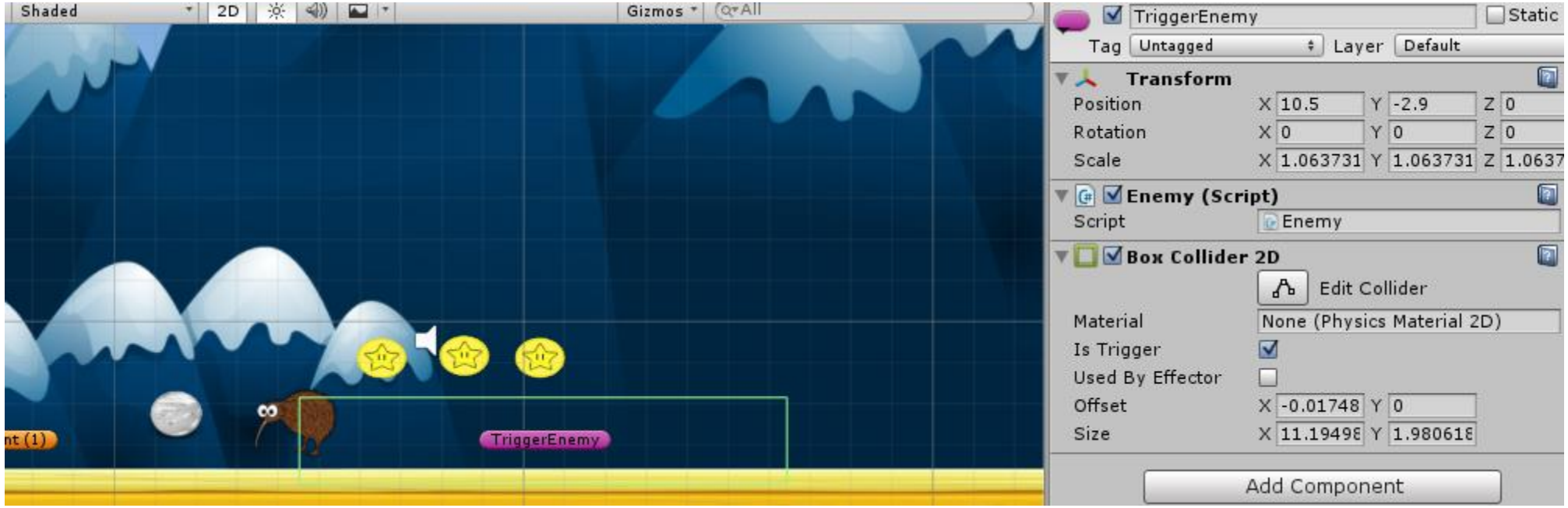
```
void OnTriggerEnter2D(Collider2D collider)
{
    if(collider.tag == "Wall")
    {
        Flip ();
    }
    else if (collider.tag == "Enemy")
    {
        Flip ();
    }
    if(collider.tag == "Player")
    {

        FindObjectOfType<PlayerStats>().TakeDamage(damage);
        Flip();
    }
}
```

Exercise #23 - Create a normal walking enemy (Cont.)

- Back in the Unity interface, don’t forget to give the enemy’s public variables values

Exercise #24 – Create an enemy that spawns at a certain point

- Create an empty gameObject and add a collider2D and a script file to it. Name it SpawnPoint
  - It is named so because the enemy appears when the player runs through the gameObject's collider!
  - Set the SpawnPoint collider to "IsTrigger"



Exercise #24 – Create an enemy that spawns at a certain point (Cont.)

- We want to make an enemy appear the moment the player character passes by a certain point. We'll need to use 3 scripts: SpawnedEnemy, Walking Enemy, and LevelManager

- Create an enemy prefab and keep it ready in the prefabs folder
- The script for SpawnedEnemy will only contain the OnTriggerEnter2D function() which does this:

```
void OnTriggerEnter2D(Collider2D other)
{
    if (other.gameObject.tag == "Player")
        FindObjectOfType<LevelManager> ().RespawnEnemy ();
}
```

- If the player game object passes through the collider, LevelManager is searched for and found. The setBool function is called and given the value of true. Back in the LevelManager script, the enemy will be spawned and will follow the player

Exercise #24 – Create an enemy that spawns at a certain point (Cont.)

- Now go to the Level Manager and define the following variable

```
public Transform enemy;
```

- The enemy is spawned from *the exact location of your the Level Manager's game object*

```
public void RespawnEnemy()
{
    Instantiate(enemy, transform.position, transform.rotation);
}
```

Exercise #24 – Create an enemy that spawns at a certain point (Cont.)

- Finally, go back to Unity and make sure you give the Level Manager the enemy prefab you have created. Drag it from the prefabs folder and drop it into the Enemy text box

Exercise #25 – Create an enemy that follows the player

- Create another prefab of an enemy game object, and create a script for it called FollowerEnemy. Wherever your player goes, this enemy follows them

- FollowerEnemy is a subclass of EnemyController

- FollowerEnemy will need a variable of type Controller (or whatever the name of your player's script is!)

```
private Controller player;
```

- This variable is necessary, because in the FollowerEnemy's Start() function, the player character will be located as seen below:

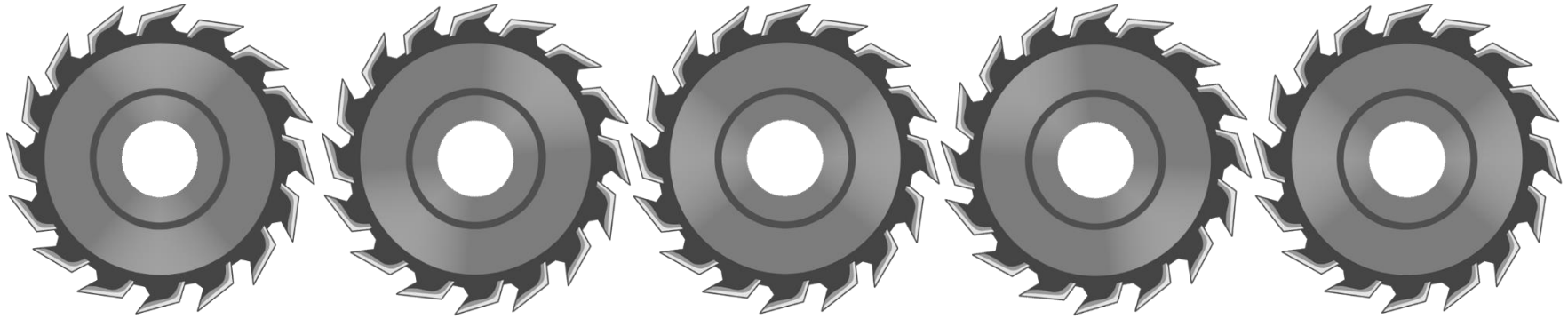
```
void Start ()
{
    player = FindObjectOfType<Controller>();
}
```

- In the Update(), MoveTowards() function is called to make the enemy follow the player even as the player's position changes every frame

```
void Update()
{
    transform.position = Vector3.MoveTowards(transform.position, player.transform.position, speedtowardplayer * Time.deltaTime);
}
```

Exercise #25 – Create an enemy that follows the player

- A good example of a FollowerEnemy is a rotating saw like this. It can be animated and follows the player left and right



Exercise #26 – Create a flying enemy

- Create an empty GameObject and name it “FlyingEnemy”
- Add a Sprite Renderer Component to the GameObject and point the “Sprite” property to your enemy image
- Add a Circle Collider 2D Component to the GameObject and check the “IsTrigger” box. Add new script to it and call it FlyingEnemy
- Note that FlyingEnemy will be a subclass of EnemyController,

so it can inherit its variables (e.g. damage) and functions

(e.g. OnTriggerEnter2D). No need to rewrite them

Exercise #26 – Create a flying enemy (Cont.)

- Declare the public variables

```
public class SinFlyEnemy : MonoBehaviour {

    public float HorizzontalSpeed;
    public float VerticalSpeed;
    public float amplitude;

    private Vector3 temp_position;

    public float moveSpeed;
    private Controller player;
```

- Assign the temporary position to the position the enemy starts at in the scene

```
// Use this for initialization
void Start () {
    temp_position = transform.position;
}
```

Exercise #26 – Create a flying enemy (Cont.)

- Draw the sin waves in the Fixed Update function

```
    // Update is called once per frame
    void FixedUpdate () {

        temp_position.x += HorizzontalSpeed;
        temp_position.y = Mathf.Sin(Time.realtimeSinceStartup * VerticalSpeed) * amplitude;
        transform.position = temp_position;
    }

}
```

- Then add the OnTriggerEnter2D function so that if this flying enemy touches the player, the player's receives damage and their health decreases

Work Tip – What are Tags?

- You can tag any game object with a keyword that the script uses in order to execute an action (For example, the player shooting an enemy)

- We will need to ‘tag’ every creature that the player can shoot as an ‘Enemy’

- To do that:
  - Select the gameObject and go to the top of the Inspector window
  - Click on the Tags drop down list
  - Create a new relevant tag or choose one from the list
  - Assign the tag to the gameObject

Useful References

• **Johnson, M., Hasankolli, R., & Henley, J. A. (2014). *Learning 2D Game Development with Unity: A Hands-on Guide to Game Creation*. Pearson Education.**

• **Adding A Health System - Unity 2D Platformer Tutorial - Part 9. URL retrieved From:** https://www.youtube.com/watch?v=F6hUIU72JwE

• **Unity RPG Tutorial 1 - Setting Up The Basics. URL retrieved from:** https://www.youtube.com/watch?v=Pk3GCgaNVTY