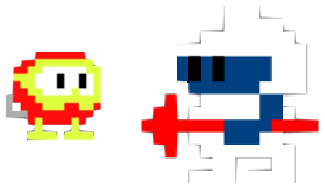
CREATE GAMES !



Welcome to issue 11.

Time flies when you are creating games doesn't it ? When you download the assets of this issue you will get the full open resource project that you can open in Unity. This way you can start right away and change/add things to your own likings. If you're new to Unity and C-Script I advise you to read and experiment with all previous issues.

In this Issue we will create a basic game of Dig Dug. You can see this all in action right here : https://youtu.be/27ZqolAIaCk

I will tell you what each script does so you will understands this example game mechanics. I am sure things could be scripted better but hey it works so go nuts and make it better when needed. The scripting of this game can also be used for other games that use similar things like Boulder dash for example.

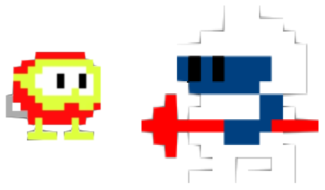
The dirt game object us just a sprite with a box collider 2D. It has a layer name Dirt. The script it has makes sure that if the player comes in range the dirt part will be removes.

Dirt script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Dirt : MonoBehaviour
{

/// Basic Dig Dug Game Issue February 2022////
/// René Pol 01-01-2022///
```

On TriggerEnter with a game object that has the name tag Player this dirt object is removed.

```
public void OnTriggerEnter2D(Collider2D collision)
{
if (collision.gameObject.tag == ("Player"))
{
Destroy(this.gameObject);
}
}
}
```



The rocks are Sprite objects with a box collider 2D. It has also a rigid body 2D with its body type set to Kinematic. This way it uses physics but will not move when touched by another object. The z freeze rotation is set so it will never rotate on this axis. The layer name given is Wall so the player can not move through it. When dirt is detected under the rock nothing happens. But when there is nothing under the rock it will fall down and gets destroyed in impact dirt or enemy.

Rocks script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Rocks : MonoBehaviour
{

/// Basic Dig Dug Game Issue February 2022////
/// René Pol 01-01-2022///
```

A bool variable is set to determine if the rock is falling or not. A layerMask is set so it can use raycast to detect those layers. Speed is set with the familiar float variable.

```
public bool IsFalling;
public LayerMask obstacleMask;
public float Speed;
```

A coroutine is created that will get started when nothing is detected under the rock so the falling bool variable is set to true.

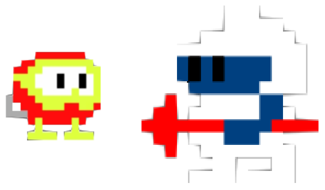
```
IEnumerator FallDown()
{
yield return new WaitForSeconds(1f);
IsFalling = true;
}
```

At start the bool variable is set to false.

```
void Start()
{
IsFalling = false;
}
```

When the bool variable is true and only then it will use a recast that aims down with a length of 0.7 looking for the layerMask that is defined in the inspector's view. The debug.drawray line makes this ray visible in the scene window.

```
void Update()
{
if (IsFalling == true)
{
RaycastHit2D hitdown2 = Physics2D.Raycast(transform.position,
transform.TransformDirection(Vector2.down), 0.7f, obstacleMask);
Debug.DrawRay(transform.position, transform.TransformDirection(Vector2.down) *
0.7f, Color.red);
transform.Translate(Vector2.down * Speed * Time.deltaTime);
```



When the raycast is hitting something and isfalling is true, so only then....

```
if (hitdown2.collider != null && IsFalling == true)
{
```

When hitdown is hitting something it will destroy itself.

```
if (hitdown2)
{
Destroy(this.gameObject);
}
```

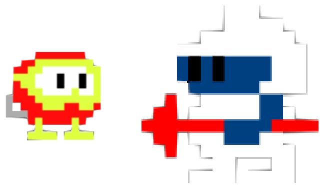
When hitdown is hitting an enemy it will destroy itself and the enemy object.

```
if (hitdown2.collider.gameObject.tag == ("Enemy"))
{
Destroy(hitdown2.collider.gameObject);
Destroy(this.gameObject);
}
}
}
```

At start the rock will stay cause it detects objects under it. When nothing is detected the fallDown coroutine is activated.

```
RaycastHit2D hitdown = Physics2D.Raycast(transform.position,
transform.TransformDirection(Vector2.down), 0.7f, obstacleMask);
Debug.DrawRay(transform.position, transform.TransformDirection(Vector2.down) *
0.7f, Color.red);
if (hitdown.collider == null && IsFalling == false)
{
StartCoroutine(FallDown());
}
}
}
```

I created a very basic AI script for the enemies. You can expand it from here if you want to.
The enemy is a sprite with a box collider 2D with is trigger box selected. It has also a rigidbody 2D with gravity set to 0 and the freeze Z rotation box selected. It got the layer name Enemy as well as a tag name Enemy. Every time the enemy hits dirt it will choose a random other direction then it went before it hitted the dirt. When hit by the players beam it will activate a coroutine that makes the enemy grow 3 times before it gets removed. When no longer hit by the beam the enemy returns to normal size and the coroutine is stopped.



Enemy script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Enemy01 : MonoBehaviour
{

/// Basic Dig Dug Game Issue February 2022////
/// René Pol 01-01-2022///
```

2 float variables are created one for movemode and one for speed. A bool variable is created to determine if the enemy is hit or not. LayerMask is created so the enemy can not walk through those objects. 4 different sprites are defined so the enemy can face left right up and down.

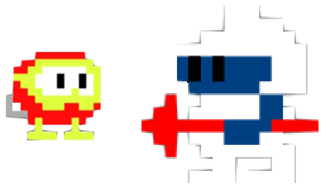
```
public float MoveMode = 0;
public float Speed;
public bool Hitted;
public LayerMask obstacleMask;
public Sprite Left, Right, Up, Down;
```

At start the Hitted bool variable is set to false.

```
void Start()
{
Hitted = false;
}
```

A coroutine is created every 0.5 seconds the object increases size by 0.1f after 3 times increasing the object is removed. This only happens when the hitted variable is true.

```
IEnumerator BlownUp()
{
Hitted = true;
while (Hitted == true)
{
yield return new WaitForSeconds(0.5f);
transform.localScale += new Vector3(0.1f, 0.1f, 0.1f);
yield return new WaitForSeconds(0.5f);
transform.localScale += new Vector3(0.1f, 0.1f, 0.1f);
yield return new WaitForSeconds(0.5f);
transform.localScale += new Vector3(0.1f, 0.1f, 0.1f);
yield return new WaitForSeconds(0.3f);
Destroy(this.gameObject);
}
yield return null;
}
```



When the trigger object with the tag name Beam is in range and the hitted variable is false the coroutine Blownup is started.

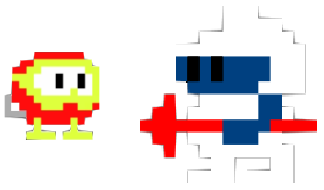
```
public void OnTriggerEnter2D(Collider2D collision)
{
if (collision.gameObject.tag == ("Beam") && Hitted == false)
{
StartCoroutine(BlownUp());
}
}
```

When the trigger object with the tag name Beam is out of range and the hitted variable is true the coroutine Blownup is stopped and hitted is set back to false. The scale is set to its normal size (1,1,1)

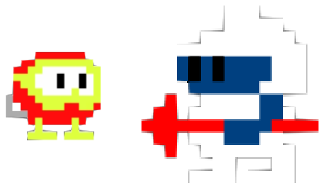
```
public void OnTriggerExit2D(Collider2D collision)
{
if (collision.gameObject.tag == ("Beam"))
{
StopAllCoroutines();
Hitted = false;
transform.localScale = new Vector3(1, 1, 1);
}
}
```

When the object is moving left and it hits something with its raycast it will random choose one out of the other 3 left directions. It will change then the MoveMode number. For each direction a similar coroutine is created.

```
IEnumerator BlockLeft()
{
int pickMove = Random.Range(0, 3);
yield return new WaitForSeconds(0);
if (pickMove == 0)
{
MoveMode = 2;
}
if (pickMove == 1)
{
MoveMode = 1;
}
if (pickMove == 2)
{
MoveMode = 4;
}
yield return null;
}
```

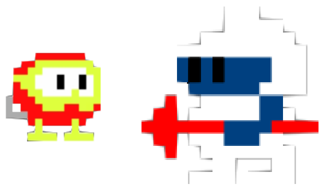


```
IEnumerator BlockRight()
{
int pickMove = Random.Range(0, 3);
yield return new WaitForSeconds(0);
if (pickMove == 0)
{
MoveMode = 2;
}
if (pickMove == 1)
{
MoveMode = 3;
}
if (pickMove == 2)
{
MoveMode = 4;
}
}

IEnumerator BlockUp()
{
int pickMove = Random.Range(0, 3);
yield return new WaitForSeconds(0);
if (pickMove == 0)
{
MoveMode = 2;
}
if (pickMove == 1)
{
MoveMode = 3;
}
if (pickMove == 2)
{
MoveMode = 1;
}
}

IEnumerator BlockDown()
{
int pickMove = Random.Range(0, 3);
yield return new WaitForSeconds(0);
if (pickMove == 0)
{
MoveMode = 1;
}
if (pickMove == 1)
{
MoveMode = 3;
}
if (pickMove == 2)
{
MoveMode = 4;
}
}
```



In the update function when hitted is set to false (So not hit by a beam) each move mode will have a direction to move the object in with a raycast to detect if there is something in the way. The right sprite is used so it faced in the direction it moves. When the raycast detects a hit the right coroutine is tarted while movement is stopped by setting it to 0.

```
void Update()
{
if (Hitted == false)
{
if (MoveMode == 1)
{
RaycastHit2D hitright = Physics2D.Raycast(transform.position,
transform.TransformDirection(Vector2.right), 0.7f, obstacleMask);
Debug.DrawRay(transform.position, transform.TransformDirection(Vector2.right) *
0.7f, Color.red);
if (hitright)
{
MoveMode = 0;
StartCoroutine(BlockRight());
}
this.GetComponent<SpriteRenderer>().sprite = Right;
transform.Translate(Vector2.right * Speed * Time.deltaTime);
}
if (MoveMode == 2)
{
RaycastHit2D hitdown = Physics2D.Raycast(transform.position,
transform.TransformDirection(Vector2.down), 0.7f, obstacleMask);
Debug.DrawRay(transform.position, transform.TransformDirection(Vector2.down) *
0.7f, Color.red);
if (hitdown)
{
StartCoroutine(BlockDown());
MoveMode = 0;
}
this.GetComponent<SpriteRenderer>().sprite = Down;
transform.Translate(Vector2.down * Speed * Time.deltaTime);
}
if (MoveMode == 3)
{
RaycastHit2D hitleft = Physics2D.Raycast(transform.position,
transform.TransformDirection(Vector2.left), 0.7f, obstacleMask);
Debug.DrawRay(transform.position, transform.TransformDirection(Vector2.left) *
0.7f, Color.red);
if (hitleft)
{
StartCoroutine(BlockLeft());
MoveMode = 0;
}
this.GetComponent<SpriteRenderer>().sprite = Left;
transform.Translate(Vector2.left * Speed * Time.deltaTime);
}
```



```
if (MoveMode == 4)
{
RaycastHit2D hitup = Physics2D.Raycast(transform.position,
transform.TransformDirection(Vector2.up), 0.7f, obstacleMask);
Debug.DrawRay(transform.position, transform.TransformDirection(Vector2.up) * 0.7f,
Color.red);
if (hitup)
{
StartCoroutine(BlockUp());
MoveMode = 0;
}
this.GetComponent<SpriteRenderer>().sprite = Up;
transform.Translate(Vector2.up * Speed * Time.deltaTime);
}
}
}
}
```

Time for the final and important script, that of the player. The player is a sprite with a box collider2D and a rigidbody2d. the gravity is set to 0 and the freeze z rotate box is selected. It has a child object named checker, another child object sprite on a distance with a box collider 2D that has the is trigger box selected and a nametag Beam. It also have a third child sprite object that shows the beam.

Player Script

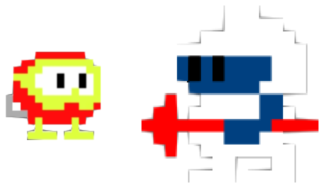
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMove : MonoBehaviour
{

/// Basic Dig Dug Game Issue February 2022////
/// René Pol 01-01-2022///
```

A float variable for speed is created. We use the checker as transform and call it movePoint. A layerMask is used and you should understand now why.

```
public float speed = 5;
public Transform movePoint;
private LayerMask obstacleMask;
```

A float variable for a horizontal and vertical grid length are created so the player will move as if it’s moving over a grid.

```
public float HorizontalGrid;
public float VerticalGrid;
```



4 different sprites are used so the player can face all 4 directions. Another transform is used and called Shooter. The bool variable is to determine if the beam is activated or not. The beam is the child sprite of the beam with the same name.

```
public Sprite Left, Right, Up, Down;
public Transform Shooter;
public bool beamActivated;
public SpriteRenderer Beam;
```

At start the bool beamactivated is set to false. The shooter deactivates it's box collider and the move point is at the center of the player but can as a child still move around.

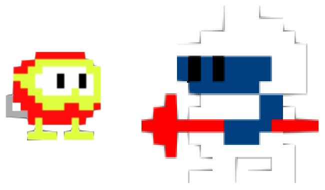
```
void Start()
{
beamActivated = false;
Shooter.GetComponent<BoxCollider2D>().enabled = false;
movePoint.parent = null;
}
```

When the movepoint is free of obstacles the player will be free to move to the movepoint this movepoint always moves first followed by the player. Any Mask layer used as obstacle will prevent the player from moving.

```
private void Move(Vector3 direction)
{
Vector3 newPosition = movePoint.position + direction;
if (!Physics2D.OverlapCircle(newPosition, 0.2f, obstacleMask))
{
movePoint.position = newPosition;
}
}
```

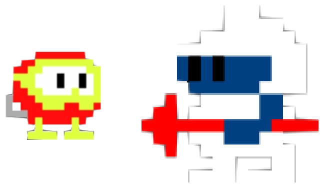
When the space key is pressed the beamactivated bool is set to true or false. When true the beam sprite will show en its box collider 2D is activated. When false the sprite will not show and the box collider 2D is deactivcated.

```
void Update()
{
if (Input.GetKeyDown(KeyCode.Space) && beamActivated == false)
{
beamActivated = true;
}
if (Input.GetKeyUp(KeyCode.Space) && beamActivated == true)
{
beamActivated = false;
}
if (beamActivated == true)
{
Beam.enabled = true;
Shooter.GetComponent<BoxCollider2D>().enabled = true;
}
if (beamActivated == false)
{
Beam.enabled = false;
Shooter.GetComponent<BoxCollider2D>().enabled = false;
}
```

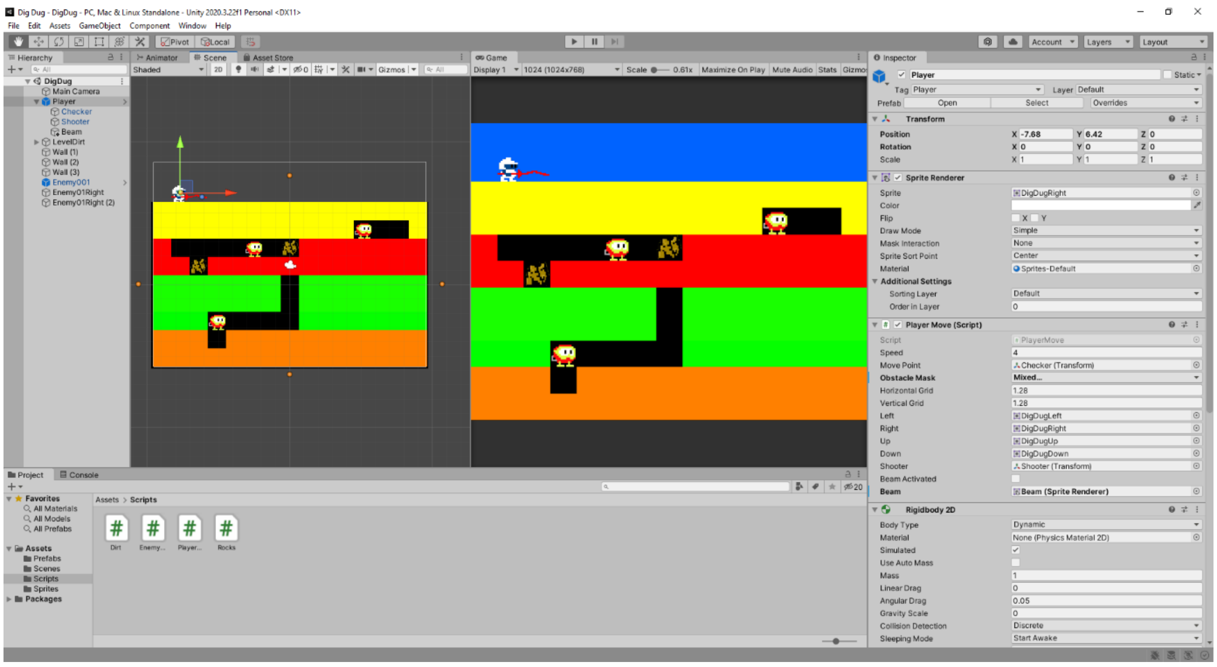


When moving Horizontal the player uses the grid size and the beam position is changed to fit it. The right sprite is used so the player faces the moving position. When moving Vertical the player uses grid size and the beam position is changed to fit it as well as its rotation.

```
float movementAmout = speed * Time.deltaTime;
transform.position = Vector3.MoveTowards(transform.position, movePoint.position,
movementAmout);
if (Vector3.Distance(transform.position, movePoint.position) <= 0.05f)
{
if (Mathf.Abs(Input.GetAxisRaw("Horizontal")) == 1f)
{
if (Input.GetAxisRaw("Horizontal") > 0)
{
Beam.transform.position = new Vector3(transform.position.x + 1.25f,
transform.position.y - 0.04f, transform.position.z);
Beam.transform.eulerAngles = new Vector3(0, 0, 0);
Shooter.transform.position = new Vector3(transform.position.x + 1.5f,
transform.position.y - 0.28f, transform.position.z);
this.GetComponent<SpriteRenderer>().sprite = Right;
}
if (Input.GetAxisRaw("Horizontal") < 0)
{
Beam.transform.position = new Vector3(transform.position.x - 1.25f,
transform.position.y, transform.position.z);
Beam.transform.eulerAngles = new Vector3(0, 0, 0);
Shooter.transform.position = new Vector3(transform.position.x - 1.5f,
transform.position.y - 0.28f, transform.position.z);
this.GetComponent<SpriteRenderer>().sprite = Left;
}
Move(new Vector3(HorizontalGrid * Input.GetAxisRaw("Horizontal"), 0f, 0f));
}
else if (Mathf.Abs(Input.GetAxisRaw("Vertical")) == 1f)
{
if (Input.GetAxisRaw("Vertical") > 0 && transform.position.y < 3.85f)
{
Beam.transform.position = new Vector3(transform.position.x + 0.02f,
transform.position.y + 1.25f, transform.position.z);
Beam.transform.eulerAngles = new Vector3(0, 0, -90);
Shooter.transform.position = new Vector3(transform.position.x - 0.28f,
transform.position.y + 1.5f, transform.position.z);
this.GetComponent<SpriteRenderer>().sprite = Up;
}
if (Input.GetAxisRaw("Vertical") < 0)
{
Beam.transform.position = new Vector3(transform.position.x - 0.02f,
transform.position.y - 1.24f, transform.position.z);
Beam.transform.eulerAngles = new Vector3(0, 0, -90);
Shooter.transform.position = new Vector3(transform.position.x - 0.28f,
transform.position.y - 1.5f, transform.position.z);
this.GetComponent<SpriteRenderer>().sprite = Down;
}
Move(new Vector3(0f, VerticalGrid * Input.GetAxisRaw("Vertical"), 0f));
}
}
}
}
```



There you have it. All the basic stuff needed to create your own levels. These scripts are giving you a good head start so happy game making !



Player — Tag: Player — Layer: Default

Transform
Position X -7.68 Y 6.42 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Sprite Renderer
Sprite: DigDugRight
Flip: X Y
Draw Mode: Simple
Mask Interaction: None
Sprite Sort Point: Center
Material: Sprites-Default
Additional Settings
Sorting Layer: Default
Order in Layer: 0

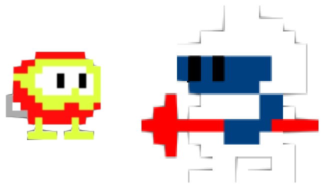
Player Move (Script)
Script: PlayerMove
Speed: 4
Move Point: Checker (Transform)
Obstacle Mask: Mixed...
Horizontal Grid: 1.28
Vertical Grid: 1.28
Left: DigDugLeft
Right: DigDugRight
Up: DigDugUp
Down: DigDugDown
Shooter: Shooter (Transform)
Beam Activated: ☐
Beam: Beam (Sprite Renderer)

Rigidbody 2D
Body Type: Dynamic
Material: None (Physics Material 2D)
Simulated: ✓
Use Auto Mass: ☐
Mass: 1
Linear Drag: 0
Angular Drag: 0.05
Gravity Scale: 0
Collision Detection: Discrete
Sleeping Mode: Start Awake

HOW TO

So this question I see asked regular and it is something that most games use.
How do you find all game Objects with a same tag and destroy them ? (Remove). This can be used for removing game objects after you finished a level or destroy enemies by dropping a bomb, your imagination is the limit.

You can see this all working at : https://youtu.be/5PJtgJWwvOo



<u>Remove All with same tag script</u>.

For this example I used an empty game object and gave it this script.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RemoveAllEnemies : MonoBehaviour
{

/// How To Issue February 2022////
/// René Pol 01-02-2022///
```

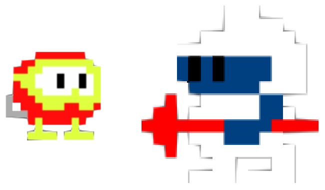
The function removal creates a list of game objects for all game objects that have the same name tag Enemy. Next all those objects will be destroyed because there in that list and we destroy that list objects.

```
public void Removal()
{
GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy");
for (int i = 0; i < enemies.Length; i++)
{
GameObject.Destroy(enemies[i]);
}
}
```

To activate the removal function I use this simple key press. You can call the function however you want depending your game creating needs.

```
void Update()
{
if (Input.GetKeyDown(KeyCode.Space))
{
Removal();
}
}
}
```

But what if I want the objects to get destroyed one by one with some kind of delay ? It is used often for clearing a level of leftover enemies or bonus points. Thanks to my good coding friend Larry Pendleton we created a ready to use script that does all of this. You can set a delay time in the Inspector's view.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RemoveAllEnemies2 : MonoBehaviour
{

/// How To Issue February 2022////
/// René Pol / Larry Pendleton 01-02-2022///
```

A float variable is set so you can choose a delay time. A bool variable is set to determine if the destroying has begun or not.

```
public float destroyDelay = 0.5f;
private bool _inDestroyed;
```

The bool is set to false.

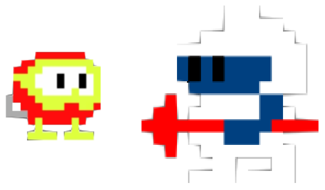
```
void Start()
{
_inDestroyed = false;
}
```

When the space key is pressed and the bool variable is false and only then it sets the bool to true and starts the coroutine. (The destroying begins.)

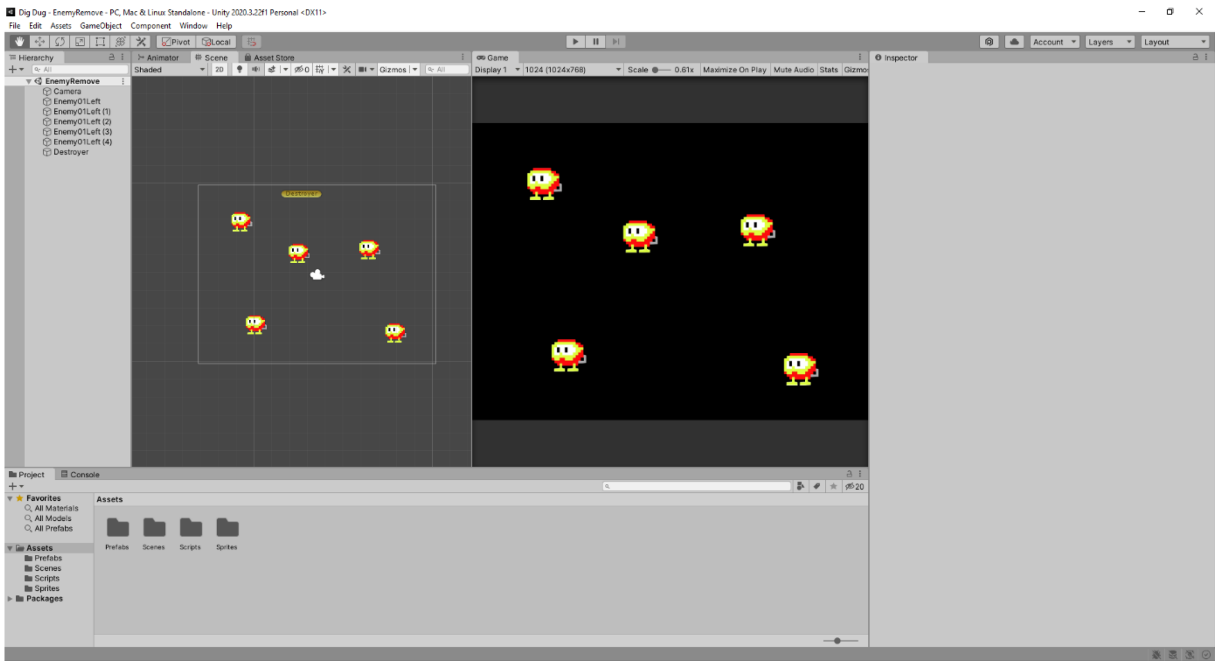
```
void Update()
{
if (Input.GetKeyDown(KeyCode.Space) && !_inDestroyed)
{
_inDestroyed = true;
StartCoroutine(RemoveAllEnem());
}
}
```

The function we had in the previous script is now a coroutine that will use the delay after a game object is destroyed before it destroys the next one.

```
private IEnumerator RemoveAllEnem()
{
GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy");
foreach (var enemy in enemies)
{
Destroy(enemy);
yield return new WaitForSeconds(destroyDelay);
}
}
}
```



You can add more things to the destroy like sound effects or scoring per enemy. You now know how To do it and use it yourself.


Dig Dug - EnemyRemove - PC, Mac & Linux Standalone - Unity 2020.3.22f1 Personal <DX11>
File Edit Assets GameObject Component Window Help
Pivot
Local
Account
Layers
Layout
Hierarchy
EnemyRemove
Camera
Enemy01Left
Enemy01Left (1)
Enemy01Left (2)
Enemy01Left (3)
Enemy01Left (4)
Destroyer
Animator
Scene
Asset Store
Shaded
2D
Gizmos
Game
Display 1
1024 (1024x768)
Scale
0.61x
Maximize On Play
Mute Audio
Stats
Gizmos
Inspector
Destroyer
Project
Console
Favorites
All Materials
All Models
All Prefabs
Assets
Prefabs
Scenes
Scripts
Sprites
Packages
Assets
Prefabs
Scenes
Scripts
Sprites
20