For this month's issue I created a small and simple game. It might be cliché but it's true, when you start at game design you have to begin with small projects. I myself wanted to do huge projects right away, but found out the hard way I was not up to it yet, so the projects failed and ended up on the shelve. This game uses only 3 Scripts I created. We will go through all of them explaining what it does and how. Experiment with these scripts and see if you can change them. You can see the game creation itself at: https://youtu.be/muZWbRSKvq4

First here are some basic things you need to know about programming in C# (C-Sharp)

- C-Sharp is an object-based programming language. So, if you want a game Object to do something it will need its own script.
- A function is called void so a function called flying is noted as `void flying()`
- A function line ends with () sometimes with instructions between the ()
- Most code (instructions) will be between an opening and closing bracket {}
- Semicolons are used to define the end of a programming (instruction) line.
- If you want to write comments in your script you have to put `//` before it. This way it won't be seen as part of your script.
- A clean new script will have 2 empty functions in it. void Start() and void Update()
- A few operators are: **>** Bigger than **<** Smaller than **==** Equal to **!=** Other than **<=** Smaller or Equal **>=** Bigger or Equal **&&** And and || = Or or.

Now let's start with the Coinfall script.

The first 3 lines are libraries that come with Unity. They will be added as soon as we create a new script. In the editor we can add things which will have scripts from those libraries.

```
//////////CoinDrop Game Script by René Pol RP-Interactive.nl© 30-03-2021/////////


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

The public class line is nothing else than a line that holds the name of the script. Make sure this is always the same otherwise your script won't be found and it will not work. It will start the script by using the opening bracket. {

```
public class Coinfall : MonoBehaviour
{
```

Next, float is a variable that stores numbers with numbers behind the comma to. For example, 1.2f . In this case it stores a number we call Falling Speed. The public part gives us the option to change it in the inspector's view of Unity and makes it available to be changed with script as well. If you use private instead of public you won't get this option. AudioClip Yahoo means that we will use an audio file that we name (Define) as Yahoo. Since we made it public, we will see it in the inspector view of Unity. This way we can drag and drop any audio file we want to use to this spot.

```
public float FallingSpeed;
public AudioClip Yahoo;
```

We come to the first real function (void). I made it public so it would be possible to change or call it by script. The name of this function is OnCollisionEnter2D. In Unity we gave the coin a Circle Collider 2D so the game Object (The coin) should be able to collide with other objects. Keep in mind in order for collision to work at least one of the colliding objects needs to have a Rigidbody2D otherwise there will be no collision at all. At the end we set that the collision should be between the object itself and another game object.

```
public void OnCollisionEnter2D(Collision2D other)
```

We gave the other object a name tag and called it Pipe so the next line means: If the other game object is the object with the name tag Pipe then do the next thing.

```
{
if(other.gameObject.tag == ("Pipe"))
{
```

It will get the audio we defined earlier and plays it at a given point in the scene. Vector3 hold 3 coordinates: x,y,z axis. -10 on the z-axis is the same spot as the Camera so the sound will be close to it. The further away you set the Vector3 the lower the volume of the sound will be.

```
AudioSource.PlayClipAtPoint(Yahoo, new Vector3(0, 0, -10));
```

The next line makes a reference to another script called Scores where a variable scorePoints is located.1 point is added to that variable. Later we will go through this Scores script as well, don't worry.

```
Scores.scorePoints += 1;
```

The colliding object itself so not the one with the name Tag Pipe will get removed by using Destroy, this makes sure it will be the game Object itself. If you want a delay before it gets removed you only have to add a comma with a number of seconds to wait written down in the form of a float variable. Example: Destroy(this.gameObject,2f); The void is closed with two closing brackets.

```
Destroy(this.gameObject);
}
}
```

A function that will be executed all the time during the game is called Update. It is a function that keeps repeating all in there for every frame. Think of a game as a movie that shows itself frame by frame. You might know the expression framerates? Well, that means the speed that the frames are shown in. There is no need to make this function public or private so it's just void Update() This function is standard there when you create a new script.

```
void Update()
{
```

To make the object move it uses transform and Translate, so move from the objects start point (transform.Translate) to a direction given (Vector2.down) with the speed of the variable we created earlier. To make sure it runs the same speed on any computer system we use Time.deltaTime (All about the framerates. The Asterisk symbol stands for multiply.

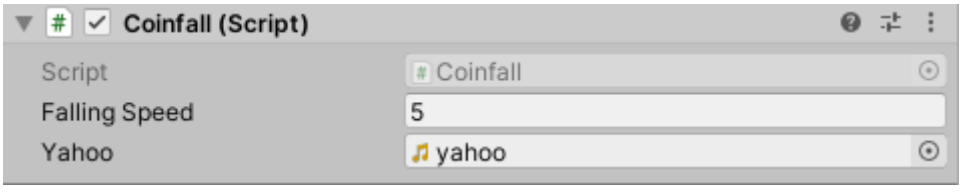
Vector2 holds 2 coordinates: x,y-axis. Instead of using numbers in this case we used the word down. This means the movement will be on the y-axis and goes down while there will be no movement on the x-axis. Try changing it into up,left or right and see what happens. Since this is in the Update function (void) it will keep moving every frame.

```
transform.Translate(Vector2.down * FallingSpeed * Time.deltaTime);
```

We can see the coordinates of the game Object in the editor. So, when you place it below you can see where it becomes no longer visible (out of screen). It's location on the Y-axis is than -6. So, if(when) the position of the Object(transform) is smaller than -6 it will be removed in a way we learned earlier.

```
if(transform.position.y < -6)
{
Destroy(this.gameObject);
}
}
}
```

So, there you have it. Script one fully explained. Experiment with it and try to change things. In this one script there are many elements of programming. When this script is added to the game Object it will show you this information in the inspector's view:



Remember the float variable and the defined music file you both made public? This is the result.

What you learned so far:

- Library lines of Unity
- The use of brackets,Seimicolons, parentheses, Comments
- Public Class
- Variable float
- Defining Audio file
- Void OnCollisionEnter2D
- Name Tag use
- Change variable in another script
- Play sound at point in scene
- Vector3 and Vector2
- Void
- Use of public/private
- Destroy (Remove Objects)
- Update()

There is no use in explaining the same things again so I will only explain the new parts. The second script we will explore is:

the Scores script.

To create a score, I made a UI text element. In order to use Unity's UI parts, we need to add one more library line. If you do not add this line to your script you will get errors because the script needs this library in order to work. Using UI elements will make a canvas that will function as an overly to your project. All you put on the canvas will stay on the same spot like a painting on the camera lens.

```
//////////CoinDrop Game Script by René Pol RP-Interactive.nl© 30-03-2021/////////
```

```
using UnityEngine.UI;
```

Next, static int is a variable that stores numbers that are integer. Static means once it's initialized it will exist to the end of our game. It's made public and you should know why. In the previous Script we saw that it uses this public static int right? (`Scores.scorePoints += 1;`*)*

```
public static int scorePoints = 0;
```

Then we define the UI text element we created and give it the name score. Because its public you should be able to drag the element to the inspector's view.

```
public Text score;
```

Then we come to this standard function that comes with a new script. It's the Start function (void). All that's in there will be executed as soon as the game object is active. There is no need to make this private or public. In this function the game Object will use the UI library to get its UI element that it should use. It retrieves the text we defined as score. GetComponent<Text>()

```
void Start()
{
score = GetComponent<Text>();
}
```

In the update function we will make sure that the Text UI element is showing out variable scorePoints by turning this int variable into a string. The score will be made visible with as many numbers as you fill in zeroes. String variables hold letters so we can add words to the score to.

```
score.text = scorePoints.ToString("00");
```

by turning the int variable into a string you can add any text to the UI element by writing it like this:

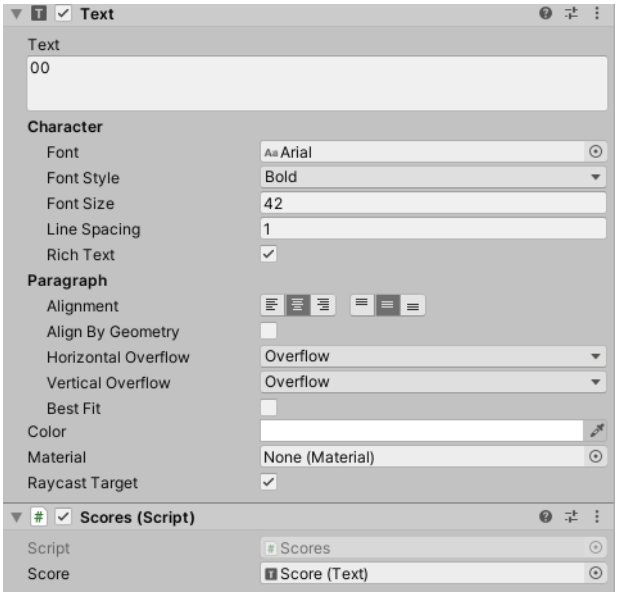
```
score.text = scorePoints.ToString("Score:" +"00");
```

If you want to add more text on a next line you write it like this:

```
score.text = scorePoints.ToString("Score:" +"00" + "\n" + "Nice");
```

In retro games you see often a score that is shown in 6 or 8 digits. Well now you know how to do that. Simple write score.text = scorePoints.ToString("00000000"); In the Inspector's view you can add fonts and set all the different options like size and color to your likings.

When you give this script to the UI element it should display its settings and the script public parts in the inspector's view.



What you learned so far:

- UI library Unity
- UI text element
- GetComponent
- Int variable
- Static
- Defining Text element
- Start function (void)
- String variable
- Turn int into string variable
- Using \n (new line) in string variable

We handled just two scripts but learned pretty much don't you agree? For each script you should make sure you understand all and try out things yourself. It's time to explore the last script.

the MarioFly script

In this script we will handle scene (level) loading. Unity has some handy tools for it but again we need its library so we add this extra line.

```
using UnityEngine.SceneManagement;
```

Two float variables are created for FlySpeed and Coinnr. The Coinnr is set to 5.

```
public float FlySpeed;
public float Coinnr = 5;
```

Next, we define 2 Game Object that are in our game project by writing Game Object followed by the names separated with a comma. You can now drag and drop those objects into the inspector's view.

```
public GameObject Coins,Restart;
```

We create a spot where the coins should come from by typing Transform(The spot) followed by a name for it. You can now drag and drop those objects into the inspector's view.

```
public Transform CoinDropper;
```

In the start function (You know what it does now) we make the game Object we defined as Restart invisible (Inactive) by turning the SetActive to false. True would make it active and visible.
You can now drag and drop this object into the inspector's view.

```
Restart.SetActive(false);
```

In the Update function a lot is happening. First when a key (in this example the R key) is pressed it sets the score to 0 by reaching out to the score script and setting it's scorePoints variable to 0.

```
if(Input.GetKeyDown(KeyCode.R))
{
Scores.scorePoints = 0;
```

Next it will use the Scene Management library line to reload our scene. You know the name of your scene as you saved it under the same name. (See it on to of unity editor)

```
SceneManager.LoadScene("SampleScene");
```

If(when) the Coinnr variable is equal to zero and only them it will make the Restart game object active.

```
if(Coinnr == 0)
{
Restart.SetActive(true);
}
```

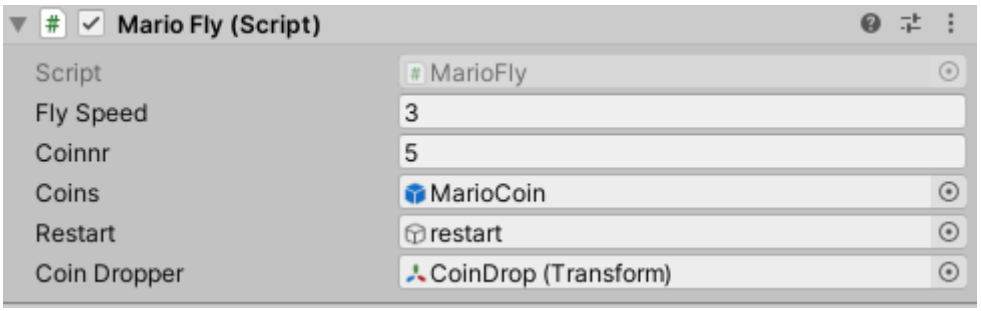
If(when) the spacebar key is pressed and the Coinnr variable is bigger than zero and only then. Create(Instantiate) the defined game object(Coins) at position of the defined Transform(spot = Coindropper.position) in its own rotated start position(Quaternion.identity). Also 1 is subtracted from the Coinnr.

```
if (Input.GetKeyDown(KeyCode.Space) && Coinnr >0)
{
Instantiate(Coins, CoinDropper.position, Quaternion.identity);
Coinnr -= 1;
}
```

The movement part you should understand how it works. It's like the coin movement but it uses right instead of down. When the position on the x-axis is bigger than 8, we teleport it to a new place. New Vector 3 followed by the coordinates on the x,y and z-axis.

```
transform.position = new Vector3(-8, transform.position.y, transform.position.z);
```

When you give this script to game Object it should display this the inspectors view.



What you learned so far:

- On key down press
- Input
- SceneManager
- LoadScene
- SetActive
- true/false
- Instantiate
- Transform
- GameObjects
- Quaternion.identity
- New Vector3 (Teleport)

Great! you came a long way and with these 3 scripts you got a good basic understanding of coding in C# and working with Unity. That's it for the Game project this issue. I hope you enjoyed it and learned from it.

Each issue I will have a section that will take just a part from any game to explain how you can do it yourself. This section will be known as How To ?

How To ?

Do you know the game Castle Crush? In order to create the players, cards are used that must be built first. This thing is used many times in card games but also strategy games where you have to click objects to build units. By using UI elements this is very easy to create yourself. You can see the creation here: https://www.youtube.com/watch?v=2VAEUsmYkiE

Let's have a look at the script which will have some new parts.

The effect is made with 2 versions of 1 card. One version (the back) is the bright version. The other is a darker version that will cover the back. Since these are UI images make sure you use the UI line for including the UI library. (Remember?)

The card on top is the one with the script. It uses the fill radial 360 effect settings that come with unity. Watch the clip to see how to set all the options for this script.

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

public class BuildingCard : MonoBehaviour
{
```

A public image is created that will use the fill effect. You can drag it into the inspector's view.

```
public Image Golemcard;
```

A bool variable is created that will decide if the building is on or off. A bool variable uses only true or false. We set the building on true. Because it's public we can see if its activated or not in the inspector's view.

```
public bool building = true;
```

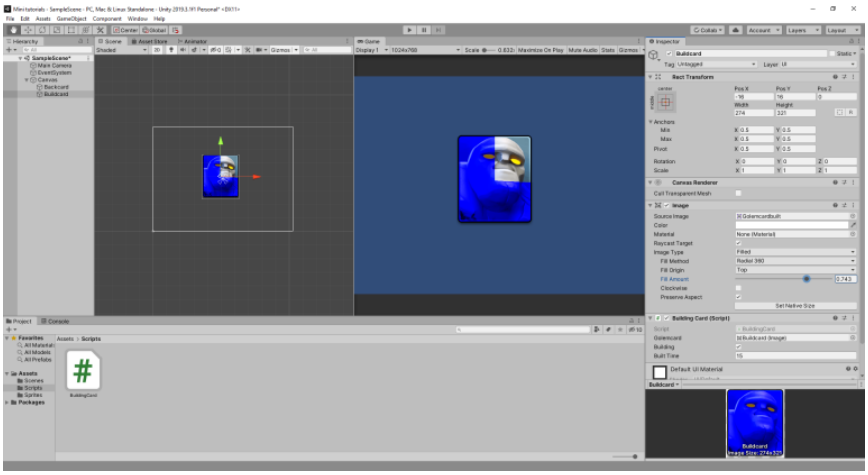
A public float is created for how long it should take to build a card. Set it to 15.0f;

```
public float builtTime = 15.0f;
```

In the update function it states that if(when) the bool variable is set to true (Equal)
The fill effect from our defined image will go down -1.0f divided by the builtTime. Time.deltaTime should be familiar to you now. If(when the fill effect from our defined image is equal to zero the fillamount is set back to 1. The fill effects will always use a range between 1 and 0 with 0 not being shown and 1 fully visible. That's why the fill amount goes down divided by our given built time variable.

```
void Update()
{
if (building == true)
{
Golemcard.fillAmount -= 1.0f / builtTime * Time.deltaTime;
if(Golemcard.fillAmount == 0)
{
Golemcard.fillAmount = 1;
}
}
}
}
```

The built time is shown in the inspector's view so you can change the built speed in any speed you like. You can use this script with all the fill effects that unity holds. Experiment with all the different use of fill effects.



All of the assets used in this issue can be downloaded here : https://rp-interactive.nl/Magazine.html

All scripts used in issue 01 are here for you.

Coinfall.

```
//////////CoinDrop Game Script by René Pol RP-Interactive.nl© 30-03-2021/////////

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

public class Coinfall : MonoBehaviour
{
public float FallingSpeed;
public AudioClip Yahoo;

public void OnCollisionEnter2D(Collision2D other)
{
if (other.gameObject.tag == ("Pipe"))
{
AudioSource.PlayClipAtPoint(Yahoo, new Vector3(0, 0, -10));
Scores.scorePoints += 1;
Destroy(this.gameObject);
}
}

void Update()
{
transform.Translate(Vector2.down * FallingSpeed * Time.deltaTime);
if (transform.position.y < -6)
{
Destroy(this.gameObject);
}
}
}
```

Scores.

```
//////////CoinDrop Game Script by René Pol RP-Interactive.nl© 30-03-2021/////////

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Scores : MonoBehaviour
{
public static int scorePoints = 0;
public Text score;

void Start()
{
score = GetComponent<Text>();
}
```

```
void Update()
{
score.text = scorePoints.ToString("00");
}
}
}
```

Mariofly.

```
//////////CoinDrop Game Script by René Pol RP-Interactive.nl© 30-03-2021/////////

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MarioFly : MonoBehaviour
{
public float FlySpeed;
public float Coinnr = 5;
public GameObject Coins,Restart;
public Transform CoinDropper;

void Start()
{
Restart.SetActive(false);
}

void Update()
{
if(Input.GetKeyDown(KeyCode.R))
{
Scores.scorePoints = 0;
SceneManager.LoadScene("SampleScene");
}
if(Coinnr == 0)
{
Restart.SetActive(true);
}
if (Input.GetKeyDown(KeyCode.Space) && Coinnr >0)
{
Instantiate(Coins, CoinDropper.position, Quaternion.identity);
Coinnr -= 1;
}
transform.Translate(Vector2.right * FlySpeed * Time.deltaTime);
if(transform.position.x > 8)
{
transform.position = new Vector3(-8, transform.position.y, transform.position.z);
}
}
}
```

BuildingCard.

```
//////////Building card Script by René Pol RP-Interactive.nl© 30-03-2021//////////


using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class BuildingCard : MonoBehaviour
{
public Image Golemcard;
public bool building = true;
public float builtTime = 15.0f;

void Update()
{
if (building == true)
{
Golemcard.fillAmount -= 1.0f / builtTime * Time.deltaTime;
if(Golemcard.fillAmount == 0)
{
Golemcard.fillAmount = 1;
}
}
}
}
```