LP2B – Multimedia: Digital Representation

- LP2B Practicum Class n°1 -

Your First steps with Unity Engine

GameObjects, Components and Assets

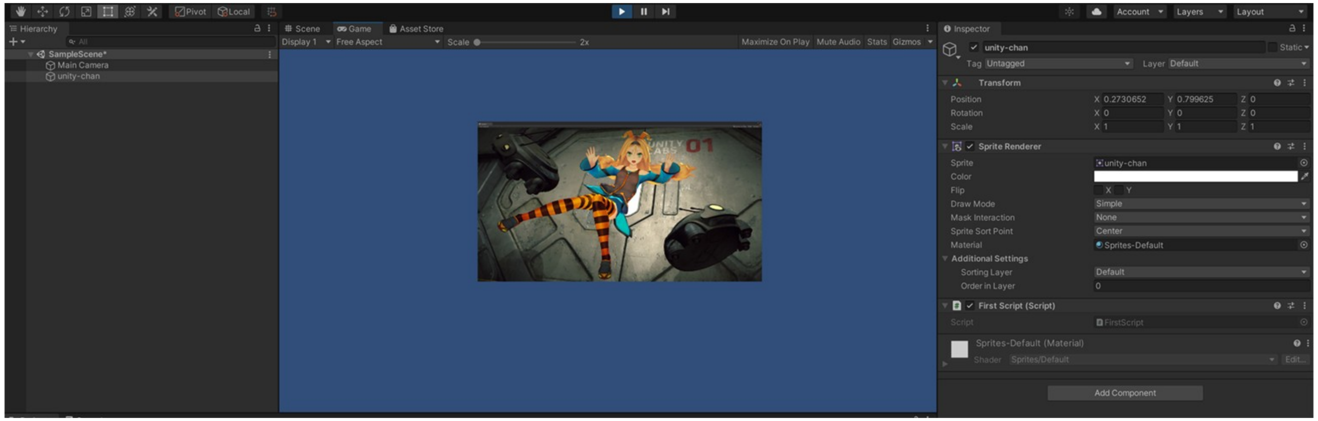
This Practicum is about applying knowledge from Tutorial (TD) n°1

You will also see the following new notions :

- Physics Materials for friction and bounciness-
- Sprite Tiling (repeat it over a surface) -
- Using Debug.Log console messages -

First, let's catch up !

The first Tutorial was a bit hard to finish in time, as problems could pop up for some people during installation *(slow connection, install problem…)*



unity-chan
Tag Untagged Layer Default

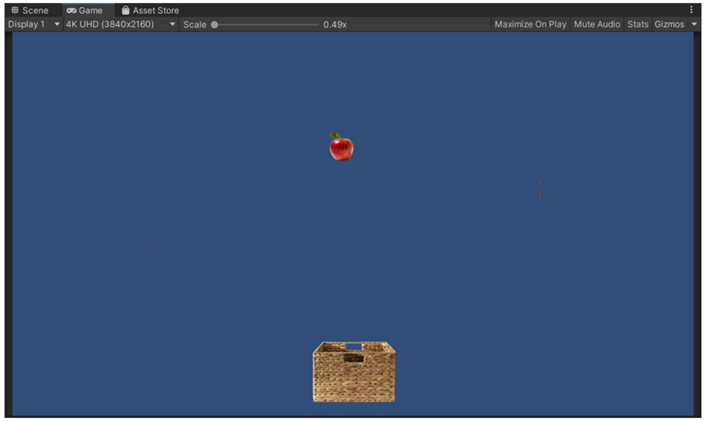
Transform
Position X 0.2730652 Y 0.799625 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Sprite Renderer
Sprite unity-chan
Color
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

First Script (Script)
Script FirstScript

Sprites-Default (Material)
Shader Sprites/Default

Add Component



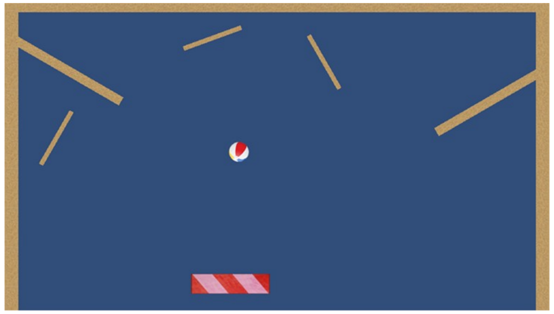
So, we're going to do some catch up first !

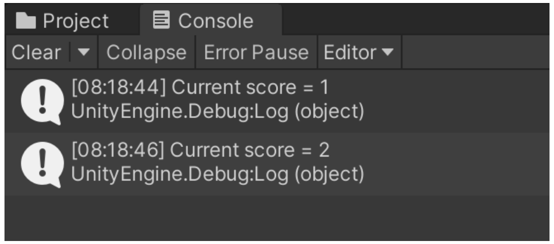
Please open the Presentation/powerpoint of the first tutorial (TD) for a while

(we'll come back here when we're done catching up)

So… What are we going to Make ?

- We want to create a « ball bouncing » mini-game.
  The player controls a bouncy pad, and will try to keep the ball in the air as long as possible
- In this game, most of the work will be done by the physics engine
  This will allow us to make something fun, despite our lack of experience
- Walls, a roof and obstacles will ensure the ball can't escape the camera field
  Except by falling down, which is obviously the lose condition of our mini game
- You will have to use MOST OF THE NOTIONS from the Tutorial Class (TD in French)
  You can use the PDF from that class as a reference today, if you forgot anything
- Image files are available in the public folder of the class
  Import them as Assets and use them as basis for your (visible) GameObjects
- Make sure TO REVIEW THE NEXT SLIDES before starting !
  They contain more guidelines on what to do, and teach you a few things you should find useful


A preview of how the game might look.
You are free to create a different « level »



```
[08:18:44] Current score = 1
UnityEngine.Debug:Log (object)
[08:18:46] Current score = 2
UnityEngine.Debug:Log (object)
```

We are going to display the score through debug messages. This is not acceptable in a "real" app, of course. But we're learning!

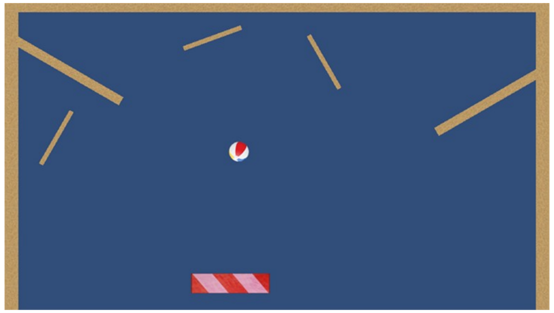
The Primary Objectives

Here are the features you must implement :

(Red text is about things we didn't see in tutorial class. Learn more in the next slides)

- A player-controlled « paddle » which moves left and right with arrow keys
- A Ball which is connected to the physics engine *(with RigidBody2D and CircleCollider2D)*
- The Ball must bounce on the paddle, without displacing the paddle
- Static walls and ceiling, connected to the physics engine as well
- The Paddle needs a physics material to make it extra bouncy
- Count the number of times the paddle collides with the ball
- Display that Number in the console through Debug.Log messages
- The walls should use tiled sprites to look good whatever their shape

…If you're fast, there are extra objectives you can do afterwards.
But make sure you complete all of those first!



Make sure you check the next 3 slides to learn how to solve the "red problems"!

They require simple, but new notions we have not seen in Tutorial!

A Few Things about C#

- Since you will start coding on your own, I will explain quickly how the basics of programming apply to the C# language

- "If - else" statements, the most basic building blocks of code, are obviously in C#
  Accolades are used to delimit the block of code if there are several instructions

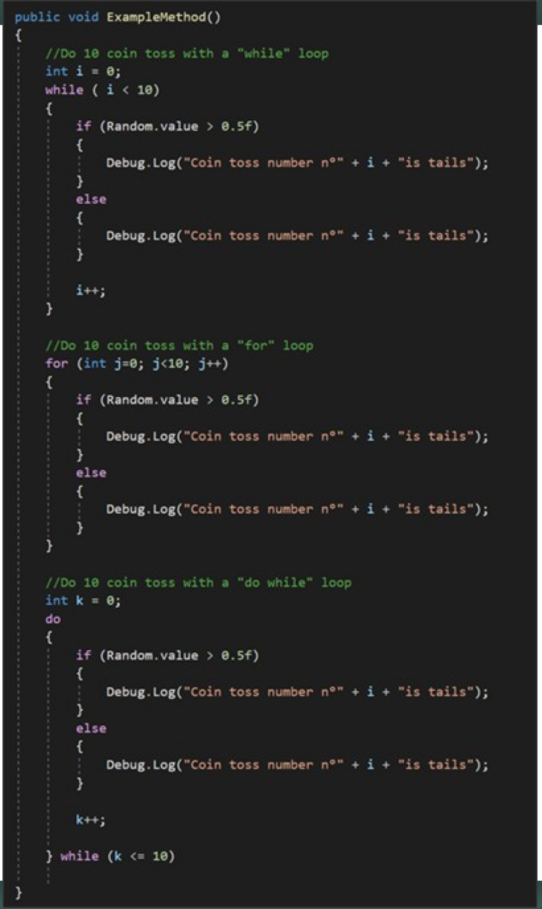
- All the usual loops are available : "for", "while", and "do…while"
  Their syntax is similar to other C languages (and to most existing languages in the world)

- You have a basic example of all those loops to the side :

- Remember that a standard loop will ENTIRELY EXECUTE WITHIN ONE FRAME
  Complex loops that execute too many times will make your app lag, as they will occur within 1 frame!

- To have a process happen over several frames, you have to use "Update()"
  …and be a bit more clever about it!
  Update() can be seen as a loop itself in a way, since it executes each frame

- You can also define your own custom Methods, like in the example
  All the loops are part of a method called ExampleMethod. See definition at the top



```
public void ExampleMethod()
{
    //Do 10 coin toss with a "while" loop
    int i = 0;
    while ( i < 10)
    {
        if (Random.value > 0.5f)
        {
            Debug.Log("Coin toss number n°" + i + "is tails");
        }
        else
        {
            Debug.Log("Coin toss number n°" + i + "is tails");
        }

        i++;
    }

    //Do 10 coin toss with a "for" loop
    for (int j=0; j<10; j++)
    {
        if (Random.value > 0.5f)
        {
            Debug.Log("Coin toss number n°" + i + "is tails");
        }
        else
        {
            Debug.Log("Coin toss number n°" + i + "is tails");
        }
    }

    //Do 10 coin toss with a "do while" loop
    int k = 0;
    do
    {
        if (Random.value > 0.5f)
        {
            Debug.Log("Coin toss number n°" + i + "is tails");
        }
        else
        {
            Debug.Log("Coin toss number n°" + i + "is tails");
        }

        k++;

    } while (k <= 10)

}
```

How can we make the Paddle Bouncy ?

- The Ball moves, hits the paddle… and it doesn't bounce back !

  By default, RigidBodies are not bouncy : objects that collide them are not pushed back !

- PhysicsMaterials are a specific Unity assets that can change this

  And because this is a 2D project, we are going to use the PhysicsMaterial2D variant

- To create one of these PhysicsMaterials :

  (right click in Project Panel => Create => 2D => Physics Material 2D)

- Select it in Project Panel so its properties appear in inspector

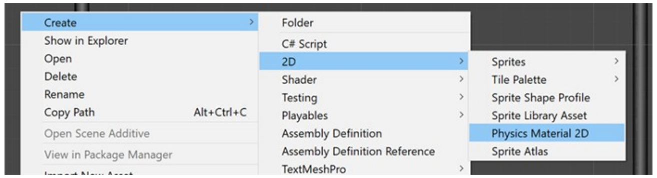
- Change « friction » of the material to 0 and « Bounciness » to 1.2

  This will create a material that is 120 % bouncy, which will re-accelerate a slow ball

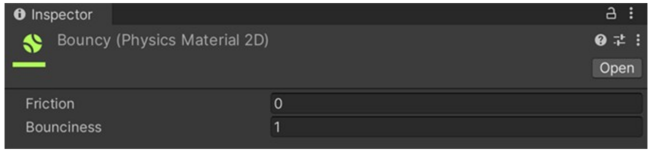
- You can rename the material to "bouncy", to keep track of what it does

- Set this Bouncy material as the PhysicsMaterial for the paddle's RigidBody

  To do that, Drag and Drop the material into the field available for that on RigidBody2D



Creating a PhysicsMaterial2D

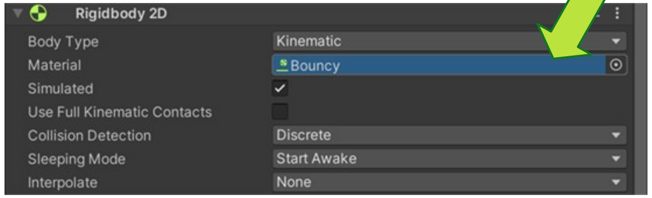


Bouncy (Physics Material 2D)

Friction 0

Bounciness 1

The Physics Material 2D in the inspector, with friction and bounciness fields



Rigidbody 2D

Body Type	Kinematic
Material	Bouncy
Simulated	✓
Use Full Kinematic Contacts	
Collision Detection	Discrete
Sleeping Mode	Start Awake
Interpolate	None

The Material field of RigidBody2D, where you must drag and drop your material

Displaying Console Messages

- Unity has a console, which you can see in the "Console" Tab

  By default, it should be at the bottom. Right next to the project tab

- This Console is similar to the ones you might have seen in other languages :

  It is a simple interface that can display text messages when prompted by the code

- To display a message in the console, use Debug.Log(*string toDisplay*)

```
Debug.Log("A string to display in Console");
Debug.Log("You can concatenate with variables : " + score);
Debug.Log("For some types, you need to convert :" + gameObject.ToString() );
```

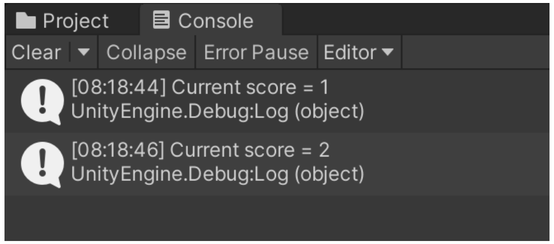
- The Console is an easy window towards your code. A great tool for debugging!

  By using Debug.Log actions, you can track the behavior of your code and identify bugs

- The Console commands will NOT be part of the final, compiled app.

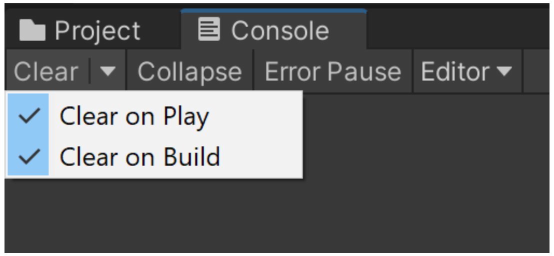
  By default, Unity will automatically remove all Console actions when compiling the app

  You do not need to delete your Debug.Log instructions before building!



```
[08:18:44] Current score = 1
UnityEngine.Debug:Log (object)
[08:18:46] Current score = 2
UnityEngine.Debug:Log (object)
```

The messages will display in the console like this as the app runs. They will stay there after the test, allowing you to check them later

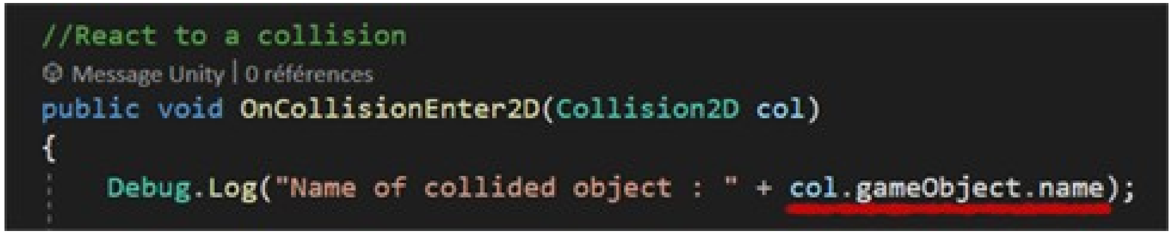


✓ Clear on Play
✓ Clear on Build

The clear button allows you to clear the console The arrow to the side of that buttons allows you to choose when the console auto-clears

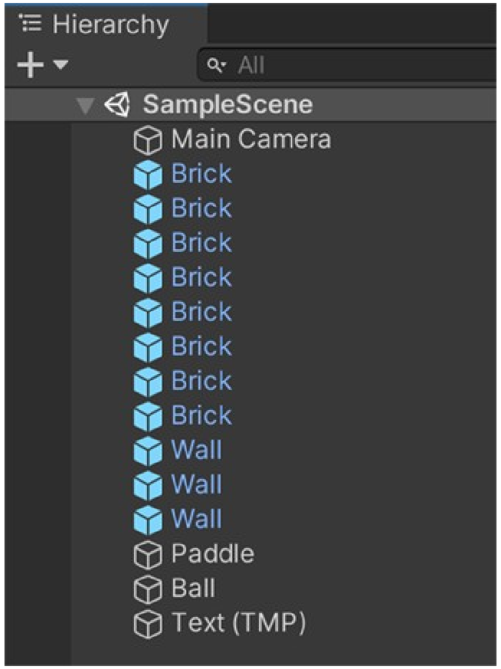
Identifying a Collided Object

- As expected, the score increases whenever a collision occurs !
  So we gain score when the paddle and/or ball touches a wall! (depends on how you coded it)
- The solution to this issue ? Identifying the object collided before giving points
  There are several ways to do this, but we will begin with the simplest : GameObject names
- The names visible in the hierarchy tab are the names of the GameObjects
  You can rename them. The same way you would rename any entity in Unity
- You can either count score from a script on the ball, or from one on the paddle
  And, of course, the code will be slightly different depending on what you choose
- In OnEnterCollision2D method, you get the name of the collided object with :



```
//React to a collision
Message Unity | 0 références
public void OnCollisionEnter2D(Collision2D col)
{
    Debug.Log("Name of collided object : " + col.gameObject.name);
```

We access the properties of the « col » parameter. It is of type Collision2D and contains all the info about the collision that occured

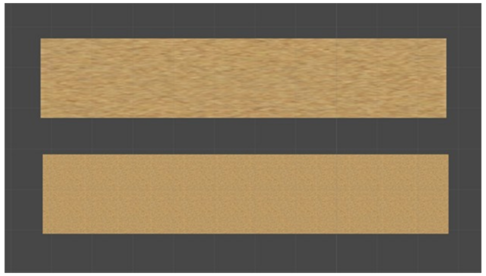


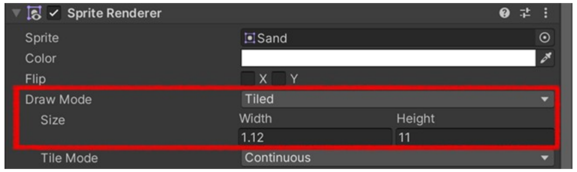
```
Hierarchy
+ ▾                 All
▼ SampleScene
    Main Camera
    Brick
    Brick
    Brick
    Brick
    Brick
    Brick
    Brick
    Brick
    Wall
    Wall
    Wall
    Paddle
    Ball
    Text (TMP)
```

As you can see on this screenshot from another project, GameObjects can have the same name. It's not a problem !

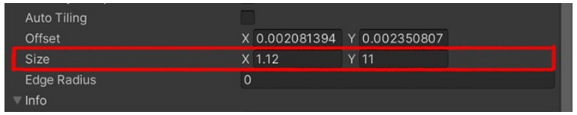
Tiling Sprites

- To create our walls, we have a seamless sand texture we want to use
  "Seamless" means it can be used as a repeating pattern and appear as if it was one image
- Problem : when we resize, the sprites are distorted rather than used as pattern
  This is because the default "Draw Mode" of a sprite is "Simple", which does that
- In SpriteRenderer, change the "Draw Mode" to "tiled", and tadah!
  If you distort the sprite on the scene, it appears we're good. But what about that warning?
- For optimum results, sprites you use for tiling should be "full rect"
  This is a setting called "mesh type", on the sprite itself. By default, it is set to "tight"
  To access the properties of the sprite itself, click it in the Project/Asset tab
- Unlike regular sprites, the size of a tiled sprite can be defined numerically through a "width" and a "height" setting. And not just by scaling.
  Note that when you add box colliders, you can use these numbers to ensure they match!


A distorted "simple" sprite (top) VS a tiled sprite with a repeating pattern (bottom)

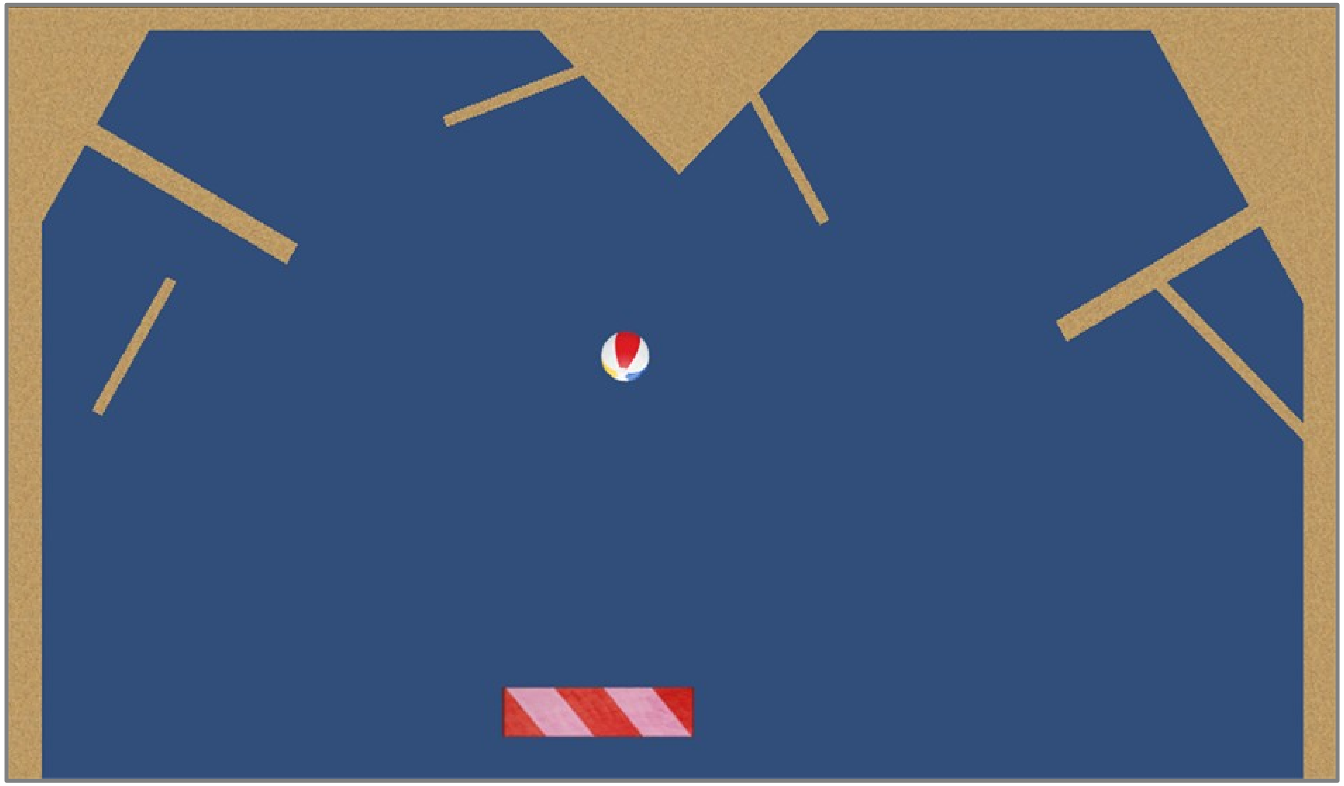

Sprite Renderer
Sprite: Sand
Color
Flip: X Y
Draw Mode: Tiled
Size: Width 1.12, Height 11
Tile Mode: Continuous

A SpriteRenderer set to "Tiled", you can see that New Width and Height properties appear


Auto Tiling
Offset: X 0.002081394 Y 0.002350807
Size: X 1.12 Y 11
Edge Radius: 0
Info

I used the same numbers for the BoxCollider, and then it matches my wall perfectly!

Now, it's your turn !



Nice ! You made your first Unity Minigame!

It is, of course, super simple. But don't forget that you managed to do this in less than 2 hours, despite being a beginner!

If you want more challenge, there is a bunch of extra objectives for you to tackle in the remaining time!

Give it a Shot!

Too easy ? Have some more Objectives !

Here are bonus objectives you can attempt if you have time :

They are not mandatory. Advice on how to accomplish them are available on the next slide

```
if ( transform.position.y < -10 )
```

```
transform.position = new Vector3(-6.65f, 3.82f, 0f);
```

```
transform.Translate( -8.5f * Time.deltaTime, 0, 0 );
```

Acting on the position is enough to achieve almost all of those objectives : Translate it, test it, or set it directly to a value!

- Adjust the speed of the paddle so the game feels as fun as possible
- Prevent the paddle from being able to move outside the screen
- Create a moving wall than will annoy the player
- Make sure the moving wall doesn't go away (with looping movement)
- If the ball falls below the screen, it re-spawns at its starting position
- When re-spawning the ball, reset its velocity to 0 with this :
- When re-spawning the ball, set the score back to 0

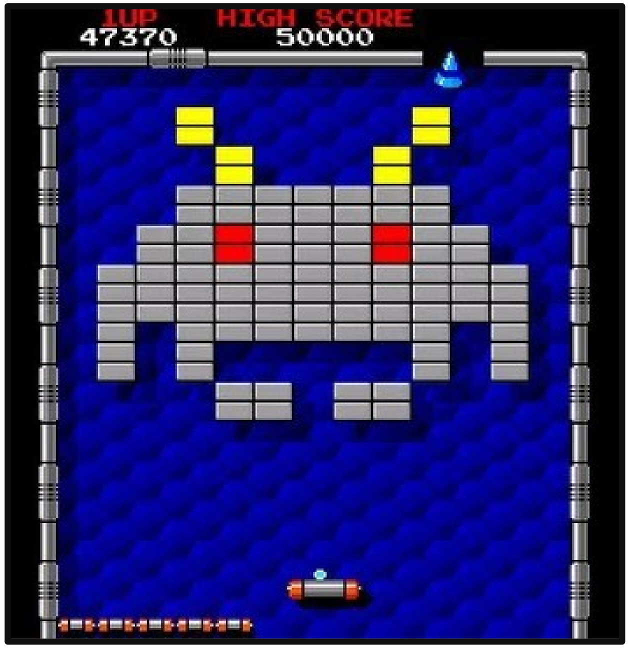
```
GetComponent<Rigidbody2D>().velocity = new Vector2(0, 0);
```

This line of code makes more sense after Tutorial (TD) n°2. Depending on your group, it might not have happened yet. So accept it as it is!

Remember that you must send me today's work within 72 hours!

You can complete those objectives as home. They are not mandatory but will give you a small boost on your final grade. Not doing the primary objectives of slide 4, however, will result in a penalty

Next time, in Practicum Class…



Arkanoid, a sight that should be familiar to people who like retro gaming!

(or old farts like myself)

- We will expand on the "bouncy paddle used to catch a ball" game mechanic that we prototyped today
- We'll use it as basis to make a "brick breaker" game!
- It shares a lot of similarities with what we did today, but we will use dynamic object instanciating to generate bricks through code, rather than manually
- Using it, we will even be able to generate random levels through an algorithm, for infinite replayability!
- You will also use sound for the first time on your own *(we will have seen it in tutorial class before, by then)*

See you Next time!