LP2B – Multimedia: Digital Representation

- LP2B Practicum Class n°2 -

« Brick Breaker » Minigame

Using more component types and understanding their C# logic further

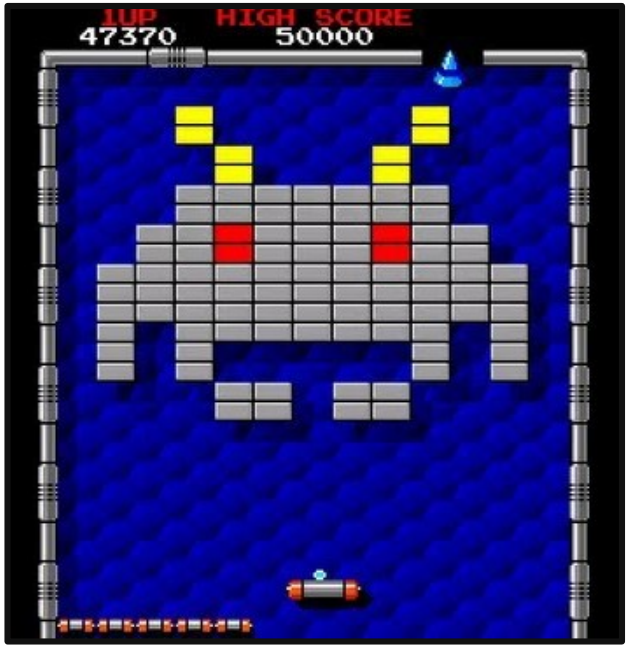
This Practicum is about applying knowledge from the first 3 Tutorials

You will also see the following new notions :

- Advanced loops for level generation-
- Communication between GameObjects -
- Randomize color (not mandatory) -

What are we going to Make ?

- We want to create a « brick breaking » mini-game. (A classic !)
  The video to the side is from « Arkanoid », a good example of this retro type of game
- Your goal : copy as many features of Arkanoid as you can in the given time
  The next slide details what features have priority, and which are « nice to have »
- Thanks to the power of Unity, a modern tool, we can do basics very quick
  Back in the day, those were done in Assembly or primitive versions of C. ...Much harder !
- You will have to use ALL THE NOTIONS from the Tutorial Classes (TD in French)
  You are free to use the Powerpoint presentations from the tutorials as reference
- 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 !



A picture is worth 1000 words !
...But please look at the objectives slide for a breakdown of what we want

Phase 1/3 : The Basics

If we implement all those features, we'll have the basics down :

- A player-controlled « paddle » which moves left and right with the arrow keys
- Static walls and ceiling, connected to the physics engine through RigidBodies
- A Ball which ignores gravity and bounces on those walls
- The Ball must bounce on the paddle, without displacing the paddle in the process
- A Brick Prefab which self-destructs when it collides with something else
- Place a few of those prefabs manually in the scene to create a level
- The Paddle should not be able to move outside the play area
- Create a Text Field for the Score and place it outside the playing field
- Have each brick destroyed grant the player 50 points, updating the display

Our Paddle goes through the Walls !

- If you followed the previous classes, your paddle should be « Kinematic »
  Reminder : A Kinematic rigidbody ignores outside forces, and only moves according to script
- But the thing is… a wall blocking the way is also an « outside force » !
- Mixing the behaviors of « Dynamic » and « kinematic » is tough for beginners…
  So we're going to cheat and block the paddle based on its X position !
  Move your paddle on the scene to check at what x coordinates are « hitting the walls »
- You can access the X position in code through : transform.position.x
- To set the position, you have to assign a full Vector3 to it
  It is not possible to edit the x, y and z numbers individually, as they are read-only

Well… that's a problem

```
transform.position = new Vector3(-10.2f, -9.3f, 2);
```

This line of code will force the position of the gameObject to (-10.2, -9.3, 2)

```
if (Input.GetKey(KeyCode.RightArrow))
{
    myRigidbody.transform.Translate(8.5f * Time.deltaTime, 0, 0);
    if ( /*position x is above the limit*/ )
    {
        //Force the position towards a new vector
        //with the current y and z, but x equal to the limit
    }
}
```

Trouble figuring it out ? This image should help you a little bit !

Remember that you have to do this for both directions of movement !

Since you can use this « cheat », basic objectives should be easy !

- You can even recycle scripts or bits of code from TP1 if you want to
- Once you're done, continue to the next slide

Phase 2/3 : the « 4 problems »

Our game is playable, but 4 big problems are easy to identify :

1) When the ball falls, we have to reset the game to replay. That's annoying.

2) Players can not influence the ball trajectory with his paddle. They have no agency.

3) The Bricks always have the same position in the level, so there is no variety

4) When we have destroyed all the bricks, there is nothing left to do !

Solving as many of those problems as possible is YOUR task !

- You should know everything you need to solve those problems. Try things and experiment to figure it out !
- The next slides will contain advices for each of those problems, but will NOT give you any direct solution !
- You can ask questions, so there's no need to panic ;)

Advice for Problem n°1

Problem : « When the ball falls, we have to reset the game to replay. That's annoying. »

Solution : « When the ball falls, we teleport it back to its starting position »

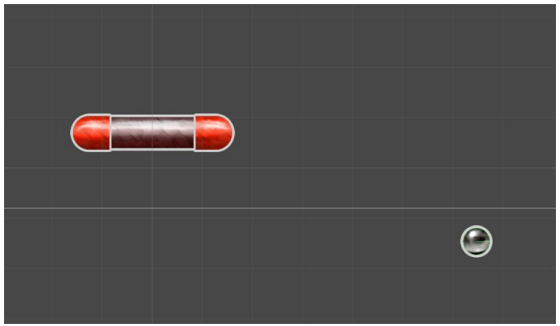
- Test the « y position » to see if the ball has fallen at the bottom
  Doing it on the Ball GameObject is easiest. You have to test each frame.

- <u>Desired 3-step behavior to use once a « fall » is detected :</u>

  1) Remove 500 points from the player. Prevent score from going below 0

  2) Teleport the ball to its initial position and set its velocity to (0,0)

  3) Wait 2 seconds and then launch it the same way as the start of the game



You can move the ball below the limit of the camera, temporarily, to read the value to use as threshold for the y position

<u>Many of you had done something similar in TP1. But now I want this « wait 2 seconds » feature !</u>
To implement it, you will need a timer, and probably some other variables to keep track of the game status...

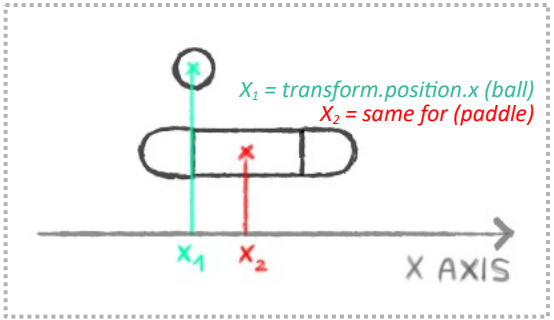
Advice for Problem n°2

Problem : « Players can not influence the ball trajectory with his paddle. They have no agency. »

Solution : « We alter the ball velocity when it collides the paddle, depending on relative position »

- By default, the position of a GameObject matches the center of its image
  As the picture to the side shows, this will be very useful to us if we do the proper math !
- In the collision method, you can access the position of the collided object, so doing this math will be possible
- Alter the velocity of the ball according to the difference in x coordinates
- RigidBody2D.velocity is a Vector2. So we can add another Vector2 to it :

```
ref_rigidBody.velocity += new Vector2(diffX * 3, 0);
```



(X_2 – X_1) gives us the ball's position relative to the center of the paddle.

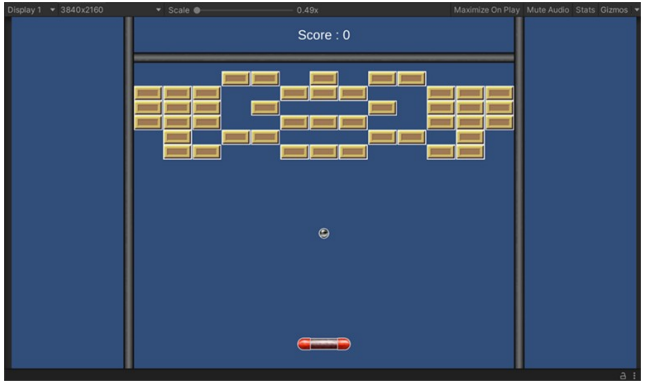
Make this feel nice to your player by tweaking the math with a multiplication!
Multiply the « X difference » by a constant number to affect the strength of the effect. *(I used 3)*

Advice for Problem n°3

Problem : « The Bricks always have the same position in the level, there is no variety »

Solution : « Create an invisible GameObject with a Script that will generate bricks »

- Create an empty GameObject to act as the « Game Master »
- Add a script to this object. Ensures it has access to the brick prefab
- Create your own algorithm to place bricks randomly within the playing field
- Place this algorithm in a custom method, which « start » will call
- Careful : the algorithm should not create bricks that overlap or that are outside the game area



A level generated by my algorithm.
which is only one algorithm amongst a large number of valid options !

<u>Several completely different algorithms are perfectly valid !</u>
You can use a grid-based logic, place bricks randomly on various lines… Choose whatever inspires you !

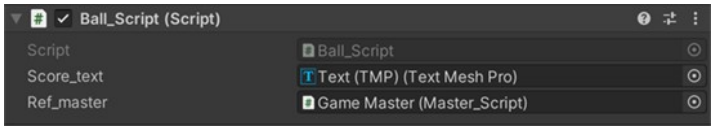
Advice for Problem n°4

Problem : « When we have destroyed all the bricks, there is nothing left to do ! »

Solution : « Keep track of the number of bricks. When all are gone, start a new level »

- « Game Master » creates the bricks, so it can know how many it created!
- Create a custom public method on the master, called « ReportBrickDeath() »
- When the ball destroys a brick, the ball's script should call this method. For that, it will need a reference to the GameMaster script.
- You can provide that reference through a clever drag-and-drop in the editor
- « ReportBrickDeath() » should decrease the number of bricks by 1, and if the number is now 0, request the creation of a new level

```
Script Unity | 0 références
public class Ball_Script : MonoBehaviour
{
    public TextMeshPro score_text;
    public Master_Script ref_master;
```



This one is pretty tough. So don't hesitate to ask questions !
This is the first time you have to use the « public attribute initialized though drag and drop » technique with a component you created yourself. Try to figure out yourself, but don't feel ashamed if you don't either !

Phase 3/3 : Sound Effects

- Get the Sound effects and Musics in the public folder
- Using what you learned in class, you should be able to implement them !

Now, it's your turn :

Implement all the following sounds so they play at the correct moment :

- « Wall Bump » should play when the ball collides a wall
- « Action Bump » should play when the ball collides the paddle or a brick
- « Intro Jingle » should be played before the ball is launched
- « Death Jingle » sould be played when the ball falls below the paddle

- You will have to alter the ball respawn timing if you want those last 2 sounds to « feel good » to the user

- You are free to add as many AudioSources as you want, either through code or GUI

All Main Objectives Complete !

If you reached this point, You completed the Main Objectives !

I split them into 3 phases, but now, you've done it ! You completed Practicum n°2 !

There are, of course, a few extra objectives you can do.

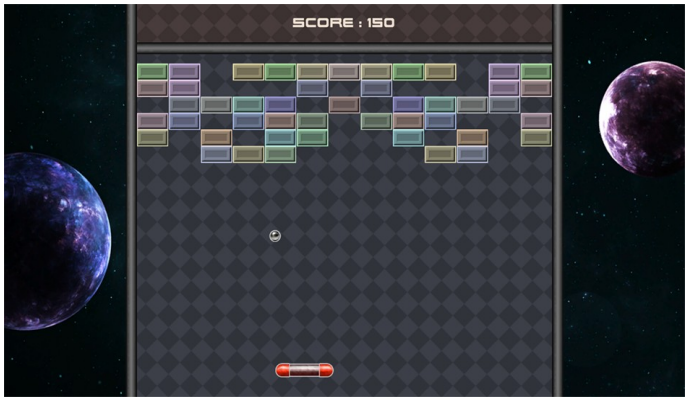
(As usual, not doing those will not count as a penalty)

Extra Objectives for this Project

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

They are not mandatory. But you should have the knowledge to do them

- Add background and decorations to your game
- Change the font of the score text to the (provided) sci-fi font
- Create a falling coin, which the paddle can collect for 100 points
- When a brick is destroyed, it has 10 % chance to spawn a coin
- Coin that fall down the screen must be properly deleted
- Hard : Randomize the color used by SpriteRenderer of each brick



What the game might look like after this extra polish !

We are going to re-use this game as part of the final project, but I still want you to send me today's work within 72 hours!

Next time… the Final Practicum !

A 3 games-in-One App !

With the power of copy paste and menus !

- We are going to make a clone of Flappy Bird… and then !
- We are going to create a big « 3 games in 1 » app, in which the user can choose to play Flappy Bird, Brick breaker, or Apple Catcher
- The Big App will start with a menu to choose a game
- During gameplay, in any of the 3 games, the user can press « escape » to return to the menu. He will also return there after a game over
- By using copy-pasting of Assets, transferring out previous games to the new big app will be easier than you might think
- The grading will put more emphasis on the new elements *(the Flappy Bird clone, the menu)* rather than on the old ones
- This is ambitious. But don't worry : you will not start from zero!

See you Next time!