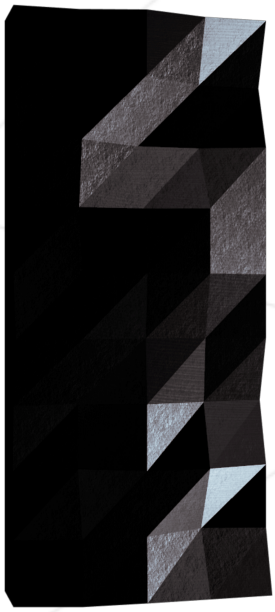

Piscine Unity - Day 00

Assets, GameObject, Behaviour, Input, Transform

Staff staff@staff.42.fr

Summary: This document contains the subject for Day 00 for the „Piscine Unity" from 42

Contents

I	Piscine Unity Starter Kit	2
II	General Instructions	3
III	Specific instructions of the day	5
IV	Exercise 00 : Balloon Simulator 2015	6
V	Exercise 01 : Quick Time Event	7
VI	Exercise 02 : Mini Golf	9
VII	Exercise 03 : Flappy Bird	10
VIII	Exercise 04 : Pong !	11

Chapter I

Piscine Unity Starter Kit

Welcome to this first day of Piscine Unity. As a good start find below a small list of advise and other link that will prove very useful during the whole piscine.

- The official Unity documentation, you will find a blue book close to every component in the inspector, that will direct you to its documentation.
- The official C# documentation
- The subject is authoritative, don't always trust demos which can contain additional non required content or features.
- If you create a generic and reusable script for everything utilitarian/not directly linked to the gameplay of the day you will save a lot of time in the following days.
- Once the day in finished, don't keep your project on your profile, Unity has a tendancy to create a billion files and your user space (as well as login time) will drastically increase.
- Days are long, don't lose too much time on details during your exercises. It will always be possible to improve your game once the day completed.
- If your MonoDevelop doens't launch, check the mega forum thread for the Piscine Unity where the whole process will be explained.
- If the peer on your right and the peer on your left don't have any clue about a technical `ISSUE` you're allowed to report it on the forum with the Piscine Unity tag or on slack.
- README are suppoosed to be read as well as the forewords...

Chapter II

General Instructions

- The Unity bootcamp has to be made entirely, exclusively and mandatorily in `C#`. No `Javascript/Unityscript`, `Boo` or any other horrors.

- The use of functions or namespace not explicitely authorised in the exercise header or ini the rules of the day will be considered cheating.

- For a optimal usage of Unity, you have to work on `~/goinfre`, which is on the local drive of your computer. Remember to make appropriate backup on your own, the local goinfre can be purged.

- Unlike any other bootcamps, each day doesn't require a folder `ex00/`, `ex01/`, ..., `exn/`. Instead you'll have to submit your project folder which will be name like the day: `d00/`, `d01/`, ... However, a project folder, by default, contains a useless folder: the `"projet/Temp/"` sub folder. Make sure to **NEVER** try to push this folder on your repository.

- In case you're wondering about it, there is no imposed norme at `42` for `C#` during this bootcamp. You can use whatever style you like without restrictio. But remember that code that can't be read or understood during peer-evaluation is code that can't be graded.

- You must sort your project's assets in appropriate folders. For every folder correspond one and only one type of asset. For exemple: `"Scripts/"`, `"Scenes/"`, `"Sprites/"`, `"Prefabs/"`, `"Sounds/"`, `"Models/"`, ...

- Make sure to test carefully prototypes provided every day. They'll help you a lot in the understanding of the subject as well as what's requested of you.

- The use of the Unity Asset Store is forbidden. You are encouraged to use the daily provided assets (when necessary) or to look for additional ones on the Internet if you don't like them, exception made of scripts obviously because you have to create everything you submit (excluding scripts provided by the staff). The Asset Store is forbidden because everything you'll do is available there in one form or another.

However the use of Unity Standard Assets is authorised and event advised for some exercises.

- From d03 for peer-evaluation you'll be required to build the games to test them. `The corrector` will have to build the game, you must therefore always push projects/sources. You project must always be properly configured for the build. `No` last minute tweaks will be tolerated.

- Warning: You'll not be corrected by a program, except if stipulated in the subject. This imply a certain degree of liberty in the way you can do exercises. However keep in mind the instructions of each exercise, don't be LAZY, you would miss a lot of very interesting things.

- It isn't a problem to had additional or useless files in your repository. You can choose to separate your code in different files instead of one, except if the exercise's header stipulate a list of files to submit. One file must define one and only one behaviour, so no namespace. Those instructions don't apply to the `"projet/Temp/"` sub-folder which isn't allowed to exist in your repositories.

- Read carefully the whole subject before beginning, really, do it.

- This document could potentially change up to 4 hour before submission.

- Even if the subject of an exercise is short, it's better to take a little bit of time to understand what's requested to do what's best.

- Sometimes you'll be asked to give specific attention on the artistic side of your project. In this case, it'll be mentioned explicitely in the subject. Don't hesitate to try a lot of different things to get a good idea of the possibilities offered by Unity.

- By Odin, by Thor ! Use your brain !!!

Chapter III

Specific instructions of the day

- The use of the Physics Unity API is strickly forbidden.
- Today, each exercise being completely different, you will require specific “ex0x” turn-in directories.

Chapter IV

Exercise 00 : Balloon Simulator 2015

	Exercise 00
Exercise 00 : Balloon Simulator 2015	
Turn-in directory : *ex*00/	
Files to turn in : `A scene file ex00, Balloon.cs, assets required for the exercise`	
Allowed functions : `Debug.Log, Mathf.RoundToInt, Input.GetKeyDown, GameObject.Destroy`	
Remarks : `n/a`	

Even though the title suggests it, let's be precise. You'll have to create a scene with a balloon. It must be possible to blow the balloon using the space key. If the balloon is too big it explodes or if it's not inflated enough the game is lost. The ballon must visibly grow based on its air level. If the space key isn't pressed anymore the balloon must deflate.

You can run the demo attached on the project page to get an idea.

You also have to consider breath. The faster we inflate the more out of breath you'll be. Reaching a certain level, it'll not be possible to blow the balloon anymore and you will have to wait for your breath to regenarate.

When the game ends, you have to display the time rounded up to the unit in the console like this:

```
Balloon life time: 120s
```

Chapter V

Exercise 01 : Quick Time Event

	Exercise 01
Exercise 01 : Quick Time Event	
Turn-in directory : *ex*01/	
Files to turn in : `A scene file ex01, CubeSpawner.cs, Cube.cs, assets required for the exercise`	
Allowed functions : `Debug.Log, Random.Range, GameObject.Instantiate, GameObject.Destroy, Input.GetKeyDown, Transform.Translate`	
Remarks : `n/a`	

You probably played Guitar Hero once, or any other game that required you to press keys at the right time. For newbies, this is what is called QTE or Quick Time Event.

You can run the demo attached on the project page to get an idea. You must use `A`, `S` and `D` to play the game.

The aim of this exercise as your probably understood is to create a QTE. Put a bar at the bottom of the screen as well as one at the top and 3 vertical lines from the top one to the bottom. Make sure that squares are randomly generated on these three vertical lines every X seconds. These squares must correspond to a key (you will find examples in the zip file attached on the project page). Every square must fall at a random speed. Once on the line at the bottom, the player must press the key at the right time. Every square must have its own line.

You must display the player's precision after each key press. Precision correspond to the distance between the square and the line when the player presses the key. It must be displayed as follow:

```
Precision: 23.454
```

Squares must have a lifetime and can't stay instantiated once out of the game field.

Chapter VI

Exercise 02 : Mini Golf

	Exercise 02
Exercise 02 : Mini Golf	
Turn-in directory : $ex02/$	
Files to turn in : `A scene file ex02, Ball.cs, Club.cs, assets required for the exercise`	
Allowed functions : `Debug.Log, Mathf.Clamp, Transform.Translate, Input.GetKey`	
Remarks : `n/a`	

There is nothing better than sport. Therefore you will create a Mini Golf. You will have to ensure that the player can shot a ball with a force defined by the space bar. The longer the press, the further the ball will go. The ball must have inertia and slow down with time. If the ball touches a wall it rebounds in the opposite direftion. If the ball goes too fast over the hole, it doesn't fall in.

The simplest is to test it! Run the demo and enjoy it a little bit.

For the scoring, we will use a simplified version or mini golf rules. The player starts with -15 points each failed attemps he wins 5 points. If the score is over 0 it's a loss, but the game doesn't stop.

At each shot the score must be displayed in the console like this:

```
Score: -5
```

Chapter VII

Exercise 03 : Flappy Bird

	Exercise 03
Exercise 03 : Flappy Bird	
Turn-in directory : *ex*03/	
Files to turn in : `A scene file ex03, Bird.cs, Pipe.cs, assets required for the exercise`	
Allowed functions : `Debug.Log, Input.GetKeyDown, Transform.Rotate, Transform.Translate, Mathf.RoundToInt`	
Remarks : `n/a`	

For this exercise, let's do a simplified version of Flappy Bird. Put the bird at the center of the scene. It must fall until the player presses the space bar to make it go up again. Run the demo attached on the project page to get an idea.

Let's quickly go over some technical considerations. The bird cannot move on the x axis. Pipes have to be at the same height because we didn't yet see how to do this properly. There cannot be more than 2 instanciated pipes, you must therefore reuse them. The speed must increase as the player goes through more and more pipes..

You must put a scoring and time system into place as well. Every pipe passed rewards 5 points. At the end of each game you must display the score in the console as follow:

```
Score: 15
Time: 25s
```

Chapter VIII

Exercise 04 : Pong !

	Exercise 04
Exercise 04 : Pong !	
Turn-in directory : *ex*04/	
Files to turn in : `A scene file ex04, PongBall.cs, Player.cs, assets required for the exercise`	
Allowed functions : `Debug.Log, Random.Range, Input.GetKey, Transform.Translate`	
Remarks : `n/a`	

To finish a concept that proved its worth and is relatively universal. You will have to create a Pong. If you don't know what it is, run the demo. The player on the left moves its racket using `W` and `S`, and the player on the right using `Up Arrow` and `Down Arrow`.

Ensure that 2 rectangles are on each side of the game. A square must rebound between the 2, its direction is reversed after each rebound including vertically.

Both rectangles must be controlled separately from another vertically using the keys of the keyboard, with as a limit the walls on the top and bottom. If the square goes over a racket, it comes back at the center and the opposing player wins a point.

The score must be displayed after each point in the console as follow:

```
Player 1: 0 | Player 2: 0
```