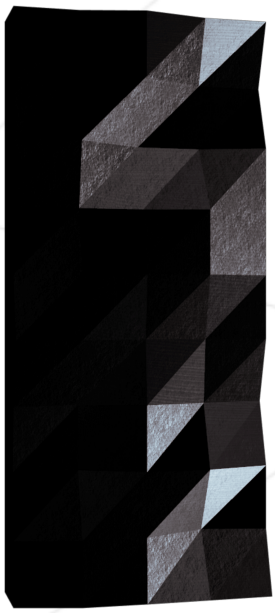

Unity Piscine - D00

Assets, GameObject, Behavior, Input, Transform

Staff staff@staff.42.fr

Summary: Here is the subject of Day 00 for the Unity piscine of 42.

Contents

I	Piscine Unity Starter Kit	2
II	Instructions	3
III	Today's specific instructions	4
IV	Exercise 00: Balloon blowing Simulator 2015	5
V	exercicese 01: Quick Time Event	6
VI	Exercise 02: Mini Golf	8
VII	Exercise 03: Flappy Bird	9
VIII	Exercise 04: Pong!	10

Chapter I

Piscine Unity Starter Kit

Welcome on this first day of the Unity Piscine. To start on the right foot, here is a list of useful advice and links that will follow you during the whole length of the Piscine.

- Official Unity documentation. You will also find a blue book in the inspector near each component that will take you to the specific documentation.
- Official C# documentation
- The subject prevails. Don't always trust the demos thant can contain additional elements that are not required by the subject.
- If you create generic scripts that are reusable for utilitarian functions/ that have nothing to do with the gameplay of the day, it will save you time for the following days.
- Once the day is done, don't keep your projects on your home. Unity has the bad habit of creating billions of files that will quickly kill your precious 5 gigas (as well as your connection time).
- Days are long so don't waste time on details during work. You will still have time to enhance your game once the mandatory parts are through and the day over.
- If you can't run your MonoDevelop, go to the Unity Piscine's mega-thread where you will find a tutorial.
- If any of your neighbor has the answer to a `TECHNICAL` problem, you can report it on the forum with the Unity Piscine tag.
- README files are meant to be read. So are the preambles...

Chapter II

Instructions

- Only this page will serve as reference. Do not trust rumors.
- The exercises have been ordered from easiest to most difficult. Under any circumstance you can submit or take into account an exercise if a previous one has failed.
- Be careful with the access rights of your files.
- You should follow the submit procedure for all you exercises.
- Your exercises will be corrected by your piscine peers.
- You cannot leave any extra file on your repository except the ones explicitly specify on you subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Everything you need can be found on the `man` or out there on Google.
- Read carefully the exercises: they may contain some features you should implement that are explicitly mentioned on the subject.
- Think about discussing on the forum Piscine of your Intra!
- Use your brain!!!

Chapter III

Today's specific instructions

- The use of Unity's physical API is strictly prohibited.

Chapter IV

Exercise 00: Balloon blowing Simulator 2015

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

Though the title is explicit, we're gonna make things clear. You must set up a scene featuring a balloon. You must be able to blow up the balloon with the use of the space bar. If you over blow it, it explodes and if it's not blown up enough after a given time, the game is lost. The balloon must visually inflate depending on its air level. When you stop pushing the space bar, the balloon deflates.

Run the demo provided with the subject to get the idea.

You must also take the blow into account. The faster you blow, the faster you get breathless. Past a breathlessness level, you cannot inflate the balloon anymore, so you'll have to catch your breath before getting back to it.

When the game ends, the time rounded up as a whole must be displayed in the console as follows:

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

Chapter V

exercicese 01: Quick Time Event

	Exercise 01
Exercise 01: Quick Time Event	
Turn-in directory : $ex01/$	
Files to turn in : `An ex01 scene file, CubeSpawner.cs, Cube.cs, and the assets specific to the exercise`	
Allowed functions : `Debug.Log, Random.Range, GameObject.Instantiate, GameObject.Destroy, Input.GetKeyDown, Transform.Translate`	

Maybe you've played Guitar Hero or any other game that demands you press a specific key at a specific moment once in your life. For the common player, it's called QTE or Quick Time Event.

Run the demo to get the idea. You must use the `A`, `S` and `D` to play this game.

The goal of this exercise is to make a QTE, as you might have guessed. Create an horizontal line at the bottom of the screen and one at the top, as well as 3 vertical lines joining both. Make sure squares are randomly generated on one of the three lines every X second. These squares must match a key (example keys are provided in today's zip). Each square must run down at a random speed towards the bottom line. When the square hits this line, the player must press the matching key at the right moment. Each square must have its own line.

You must display the player's accuracy for each hit. The accuracy will match the distance between the square and the line when the player pressed the key. It must be displayed this way:

```
Precision: 23.454
```

Cubes must have a lifespan and must not remain instantiated in the scene for ever once they’re out of the game zone.

Chapter VI

Exercise 02: Mini Golf

	Exercise 02
Exercise 02: Mini Golf	
Turn-in directory : *ex*02/	
Files to turn in : `A ex02 scene file, Ball.cs, Club.cs, and the assets specific to the exercise`	
Allowed functions : `Debug.Log, Mathf.Clamp, Transform.Translate, Input.GetKey`	

A little sports never hurt no one. You're going to create a mini golf. You will have to be able to hit a ball with a power defined by the space bar. The longer the pressing, the further the ball will go. The ball must have an inertia and slow down in time. The goal, of course, is to put the ball in a hole. If it touches a wall, it will bounce in the opposite direction. If the ball is too fast when it rolls over the hole, it won't fall into it.

Test it, you'll understand! Run the demo and have fun for a moment.

For the score, you will use a simplified version of the mini golf rules. The player starts with -15 points, each missed hit grants them 5 points. If their score goes higher than 0, they lose. However, the game doesn't stop.

After each hit, the score will have to be displayed in the console as follows:

```
Score: -5
```

Chapter VII

Exercise 03: Flappy Bird

	Exercise 03
Exercise 03: Flappy Bird	
Turn-in directory : $ex03/$	
Files to turn in : `An ex03 scene file, Bird.cs, Pipe.cs, the assets specific to the exercise`	
Allowed functions : `Debug.Log, Input.GetKeyDown, Transform.Rotate, Transform.Translate, Mathf.RoundToInt`	

For this exercise, you will create a simplified version of Flappy Bird. Set the bird in the middle of the scene. It must fall until the player presses the space bar to make it go up again. If the bird touches the ground or a pipe, the game is lost. Run the demo if you've never played the cult hit.

Let's get technical now. The bird must never move on the x axis. Pipes will always be set at the same height because you have not yet learned how to make it properly. They can only be two pipe instances. You have to reuse them. Speed must increase progressively after each pipe passed.

You must also set a score and time system. Each pipe passed grants 5 points to the player. Each time the game ends, the score must be displayed as follows:

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

Chapter VIII

Exercise 04: Pong!

	Exercise 04
Exercise 04: Pong!	
Turn-in directory : $ex04/$	
Files to turn in : `An ex04 scene file, PongBall.cs, Player.cs, the assets specific to the exercise`	
Allowed functions : `Debug.Log, Random.Range, Input.GetKey, Transform.Translate`	

And now, you're in for a timeless classic. You're going to design a... Pong! If you don't know what a Pong is, run the demo. The player on the left moves the racket with the `W` and `S` keys. The player on the right, moves the racket with the `Up Arrow` and `Down Arrow`.

Make sure you create two rectangles on both sides of the field. A square must bounce between them, on the opposite direction after each bounce. That includes vertical rebounds.

Each rectangle must be controlled independently and move vertically thanks to the keyboard keys. They must be stopped by top and bottom walls. If the square touches one of the edges on the right or left of the field, it is set back in the middle of the field and the opposite player scores a point.

The score must be displayed and modified after each point scored in the Unity console as follows:

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