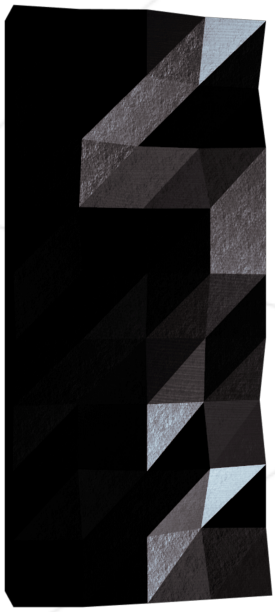

Unity Piscine - D04

PlayerPrefs and Coroutines

Staff staff@staff.42.fr

Summary: Here is the D04's subject for the Unity Piscine of 42.

Contents

Chapter I

Consignes

Sauf contradiction explicite, les consignes suivantes seront valables pour tous les jours de cette Piscine.

- Seul ce sujet sert de référence : ne vous fiez pas aux bruits de couloir.
- Le sujet peut changer jusqu'à une heure avant le rendu.
- Les exercices sont très précisément ordonnés du plus simple au plus complexe. En aucun cas nous ne porterons attention ni ne prendrons en compte un exercice complexe si un exercice plus simple n'est pas parfaitement réussi.
- Attention aux droits de vos fichiers et de vos répertoires.
- Vous devez suivre la procédure de rendu pour tous vos exercices. L'url de votre dépot `GIT` pour cette journée est disponible sur votre intranet.
- Vos exercices seront évalués par vos camarades de Piscine.
- En plus de vos camarades, vous pouvez être évalués par un programme appelé la Moulinette. La Moulinette est très stricte dans sa notation car elle est totalement automatisée. Il est donc impossible de discuter de sa note avec elle. Soyez d'une rigueur irréprochable pour éviter les mauvaises surprises.
- Les exercices shell doivent s'éxcuter avec `/bin/sh`.
- Vous ne devez laisser aucun autre fichier que ceux explicitement specifiés par les énoncés des exercices dans votre dépot de rendu.
- Vous avez une question ? Demandez à votre voisin de droite. Sinon, essayez avec votre voisin de gauche.
- Toutes les réponses à vos questions techniques se trouvent dans les `man` ou sur Internet.
- Pensez à discuter sur le forum Piscine de votre Intra et sur Slack !
- Lisez attentivement les exemples car ils peuvent vous permettre d'identifier un travail à réaliser qui n'est pas précisé dans le sujet à première vue.
- Réfléchissez. Par pitié, par Thor, par Odin !

Chapter II

Foreword

- Sonic 3 + Sonic and Knuckles (1994) is one of the first video games in history that has was sold as a kit. Indeed, in a time when DLC did not yet exist, SEGA released both games with a 6 months gap, both full priced. The Sonic and Knuckles cartridge was unique because it featured a cartridge port. You could stick it in the Sega Genesis and stick the Sonic 3 cartridge over it to get the full game, its 12 worlds and 2 sets of chaos emeralds and super emeralds.
- Michael Jackson did not compose the Sonic 3 soundtrack. This is an urban legend that was debunked years ago. He is indeed not credited in the game's credits because the sound chip quality did not fit his demands. Which is rather ironic since his Genesis game (Moonwalker, 1990) was released a couple of years before with one of his hits featuring in the soundtrack. You can also hear some samples from his voice in some of Sonic 3 tracks. Especially the end title.
- Sonic 3 + Sonic & Knuckles, once again, was one of the first games that featured a saving system. The player didn't need to run the whole game to finish it. They could start from any unlocked areas. This was a revolution, then. However, Zelda, released 7 years prior on NES already featured an implemented saving system.
- Now, about Sonic, the first. The Sega sound used to take 1/8th of the total cartridge's memory. At the time, cartridges' memory was ridiculously small. 2 seconds of sound used more memory than a whole game level. We're talking about a sound that was so compressed it was barely audible.
- The game's lore tells us Sonic's speed is faster than light. Sonic Generations famously made him run than light (in meters/second ingame) while letting the game remain playable.
- Sonic has the same hairdo as Sangoku. Both see their hair turn yellow and glowing when the get mad. Officially, Goky became a Super Sayan just 4 days before the release of Sonic 1 on Genesis... so who came first? The chicken or the Edge(hog)?

Chapter III

Exercise 00: Data select!

	Exercise : 00
Exercise 00: Data select!	
Turn-in directory : *ex*00/	
Files to turn in : `The "TitleScreen" scene, the "DataSelect" scene and anything relevant`	
Forbidden functions : `None`	

`Today's program is LOADED, and you will have many levels to create, so keep up the pace! Don't waste your time on details and remember you will be graded on what's required in the subject. You will have time to polish your levels/menus/etc later is you want to.`

`You can modify all the assets provided, including the scripts (especially Sonic's one is you want to add something to it or upgrade it) as you see fit. Make sure you don't break anything, though, since many elements are linked and one modification might create another one you may not be aware of.`

You must create a user profile that will be saved in the players prefs so it can be reloaded if you quit the game and relaunch it. This profile must contain the list of each level unlocked by the player, the number of lives they lost since the game first launched, the total number of rings they earned, and the score for their best run on each level.

You must also create a DataSelect scene that helps visualize all these infos as well as the locked and unlocked playable levels thanks to a GUI. Test the demo to check the example. Remember you can create the interface you like as long as it features all the necessary infos.

You will then create a title screen (a title screen is always welcome!) with a clickable button that will allow you to reset the player's profile. You can also get to the DataSelect screen pressing the return key to be able to choose a level and launch a game.

You will use the playerprefs only for educational purpose. You should NEVER use it to store infos the player is not supposed to modify (their progress for instance). The playerprefs are stored in a file, hence they're editable. They will be mostly used to keep the player settings regarding the game options in memory: the keybindings, the audio and video prefs, etc...

Chapter IV

Exercise 01: A basic level

	Exercise : 01
Exercise 01: A basic level	
Turn-in directory : $ex01/$	
Files to turn in : `A scene named after an area of your choice and anything relevant`	
Forbidden functions : `None`	

`In today's assets, you will find 4 sets of tiles/prefabs already already preconfigured to help you create your levels. If you have not done so already, go read the ReadMe for more detailed explanations on how they work and how you can modify them freely.`

Sonic's principle is ready: a character starts the level it must end as fast as possible gathering as many rings as possible. The level's end is always marked by a goal plate spinning when Sonic runs by. Generally, levels are supposed to be fast races. The player can choose among different paths (unlike its historical rival, Mario, who's slower and features more linear levels). A good level design is a mix of curves, bonus and wisely placed enemies so the course is fluid. Add several traps so that the course is a little dangerous. When Sonic is hit, he loses his rings, that are scattered around him. He can try to catch them before they vanish. If he's hit when he doesn't own any ring, he loses one life and starts the level again at the previous checkpoint.

This being stated, you will create a simple level (no trap, no hole, no enemy) with interesting various paths. The goal will be grab as many rings as possible and complete the level as fast as possible. Once you're through with your level design, you will have to:

- Implement a time counter in the GUI. It will count minutes and seconds from the level start to the end as follows: 0:00. There will be NO float in the minutes.
- Create rings. You will find a prefab animated ring you will have to modify so that

Sonic can grab them. You will also have to store thes rings somewhere. You will choose to modify an existing script or create a new one. Several solutions exist. Choose the one you see fit.

- Use the provided sound when Sonic grabs a ring. This is a trademark sound that will let anyone hearing it know you're playing Sonic.
- Implement a ring counter that shows how many rings Sonic has gathered in real time.
- Implement the level's music (to each level its own). Choose the tracks in the assets or go find one on the web. There are hundreds of remixes. The only constraint will be that the musical theme matches the area played (or another cool one, but it's got to be Sonic!).
- Implement the spinning goal plate at the end of the level as well as the matching music that will be triggered when Sonic runs by the board.
- Calculate a final score at the end of the level. It must appear in the middle of the screen exactly 6 seconds after the musics starts so it's synchronized to it. It might sound strange at first but it's the kind of tiny detail that all historical video games have in common.

Calculate the score as you like but it must take into account the number of killed enemies, rings at arrival and the time the player took to complete the level. Here is a scale example: 500pts for each enemy, 100pts for each ring, and 20000pts - 100pts for each second passed after the level started (with a minimum of 0pts past 200s).

Chapter V

Exercise 02: The end of the joyride

	Exercise : 02
Exercise 02: The end of the joyride	
Turn-in directory : *ex*02/	
Files to turn in : `A scene named after an area of your choice, different from the previous one and anything relevant.`	
Forbidden functions : `None`	

That's quite a bit you've achieved! Let's now add a some traps to make the player's ride a little nastier.

Add the spikes you'll find in today's sprites. Create your own prefabs with the working colliders and scripts. You will also create holes that instantly take a life if the player falls into it. It's already managed by the script that launches a dead() when Sonic's .y position is too low. You just have to set the holes at will and watch the magic do the work!

`For the holes, Dead() is launched when the .y position goes below -15. Keep that in mind when designing your level and try to avoid the unwilling Trigger Dead().`

We have provided a fancy Sonic sprite in the assets so you don't have to waste precious time - let's say, a day - recoding Sonic's physics and animations. Unfortunately, a part of the script has been corrupted and a method has vanished. Your mission will be to find the method named getHit() in the Sonic.cs script and to rewrite its content.

`From now on, all the variables and methods I will mention are already implemented int the Sonic.cs. You won't have to create them. Just call/ attribute them, except of explicitly stated otherwise.`

You will have to:

- Make sure Sonic is not invincible testing the isInvincible bool.
- Call the Dead() method if Sonic doesn't carry any ring. Straight, simple. If he does carry rings, you will have to:
- Stop the rigidbody velocity attached to Sonic (you can access it with the rbody variable).
- Apply an impulse on rbody to throw Sonic in the air in the direction opposite to the one he was going when hit.
- Pass the isHit variable to true. That will stop the player from controlling Sonic when he's bumped back.
- Invoke the stopHit method (already in the script) with a 2 seconds delay so that the player can retrieve Sonic's commands.
- Pass the animator's getHit bool to true to launch the matching animation (if the sentence doesn't make any sense to you, watch the Dead() method a little further in the script to find an example).
- Create and launch a coroutine that makes Sonic invincible for 5 seconds. This coroutine must make the sprite blink, pass isInvincible to true when it is called and pass it back to false after 5 seconds.
- Play the sound that illustrates the ring loss stored in the aLoseRings variable.
- Set the number of rings Sonic owns to 0 and create a ring explosion around him. You must instantiate half the rings he was carrying (10 rings if he carried 20 for instance). Rings are scattered in the air and pass through Sonic. After 2 seconds, they start blinking and Sonic can recover them touching them like standard rings. They disappear after blinking for 4 seconds.

A little recap about the ring explosion to be perfectly clear: Sonic is hit, half the rings he was carrying are scattered around him. For 2 seconds, the rings pass through him, then, for 4 seconds, they blink and then disappear. Test the examples provided to see the expected behavior.

Chapter VI

Exercise 03: Fast as lightning

	Exercise : 03
Exercise 03: Fast as lightning	
Turn-in directory : $ex03/$	
Files to turn in : `A scene named after an area of your choice, different from the previous ones and anything relevant.`	
Forbidden functions : `None`	

Congratulations! If you read this, it means you have defeated the coroutines or you read the whole subject before you begin it. Either way, good for you! Go relax and check an interesting example of what can be done with Unity ou que vous lisez tout le sujet avant de commencer. Dans les deux cas bien joué ! Pour se détendre et voir un exemple intéressant des capacités d'Unity here is a Sonic remake made years ago (on Unity 3 at the time). Graphic design is not its prime quality but the technical skill is dead on. The game was made in a couple of months by two people.

Ok, back to work! Let's get back to OUR Sonic. You will now implement to other key elements of the series: bumpers and tv's.

Let's start with the bumpers. To avoid recoding 50 times the same things, and reading endless lists of triggers you will have to set to ON, I've made a function that will help you manage the bump in the Sonic.cs script:

```
public void bumper(float boostX, float boostY);
```

You just have to call it, setting the boost in X and Y in which Sonic will fly/jump/roll. Your job will be to create the bumper, because no prefab will be provided this time. You will just have cut-out sprite sheets. For each of the 8 bumper directions, you will have to:

- Create a prefab with a collider and a trigger: the collider will allow the character to stop or walk by the bumper side (and give it an independent physical collision) and the trigger will be used to trig the real bump. The goal is to isolate the bouncing surface to stop it from triggering the bump when the sprite is touched somewhere else.
- Create two bumper states. In normal mode, it is folded and when it is triggered, its position is unfold (watch the sprite sheet if you don't understand). After 0.3 second, it returns to its initial position.
- Play the "bumper.wav" sound provided in the assets when the bumper is active.

And now, the TV's. You will create the following tv's:

- Rings: Breaking this TV grants 10 rings.
- Power sneakers: Breaking this TV grants super speed. For 15 seconds, you will increase Sonic's max speed to 30 and the music pitch will be increased by 20and Sonic's speed returns to 20.
- Shield: Breaking this TV offers a shield to Sonic. You must instantiate the shield's animated prefab on Sonic. It possesses a currentShield variable that's been set to this use. You must also pass its isShielded variable to true. Don't forget to modify your getHit() to take the shield into account. It would be dumb to own a shield that doesn't protect you from the hits, right?

And now for the big question: how do we break a TV? The answer is simple:

- Each TV features a collider. If Sonic touches it with the isRolling or isJumpBall set to True, the tv is destroyed and the bonus is gained.
- TV's are animated with 2 images. One is displaying the bonus, the other is a glitched screen. Once again, you will find what you need in the sprite sheets.
- A broken TV must not disappear. You will change its sprite. There is a broken tv sprite in the SpriteSheets. A broken tv has no more collider, hence, you can run through it.
- When a tv is broken, you must call Sonic's destroy() method. This method makes Sonic bounce with a matching sound. You will also use it in the next exercise when destroying enemies.

Chapter VII

Exercise 04: Enemies!

	Exercise : 04
Exercise 04: Enemies!	
Turn-in directory : $ex04/$	
Files to turn in : `A scene named after an area of your choice, different from the previous ones and anything relevant.`	
Forbidden functions : `None`	

`Beware! I remind you you must create a new level for each exercise. This should be you 4th level.`

You'll be a little more free now, and if you made it to here, it's time to become a game designer.

You will find enemy sprites in all the provided spritesheets. You can proceed as you will as long as you create 3 enemies with different behaviors.

- A still enemy shooting projectiles.
- A slow enemy featuring 2 phases: whether it's moving slow, whether it stands still but is surrounded by spikes - hence, turning invincible (and dangerous).
- A fast moving enemy without any special skill.

To destroy an enemy, the conditions are exactly the same as the TV's. Sonic must jump/roll and when the enemy is destroyed, you must launch his Destroy() method (don't forget to add points to the score!).

Chapter VIII

Bonus exercise: Dr Robotnik

	Exercise : 05
Bonus exercise: Dr Robotnik	
Turn-in directory : $ex05/$	
Files to turn in : `A final scene and anything relevant`	
Forbidden functions : `None`	

If you've reached this point, congratulations! You have deserved to shine and create the final battle as you see fit.

This is an optional exercise. It will not grant you any point during the evaluation. But you will certainly have something to brag about.

Choose your favorite Robotnik in the assets. Choose an epic boss battle music and have fun! Create a memorable battle and dazzle your assessors!

Thor will pay you coffee on the terrace if you complete this exercise.