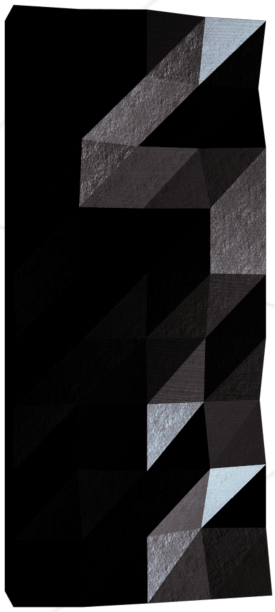

Piscine Unity - Day 04

PlayerPrefs ands Coroutines

Staff staff@staff.42.fr

Summary: This document contains the subject for Day 04 for the „Piscine Unity" from 42

Contents

Chapter I

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 II

Foreword

- Sonic 3 and Sonic & Knuckles (1994) is one of the first game sold as a kit in the video game history. At a time when DLC didn't exist yet, SEGA did something crazy by releasing both game 6 months from each other selling both at full price. The Sonic & Knuckles cartridge was a little special because it was equipped with a cartridge reader. It was therefore possible to put the game into the megadrive and put Sonic 3 on it to enjoy a full game with 12 level, 2 set of emaralds as well as super choas emaralds to be found. A whole program ...

- Michael Jackson composed songs for Sonic 3. It is an urban legend confirmed couple of years ago. He wasn't mentionned in the credits because the sound chip of the console wasn't good enough for him. It is rather funny when you know that he released his one megadrive game (Moonwalker, 1990) couple of years before including some of his successful songs as soundtrack. A shard ear might be able to glimse couple of samples of his voice in some of Sonic 3 songs, especially the ending.

- Sonic 3 + Sonic & Knucles, them again, are part of the first games implementing a backup system. No need to redo the whole game completely it was possible to start from any unlocked area, which was a small revolution for the franchise. But don't forget that Zelda on NES 7 years before already had an integrated backup system.

- Sonic 1 this time: The SEGA's intro sound takes 1/8th of the total cartridge memory. At that time the memory in the cartridge was ridiculy small and 2s of sound used more memory that every areas of the game put together. Worse is that the sound was compressed to the limit of audible range.

- According to the game's lore, Sonic runs so fast that he can go faster than light. Sonic Générations was able to make him reach the sound barrier (in meter per second ingame) while remaining playable.

- Sonic has the same haircut as Goku and when angry both hairs style goes golden with an energy aura around them. Officially Goku became a super saiyajin only 4 days before the Sonic 1's release, so who's the egg and who's the chicken?

Chapter III

Exercise 00 : Data select !

	Exercise 00
Exercise 00 : Data select !	
Files to turn in : `The "TitleScreen" scene, the "DataSelect" scene and everything that seems relevant`	
Forbidden functions : `None`	
Remarks : `n/a`	

There is a lot of things to do today and a lot of levels to create so don't slack off! Don't lose time on details and remember that youre graded on what's requested. You'll have all the time you want to polish your levels/menus/etc. later if you want to.

You are allowed to modify every provided asset as much as you want, including the scripts (especially Sonic's script if you want to add or upgrade something). Be careful though not to break anything, a lot of elements are linked together and a small update somewhere might have complicated repercussions somewhere else.

You must create a user profile which will be saved in the players prefs to be able to be loaded when exiting and rerunning the game. This profil must countain the list of unlocked level per player, the number of life lost every game included as well as the player's best score on every level.

You must create a DataSelect scene as well which will allow to visualise thanks to a GUI every information as well as the list of every playable, locked and unlocked levels. Don't hesitate to test the demo to have an idea about a possible set up, knowing that you are free to do any interface you want, as long as it has every information.

You will then have to create a title screen (It is always nice to have a title screen for a game!) with a clickable button which will allow the player to reinitialise his profile. It is also possible to go directly to the DataSelect screen directly by pressing Enter/Return and then choose a level to start a game.

We are playing with playerprefs in a pedagogical context here but you are usually NEVER supposed to store any information that the player isn't supposed to update ingame (progression for example). Playerprefs are stored in a file and therefore can be edited. It is usually used to save player's preference such as: keybindings, audio and video settings, etc.gfg

Chapter IV

Exercise 01 : A basic level

	Exercise 01
Exercise 01 : A basic level	
Files to turn in : `A scene named after the area of your choice and everything that seems relevant.`	
Forbidden functions : `None`	
Remarks : `n/a`	

`You will find in the provided files today 4 sets of tiles/prefabs preconfigured to create your levels. If you didn't already please check out the ReadMe for more detailed explanations on how they work and how to update them as you see fit.`

In Sonic the principe is straighfoward, the character starts at the beginning of the level and must go through it fast grabbing as much golden rings as possible. The end of a level is always marked by a rotating sign when Sonic pass it. The principle usually is that level are designed to be pass very quickly by choosing a path between several available (unlike his historical rival Mario whose levels are slower, more technical and often more linear). A good level design is a mix of cuvers, bonuses and well placed enemies for the path to be fluid with just a little bit of properly placed traps to avoid the game being a walk in the park. When Sonic is hit, he loses all his rings which scatter all around him. He can try to get some back before they disappear. If he is hit when he doesn't have any rings he then loses a life and has to restart from the beginning of the level or the last checkpoint.

Those basics set you have to create a first simple level (without traps, holes or ennemies) with different and interesting path to go through. The aim is to get as much rings as possible and to finish le level the fastest possible. Once your level design finished you have to:

- Integrate a time meter as GUI which will deplay secondes and minutes elapsed since the beginning of the level in the following form 0:00. No floats for the minutes, we

insist.

- Create the rings. You will already find an animated prefab of a ring which you will have to update to allow Sonic to grab them. You will also store those rings somewhere. It is up to you to decide if you rather modify an existing script or create a new one. Several solution are available, choose the one you think is more adapted.

- Use the provided soun each time Sonic grabs a ring. It's the branding of the game which make everybody in the room know that you are playing Sonic.

- Integrate a ring counter which indicates in real time how many rings Sonic has.

- Integrate the music of the level (each level has its own music) either by selecting in the provided musics, or by searching online for one of the hundreds of available remixes. The only constraint is that the music must be of the same zone of the level played (or the music of a really cool one, but it has to be from Sonic!).

- Integrate the rotating sign at the end of the level, as well as the corresponding music, which will be triggered by Sonic passing it.

- Calculate a score at the end of the level which must be displayed in the middle of the screen exactly 6 secondes after the beginning of the music to be synched to it. Said like that it seems strange but these small insignificant details are common to every game that did put their mark in the video game history.

You are free to calculate the score as you want but the equation must consider the number of ennemies killed, the number of rings at the end of the level as well as the time spent to go through the level. Here is a suggested scale: 500pts per ennemy killed, 100pts per ring at the end and 20000pts - 100pts per secondes spent since the beginning of the level (with a minimum of 0pts after 200s).

Chapter V

Exercise 02 : Let's stop running?

	Exercise 02
Exercise 02 : Let's stop running?	
Files to turn in : `A scene named after the area of your choice, different from the last exercise and everything that seems relevant.`	
Forbidden functions : `None`	
Remarks : `n/a`	

Here goes a big chunck of the work! Now let's add some traps to make the player's life a little more complicated.

You have to add spikes privided in the ressources today. It is up to you to create your own prefabs with every colliders and scripts required. You have to create holes as well which will instantly make the player lose a life. It is already automatically managed by the script which runs Dead() when Sonic's Y position is too low, you just have to add holes where you want and watch the magic happen.

`For the holes, Dead() is executed when Sonic's Y position goes under -15. Take it into consideration when creating your own maps to avoid triggering Dead() by mistake.`

We provided you with a beautiful Sonic in the ressources to avoid you spending a day recoding the physics and all its animations. Unfortunately part of its script is corrupted and a method disapeared. Your mission is to find that method called getHit() in the Sonic.cs script and rewrite it.

`Every variable and method we are referring from now own are already in the Sonic.cs script, you don't have to create them just to call/assign them, except if specifically mentionned.`

Vous devrez donc :

- Check that Sonic isn't invincible by testing the isInvincible boolean.
- Call the Dead() method if Sonic is hit and has no rings. Simple and efficient. If he has rings you will have to:
    - Stop the velocity of the rigidbody attached to Sonic (which you can access using the rbody variable).
    - Apply an impulse on rbody to send Sonic in the air in the opposite direction when hit.
    - Put the variable isHit to true. That's what will prevent the player to control the character while it's being bumped back.
    - Invoke the stopHit method (already in the script) with a 2s delay so that the player can get Sonic's control back.
    - Put the animator's getHit boolean to true to launch the corresponding animation (if this sentence seems confusing take a look at Dead() a little further in the script to find an example).
    - Create and run a coroutine which makes Sonic invincible for 5 secondes. This coroutine must make the sprite blink, put isInvincible to true when called and put it back to false after 5 secondes.
    - Play the ring losing sound already saved in the aLoseRings variable.
    - Reset to 0 the number of rings Sonic has and create an explosion of rings around him. You need to instanciate half of the rings he has (for example 10 rings if he had 20). Rings are ejected in the air in every direction going through Sonic. After 2 secondes they start to blink and Sonic can pick them up but touching them like standard rings. They have to disapear after blinking for 4 secondes.

Small recap for the ring explorion to be as clear as possible: Sonic gets hit, half of his rings are ejected around him. For 2 secondes go through him, then for 4 seconde they blink before disappearing. Don't hesitated to test the prototype to see how it works.

Chapter VI

Exercise 03 : As fast as light

	Exercise 03
Exercise 03 : As fast as light	
Files to turn in : `A scene named after the area of your choice, different from the last exercises and everything that seems relevant.`	
Forbidden functions : `None`	
Remarks : `n/a`	

Félicitations ! si vous lisez ceci c'est que vous avez vaincu les coroutines 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 voici une vidéo d'un remake de Sonic fait en HD il y a 5 ans (sur Unity 3 à l'époque). La qualité artistique n'est pas forcément de bon goût (c'est un peu Overkill...) mais le rendu technique est bluffant. Le jeu ayant été fait par seulement deux personnes en quelques mois.

Fin de la récré ! Revenons à NOTRE Sonic. Vous allez maintenant devoir implémenter deux autre éléments phares de la franchise : les bumpers et les télés.

Commençons par les bumpers. Pour vous éviter de devoir recoder 50 fois les mêmes choses et m'éviter de faire des listes interminables de triggers à mettre sur On, je vous ai déjà fait une fonction pour gérér le bump dans le script Sonic.cs :

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

Vous n'avez qu'à l'appeler en précisant le boost en X et en Y dans lesquels faire voler/sauter/rouler Sonic. Là où vous entrez en scène cette fois, c'est dans la création du bumper, car aucun préfab ne vous est fourni et vous n'avez donc que les feuilles de sprites prédécoupés comme base. Vous devez donc pour chacune des 8 directions de bumper :

- Créer un préfab avec un collider et un trigger : le collider va permettre d'arriver ou de marcher sur le côté du bumper (de lui donner une collision physique indépendante) et le trigger va servir à déclencher le bump à proprement parler. Le but est d'isoler la surface rebondissante pour éviter de déclencher le bumper en touchant le sprite à un autre endroit.

- Créer deux états au bumper. En temps normal il est replié et lorsqu'il se déclenche son sprite change en position dépliée (regardez la feuille de sprites si vous ne comprenez pas). Au bout de 0.3 seconde il reprend sa position repliée initiale.

- Jouer le son "bumper.wav" disponible dans les assets lorsque le bumper est activé.

Maintenant passons aux TVs. Vous allez devoir créer les TV suivantes :

- Pieces : Casser cette TV donne 10 anneaux.

- Super-bottes : Casser cette TV donne la super vitesse. Pendant 15 secondes vous devez augmenter la vitesse max de Sonic à 30 ainsi que le pitch de la musique jouée de 20 pourcents. Au bout des 15 secondes la musique redevient normale et la vitesse max retombe à 20.

- Shield : Casser cette TV donne un bouclier à Sonic. Vous devez instancier le prefab animé du bouclier sur Sonic. Il possède une variable currentShield prévue à cet effet. Vous devez également passer sa variable isShielded à true. N'oubliez pas également de modifier votre getHit() pour tenir compte du shield. Ce serait bête d'avoir un bouclier qui ne protège pas des coups, non ?

La grande question que vous devez vous poser c'est : comment casser une TV ? La réponse est simple :

- Chaque TV possède un collider. Si Sonic entre en contact avec les bool isRolling ou isJumpBall à true la TV est détruite et il gagne le bonus.

- Les TVs sont animées en 2 images, une affichant le bonus et l'autre une image parasitée. Encore une fois vous trouverez tout ça dans les sprites.

- Une TV cassée ne doit pas disparaître mais vous devez changer son sprite. Il y a un sprite de TV détruite dans les SpriteSheets. Une TV cassée n'a plus de collider et on peut donc passer à travers.

- Lorsqu'une TV est cassée vous devez appeler la méthode destroy() de Sonic. Cette méthode fait rebondir Sonic et joue le son approprié. Vous l'utiliserez d'ailleurs également dans l'exercice suivant pour la destruction des ennemis.

Chapter VII

Exercise 04 : Ennemies!

	Exercise 04
Exercise 04 : Ennemies!	
Files to turn in : `A scene named after the area of your choice, different from the last exercises and everything that seems relevant.`	
Forbidden functions : `None`	
Remarks : `n/a`	

`Attention ! Je vous rappelle que vous devez créer un nouveau niveau à chaque exercice. Vous devez donc ici en être à votre 4e niveau.`

Maintenant vous êtes un peu plus libre et si vous êtes arrivés jusqu'ici c'est le moment de devenir game designer.

Vous allez trouver dans toutes les spritesheets fournies différents sprites d'ennemis. Vous avez carte blanche sur la façon de procéder mais vous devez créer 3 ennemis aux comportements différents.

- Un ennemi fixe qui tire des projectiles.
- Un ennemi peu mobile qui possède deux phases : soit il bouge lentement, soit il est immobile mais entouré de pics et est donc invincible (et dangereux).
- Un ennemi assez mobile sans autre compétence particulière.

Pour tuer un ennemi les conditions sont exactement les mêmes que pour détruire

une TV. Il faut que Sonic saute/roule et lorsqu'il détruit l'ennemi vous devez lancer sa méthode Destroy() (et n'oubliez pas d'ajouter les points au score !).

Chapter VIII

Bonus Exercise: Doctor Eggman

	Exercise 05
Bonus Exercise: Doctor Eggman	
Files to turn in : `A final scene and everything that seems relevant.`	
Forbidden functions : `None`	
Remarks : `n/a`	

Si vous en êtes là bravo. Vous avez mérité votre moment de gloire et vous pouvez créer le combat contre le boss final de la manière que vous préférez.

Cet exercice est totalement optionnel et ne rapportera aucun point bonus. C'est vraiment pour conclure le jeu et vous la pèter en soutenance.

Choisissez parmi les assets ceux de robotnik qui vous parlent, choisissez une musique de boss épique et faites vous plaisir ! A vous de faire un combat mémorable et d'en mettre plein la vue à vos correcteurs !

Un café sur la terrasse du bocal offert par Thor si vous validez cet exercice.