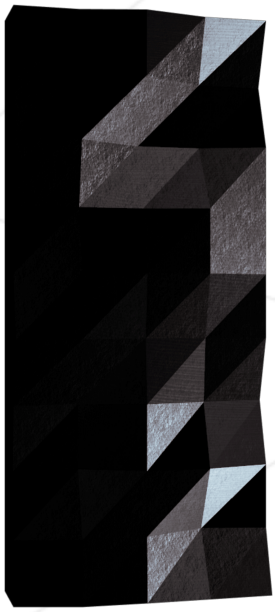

Piscine Unity - D04

Les PlayerPrefs et les Coroutines

Staff staff@staff.42.fr

Résumé: Ce document contient le sujet du jour 04 de la piscine Unity de 42.

Table des matières

Chapitre I

Consignes generales

- La piscine Unity est à faire entièrement et obligatoirement en `C#` uniquement. Pas de `Javascript/Unityscript`, de `Boo` ou autres horreurs.
- L'utilisation de fonctions ou de namespaces non autorises explicitement dans le header des exercices ou dans les regles de la journee sera considéré comme de la triche.
- Pour une utilisation optimale de Unity, vous devez travailler sur le `~/goinfre`, qui est en local sur le mac que vous utilisez. Pensez à bien récupérer vos projets avant de vous delog car le goinfre local est vidé régulièremment.
- Contrairement aux autres piscines, chaque journée ne demande pas un dossier `ex00/`, `ex01/`, ..., `exn/`. A la place pour la piscine `Unity`, vous devrez rendre votre dossier projet qui aura pour nom le nom de la journee : `d00/`, `d01/`, ... Toutefois, un dossier de projet contient par defaut un sous-dossiers inutile : le sous-dossier `"projet/Temp/"`. Assurez-vous de ne **JAMAIS** pusher ce dossier dans votre rendu.
- Au cas ou vous vous poseriez la question, il n'y a pas de norme imposée à `42` pour le `C#` pendant cette piscine Unity. Vous pouvez utiliser le style qui vous plaît sans restriction. Mais rappelez-vous qu'un code que votre peer-evaluateur ne peut pas lire est un code qu'elle ou il ne peut noter.
- Vous devez trier les assets de votre projet par dossier. Chaque dossier correspond à un et un seul type d'asset. Par exemple : `"Scripts/"`, `"Scenes/"`, `"Sprites/"`, `"Prefabs/"`, `"Sounds/"`, `"Models/"`, ...
- Assurez-vous de tester attentivement les prototypes fournis chaque jour. Ils vous aideront beaucoup dans la compréhension du sujet et du travail attendu.
- L'utilisation de l'Asset Store d'Unity est interdite. Vous êtes encouragés à utiliser les assets fournis chaque jour (quand nécessaire) ou à en chercher d'autres sur le net s'ils ne vous plaisent pas, sauf bien entendu pour les scripts car vous devez avoir écrit tout ce que vous rendez (hors scripts fournis par le staff, obviously). L'Asset Store est interdit car quasiment tout le travail que vous avez à faire s'y trouve déjà sous une forme ou sous une autre. Néanmoins l'utilisation des Standard Assets de Unity est autorisée voir meme conseillée pour certains exercices.

- Pour les corrections à partir du d03 il vous sera demandé de builder les jeux pour les tester. `C'est le correcteur` qui doit build le jeu vous devez donc évidemment toujours push vos projets/sources. De ce fait votre projet doit correctement configuré pour le build. `Aucun` réglage de dernière minute ne doit être toléré.
- Important : Vous ne serez pas évalués par un programme, sauf si le contraire est explicite dans le sujet. Cela implique donc un certain degré de liberté dans la façon que vous choisissez de faire les exercices. Toutefois, gardez en tête les consignes de chaque exercice, et ne soyez pas FAINÉANTS, vous passeriez à coté de beaucoup de choses intéressantes.
- Ce n'est pas grave d'avoir des fichiers supplémentaires ou inutiles dans votre dossier de rendu. Vous pouvez choisir de séparer votre code en différents fichiers au lieu d'un seul, sauf si le header d'un exercice mentionne explicitement les fichiers à rendre. Un fichier ne doit définir qu'un et un seul comportement, pas de namespaces donc. Toute cette consigne ne s'applique bien evidement pas au sous-dossier `"projet/Temp/"` qui n'a pas le droit d'exister dans vos rendus.
- Lisez le sujet en entier avant de commencer. Vraiment, faîtes-le.
- Le sujet pourra être modifié jusqu'à 4h avant le rendu.
- Meme si le sujet d'un exercice est relativement court, ca vaut le coup de passer un peu de temps à comprendre parfaitement le travail attendu pour le faire au mieux.
- Parfois il vous sera demandé un soin particulier sur la qualité artistique de votre rendu. Dans ce cas, cela sera mentionné explicitement dans le sujet correspondant. N'hésitez alors pas à tester plein de choses différentes pour vous donner une idée des possibilités offertes par Unity.
- Par Odin, par Thor ! Refléchissez ! ! !
-

Chapitre II

Foreword

- Sonic 3 et Sonic & Knuckles (1994) est l'un des premiers jeux vendus en kit de l'histoire du jeu vidéo. En effet, a une époque où les DLC n'existaient pas encore, SEGA a fait fort en sortant les deux jeux à moins de 6 mois d'intervale l'un de l'autre, vendant les deux au prix fort. La cartouche de Sonic & Knuckles était spéciale car elle contenait .... un port cartouche. On pouvait donc placer celle-ci dans la mégadrive puis enfoncer la cartouche de Sonic 3 par dessus pour profiter du jeu complet réuni et de ses 12 mondes et ses 2 set d'émeraudes et super émeraudes du chaos a trouver. Tout un programme ...

- Michael Jackson a composé des musiques de Sonic 3. C'est une légende urbaine qui a été prouvée il y a quelques années. Il n'est en effet pas crédité au générique du jeu car apparemment le son de la puce sonore de la console n'était pas d'assez bonne qualité à son goût. Ce qui est plutôt drôle quand on sait qu'il a sorti SON jeu mégadrive (Moonwalker, 1990) quelques années plus tôt avec quelque uns de ses tubes en guise de bande son. Une oreille attentive pourra aussi entendre des samples de sa voix dans certaines pistes de Sonic 3, notamment le générique de fin.

- Sonic 3 + Sonic & Knucles, encore eux, font partie des premiers jeux console dotés d'un système de sauvegarde. Plus besoin de faire tout le jeu d'une traite et possibilité de recommencer à n'importe quelle zone débloquée, c'était une révolution pour la franchise. Il ne faut cependant pas oublier le premier Zelda sorti sur NES 7 ans plus tôt avec lui aussi un système de sauvegarde intégré.

- Sonic 1 cette fois : Le jingle sonore d'intro de SEGA prend 1/8e de la taille totale de la mémoire de la cartouche. A l'époque la taille des cartouches était ridiculement petite et deux secondes de son prenaient plus de mémoire que des zones de jeu entières. Et encore, le son était compressé à la limite de l'audible.

- D'après le lore du jeu, sonic court tellement vite qu'il peut dépasser la vitesse de la lumière. Sonic Générations a réussi le tour de force de lui permettre dépasser le mur du son (en mètres par seconde ingame) tout en restant jouable.

- Sonic a la même coupe que Sangoku et quand ils s'énervent les deux ont les cheveux qui deviennent jaune et une aura d'énergie autour d'eux. Officiellement Sangoku est devenu super Saiyan seulement 4 jours avant la sortie de Sonic 1 sur Mégadrive, alors qui est l'oeuf et qui est la poule ?

Chapitre III

Exercice 00 : Data select !

	Exercice : 00
Exercice 00 : Data select !	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `La scène "TitleScreen", la scène "DataSelect" et tout ce qui vous semble pertinent`	
Fonctions interdites : `Aucune`	
Remarques : `n/a`	

`Il y a énormément de choses à faire aujourd'hui et beaucoup de niveaux à créer donc ne trainez pas ! Ne perdez pas de temps sur des détails et rappellez vous que vous êtes notés sur ce qui est demandé. Vous aurez tout le temps de peaufiner vos niveaux/menus/etc plus tard si vous en avez envie.`

`Vous avez le droit de modifier comme bon vous semble tous les assets fournis aujourd'hui y compris les scripts (notamment celui de Sonic si vous voulez y rajouter quelque chose ou l'améliorer). Faites attention cependant à ne rien casser, beaucoup d'éléments sont liés entre eux et une modification à un endroit peut avoir des répercussions ailleurs.`

Vous devez créer un profil utilisateur qui sera enregistré dans les players prefs pour pouvoir être rechargé si on quitte et relance le jeu. Ce profil doit contenir la liste des niveaux débloqués par le joueur, le nombre de vies qu'il a perdu toutes parties confondues, le nombre d'anneaux gagnés toutes parties confondues ainsi que son meilleur score de run sur chaque niveau.

Vous devez également créer une scène DataSelect qui permet de visualiser grâce à une GUI toutes ces informations ainsi que la liste des niveaux jouables, vérrouillés et dévérrouillés. N'hésitez pas à tester la démo pour voir un exemple de mise en place, sachant que vous êtes libres de faire l'interface qui vous plait, tant que toutes les infos y sont.

Vous devez ensuite créer un écran titre (c'est toujours bien d'avoir un titre pour un jeu !) avec un bouton cliquable qui vous permettra de réinitialiser le profil du joueur. On peut également passer à l'écran DataSelect en appuyant sur la touche Entrée/Return pour pouvoir ensuite choisir un niveau et lancer la partie.

On joue ici avec les playerprefs dans un but pédagogique mais en temps normal ne les utilisez JAMAIS pour stocker des infos que le joueur n'est pas censé pouvoir modifier ingame (comme sa progression par exemple). Les playerprefs sont stockées dans un fichier et donc éditables. Elles sont surtout utilisées pour garder en mémoire les réglages du joueur correspondant au menu Options d'un jeu : les keybindings, les prefs audio et vidéo, etc ...

Chapitre IV

Exercice 01 : Un niveau de base

	Exercice : 01
Exercice 01 : Un niveau de base	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `Une scène nommée avec le nom de la zone de votre choix et tout ce qui vous semble pertinent`	
Fonctions interdites : `Aucune`	
Remarques : `n/a`	

`Vous trouverez dans les fichiers fournis aujourd'hui 4 sets de tiles/prefabs déjà préconfigurés pour créer vos niveaux. Si ce n'est pas fait allez lire le ReadMe pour des explications plus détaillées sur leur fonctionnement et la façon de les modifier à votre convenance.`

Dans Sonic le principe est simple, le personnage commence au début du niveau qu'il doit traverser rapidement en ramassant un maximum d'anneaux. La fin du niveau est systématiquement marquée par un panneau tournant lorsque Sonic passe à côté. Le principe de base est que généralement les niveaux sont conçus pour être traversés à grande vitesse en choisissant sa route parmis plusieurs chemins (contrairement à son rival historique Mario dont les jeux sont plus lents, plus techniques et souvent avec des niveaux linéaires). Un bon level design c'est un mélange de courbes, de bonus et d'ennemis bien placés pour que le parcours soit fluide, avec quelques pièges savament disposés pour éviter que tout ça ne devienne une promenade. Quand Sonic est touché il perd tous ses anneaux qui tombent autour de lui. Il peut tenter d'en ramasser quelques uns avant qu'ils disparaissent. S'il est touché alors qu'il n'a pas d'anneaux alors il perd une vie et recommence au début du niveau ou au dernier point de contrôle.

Ces bases posées vous allez devoir créer un premier niveau simple (sans pièges, trous ni ennemis) avec des chemins intéressants et différents à parcourir. Le but sera d'attrapper un maximum d'anneaux et de finir le niveau le plus rapidement possible. Une fois votre level design terminé vous allez devoir :

- Intégrer un compteur de temps en GUI qui affiche les secondes et les minutes écoulées depuis le début du niveau sous la forme suivante 0 :00. Pas de float dans les minutes donc, j'insiste.
- Créer les anneaux. Vous trouverez déjà un préfab animé d'anneau qu'il va falloir modifier pour pouvoir permettre à Sonic de les attrapper. Vous devez aussi stocker ces anneaux quelque part. A vous de voir si vous préférez modifer un script existant ou en créer un nouveau. Plusieurs solutions existent, choisissez celle qui vous semble la plus adaptée.
- Utiliser le son fourni à chaque fois que Sonic attrappe un anneau. C'est la marque de fabrication du jeu qui fait que quelqu'un dans la pièce d'à côté sait que vous jouez à Sonic.
- Intégrer un compteur d'anneaux qui indique en temps réel le nombre d'anneaux qu'a Sonic.
- Intégrer la musique du niveau (chaque niveau a sa propre musique), soit en choisissant dans les musiques proposées dans les assets, soit en allant fouiller sur le net, les remix se comptant par centaines. La seule contrainte est que le thème musical doit être celui de la zone jouée (ou alors celui d'une autre zone particulièrment cool, mais il faut que ça reste du Sonic !).
- Intégrer le panneau tournant de fin de niveau, ainsi que la musique correspondante, qui sera déclenchée lorsque Sonic passe à côté du panneau.
- Calculer un score final à la fin du niveau qui doit apparaître au milieu de l'écran exactement 6 secondes après le début de la musique pour être en rythme avec celle-ci. Dit comme celà ça semble étrange, mais c'est ce genre de petits détails insignifiants qu'ont en commun tous les jeux ayant marqué l'histoire du jeu vidéo.

Vous êtes libres de calculer le score comme il vous plait mais l'équation doit tenir en compte des ennemis tués, du nombre d'anneaux à l'arrivée et du temps mit à parcourir le niveau. Voici par exemple un barème conseillé : 500pts par ennemi tué, 100pts par anneau à l'arrivée et 20000pts - 100pts par seconde écoulée depuis le début du niveau (avec un minimum de 0pts au bout de 200s).

Chapitre V

Exercice 02 : On arrête la balade ?

	Exercice : 02
Exercice 02 : On arrête la balade ?	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `Une scène nommée avec le nom de la zone de votre choix, différente du niveau précédent et tout ce qui vous semble pertinent`	
Fonctions interdites : `Aucune`	
Remarques : `n/a`	

Voilà déjà un gros morceau de fait ! Maintenant nous allons ajouter quelques pièges pour compliquer la tâche du joueur.

Vous devez à présent rajouter des pics que vous trouverez dans les sprites fournis aujourd'hui. A vous de créer vos propres préfabs avec tous les colliders et scripts qui vont bien. Vous devez aussi créer des trous qui font perdre directement une vie. C'est déjà automatiquement géré par le script qui lance un dead() lorsque la position.y de Sonic est trop basse, vous n'avez donc qu'à placer des trous où vous souhaitez et regarder la magie opérer.

`Pour les trous Dead() est lancé quand la position.y de Sonic passe en dessous de -15. Tenez en compte lors de la création de vos maps pour éviter de Trigger Dead() par inadvertance.`

Nous vous avons fourni un Sonic tout beau dans les sources pour vous éviter de passer la journée à simplement recoder sa physique et sa bonne dizaine d'animations. Malheureusement une partie de son script a été corrompue et une méthode à disparue. Votre mission est de trouver cette méthode intitulée getHit() dans le script Sonic.cs et de réécrire son contenu.

Toutes les variables et méthodes dont je vais parler à partir de maintenant sont déjà implémentées dans le script Sonic.cs, vous n'aurez donc pas à les créer mais simplement à les appeler/attribuer, sauf mention contraire explicite.

Vous devrez donc :

- Vérifier que Sonic n'est pas invincible en testant le bool isInvincible
- Appeler la méthode Dead() si Sonic n'a pas d'anneaux sur lui. Simple et efficace. S'il a des anneaux par contre vous allez devoir :
- Stopper la vélocité du rigidbody attaché à Sonic (auquel vous pouvez accéder avec la variable rbody).
- Appliquer une impulsion sur rbody pour envoyer Sonic en l'air et dans le sens opposé à sa direction au moment du hit.
- Passer la variable isHit sur true. C'est ce qui va empêcher le joueur de pouvoir controller le personnage pendant qu'il est bumpé en arrière.
- Invoker la méthode stopHit (déjà présente dans le script) avec un délai de 2 secondes, pour que le joueur puisse récupérer les commandes de Sonic.
- Passer le bool getHit de l'animator à true pour lancer l'animation correspondante (si cette phrase n'a aucun sens regardez la méthode Dead() un peu plus loin dans le script pour y trouver un exemple).
- Créer et lancer une coroutine qui rend Sonic invincible pendant 5 secondes. Cette coroutine doit faire clignoter le sprite, passer isInvincible a true lorsqu'elle est appelée et le repasser à false à la fin des 5 secondes.
- Jouer le son de perte des anneaux qui est déjà stocké dans la variable aLoseRings.
- Mettre à 0 le nombre d'anneaux que Sonic possède et créer une explosion d'anneaux autour de lui. Vous devez instancier la moitié des anneaux qu'il possédait (par exemple 10 anneaux s'il en avait 20 sur lui). Les anneaux sont éjectées en l'air dans tous les sens et passent à travers Sonic. Au bout de deux secondes ils se mettent à clignotter et Sonic peut les ramasser en les touchant comme des anneaux standard. Ils disparaissent après avoir clignotté pendant 4 secondes.

Petit récap pour l'explosion d'anneaux afin d'être le plus clair possible : Sonic se fait toucher, la moitié des anneaux qu'il possédait est éjectée tout autour de lui. Pendant 2 secondes les anneaux lui passent à travers, puis, pendant 4 secondes ils clignottent avant de disparaître. N'hésitez pas à tester le prototype d'exemple pour voir le comportement attendu.

Chapitre VI

Exercice 03 : Rapide comme la lumière

	Exercice : 03
Exercice 03 : Rapide comme la lumière	
Dossier de rendu : *ex*03/	
Fichiers à rendre : `Une scène nommée avec le nom de la zone de votre choix, différente des niveaux précédents et tout ce qui vous semble pertinent`	
Fonctions interdites : `Aucune`	
Remarques : `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.

Chapitre VII

Exercice 04 : Des ennemis !

	Exercice : 04
Exercice 04 : Des ennemis !	
Dossier de rendu : $ex04/$	
Fichiers à rendre : `Une scène nommée avec le nom de la zone de votre choix, différente des niveaux précédents et tout ce qui vous semble pertinent`	
Fonctions interdites : `Aucune`	
Remarques : `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 !).

Chapitre VIII

Exercice Bonus : Dr Robotnik

	Exercice : 05
Exercice Bonus : Dr Robotnik	
Dossier de rendu : $ex05/$	
Fichiers à rendre : `Une scène finale et tout ce qui vous semble pertinent`	
Fonctions interdites : `Aucune`	
Remarques : `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.