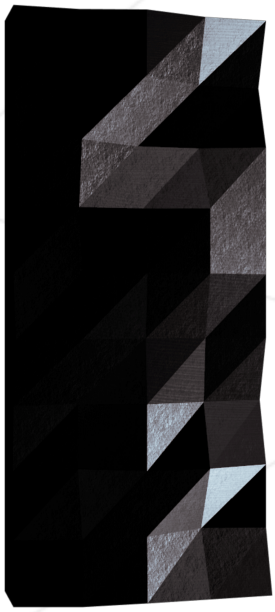

Piscine Unity - D00

Assets, GameObject, Behaviour, Input, Transform

Staff staff@staff.42.fr

Résumé: Ce document contient le sujet du jour 00 de la piscine Unity de 42.

Table des matières

Chapitre I

Piscine Unity Starter Kit

Bienvenue dans ce premier jour de Piscine Unity. Pour bien commencer voici une petite liste de conseils et autres liens qui vous seront très utiles tout au long de votre piscine.

- La documentation officielle de Unity, vous trouverez également un livre bleu présent dans l'inspecteur à coté de chaque component qui vous renverra vers sa documentation.
- Documentation officielle du C#
- Le sujet fait foi, ne vous fiez pas toujours à la lettre aux demos qui peuvent contenir des ajouts supplémentaires non demandés.
- Si vous créer des scripts génériques et réutilisables pour tout ce qui est utilitaire/non lié directement au gameplay de la journée vous gagnerez beaucoup de temps sur les jours suivants.
- Une fois la journée finie ne gardez jamais vos projets sur votre home, Unity a la sale manie de créer des milliards de fichiers et vos 5 précieux gigas (ainsi que vos temps de connexion) y passeront.
- Les journées sont longues donc ne perdez pas trop de temps sur des détails pendant vos exercices. Il sera toujours temps d'ameliorer votre jeu une fois les exercices de la journée terminés.
- Si votre MonoDevelop ne se lance pas, rendez-vous sur le mega-thread de la Piscine Unity où la marche à suivre vous sera expliquée.
- Si votre voisin de gauche et votre voisin de droite n'ont pas la réponse à un problème `TECHNIQUE` vous êtes invités à le reporter sur le forum avec le tag Piscine Unity.
- Les README sont la pour être lus, les préambules aussi ...

Chapitre II

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.
- 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 evidemment 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 III

Règles spéficiques à la journée

- L'utilisation de l'API Physique d'Unity est strictement interdite.

Chapitre IV

Exercice 00 : Gonflage de Ballon Simulator 2015

	Exercice : 00
Exercice 00 : Gonflage de Ballon Simulator 2015	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `Baloon.cs`	
Fonctions Autorisées : `Debug.Log, Mathf.RoundToInt, Input.GetKeyDown`	
Remarques : `n/a`	

Bien que le titre soit relativement explicite nous allons quand même préciser. Vous devez mettre en place une scène comportant un ballon. On doit pouvoir gonfler ce ballon à l'aide de la barre d'espace. Si le ballon est trop gros gonflé il explose ou si on ne le gonfle pas assez au bout d'un certain temps le jeu est perdu. Le ballon doit visuellement grossir en fonction de son niveau d'air.

Lancez la demo presente en pièce jointe à ce sujet pour vous faire une idee.

Vous devez également prendre en compte le souffle. Plus on gonfle vite plus on s'essouffle. A partir d'un certain niveau d'essouflemment on ne peut plus gonfler le ballon il faudra donc attendre que notre souffle se régénère.

Quand le jeu se termine vous devez afficher le temps arrondi en entier dans la console de la manière suivante :

```
Baloon life time: 120s
```

Chapitre V

Exercice 01 : Quick Time Event

	Exercice : 01
Exercice 01 : Quick Time Event	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `CubeSpawner.cs, Cube.cs`	
Fonctions Autorisées : `Debug.Log, Random.Range, GameObject.Instantiate, GameObject.Destroy, Input.GetKeyDown, Transform.Translate`	
Remarques : `n/a`	

Vous avez surement déjà joué à Guitar Hero, ou même à des jeux d'actions vous demandant à un moment où à un autre d'appuyer sur certaines touches au bon moment. C'est ce qu'on appelle pour les plus néophytes d'entre vous des QTE ou Quick Time Event.

Lancez la démo pour vous faire une idée. Vous devez utilisez les `A`, `S` et `D` pour jouer à ce jeu.

Le but de cet exercice vous l'aurez compris c'est donc de réaliser un QTE. Mettez en place une ligne vers le bas de l'ecran et une autre vers le haut ainsi que trois lignes verticales rejoignant ces deux. Faites en sorte que des carrés soient générés de manière aléatoire sur une des trois lignes verticales tous les X secondes. Ces carrés doivent chacun correspondre à une touche (des touches d'exemples vous sont fournis dans le zip de la journée). Chaque carré doit descendre avec une vitesse aléatoire en direction de la ligne horizontale du bas. Lorsqu'il atteint cette ligne le joueur doit appuyer sur la touche correspondante au bon moment. Chaque carré doit avoir sa propre ligne.

Vous devrez afficher la précision du joueur à chaque pression. La précision correspond à la distance entre le carré et la ligne au moment où le joueur appuie sur la touche. Elle doit etre affichée de cette manière :

```
Precision: 23.454
```

Les cubes doivent avoir une durée de vie et ne pas rester instantiés dans la scène éternelement une fois sortis de la zone de jeu.

Chapitre VI

Exercice 02 : Mini Golf

	Exercice : 02
Exercice 02 : Mini Golf	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `Ball.cs, Club.cs`	
Fonctions Autorisées : `Debug.Log, Mathf.Clamp, Transform.Translate`	
Remarques : `n/a`	

Rien de tel qu'un peu de sport. Vous allez donc créer un Mini Golf. Vous devrez donc faire en sorte qu'on puisse tirer dans une balle avec une force définie par la barre d'espace. Plus la pression est longue, plus la balle ira loin. La balle doit avoir une inertie et ralentir avec le temps. Le but etant donc de mettre la balle dans un trou. Si la balle touche un mur elle rebondit dans la direction opposée. Si la balle passe trop vite au dessus du trou, elle n'y tombe pas.

Le plus simple, c'est encore de tester ! Lancez la démo et amusez-vous quelques instants.

Pour ce qui est du score nous utiliserons une version simplifiée des regles du mini golf. Le joueur part avec -15 points, à chaque coup manqué il en gagne 5. Si son score est supérieur à 0 il perd, mais la partie ne s'arrete pas pour autant.

A chaque coup le score devra être affiché dans la console de la manière suivante :

```
Score: -5
```

Chapitre VII

Exercice 03 : Flappy Bird

	Exercice : 03
Exercice 03 : Flappy Bird	
Dossier de rendu : $ex03/$	
Fichiers à rendre : `Bird.cs, Pipe.cs`	
Fonctions Autorisées : `Debug.Log, Input.GetKeyDown, Transform.Rotate, Transform.Translate, Mathf.RoundToInt`	
Remarques : `n/a`	

Pour cet exercice c'est une version simplifiée de Flappy Bird que vous allez réaliser. Mettez donc en place l'oiseau au centre de la scène. Il doit tomber jusqu'à ce que le joueur appuie sur espace pour le faire remonter. Si l'oiseau touche le sol ou un des tuyaux la partie est perdue. Lancez donc la démo pour vous fair eune idée.

Quelques considérations techniques maintenant. L'oiseau ne doit jamais bouger sur l'axe x. Les tuyaux seront toujours à la meme hauteur parcequ'on a pas encore vu comment faire ca proprement. Il ne peut y avoir plus de deux tuyaux instancies, vous devez donc les reutiliser. La vitesse doit augmenter progressivement au fur et à mesure que le joueur passe des tuyaux.

Vous devrez également mettre en place un systeme de score et de temps. Chaque tuyau passé rapporte 5 points. À chaque fin de partie vous devez donc l'afficher de la manière suivante dans la console :

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

Chapitre VIII

Exercice 04 : Pong !

	Exercice : 04
Exercice 04 : Pong !	
Dossier de rendu : $ex04/$	
Fichiers à rendre : `Cube.cs, Player.cs`	
Fonctions Autorisées : `Debug.Log, Random.Range, Input.GetKeyDown, Transform.Translate`	
Remarques : `n/a`	

Pour terminer un concept qui a fait ses preuves et qui est relativement universel. Vous devrez réaliser un Pong. Si vous savez pas ce que c'est, lancez la démo. Le joueur de gauche déplace sa raquette avec les touches `W` et `S`, et le joueuyr de droite déplace sa raquette avec les touches `Up Arrow` et `Down Arrow`.

Faites en sorte qu'il y ai donc deux rectangles disposés de part et d'autre d'un terrain. Un carré doit rebondir entre eux deux, sa direction est inversee a chaque rebond, y compris verticalement.

Les deux rectangles doivent pouvoir être controlé indépendamment l'un de l'autre de manière verticale à l'aide des touches du clavier, avec pour limites, les murs du haut et du bas. Si le carré atteint un des bouts du terrain il est remis au centre et le joueur opposé gagne un point.

Le score doit être affiché à chaque point dans la console de Unity de la manière suivante :

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