Dossier Créatif

YAO Qingyuan

(étudiant non-francophone)

Partie commune

Presentation

Ceci est un jeu sérieux/éducatif en side-scrolling pour un musée de tissus.

Grèce ancienne, Arachné a été transformé en une araignée après avoir perdu à Athéna. 1560, une jeune reine de la France brodait le motif qui lui rappelle de son enfance à Florence. 1836, les peuples sur une ile en Ecosse luttait contre la révolution industrielle. 1960, les panneaux "Make love, not war!" flottaient dans l'océan de jeans à New York… 2010, Chine moderne, les vêtements en tissu mixte sont produites dans les usines…

Une jeune fille découvre que les vêtements qu'elle a acheté lors d'un vide-grenier à Shanghai la transportent dans les mémoires des tissus. Ayant obtenu les indices dans ces mémoires, elle retrouve leur vendeuse Arachné qui ne croit plus à la représentation d'humanité par les tissus à cause de la mondialisation et la surconsommation. La fin dépend des choix du joueur.

Expérience et fonctionnement de l'interactivité

L'interaction du jeu sera principalement dans les morceaux de mémoire (Italie&France, Ecosse, New York), le reste du jeu sera un visual novel aux choix multiples (Grèce, Chine).

Lors d'une séquence de mémoire, le joueur est en forme d'une âme sur le tissu. Il est capable de sauter sur les autres tissu à côté (50cm) du même type. Ceci est inspirée par l'obsession d'un objet dans *Ghost Tricks: Fantom Detective.*

Le joueur possède également le pouvoir de rembobiner le temps jusqu'au début de la séquence. Tous les objets dans la mémoire sera affectés par le rembobinage sauf l'âme du joueur (inspiré par *Life Is Strange*). La maîtrise du rembobinage permet l'âme d'aller plus loin (ex. sauter sur un personnage qui entre et puis rembobiner pour sortir) (inspiré par *Braid*)

Inspirations sonores

La bande sonore sera adapté pour chaque époque avec l'instrument et la mélodie correspondants(**Grèce**[1]: tragique avec la lyre, **Italie&France**[2]: les violes, les luths, etc.. **Etats Unis**: rock avec la guitare électrique. **Ecosse**: folk avec la grande cornemuse écossaise. **Chine moderne**[3]: musique pop mais avec Guzheng et Erhu). De plus, il y aura une mélodie commune jouée avec l'instrument different de chaque époque (inspiré par *Assassin's Creed)*.

---

[1] https://youtu.be/9RjBePQV4xE
[2] http://www.vam.ac.uk/content/articles/r/renaissance-music/
[3] https://youtu.be/v_hbbuHVtkQ

Inspirations visuelles

Le style est inspiré par *Father And Son*, avec les lieux et les vêtements de chaque époque.

Problématique technique

La problématique la plus difficile sera la mécanisme du rembobinage. Puisqu'il est possible de rembobiner à chaque instant, il faut garder les traces de mouvement de chaque object à chaque moment pour y revenir en optimisant la mémoire pour garantir la performance du jeu.

Public cible

Ce jeu est un jeu éducatif destiné aux visiteurs du musée. Voici leurs 3 caractéristiques:

1. La plupart des visiteurs amènent leur propre smartphone - il est préférable de développer le jeu pour les écrans à l'exposition, mais aussi une version pour iOS et Android.
2. Ils sont multiculturels et de tous âges - le jeu sera multilingue avec le vocabulaire simple pour les enfants, mais aussi avec les informations supplémentaires pour les adultes.
3. Les visiteurs visitent le musée - le jeu sera fournit avec les fonctionnalités exclusives lors de la visite au musées. Par exemple, un scanneur de code QR qui affiche les informations comme l'application de Centre Pompidou, ou le GPS qui ouvre le contenu supplémentaire du jeu comme *Father And Son* pour le Musée archéologique national de Naples

Partie spécifique

Problématique: rembobinage

Supposons d'abord que le jeu est fait dans Unity et les animations sont squelettiques.

Le rembobinage du jeu a les caractéristiques ce-dessous:

1. Le rembobinage est possible dès le début de la séquence, qui est aussi le plus loin possible qu'on peut rembobiner.
2. Les mouvements de chaque objets dans la séquence sont scénarisées, il n'y aura pas donc la variation de mouvement pour chaque rembobinage.
3. Le temps total d'une séquence est fixé (5 - 10 minutes)
4. L'âme n'est pas affecté directement par le rembobinage, mais il déplace avec l'objet possédé.

Il existe plusieurs jeux avec la mécanisme du rembobinage similaire, mais pas tout à fait identique à ce jeu. Pourtant, il est possible de partir sur les notes et les remarques de ses développeurs pour chercher la meilleure façons d'appliquer.

Bibliographie

Pour cette problématique, j'ai référencé 2 jeux qui implante le rembobinage et 4 tutoriels.

Retours visuels

Dans *Life Is Strange*[4], le monde sauf la protagoniste Max est rembobinable, de plus, Max n'est pas affecté par le rembobinage. Quand elle rembobine, les développeurs rendent Max invisible au monde, presque comme un fantôme, pour ne pas affecter le monde et ne pas être affecté par le monde. Pour indiquer les états aux joueurs, les artistes ajoute les effets de fuite de lumière lors du rembobinage et les effets de brûlage pour le bout du rembobinage.

Remarque: L'invisibilité et l'isolation de l'âme seront toujours activés dans mon jeu, de plus, des retours d'informations aux joueurs de quelques sortes seront appliqués.

Implementations Possibles

Dans le jeu *Braid*, le joueur et les ennemies sont rembobinables, mais pas l'environment. Dans une conférence des développeurs de jeux (GDC)[5] le développeur Jonathan Blow a expliqué qu'il a exploré 3 façons à implémenter le rembobinage:

---

[4] https://youtu.be/NVsMERVjEUg
[5] https://youtu.be/8dinUbg2h70

1. Enregistrer les entrées par les joueurs (refusé, pas assez précis)
2. Simuler dans le moteur du jeu (refusé, complique le moteur du jeu qui est déjà compliqué)
3. Enregistrer les états du monde à chaque image (idéalement 60 image et donc 60 appelles par seconde)

Il a décidé de partir sur la troisième et améliorer la mémoire en compressant les états enregistrés (ex. Ne enregistrer que les états des objets qui bougent, réduire les données des images intermédiaires).

Remarque: Blow présenté une implémentation simple et robuste qui peut être optimisé après.

Tutoriels dans Unity

Il y a plusieurs tutoriels qui partent sur la troisième façons présenté par Blow avec quelques différences en optimisation.

Brackeys[6] enregistre la position et la rotation comme l'état à chaque FixedUpdate (séparé aux images, 50 appelles par seconde par défaut) sans faire l'optimisation, pour rembobiner, il suffit de récupérer le dernier état un par un et appliquer sur les objets. Pour sa démo qui n'a besoin des 5 dernières secondes, la non-optimisation n'est pas grave, mais ce n'est pas le cas de mon jeu.

Cameron LeBlanc[7] a fait l'ingénierie inverse de Braid pour optimiser, il divise les images à enregistrés par 10 et il compresse les float à suivre (vélocité à 1 sbyte, position à 3 short).

Remarque: L'optimisation est différentes selon le besoin différent de chaque jeu.

Animation

Après avoir lu le tutoriel d'Alexander Grishanin[8] qui a décidé de construire un éditeur d'animation (ou un timeline) pour simplifier la vie du game designer, je m'intéresse sur la possibilité de rembobiner directement depuis Animator, celui-ci n'est pas assez tenté par les autres puisque leurs jeux ne sont pas déterministes ou il y a toujours des variations de mouvement à chaque rembobinage. Dans Animator d'Unity, selon ce tutoriel[9], il faut simplement changé la vitesse à -1 pour inverser l'animation, il est également possible de remplacer Animator par un script.

Remarque: Une implémentation qui combine la fanons de Blow et la manipulation d'animation peut être mis en place.

---

[6] https://youtu.be/eqlHpPzS22U
[7] https://gamasutra.com/blogs/CameronLeBlanc/20130320/188921/Recreating_the_time_mechanics_of_Braid_Part_3.php
[8] https://letsmakeagame.net/level-creation-utility-unity/
[9] http://gyanendushekhar.com/2016/10/28/reverse-animation-play-unity3d/

Solution

Après avoir vu les décisions pris par les développeurs, voici mes choix:

(Pré)Enregistrement

Grace à l'invariabilité des mouvements, il est possible de prédire tous les mouvements de tous les objets sauf l'âme. Donc, pour améliorer la performance au maximum, j'ai décidé de préenregistrer tous les mouvements en avance. Lors du jeu, le moteur n'a pas besoin d'enregistrer aucun mouvement.

Séquences d'animation

Comme le jeu est en 2D sans physiques, les animations ne manipuleront que: transform.position(Vector2), transform.rotation(une seule angle), et aussi transform.scale(2D) pour certains objets.

Puisque les animations de chaque objet se joue en boucle la plupart du temps, j'ai décidé d'en profiter pour optimiser le stockage des mouvements. Pour chaque boucle d'une animation, je le remplace par une classe AnimLoop

Class AnimLoop

L'animation seront contenus dans une classe AnimLoop avec les attributs ci-dessous:

animation (Animation): l'animation à boucler

start et end (float): 2 float qui décrivent le début et la fin d'une AnimLoop (plutôt pour debug)

repeat (int): un int qui décrit combien de fois une animation va répéter

cpt (int): un compteur qui compte les fois que l'animation a joué

Les attributs dois satisfait aux règles ci-dessous:

1. end - start = repeat * (durée de l'animation)
2. start = end du AnimLoop précédent, end = start du prochain AnimLoop

Dans FixedUpdate(), il vérifie à chaque fois si GetKeyDown() sur la touche qui active le rembobinage est vrai, sinon il commence à jouer les animations et il augmente le compteur à la fin de chaque animation. À la détection de GetKeyDown()=true, il lance Rewind() qui modifie la vitesse de l'animation en cours à -1 pour qu'elle revient au début, et rejoue cpt fois l'animation en arrière en décrémentant le compteur jusqu'à ce qu'il atteigne 0.

Class des objets

Chaque objet contient une List<AnimLoop> qui contient tous les AnimLoop que l'objet va réaliser lors d'une séquence.

Ex: [Walk x 5, Enter, Walk x 10, GetUpstairs, Walk x 5, PutDownCloth, Turn, Walk x 5]

Ex: [Protest x 100] (x 100 veut dire un seul AnimLoop avec repeat=100)

Il y aura aussi une fonction Rewind() qui sera lancé par FixedUpdate() pour gérer le rembobinage de l'ensemble d'AnimLoop.

Démarche & rembobinage

Dès le début, la List<AnimLoop> de chaque objet commence à dérouler en même temps. À la détection du rembobinage, les Rewind() des objets et d'AnimLoop seront lancé. Au relâchement de la touche, les animation continueront en avant.

Ex: Walk1->Walk2->Walk3->Walk4->Walk5->Enter->Walk1->Wal->REMBOBINE->laW->1klaW->retnE->5klaW->4klaW->3klaW->2kla->CONTINUE->alk2->Walk3->Walk4->Walk5… (Walk1->…Walk5 est une décomposition de Walk x 5)

Position de l'âme

La position de l'âme sera toujours la même avec la position de l'objet qu'il possède,

Retours visuels & sonores

Afin que le joueur rende compte du rembobinage, l'effet visuel du "glitch" de la cassette VHS et l'effet sonore du rembobinage de la cassette sera ajouté lors du rembobinage. Au bout du rembobinage, il y aura l'effet visuel et sonore de l'absence de signal.