Rapport Projet Programmation 2 - Partie 2

HURIOT Sacha, YAX Nicolas

April 11, 2019

A Turtle's Holistic Adventures

Les lumières venaient de s'éteindre, les chercheurs rentraient chez eux après une nouvelle journée infructueuse de tests. Seuls les animaux cobayes restaient à l'intérieur du laboratoire, témoins de l'obscurité et du silence froid de leur cellule respective. La tortue, dernière représentante de son espèce dans cette installation souterraine, s'apprêtait à fermer les yeux quand soudain, une lumière forte, un bruit assourdissant, les minces barreaux de sa cage se brisent net. Éblouie par les lumières perçantes, elle peine à discerner la silhouette qui se penche au-dessus de sa tête et y dépose un casque léger. La silhouette disparaît aussi vite qu'elle est apparue mais la tortue ne le remarque même pas, un flot d'informations et de pensées surgit là où quelques instants auparavant il n'y avait que de simples idées. La tortue reprend ses esprits et, pour la première fois, comprend la situation dans laquelle elle se trouve. Tandis que l'alarme du système de sécurité résonne toujours aussi forte, la tortue ne se focalise plus que sur un objectif : sortir d'ici. Elle fait le tri dans ces nouveaux savoirs dont elle ignore l'origine mais est certaine de la validité : la seule sortie est par l'ascenseur dont l'utilisation requiert un badge, suite à l'explosion les androïdes du système de sécurité du laboratoire ne vont pas tarder à être déployés et s'ils la trouve hors de sa cage, s'en est fini. Elle décide de se réfugier dans la cage d'ascenseur qui paraît être sure pour l'instant. Ensuite chemin, elle aperçoit deux prototypes des armes développées dans ce laboratoire, des objets intelligents commandés par la pensée. En se concentrant, et grâce à ses capacités tout juste acquises, elle en prendre le contrôle et réussit là où tous les tests de ces dernières semaines échouaient. Dans l'ascenseur, elle trouve un oiseau également libéré par l'explosion et elle découvre une nouvelle fonction de son casque : elle peut transmettre ses pensées à son nouveau compagnon. Elle donne une des deux armes à l'oiseau et commence à former un plan pour s'échapper : rassembler une armée parmi les cobayes et descendre au quinzième sous-sol, là où se trouve un prototype de téléporteur.

Problèmes de temps Nous avons eu de serieux problèmes de temps pour travailler sur le projet suite à de nombreux DM/Exams/Autres projets (comme celui de Compilation qui demande énormément de temps). C'est pourquoi nous n'avons pas eu le temps de tout corriger pour le rendu et que nous présentons une version "Démo" de ce que nous voulions faire. Ainsi elle n'implémente malheureusement pas tous les aspects demandé pour le moment bien qu'ils aient été beaucoup travaillés (voir le code source qui a plus que doublé de taille depuis la Partie 1).

1. Boss Cette partie a été difficile a implémentée car créer un boss interessant demande beaucoup d'énergie tant du point de vue de la conception que de la programmation (et surtout la programmation). Pour implémenter une telle feature, j'ai utilisé le systeme d'IA que j'ai mis au point : une IA est un event (fonction Unit=¿Int) qui tourne tant que le jeton concerné n'est pas mort et le controle comme une marionette et qui caracterise son comportement pendant sa durée de vie. Ainsi le boss a une IA assez complexe car elle alterne 3 patternes de combat (cf fichier d'aide pour battre le Boss). Vu le peu de temps que j'ai eu pour travailler sur le projet, je n'ai pas eu le temps de finir son systeme d'energie ni de corriger les bugs du systeme d'eau (voir le fichier pour expliquer comment battre le boss). Par conséquent j'ai ajouté des kits de soins pour qu'il soit faisable. Le code source du boss est très complexe et utilise beaucoup de programmation fonctionnelle (cf code source) nottement sur la notion d'events composés les uns sur les autres.

2. Donjon L'idée était de proposer une progression dans un donjon, en partant du 1er étage avec une petite équipe et de monter dans les étages en accumulant de l'équipement afin de battre les 3 boss prévus du donjon. Ainsi la progression dans les étages a été codée encore une fois avec des events.

3. LayerSet Afin d'avoir un code encore plus polyvalent et plus proche de la programmation objet, j'ai opté pour une reconversion du code source vers un systeme de layers (couches contenant les sprites) afin de pouvoir afficher des sprites plus larges (avant ils ne pouvaient que etre 32x32). Cette conversion a prit une grande partie de notre temps mais permet d'avoir un code bien plus jolie et efficace.

4. Compétences Pour diversifier le gameplay, des compétences sont données aux personnages afin de donner une importance à l'équipement. Ces compétences ont des Array[Int] =¿ Unit pour les initialiser et utilisent des events pour calculer leurs effets sur le temps (comme une brulure ou un empoisonnement).

De mon côté, je me suis occupé de la gestion des objets et de l'inventaire, de la création des nouveaux sprites (effets sur l'eau, objets et ennemis), de l'affichage de la fiche des personnages, de la base de données des objets ainsi que de leur hiérarchie de classes et, enfin, de la phase de récompenses et de gestion d'inventaire après les combats. On s'est fixé beaucoup d'objectifs pour ce deuxième rendu, ce qui n'a pas forcément été en accord avec notre emploi du temps. Beaucoup des nouvelles implémentations dépendent les unes des autres ce qui nous empêchait de vraiment en retirer une sans devoir faire une croix sur autre chose.

1. Gestion des objets et de l'inventaire ('Inventory.scala') J'ai construit une hiérarchie de classes pour les objets : Item se divise en Weapon, Armor et OneUse, ce qui permet de gérer les objets de manière générale dans l'inventaire et de manière spécifique dans leur utilisation spécifique (équipement qui donne une compétence ou recharge qui change les stats). La classe mainInventory permet de stocker la reserve globale de l'équipe (accessible entre les niveaux) et les objets portés par chaque membre (utilisables en combat).

2. Affichage de la fiche des personnages et de l'inventaire ('Display.scala') J'utilise une classe Sheet qui a une méthode pour modifier le layer réservé aux fenêtres d'interface. Elle affiche le sprite du dernier animal sélectionné

avec son inventaire et son équipement en bas à gauche du niveau. L'inventaire est affiché sous forme de plusieurs onglets : un pour le sac principal ou se trouve toutes les récompense et un pas personnage de l'équipe où se trouve sont inventaire personnel. On peut manipuler des curseurs pour naviguer dans l'inventaire et déplacer des objets.

3. Base de données des objets ('bddItem.scala') Je rassemble ici les fonctions pour créer une instance de chaque objet lorsque celui-ci est reçu comme récompense dans la phase de loot. Par rapport au dernier rendu, On a maintenant des objets à usage unique pour recharger les armes du type correspondant ou pour se soigner. Les armes et armures donnes des aptitudes en plus de boost et de réduction de dégâts.

4. Phase de récompense et gestion des sprites ('Loot.scala' et 'Mechanisms.scala') J'ai mis en place un moyen d'activer les mécanismes qui correspondent à l'équipement porté par l'équipe et de rajouter les objets directement dans l'inventaire général de l'équipe. De plus, j'ai converti mon code de génération des mécanismes de la dernière version pour renvoyer une liste de Layers. Cela permet lors de la phase de récompenses de ne mettre à jour que les sprites des mécanismes et donc de fluidifier le jeu.

5. MàJ de la génération aléatoire de niveau ('Schematics.scala') J'ai modifié la classe Tile pour y incorporer la cellule d'eau. Pour chaque couloir créé entre le 'grand cadre' de la pièce et les cadres intérieurs, il y a une chance sur deux que le sol soit recouvert d'eau qui interagit avec les compétences élémentaires des alliés et ennemis. Cela a augmenté le nombre de configurations possibles de pièce (sans les mécanismes) à plus de 4 milliards.