Rapport Unity : Xamaton

Description du jeu

Xamaton est un jeu de combat au tour par tour à la 3ème personne. Votre héros se retrouve enfermé dans un donjon hostile, infesté de monstres impitoyables. A chaque étage, de nouvelles créatures de plus en plus puissantes vous agressent. Mais jusqu'où aurez-vous le courage d'aller ?

Comment jouer ?

Le personnage apparaît dans une salle, il est possible de se déplacer en cliquant sur les cellules libres. L'inventaire du joueur est disponible à gauche dans le GUI, chaque touche du clavier est définie dans l'inventaire. Lorsque vous sélectionnez un item avec sa touche correspondante, les cellules ciblables deviennent bleues. Le personnage peut se déplacer sur les items au sol, ce qui permet d'ajouter ces derniers dans son inventaire.

Les feux follets présents dans les salles sont des ennemis, il est possible de les attaquer en sélectionnant par exemple l'épée (touche A) pour une attaque au corps à corps ou les flèches pour une attaque à distance (touche Z).

Le changement de salle, se fait en passant dans les couloirs Nord, Sud, Ouest et Est. Le but du jeu est de trouver l'escalier dans une des salles de l'étage. Il vous téléporte dans l'étage suivant. Vous améliorez votre score en passant d'étage en étage en trouvant à chaque fois les escaliers. Attention, plus vous franchirez d'étages et plus les ennemis seront fort et nombreux. Et de même, le joueur monte de niveau en tuant des ennemis, il vaut mieux en tuer au fur et à mesure que d'arriver à l'étage 10 et ne pas être à la hauteur.

Mécaniques de jeu

- **Système d'étage**

Chaque étage est généré de manière aléatoire et se compose d'un nombre de map aléatoire. Le *FloorManager* gère la création de l'étage et les spawners d'items et monstres.

- **Entités**

Le jeu possède des entités : le Joueur et les Monstres. Ils ont un système de leveling, de vie et d'inventaire. Lorsque le joueur tue un monstre, il obtient de l'XP pour monter de niveau. Le monstre spawn avec un niveau initial en fonction de l'étage actuel du donjon.

- **Système de tour par tour**

Nous avons une classe *ActionManager* (Singleton) qui gère la liste des entités (mobs et joueur). Chaque action passe la main à l'entité suivante dans la liste. Exemple d'actions possibles : Attaque, Déplacement, Heal.

- **Système d'actions de combat**

Nous avons une classe abstraite Action, que l'on peut exécuter sur une case donnée. Ces actions sont aussi bien géré par le joueur que par les mobs. Le déplacement, les attaques, sont des actions.

- **Système de déplacement**

Nous avons utilisé l'algorithme A* pour déplacer une entité sur la carte. Lorsque l'on clique sur une cellule, le système va générer plusieurs actions de déplacement jusqu'à atteindre la case sélectionnée. Les monstres suivent simplement le joueur.

- **Système d'item**

Chaque item est lié à une *Action*. Par exemple : la potion de heal, est lié à *HealthAction*, qui définit l'action de santé, des soins. Un item a des caractéristiques, s'il est consommable, droppable, sa rareté (drop rate). Les entités possèdent un inventaire qu'ils utilisent au moyen de "key", représentant une touche du clavier. Si il utilise un item, son Action sera exécuté, de même que si l'item est consommable, sa quantité dans l'inventaire sera décrémentée.

Dans le cas du joueur, son inventaire est affiché à gauche dans le GUI, et chaque item est

- **Système de spawn de mob**

Pour chaque étage, Il y a un nombre de mob initial et un nombre de mob restant à spawner. Ce système à été mis en place afin que le joueur ne soit pas inondé en entrant dans une salle, mais garde un challenge.

- **Système de spawn d'items**

Pour chaque étage, il y a un nombre minimum et maximum d'items généré au sol. Les entités (monstres et joueur) peuvent les ramasser et les utiliser. Si une entité meurt, le spawner d'items se charge de vider l'inventaire de l'entité au sol avec les items droppables.

- **Système de changement de map**

Une map possède une liste de téléporteurs chargeant une nouvelle map lorsque le joueur passe dessus.

- **Système de score**

Le score du joueur est représenté par le nombre d'étage parcouru.

- **Système de MeshMap**

Chaque cellule possède un contenu (type *Placable*) et un item (type *ItemObject*) et connaît ses voisines.

Organisation

Maxime	Antonin
Algorithme A* et zone d'attaque : 4h Système de combat/sort : 6h30 Système tour par tour : 12h Système d'entités : 6h Système de spawn de mob : 6h Etage procédural : 9h Système d'item : 10h Système de Spawners : 6h	Système de cases : 8h Système de déplacement : 4h Système de changement de Map : 8h Système de chargement de Map : 8h GUI : 4h Système de difficulté par étage : 4h Diversité de monstres : 4h

Outils

Unity, Monodevelop, Github : https://github.com/antoriche/Xamaton

Le code source, ainsi qu'une version Build, est à disposition sur le Github.

Conclusion

A l'origine, nous étions partis sur un jeu de type "Dofus". Un mode exploration et un système de combat tour par tour, avec des Point d'Action et des Point de Mouvement. Notre vision du projet a évolué au fil du temps, et nous nous sommes plutôt tournés vers un "Pokémon donjon mystère". C'est-à-dire que le mode combat et exploration sont fusionnés, et les entités n'ont droit qu'à une action par tour, attaquer ou se déplacer. Nous avons également dû donner un but à notre jeu, par exemple, aller le plus loin possible dans les étages du donjon.

Problèmes rencontrés

L'un des problèmes rencontrés a été l'erreur de partir avec des composants 3D pour faire un jeu 2D ce qui a posé des problèmes de rendu.

Lors du Build, Unity ne trouve plus le namespace "UnityEditor", nous avons dû commenter toutes les parties du code où UnityEditor était utilisé. De plus, nous avons eu un problème avec l'affichage des Lifebars dans le canvas, celles-ci fonctionnaient dans Unity même, mais pas dans le jeu Build, un "Reimport All" à régler le problème.

Modifications et améliorations

- **Utilisation de sprites pour les cellules** : nous sommes partis dès le départ sur du simple Mesh Renderer, au lieu de Sprite Renderer.
- **Optimisation des spawners**, par exemple faire en sorte d'avoir un prefab de monstre et d'appliquer un Scriptable Object sur ce prefab définissant les caractéristiques de ce monstre. De la même façon que les items, il existe un prefab et c'est le ScriptableObject Item qui défini ses caractéristiques.
- **Ajout d'un spawner de coffres** : nous voulions ajouter un 3ème spawner qui s'occuperait de l'apparition des coffres sur l'étage et ajouter une action "OpenChestAction" appliquable sur un nouvel item "Clé". Par exemple, les boss auraient droppé des clés.
- **Amélioration de l'écran de Game Over et ajouter un écran d'accueil**.
- **Réalisation de nouveaux sorts** : sort d'attaque en zone, heal en % de vie.

Annexes

Asset utilisé pour les imagess : https://www.assetstore.unity3d.com/en/#!/content/19553