Rapport Projet Programmation 2

Antoine Grimod, Bastien Lhopitallier, Paul Robert

6 avril 2021

Changements dans le code de la partie 1

- C'est maintenant au joueur de choisir lorsqu'il termine son tour (sur pression de la touche `Entrée`), permettant l'utilisation d'objets après avoir attaqué (attaquer consomme tous les points d'action du joueur et le tour se terminait donc automatiquement).
- Les cases touchées par une attaque sont maintenant surlignées en orange afin de les distinguer des cases de déplacement.
- L'affichage des commandes peut maintenant être caché ou affiché au choix du joueur.
- Les sorts nécessitent maintenant un coût en points de vie pour être lancés.
- Le joueur a maintenant une capacité de port limitée.

Ajouts pour la partie 2

Fichiers JSON

Les items et les ennemis sont maintenant définis dans des fichiers JSON extérieurs au code. Cela permet de pallier au problème des entités nécessitant beaucoup de valeurs numériques à la création (obligeant pour cela de devoir regarder la définition de la classe pour savoir ce qu'elles représentent).

En effet, dans les fichiers JSON, les noms des variables apparaissent aussi, ainsi il est bien plus facile de s'y retrouver. De plus, toutes les valeurs sont maintenant optionelles car des valeurs par défaut sont définies (afin d'éviter d'éventuelles erreurs dues à des définitions de valeurs manquantes).

Cependant, la mise en place du JSON a été compliquée, en effet, Scala ne dispose pas dans la bibliothèque standard d'un module pour gérer le JSON. Notre choix s'est donc porté sur μpickle (ou simplement upickle), mais le manque de documentation a fortement ralenti notre travail et tout le code pour la création de Weapon est calqué sur un unique exemple de la documentation. Qui plus est, le manque d'information sur la bibliothèque nous a aussi contraint à implémenter certaines fonctions qui sont probablement déjà présentes dans la bibliothèque, comme par exemple une fonction pour obtenir la longueur d'un tableau JSON, ou pour savoir si le code JSON contient la définition d'un certain champ.

Salles

Après le premier rendu, nous avons changé de structure pour nos tuiles. Nous avions un `vector` à 2 dimensions et sommes passés à un `Map[(Int, Int), Tile]` afin de profiter de la méthode `foreach` très pratique.

La classe `Room` a ensuite été crée afin de représenter les salles. Ainsi, chaque instance de cette classe possède son propre ensemble de tuiles et par conséquent l'objet `Map` contenant toutes les tuiles

a été supprimé.

De plus, chaque porte possède maintenant une condition d'ouverture parmi :

- Avoir tué tous les ennemis de la salle
- Dévérouiller la porte à l'aide d'une clé laissée tombée par un ennemi à sa mort
- Placer un ou plusieurs objet(s) sur un ou plusieurs réceptacle(s)

Condition de victoire

Pour gagner le jeu, tous les objectifs de salle doivent être atteints.

Nouveaux objets

Le jeu inclut maintenant :

- Des parchemins, permettant de lancer un sort en les consommant (usage unique)
- Des grimoires, permettant de lancer un sort sans les consommer. En revanche, tandis que tout être peut utiliser un parchemin, chaque grimoire possède une valeur seuil de pouvoir `Pow` : il peut être utilisé uniquement si la caractéristique de pouvoir du joueur est au moins égale à celle requise par celui-ci.
- Des armures, permettant d'augmenter la valeur d'armure (`ArmorClass`) du joueur. Il en existe 4 types : tête, corps, jambes et pieds. Il n'est possible d'équiper qu'une seule pièce de chaque type.

Nouveaux types d'entités

L'ennemi fuyard : il s'agit d'un type d'ennemi qui, à chaque tour de jeu, n'attaque pas et se déplace de manière à maximiser la distance qui le sépare du joueur.

L'ennemi neutre : c'est une entité qui, initialement, ne va ni attaquer le joueur, ni se déplacer. En revanche, si le joueur se montre hostile envers elle, elle se mettra à agir comme un ennemi normal, c'est-à-dire se déplacer pour se mettre à portée du joueur et l'attaquer dès que possible (même si ces entités sont neutres de base, elles possèdent quand même une arme).

Le marchand : il hérite de la classe des ennemis neutres, il est possible de lui acheter et de lui vendre des objets. Comme un ennemi neutre, il se mettra à attaquer si le joueur se montre hostile.

L'ennemi volant : inutilisés pour le moment, les ennemis volants seront capables de se déplacer sur des tuiles inaccessibles au joueur (exemple : des tuiles représentant un trou dans le sol)

Effets de statut

Les armes peuvent maintenant enflammer, empoisonner, geler et paralyser leur cible. Voici les effets :

- Le feu et le poison font des dégâts à la fin de chaque tour.
- Le gel réduit les points d'actions de l'entité touchée de moitié.
- La paralysie les réduit à 0.

Cependant, le système n'est pas encore opérationnel à cause d'un bug que nous n'avons pas encore identifié.

Pour la suite

Système de sauvegarde

Grâce au JSON, il sera facile d'implémenter un système de sauvegarde (salle par salle puis tuile par tuile avec ce qu'il y a dessus) en enregistrant des fichiers JSON au même format que celui utilisé pour définir les objets.

Génération aléatoire

Pour l'instant, les positions des salles sont prédéfinies et elles sont toutes carrées (apparaissant en forme de parallélogramme grâce au système hexagonal). Nous avons prévu de générer des cartes aléatoirement à l'aide de fonctions renvoyant les coordonnées d'une salle de forme géométrique choisie (triangle, anneau, rectangle, hexagone, ...) puis de relier ces salles entre elles à l'aide de couloirs plus ou moins larges.