Projet Programmation 2 : Rapport Partie 2

MANGEL Léo et MICHEL Thomas

1 Fonctionnalités ajoutés

1.1 Personnages et entités

1.1.1 Nouveaux traits

- Le trait `InteractWithItems` : Si le joueur interagit avec une entité ayant ce trait, la méthode itemInteraction est appelée avec en paramètre l'objet actuellement sélectionné par le joueur. Cela permet d'intégrer de nouvelles interactions en fonction des objets possédés par le joueur.
- Le trait `Hackable` : Dérive de `InteractWithItems` et ajoute un appel à la méthode hack lorsque l'objet sélectionné est "Hacking Tools".
- Le trait `FleeingWhenDamaged` : Dérive de `AIControlled`. Si le personnage reçoit des dommages, il se met à fuire. Pour déterminer vers quelle case le personnage fuit, on utilise le même algorithme que pour calculer la première case du plus court chemin entre le joueur et le personnage, mais on prend la case opposée à la place. Si la case opposée est occupée, par un mur ou un autre personnage, on regarde les cases adjacentes jusqu'à en trouver une libre.
- Le trait `RangedEnemy` : Implémente un nouveau comportement pour les ennemis. Le personnage ne se déplace pas et attaque lorsque le joueur est à portée de tir. Pour savoir s'il est possible de toucher le joueur (si la ligne de visée n'est pas obstruée), nous avons choisi de nous basé sur l'algorithme calculant les cases visibles par le joueur implémenté durant la partie 1 du projet. Nous faisons l'hypothèse que si le joueur peut voir un ennemi alors celui-ci peut aussi voir le joueur. Nous nous étions en effet efforcé de rendre l'algorithme symétrique lorsque nous l'avions implémenté. De cette façon, il n'est pas nécessaire de calculer les cases visibles par chaque ennemi mais seulement celles visibles par le joueur puis d'autoriser les tirs ennemis si ceux-ci sont visibles sur la carte et la distance séparant le joueur de l'ennemi est inférieures à la portée de ce dernier.

1.1.2 Système de statuts

Les personnages possèdent maintenant une liste de statuts actifs. Les statuts sont des classe héritant de la classe de base `Statut` qui gère la durée restante du statut. Si la durée restante est nulle alors le statut est supprimé. Au début du tour de chaque personnage, la méthode `updateStatus` rassemble les effets des différents statuts dans une instance de la classe `StatusResult`. Cet objet va être conservé durant le tour et va influencer les actions que le personnage pourra entreprendre.

Il y a pour l'instant trois statuts implémentés : l'étourdissement empêche le personnage de bouger durant ce tour, le saignement lui fait perdre des points de vie et la régénération lui rend des points de vie.

1.1.3 Nouvelles entités et nouveaux personnages

- *Boutique* : Afin de diversifier les personnages non-joueurs, nous avons décidé de mettre en place des boutiques accessibles par l'intermédiaire de marchands ainsi qu'une monnaie permettant d'acheter des objets dans les boutiques. La classe `Shopkeeper` dérive de la classe `Character` et possède le trait `FleeingWhenDamaged`. L'interaction par défaut est remplacée par l'ouverture d'un menu (classe `ShopMenu`) affichant les objets disponibles à la vente et se mettant à jour à chaque nouvel achat du joueur. De plus, bien qu'il ne soit pas possible d'attaquer le marchand par une attaque au corps à corps, il est tout de même possible de lui infliger des dégâts auquel cas celui-ci se met à fuir le joueur.
- *Ordinateur* : Entité héritant du trait `Hackable`. Cette entité révèle la carte du niveau actuel lorsqu'il est piraté par le joueur.
- *Cadenas* : Entité héritant du trait `Hackable`. Cette entité bloque l'accès à l'ascenseur permettant de passer d'un niveau à l'autre.
- *Tourelle* : Ennemi héritant du trait `RangedEnemy` qui tire sur le joueur dès qu'il est à portée.

1.2 Objets et équipement

1.2.1 Nouveaux traits

— Le trait `Upgradable` : Attribut un niveau à chaque objet possédant ce trait. La méthode upgrade qui peut être redéfinit par chaque classe héritant du trait. Un annotation est ajoutée à côté du nom d'un objet amélioré afin d'indiquer son niveau.
— Le trait `RangedWeapon` : Ce trait définit la méthode shoot qui permet d'attaquer des ennemis à distance ainsi qu'une nouvelle statistique d'attaque spécifique aux armes à distance influençant les dégâts infligés aux adversaires.
— Le trait `Throwable` : Trait modifié par rapport à la partie 1 de façon à utiliser le curseur. Il s'agit le trait de base pour les objets pouvant être lancés.
— Le trait `ThrowableWithAoE` : Redéfinit la méthode `throwItem` du trait `Throwable` afin d'appliquer l'effet de l'objet sur chacune des cases d'une certaine zone.

1.2.2 Nouveaux objets

— *Tournevis* : Objet pouvant être utilisé pour démonter une pièce d'équipement améliorable. Ouvre un menu affichant les objets que l'on peut démonter. Chaque objet démonté permet d'obtenir un objet *Composants électroniques*
— *Composants électroniques* : Objet consommable permettant d'améliorer les objets qui peuvent l'être. Ouvre un menu permettant de sélectionner l'objet à améliorer.
— *Argent* : Lorsque cet objet est ramassé par le joueur. Le montant d'argent représenté par l'objet est directement ajouté à la statistique *Argent* du joueur. Une quantité aléatoire de cet objet apparaît à la mort d'un ennemi.
— *Outils de piratage* : Objet consommé lors de l'interaction entre le joueur et une entité héritant du trait `Hackable`. L'effet de l'interaction dépend de l'entité avec laquelle le joueur interagit.
— *Bandage* : Consommable soignant le statut *Saignement*.
— *Seringues* : Les seringues héritent d'une classe de base `Syringe`. Une seringue pleine est un objet consommable se transformant en seringue vide lors de la consommation. Les seringues héritent du trait `Throwable` et infligent un statut d'étourdissement si elles sont lancées sur un personnage. La seringue de morphine permet de récupérer quelques points de vie.
— *Yeux laser, bras canon* : Armes à distance s'équipant sur différentes parties du corps.
— *Chapeau de Cowboy, veste épaisse, casque* : Classes héritant de la classe Armor, équipables sur différentes parties du corps et avec différentes statistiques.
— *Point américain* : Arme au corps à corps héritant du trait `Weapon`.
— *Grenade, Grenade EMP* : Armes de lancé appliquant un effet sur une zone héritant de `ThrowableWithAoE`. La première fait des dégats et la seconde étourdit les personnages.

1.3 Menus

Nous avons ajouté une nouvelle classe de base pour les menus ainsi qu'une façon de les afficher grâce à l'objet `SplashScreenRenderer`. Cette classe définit les différentes méthodes permettant la navigation dans le menu et le joueur peut interagir avec au moyen de l'objet `UI`. Les menus sont gérés au moyen d'une pile et seul le menu au dessus de la pile reçoit des commandes. La création de menus est ainsi facilitée, il suffit de remplir la liste des options sélectionnables du menu et de redéfinir la méthode `confirm` qui est appelée lorsque le joueur valide son choix. Certains menus particuliers dérivent de la classe de base tels que le menu principal ou encore le menu de la boutique.

1.4 Curseur, armes à distance et objets lançable

Afin de pouvoir viser de manière précise, nous avons ajouté un curseur. Il devient visible et controlâble par le joueur lorsqu'on tire avec une arme ou lorsqu'on lance un objet.
Pour cela, nous avons ajouté des modes de contrôle inspiré par l'éditeur de texte vim. Il y a pour l'instant 5 modes, dont 3 qui utilisent le curseur. Il s'agit des modes `Cursor`, `Throw` et `Fire`.
Le mode `Cursor` n'est pour l'instant pas utile et permet simplement de déplacer le curseur. Nous n'avons pas eu le temps de nous concentrer sur ce mode, mais l'objectif de ce mode serait de permettre d'inspecter les cases autour du joueur et d'afficher une description des ennemis.
Le mode `Throw` permet de lancer certains objets. Pour entrer dans ce mode, il faut sélectionner l'objet à lancer et appuyer sur T. Ensuite, on peut déplacer le curseur vers la case sur lequel on veut envoyer l'objet, puis appuyer à nouveau sur T. On ne peut pas lancer d'objet sur une case qu'on ne voit pas, ni sur un mur. Certains objets sont consommés lorsqu'ils sont lancés, comme les grenades.
Le mode `Fire` fonctionne de la même manière que le mode `Throw` mais permet lui de tirer avec des armes à

distance. Pour y entrer, il faut avoir une arme à distance équipée et appuyer sur F. Si on a plusieurs armes à feu équipées, on peut soit sélectionner une de ses armes pour tirer avec, soit tirer avec l'arme principale, qui est la première arme à distance équipée, lorsque l'objet selectionné n'est pas une arme à distance équipée. Les restrictions imposées aux objets à lancer s'appliquent également aux armes à distance. À ces restrictions s'ajoute la condition qu'il n'y ai pas un autre personnage entre le joueur et la case visée. Pour visualiser ces conditions, on affiche un halo jaune sur les cases entre le joueur et la case visée. Pour calculer les cases concernées, on a préféré utilisé un algorithme naïf plutôt qu'un algorithme plus optimisé comme celui de Bresenham. En effet, vu qu'on ne calcule les cases qu'un seul segment par tour, la complexité plus élevée ne se fait pas ressentir.

1.5 Sauvegarde

Pour sauvegarder les informations du jeu en cours, nous utilisons la sérialisation native de Scala. Cela nous permet de rendre chaque classe sauvegardable dans un fichier en ajoutant le trait `Serializable`. Ainsi, lorsqu'on sauvegarde, on enregistre dans un fichier un objet de la classe `Game` qui contient toutes les informations sur la partie en cours. De même, lorsqu'on charge, on désérialise le contenu du fichier pour obtenir l'objet de la classe `Game` initial.
Pour sauvegarder et charger une partie, on utilise un menu principal, auquel on a accès en début de jeu, ainsi qu'à n'importe quel moment de la partie en appuyant sur Echap. Nous avons choisi d'avoir une seule sauvegarde possible, afin de ne pas avoir besoin d'implémenter un système pour sélectionner une sauvegarde.

1.6 Niveaux et objectifs

Pour changer de niveau, il faut trouver un ascenseur descendant. Il y en a un par niveau. Au dessus de l'ascenseur, il y a un cadenas qui empêche d'accéder à l'ascenseur. Pour supprimer le cadenas, le joueur a deux choix. Soit il utilise un outil de hacking pour déverouiller le cadenas, soit il utilise une arme à distance pour détruire le cadenas. S'il détruit le cadenas, il abîme aussi l'ascenseur. Cela se traduit en jeu par le fait qu'après avoir utilisé l'ascenseur abimé pour descendre d'un étage, il ne pourra plus l'utiliser pour monter.
Afin d'empêcher le joueur d'être bloqué, il commence avec un "Arm-cannon" qui est une arme à distance, ce qui lui permettra de détruire le cadenas.
Pour la suite, nous aimerions augmenter la difficulté progressivement lorsqu'on descend. Cela se fera probablement en augmentant les statistiques des ennemis.

2 Difficultés rencontrées

Malgré sa relative simplicité de mise en place dans le code, nous avons eu des difficultés à faire fonctionner correctement la sérialisation. En effet, sbt empêche initialement au programme de faire des forks qui sont utilisés par Scala pour la désérialisation. Pour régler le problème, il faut changer le fichier `build.sbt`. Nous avons pensé un peu de temps à comprendre d'ou venait le problème et à le résoudre.