Devoir Maison d'Architecture Logicielle

Sarah Parant Raphael Mora
Guillaume Guerin de Tourville

January 31, 2014

1 Histoire

Il était une fois dans un pays très très lointain deux châteaux sur une île. L'île avait été séparée en deux en réponse à la jalousie du frère du roi. Des brigands et des bandits ont été engagés pour piller et détruire le château du roi.

Aujourd'hui, votre heure a sonné ! Vous avez été appelé par le roi pour guider vos hommes vers la victoire. Vous hissez votre drapeau sur lequel apparaît un lapin, l'emblème du roi. Levez vos armes et partez dans la vallée, partez vers votre destinée !

2 Fonctionnalités

2.1 Carte

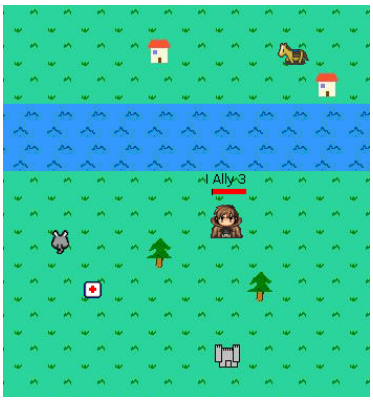
Les éléments de la carte sont générés aléatoirement sauf les deux châteaux. Les personnages ont une position aléatoire, mais nous vérifions que le personnage ne se retrouve pas sur une case bloquée à l'initialisation.

2.2 Entité

Il existe deux types d'entités dans notre jeu : les soldats et les objets.

2.3 Objet

Dans les objets, il existe des animaux, des éléments du décor et des packs de soins.
Les packs de soins et les chevaux peuvent être récupérés par les soldats. Les lapins, les arbres et les maisons sont des éléments décorateurs.



2.4 Curseur souris

Pour gérer le déplacement à la souris une classe MouseCursor à été créee. Elle hérite de GameMovable. Le principe auquel nous avons pensé est de se servir des classes déjà existantes et donc pour gérer le curseur nous allons créer un objet qui se déplace lorsque l'on clique avec la souris. L'utilisation de fonctions de GameOverlapRules va nous permettre de savoir avec quel objet l'on intéragit.

2.5 Soldats

Les soldats ont des factions. Dans notre niveau, il n'y a que deux types d'équipe. Nous avons mis une troisième équipe constituée de princesse (il faut mettre un nombre de princesses supérieur à 0) pour montrer que les IA peuvent s'attaquer entre elles si elles ne font pas partie de la même équipe.

Il existe deux types de personnages, les hommes d'infanterie et les cavaliers. La distinction des types de soldat se fait avec leurs noms.

Les soldats alliés peuvent être sélectionné/désélectionné avec un clic gauche de la souris puis se déplacent avec un clic droit de la souris.

Les soldats ennemis se déplacent aléatoirement en faisant des pauses. Ce déplacement a aussi été assigné aux lapins et aux chevaux.

Les soldats combattent lorsqu'ils sont en contact avec un soldat ennemi. Un soldat allié identique à un soldat ennemi gagne le combat, car ils tapent en premier.

Suite à une victoire, les soldats peuvent recevoir des récompenses. Il existe un pourcentage de chance pour qu'un soldat victorieux gagne une épée, un bouclier ou les deux. Les soldats qui possèdent des équipements ont une icône de leurs équipements sur eux.

Après un combat, un pack de soins peut apparaitre sur la carte.
Les soldats ennemis peuvent être équipés ou non au début de la partie.

Les soldats alliés sont dans le tiers inférieur de la carte à côté du château du roi. Les soldats ennemis sont dans la moitié supérieure de la carte à côté du château du frère du roi.

3 Architecture du jeu

Dans un premier temps, les différentes variables, par exemple la taille de la carte du jeu a été rassemblée au sein d'un même fichier de configuration GameConfig.java afin de pouvoir les modifier facilement.

Dans le package game, une classe GameMap.java a été créée pour les fonctions concernant la carte du jeu. Dans cette classe, nous avons la génération de la carte.
Dans la classe GameDefaultImpl, nous avons modifié les observers de fin des parties.

Actuellement, nous avons 2 possibilités de fin de partie : lorsque tous les ennemis sont morts ou lorsque tous les alliés sont morts.
Dans la version de Pacman, nous avions un observer qui envoie la fin de partie et un observer qui vérifie s'il reste de la vie. Nos conditions de fin de partie sont plus proches de la fin de partie de Pacman avec l'observer sur la vie. Nous avons donc rajouté un observer sur le nombre d'alliés en jeu et un observer sur le nombre d'ennemis en jeu. Ainsi, avec l'observer, nous avons les informations sur le nombre de personnages en jeu. Nous avons laissé l'observer endOfGame bien que nous ne l'utilisons pas, car nous avons pensé à une règle de jeu qui nous demanderez à appeler la fonction notify() de l'observer endOfGame pour finir le jeu. La règle en question est que le joueur doit faire attention que les

soldats ennemis n'attaquent pas le château. Si le château était touché, la partie serait finie.
L'observer des scores aussi est resté si l'on voulait rajouter du score. Actuellement, notre jeu faisant qu'un seul niveau, nous n'avons pas pensé utile de comptabilisé celui-ci.

3.1 Architecture des soldats

Lors de l'insertion de notre architecture avec l'architecture du framework de Pacman, nous avions deux choix :

- utiliser une nouvelle classe dans les entités qui contiendrait par délégation un soldier.
- accrocher notre framework sur notre design pattern composite des armées.

Nous avons choisi d'implémenter la deuxième solution.
Notre première implémentation a été la première solution, car elle était plus rapide à implémenter ce qui nous a permis de tester nos personnages dans le jeu rapidement. Par la suite, par manque de manœuvre dans notre architecture, nous avons implémenté la deuxième.

L'observer des ArmedUnit n'étant pas utilisé dans notre gameplay, nous avons fait le choix de supprimer la relation et ainsi nous avons pu extends GameMovable notre ArmedUnitSoldier.

Nous avons voulu montrer que le choix d'implémentation dans un logiciel est important. Nous avons choisi de faire le scénario d'un homme d'infanterie qui devient un cavalier lorsqu'il rencontre un cheval. Dans l'implémentation choisie en cours, les modifications des soldats ne sont pas prises en compte.
Un soldat est crée, décoré facultativement et est détruit. Le soldat en lui-même ne peut pas être modifié.
Nous avons donc créé un nouveau soldat qui est un cavalier qui récupère toutes les informations de notre homme d'infanterie puis nous détruisons l'homme d'infanterie.
La modification n'ayant pas été prise en compte dans notre implémentation a des conséquences sur la complexité de notre fonction.

4 Difficultés rencontrées

Nous avons rencontré des difficultés au cours de la sélection des personnages.

Le déplacement des unités se fait par des vecteurs. Sans modifier le framework, nous n'avons pas trouvé de solution pour que notre personnage s'arrête lorsqu'il est au point du clic droit de la souris. Nous avons donc choisi que les personnages sélectionnés doivent se diriger vers la coordonnée de notre souris lorsque le bouton droit est enfoncé. Lorsque le bouton est relâché, les personnages sélectionnés s'arrêtent.
Nous avions pensé à une solution pour que les personnages avancent et s'arrêtent à la dernière coordonnée du clic souris droit. Pour cela, nous aurions un observer

dédié à la vérification des coordonnés des personnages pour l'arrêter lorsqu'il est arrivé à destination. Cette implémentation est une des améliorations possibles.

Dans notre cahier des charges, les personnages ne doivent être bloquants. Nous n'avons pas réussi à rendre les personnages bloquants. Un booléen isFighting avait été rajouté dans la classe ArmedUnitSoldier et mis à true lorsque la fonction d'overlap entre 2 soldats d'une équipe différente avait été appelée. Lorsque cela se produisait on mettait également la direction du SpeedVector à (0,0) et reprennait un comportement normal quand l'un des deux était mort (passage de la variable isFighting à false) ce qui fonctionnait mais dans un cas particulier, si l'on était entrain de se déplacer il fallait relacher le bouton. Si l'on restait appuyé sur le clic droit de la souris le personnage continuait de se déplacer et se bloquait une fois relaché et si il n'était plus a portée du Soldat qu'il attaque les deux restaient bloqués dans l'état de combat sans pouvoir en sortir (La fonction d'overlap n'était plus appelée, et pas de perte de vie).

Suite à notre difficulté sur les unités bloquantes, nous n'avons pas pu implémenter correctement les combat. Les combats devaient se déroulés avec des armées. Chaque personnage de la même équipe qui est côte à côte forme une armée. Si un personnage récupère un objet, l'armée de celui-ci récupère l'objet.

Une autre difficulté rencontrée par le groupe est que les combats s'effectuent en même temps via GameOverlapRules ce qui a pour conséquence si plusieurs Soldats alliés attaquent un enemi et que ce dernier meurt il sera compté comme tué plusieurs fois par l'Observer et entraine une victoire plus rapidement que prévue.

5 Fonctionnalités futures

Certaines fonctionnalités n'ont pas été implémentées. Par exemple, nous voulions faire des armées. Lorsque des personnages de même équipe se touchent, ils sont considérés comme faisant partie de la même armée. Ainsi, nous aurions utilisé les fonctions du composite tout en rajoutant de la stratégie dans notre jeu. Le but apporté pour le joueur est de prendre en compte qu'un groupe d'ennemi est plus difficile à tuer qu'un personnage solitaire.