Projet Fire Emblem

Thanusan SATHIAKUMAR – Léo CHAZALLET

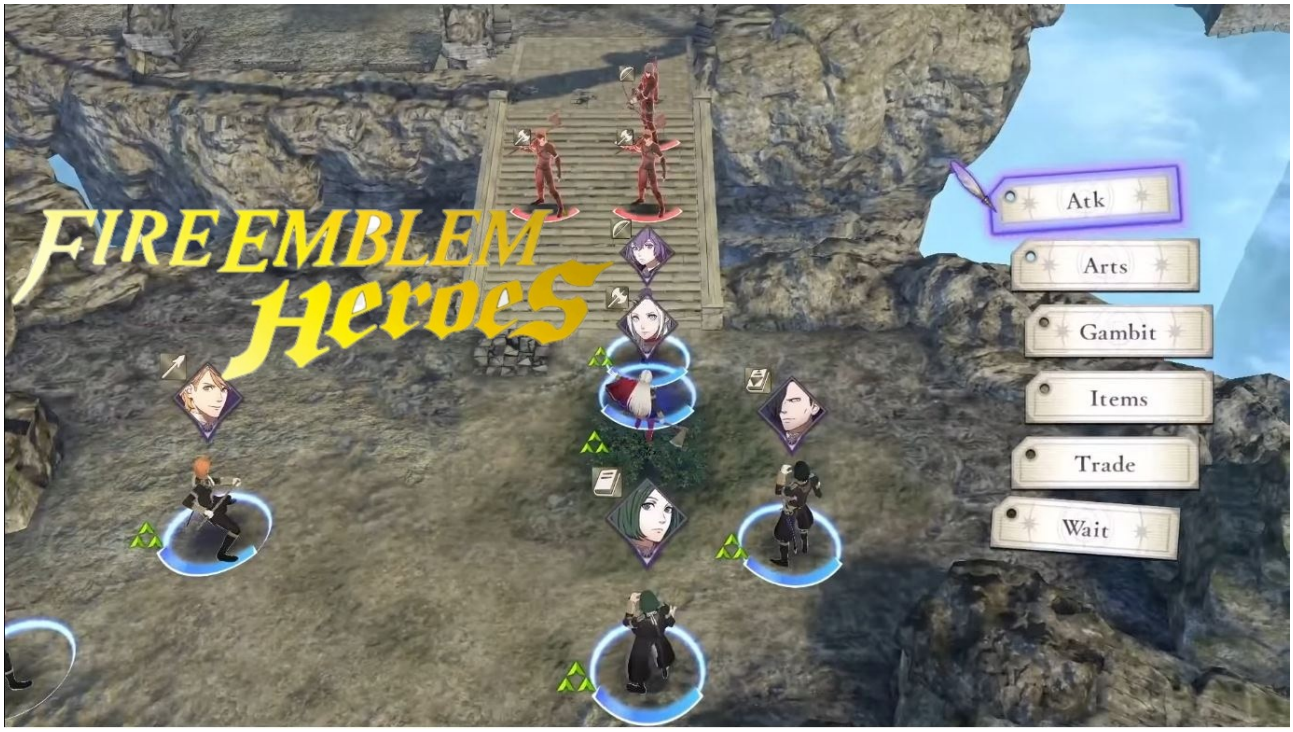

Table des matières

1 Objectif

1.1 Présentation générale

Le jeu va être un jeu de combat stratégique en tour par tour. Le principe se base sur la partie combat de la série de jeux Fire Emblem.

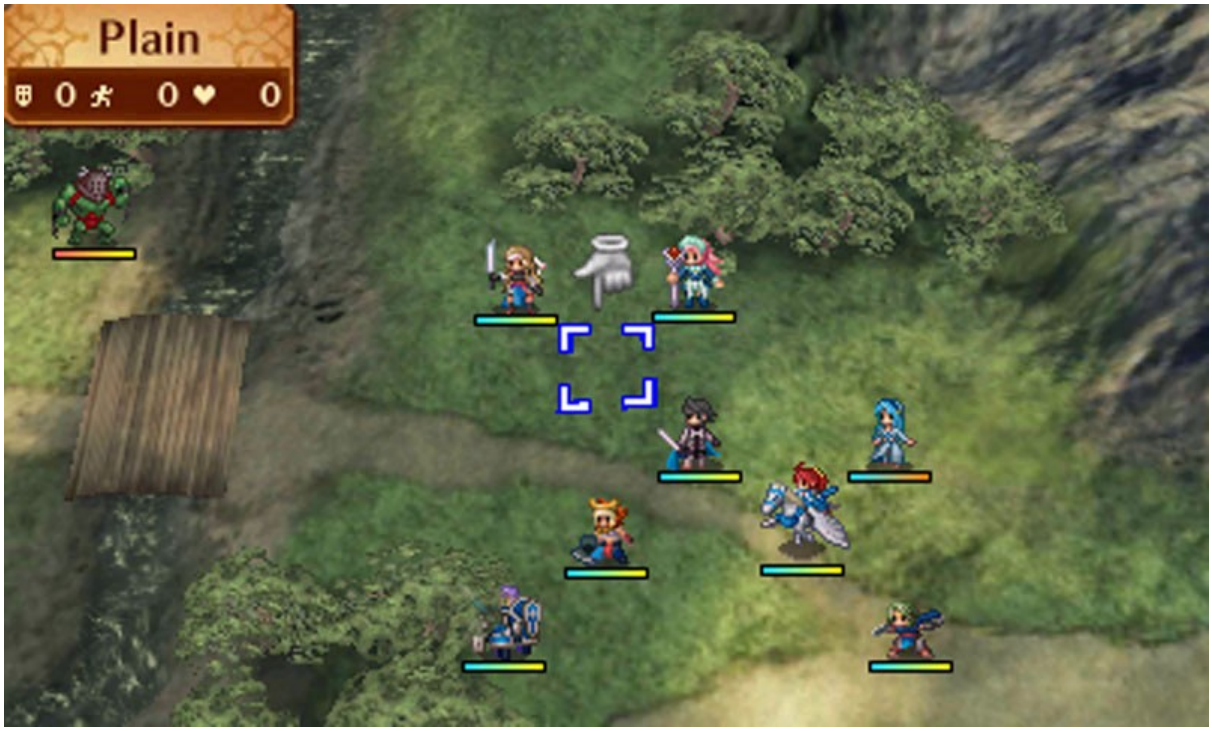

FIGURE 1 - Image tirée du jeu Fire Emblem Fates : Birthright

Le joueur possède une ou plusieurs unités qui sont placées sur un espace quadrillé. Chacune des unités aura ses propres caractéristiques (points de vie, point de mana, sorts, portée de déplacement, portée d'attaque, valeur d'attaque, valeur de défense). Des ennemis sont présents sur chaque carte/niveau.possédant eux aussi des caractéristiques spécifiques. Le but va donc d'éliminer tous les ennemis présent sur la carte pour obtenir une victoire. A chaque tour le joueur va pouvoir décider pour une unité une action (se déplacer, attaquer, utiliser un sort).

1.2 Règles du jeu

L'utilisateur jouera contre des unités ennemis contrôlées par une IA par défault. Il y aura cependant la possiblité de passer en mode 2 joueurs par la suite.

Voici une liste des règles principales du jeu :

- le/les unité/s du joueur apparaissent du côté opposé de la carte à celle des unités ennemies
- le joueur ne peut effectuer qu'une seule action par unité par tour
- une action peut être soit un déplacement, une attaque ou l'utilisation d'un sort/objet
- la partie se termine quand tous les ennemis sont éliminés ou lorsque le joueur n'a plus d'unité

Le jeu est constitué d'un seul niveau pour l'instant. Il y aura une unité alliée (un héros) et une unité ennemie (un monstre). Ces unités auront des caractéristiques basiques et communes. Chacune aura des points de vie, une capacité de déplacement, une portée d'attaque, une valeur d'attaque et une valeur de défense.
Au fur et à mesure des niveaux, le nombre d'unités ainsi que la difficulté va augmenter. D'autres unités seront aussi disponible comme des mages qui pourront avoir en plus différents sorts disponibles, auront des points de mana et qui pourront attaquer à distance. Chaque niveau aura une carte dédiée qui aura une taille adaptée au nombre d'unités.

S'il reste du temps, voici les fonctionnalités que nous aimerions implémenter :

- présence sur le terrain d'éléments apportant des bonus (points de vie, attaque ou défense) répartis aléatoirement sur la grille du jeu. Le joueur doit choisir de manière stratégique les éléments dont il aura besoin pour vaincre l'ennemi
- présence sur le terrain d'éléments apportant des malus au joueur (par exemple des trous à éviter pour ne pas perdre de la vie…)
- présence de cases permettant la téléportation du joueur d'une case à une autre prédéfinie ou aléatoire (permettant d'offrir au jeu un aspect stratégique)

1.3 Conception Logiciel

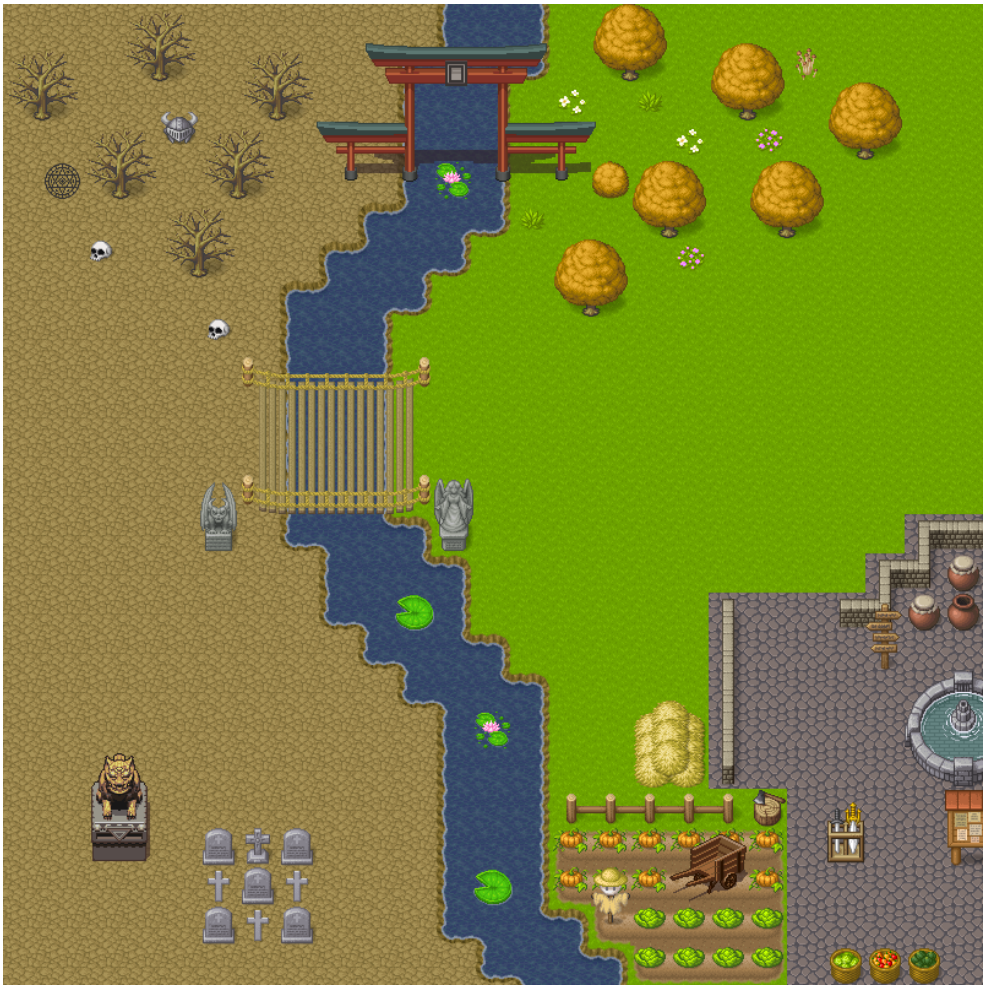

FIGURE 2 – Carte de la zone de combat

La zone de combat est une image de 800 par 800 pixels décomposée en 25 tiles sur la longeur et la largeur (un tile est un carré de 32 par 32 pixels).

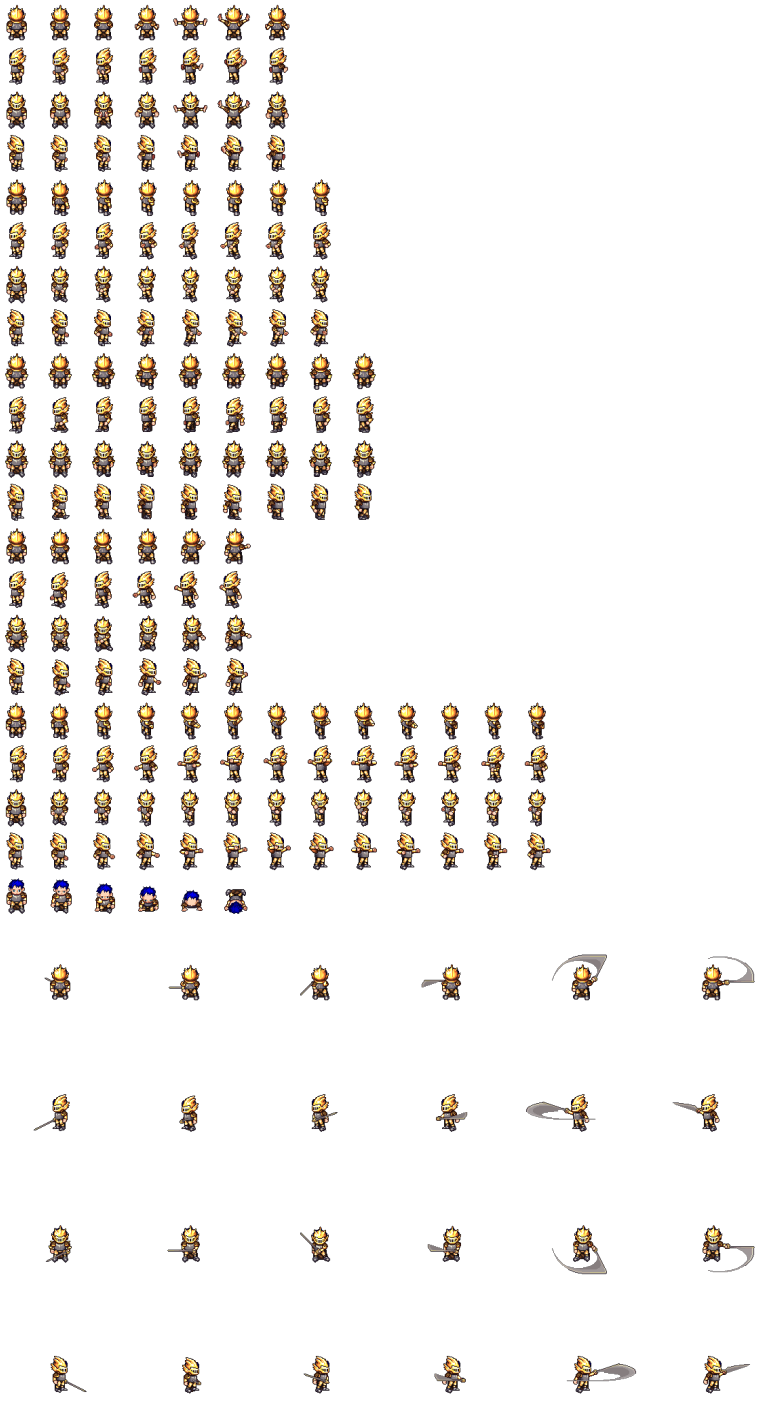

FIGURE 3 - Sprite d'un personnage

Pour tout ce qui est gestion du texte, nous utiliserions les fonctionnalités proposées par la bibliothèque SFML.
Nous n'avons pas pu mettre l'ensemble des ressources que nous allons utiliser pour le jeu pour éviter de surcharger le rapport. Il est possible de tous les trouver dans le dossier « rsc » récupérable sur le GitHub: https://github.com/LeoChazl/plt/tree/master/rsc .

2 Description et conception des états

L'objectif de cette section est une description très fine des états dans le projet. Plusieurs niveaux de descriptions sont attendus. Le premier doit être général, afin que le lecteur puisse comprendre les éléments et principes en jeux. Le niveau suivant est celui de la conception logiciel. Pour ce faire, on présente à la fois un diagramme des classes, ainsi qu'un commentaire détaillé de ce diagramme. Indiquer l'utilisation de patron de conception sera très appriécé. Notez bien que les règles de changement d'état ne sont pas attendues dans cette section, même s'il n'est pas interdit d'illustrer de temps à autre des états par leur possibles changements.

2.1 Description des états

2.2 Conception logiciel

2.3 Conception logiciel : extension pour le rendu

2.4 Conception logiciel : extension pour le moteur de jeu

2.5 Ressources

Illustration 1: Diagramme des classes d'état

3 Rendu : Stratégie et Conception

Présentez ici la stratégie générale que vous comptez suivre pour rendre un état. Cela doit tenir compte des problématiques de synchronisation entre les changements d'états et la vitesse d'affichage à l'écran. Puis, lorsque vous serez rendu à la partie client/serveur, expliquez comment vous aller gérer les problèmes liés à la latence. Après cette description, présentez la conception logicielle. Pour celle-ci, il est fortement recommandé de former une première partie indépendante de toute librairie graphique, puis de présenter d'autres parties qui l'implémente pour une librairie particulière. Enfin, toutes les classes de la première partie doivent avoir pour unique dépendance les classes d'état de la section précédente.

3.1 Stratégie de rendu d'un état

3.2 Conception logiciel

3.3 Conception logiciel : extension pour les animations

3.4 Ressources

3.5 Exemple de rendu

Illustration 2: Diagramme de classes pour le rendu

4 Règles de changement d'états et moteur de jeu

Dans cette section, il faut présenter les événements qui peuvent faire passer d'un état à un autre. Il faut également décrire les aspects lié au temps, comme la chronologie des événements et les aspects de synchronisation. Une fois ceci présenté, on propose une conception logiciel pour pouvoir mettre en œuvre ces règles, autrement dit le moteur de jeu.

4.1 Horloge globale

4.2 Changements extérieurs

4.3 Changements autonomes

4.4 Conception logiciel

4.5 Conception logiciel : extension pour l'IA

4.6 Conception logiciel : extension pour la parallélisation

Illustration 3: Diagrammes des classes pour le moteur de jeu

5 Intelligence Artificielle

Cette section est dédiée aux stratégies et outils développés pour créer un joueur artificiel. Ce robot doit utiliser les mêmes commandes qu'un joueur humain, ie utiliser les mêmes actions/ordres que ceux produit par le clavier ou la souris. Le robot ne doit pas avoir accès à plus information qu'un joueur humain. Comme pour les autres sections, commencez par présenter la stratégie, puis la conception logicielle.

5.1 Stratégies

5.1.1 Intelligence minimale

5.1.2 Intelligence basée sur des heuristiques

5.1.3 Intelligence basée sur les arbres de recherche

5.2 Conception logiciel

5.3 Conception logiciel : extension pour l'IA composée

5.4 Conception logiciel : extension pour IA avancée

5.5 Conception logiciel : extension pour la parallélisation

6 Modularisation

Cette section se concentre sur la répartition des différents modules du jeu dans différents processus. Deux niveaux doivent être considérés. Le premier est la répartition des modules sur différents threads. Notons bien que ce qui est attendu est un parallélisation maximale des traitements: il faut bien démontrer que l'intersection des processus communs ou bloquant est minimale. Le deuxième niveau est la répartition des modules sur différentes machines, via une interface réseau. Dans tous les cas, motivez vos choix, et indiquez également les latences qui en résulte.

6.1 Organisation des modules

6.1.1 Répartition sur différents threads

6.1.2 Répartition sur différentes machines

6.2 Conception logiciel

6.3 Conception logiciel : extension réseau

6.4 Conception logiciel : client Android

Illustration 4: Diagramme de classes pour la modularisation