Projet Logiciel Transversal

Steven SENG – Francis KINAVUIDI

Table des matières

1 Objectif

1.1 Présentation générale

Ce jeu base ses mécaniques sur des jeux comme Fire emblem, un tactical RPG, et Dofus, un MMORPG. C'est un tactical RPG au tour par tour où l'on contrôle 1 personnage.

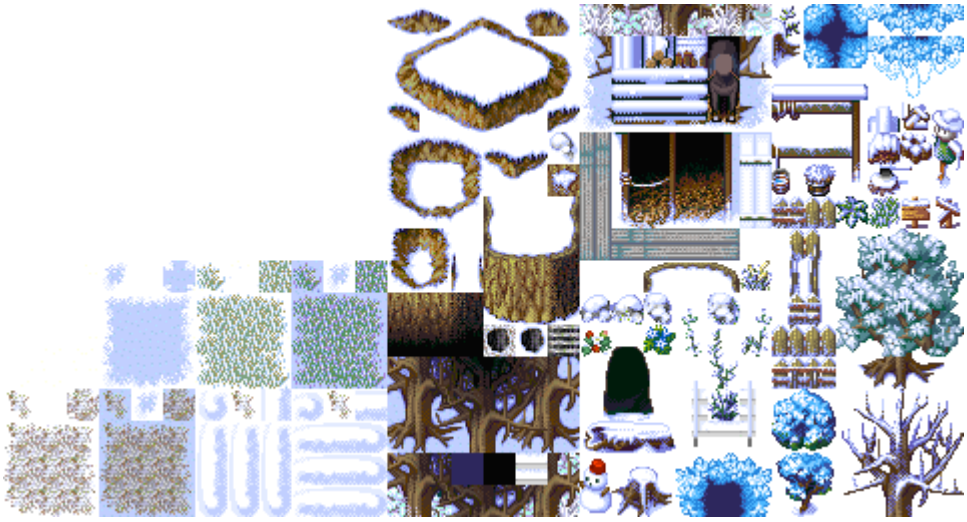
1.2 Règles du jeu

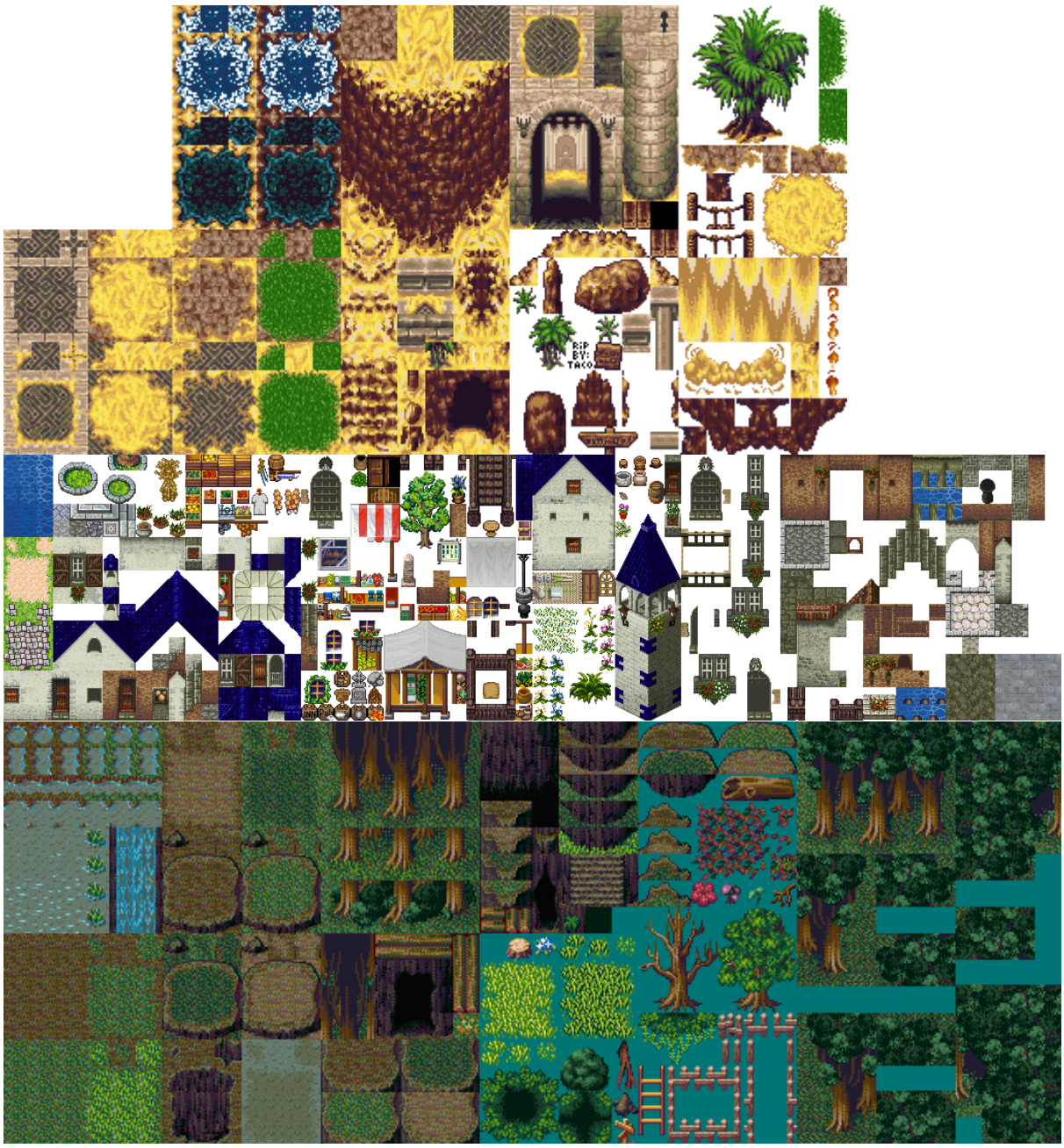
En début de partie, le joueur sélectionne une classe de personnage, possédant des caractéristiques propres (point de déplacement de base, type de déplacement, point de vie de base, attaque de base, initiative, etc.), et une liste 4 attaques dépendant de la classe choisie. Chaque attaque a des caractéristiques de base qui peuvent être modifiées par l'état du joueur ou le terrain : dégât, porté, précision et effet (sur le terrain, le lanceur ou la victime). Une fois le choix fait, le joueur est placé sur un terrain et a pour objectif d'éliminer son/ses adversaire(s).
La carte du jeu, ou le terrain, est constituée de nombreuses cases, chacune possédant un type prédéfini (forêt, ville, plaine, etc.) et un état (normal, neige, brume, etc.). Ces cases influencent les statistiques, états et attaques des personnages.
Un tour de jeu se déroule donc de la manière suivante : en fonction de leur statistique d'initiative, chaque personnage encore en vie joue un par un leur tour de joueur. Une fois qu'ils ont tous fini, les événements aléatoires du terrain s'activent.
Un tour de joueur se divise en 2 parties, la partie active et l'application des effets. Lors de la partie active, le joueur peut se déplacer autant de fois qu'il le veut tant qu'il a encore des points de déplacement mais ne peut utiliser que 2 attaques. Pour se déplacer vers une case adjacente, le personnage doit dépenser ses points de déplacement en fonction du type de case qu'il traverse et de son type de déplacement. Cette phase se finit si le joueur n'a plus d'action à effectuer ou s'il le décide. L'application des effets se fait automatiquement après la partie active. Les points de mouvement du personnage se régénèrent et les effets affectant ses points de vie et les changements d'état aléatoires dû au terrain s'appliquent.
Lorsque les points de vie d'un personnage tombent à 0 suite à une attaque ou à un effet, il meurt et disparaît du terrain.

1.3 Ressources

Chaque personnage possède un ensemble de sprites représentant celui-ci dans différentes positions : immobile, se déplaçant, attaquant et lançant un sort, et ce dans les quatre directions. Les mouvements de déplacement, d'attaque et de sortilège se décomposent chacun en une animation de 3 sprites.

Nous avons ensuite divers affichages représentant les différents types de terrains.





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