Projet Logiciel « »

Kusan THANABALASINGAM – Benoît LAFON

Table des matières

1 Objectif

1.1 Présentation générale

L'objectif de ce projet est la réalisation d'un jeu de stratégie tour par tour inspiré de jeux semblable à « Dofus ».

ena Pocket

Illustration 1: Dofus

Illustration 3: Final Fantasy Tactics

1.2 Règles du jeu

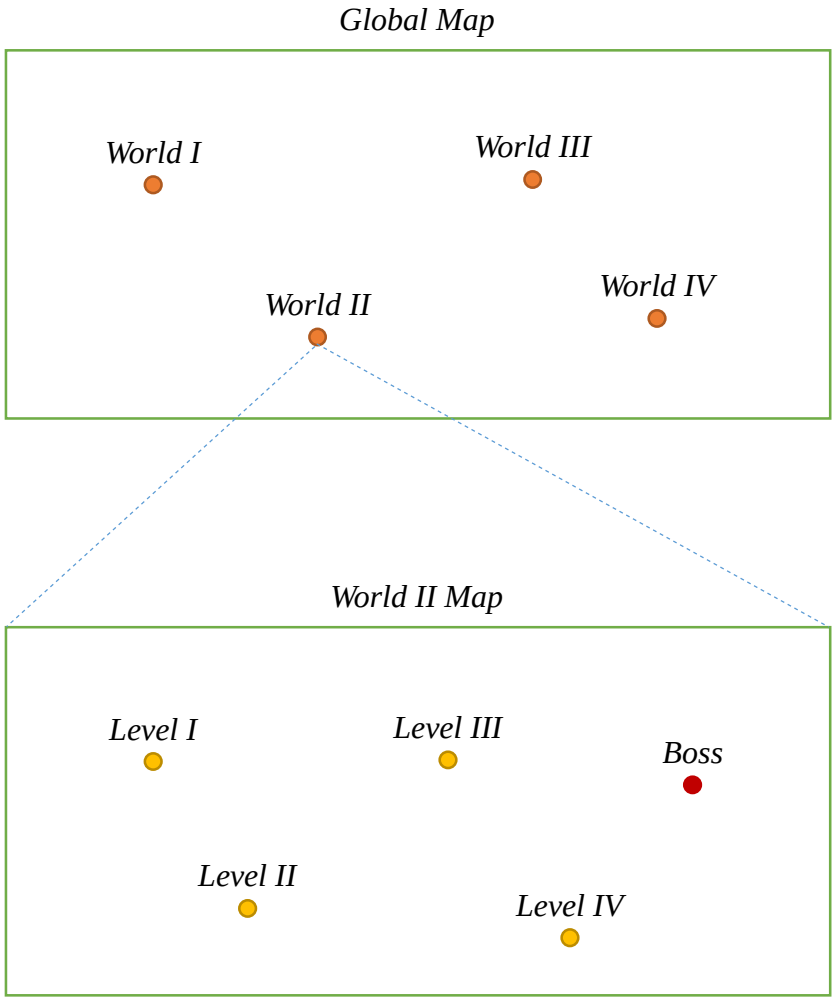
Le joueur déplace une équipe de personnages sur une carte et doit vaincre l'équipe adverse. Chaque personnage a des compétences et des propriétés (points de vies, attaque, défense, etc) différentes. Le joueur avance de niveau en niveau, le joueur ayant la possibilité de refaire un niveau pour augmenter son expérience.
Ainsi une montée de niveau donne des points de compétences au personnage que le joueur peut librement attribuer (pv, attaque, défense, etc). De plus le joueur gagne de nouveaux personnages et sorts au fur et à mesure de son avancée.

- Fonctionnement des cartes et des niveaux :

Le jeu est constitué de différentes cartes dont la *Global Map* est la plus générale. La *Global Map* recense tous les mondes possibles.
Chaque monde est constitué d'une carte qui lui est propre tel que la *World I Map* pour le premier monde. Sur chacune des cartes est représentée des niveaux que le joueur doit surmonter pour atteindre le prochain niveau. Une carte s'achève avec un *boss* de fin.

➢ Règles générales :

- Le monde *n+1* est débloqué si le *boss* du monde *n* a été vaincu.
- Le *boss* d'un monde *n* est débloqué si tous les niveaux de la même carte ont été terminés avec succès.
- Le niveau *n+1* d'un monde *n* est débloqué est débloqué si le niveau *n* a été terminé avec succès.



- Constitution et évolution des personnages :

Le joueur démarre une partie avec un seul personnage. Au fur et à mesure de l'avancée des mondes, le joueur gagne de nouveaux personnages.
Chaque personnage a des attributs et spécificités qui leurs sont propre en fonction de leur classe (points de vies, attaque, défense…).
Au terme d'un combat, chaque personnage gagne de l'expérience en fonction de son utilisation lors du combat ainsi que du niveau de l'ennemi.
Le joueur peut rejouer un même combat autant de fois qu'il le souhaite pour monter le niveau de ses personnages.
Le passage au niveau supérieur d'un personnage est accompagné d'un certain nombre de point que le joueur est libre d'attribuer au personnage.
De plus, le passage au niveau supérieur peut s'accompagner de l'apprentissage de nouvelles compétences.

- Déroulement d'un combat

Le déroulement d'un combat s'effectue en équipe et au tour par tour. L'objectif étant de vaincre l'équipe adverse. A chaque tour le joueur doit choisir une action selon le personnage sélectionné sachant que deux personnages au maximum peuvent effectuer une action.
Chaque attaque/sort a des portés différents, ainsi le joueur doit placer son personnage à porter du personnage qu'il veut atteindre pour pouvoir effectuer l'action désiré sachant que chaque personnage possède un nombre de case de déplacement possible fixe réinitialiser à chaque tour.
Une fois le nombre de personnages d'une équipe réduit à zéro ; le combat s'achève.

1.3 Conception Logiciel

Présenter ici les packages de votre solution, ainsi que leurs dépendances.

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 4: 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'implémentent 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 5: 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 6: 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 7: Diagramme de classes pour la modularisation