SLR202

TD2 UML Design Pattern : Modélisation d'un jeu de stratégie temps réel

Sylvie Vignes, Etienne Borde

Dans ce jeu, les joueurs contrôlent chacun une civilisation. Une civilisation comprend un ensemble de bâtiments et de différents types de personnages. Certains bâtiments permettent de créer les personnages. Le but du jeu est de détruire les unités adverses. Pour cela, le joueur doit amasser des ressources (nourriture, bois, or...), qui lui permettront de construire des bâtiments ou de créer plus de personnages.
Nous définissons trois types de bâtiments :

1. Le forum, bâtiment de base, qui permet de créer des villageois, et de stocker des ressources.
2. La caserne, qui permet de créer des guerriers.
3. Les entrepôts, qui permettent de stocker des ressources.

Il y a donc deux types de personnages :

1. Les villageois constituent les unités de base du jeu. Ils peuvent construire les bâtiments, et récupérer des ressources. Ils ne peuvent pas se battre.
2. Les guerriers sont les seuls personnages capables de combattre. Ils ne peuvent ni construire, ni récupérer des ressources.

La partie débute avec un forum déjà construit, et un petit nombre de villageois. Pour ramasser une ressource, le villageois doit se déplacer vers celle-ci, passer du temps à récolter, et ramener la ressource dans l'entrepôt le plus proche, ou le forum. La construction d'un bâtiment prend elle aussi un certain temps, de même que la création de nouveaux personnages. Il est possible d' « empiler » la création de personnages, c'est-à-dire demander à un bâtiment de produire plusieurs personnages à la suite. Dans ce cas, on met en place un système de file d'attente.
Chaque bâtiment peut être vu comme une fabrique de personnages. Une exception : les entrepôts ne produisent rien.

Pour simuler l'avancée d'une civilisation, on met en place un système d'époque/âge. Le premier âge constitue un stade de technologie peu avancée, ce qui signifie que les guerriers ont peu de points de vie, d'attaque et de défense, et que les villageois récoltent lentement les ressources. Le changement d'âge se fait en consommant des ressources depuis le forum, et permet de produire des guerriers plus forts, et des villageois plus rapides ; de même, les bâtiments changent d'aspect. Les guerriers créés avant le changement d'âge n'évoluent pas, mais peuvent retourner dans une caserne pour suivre un entraînement et être remis à niveau.

Les guerriers sont les seuls à pouvoir se battre, ils peuvent s'attaquer à tout ce qui constitue la civilisation d'un adversaire : bâtiments, villageois, guerriers. Ils peuvent avoir un rôle offensif (aller attaquer d'eux même les ennemis les plus proches), ou défensif (rester près d'une ressource pour empêcher les adversaires de l'exploiter).

Design pattern « State »

Comme décrits ci-dessus, les bâtiments peuvent potentiellement produire chacun des unités. Pour simplifier on considère qu'un bâtiment ne produit qu'un type de personnage, mais qu'il est possible de demander la création de plusieurs personnages en même temps. On crée donc une file d'attente pour les demandes de création. Le pattern state va nous permettre de déterminer trois états :

1. Le bâtiment ne peut créer aucune unité. C'est le cas de l'entrepôt.
2. Le bâtiment peut produire des unités et sa file de production n'est pas pleine.
3. La file d'attente du bâtiment est pleine. Il faut donc indiquer à l'utilisateur qu'il ne peut temporairement plus commander de personnages dans ce bâtiment.

Normalement, les seules transitions d'états se feront du 2 au 3 et vice-versa.

Design pattern « Method Factory »

Les casernes produisent des guerriers, qui ont des caractéristiques fixes. Toutefois, à chaque changement d'âge, les guerriers deviennent plus forts. On doit donc munir les casernes d'une fabrique de guerriers. On implémentera une fabrique « concrète » par âge qui reflète le fait que les guerriers deviennent de plus en plus forts quand on avance dans les âges. A chaque âge correspondra une fabrique concrète de guerriers.
Remarque : si on imagine différentes familles de guerriers (infanterie lourde, infanterie légère, archers, etc) on implémentera plutôt le DP « Abstract factory ».

Design pattern « Strategy »

Les guerriers peuvent adopter des comportements différents. Mais contrairement à un bâtiment, dont les changements d'états peuvent être automatisés, le changement de comportement ne dépend que du bon vouloir de l'utilisateur. On pourra modéliser un comportement offensif, et un défensif dans un premier temps. Le comportement par défaut d'un guerrier est offensif.

Design pattern « Observer »

Le changement d'époque se fait par le forum, sur intervention de l'utilisateur. Le forum appelle une méthode de changement d'âge de la civilisation. Un changement d'âge signifie un changement d'aspect graphique pour les bâtiments entre autres. Les classes gérant l'interface graphique vont donc observer la Civilisation pour détecter les changements d'âge.

Design pattern « Singleton »

Une partie oppose deux (ou plus) joueurs, donc civilisation. On part du principe qu'on ne peut jouer qu'une partie à la fois. On applique donc le pattern singleton, à une classe « Partie », qui contiendra les différentes civilisations engagées. On s'assure ainsi qu'il n'existera, à tout moment, qu'une unique instance de la classe Partie.

Design pattern « Flyweight »

On note que les villageois ont des caractéristiques identiques : points de vie maximums, défense, et ce quelque soit l'âge de la civilisation. On peut donc utiliser le pattern « poids mouche », en implémentant une classe « ProprietesVillageois » qui contiendra ces informations communes (ce sont des données partagées entre instances).

Design pattern « Visitor »

On aimerait mettre au point un système de statistiques, qui indiquent le nombre de villageois, guerriers, bâtiments de chaque joueur en temps réel. Pour cela, on utilise le « Visiteur ». A intervalles réguliers, celui-ci va visiter les civilisations, et tous les éléments qui les composent. Chaque sous classe d'*ElementCiv* va donc implémenter une méthode *accept()* qui permettra au visiteur de savoir de quel type sont chacun des éléments visités.

Références DP :

http://rpouiller.developpez.com/tutoriel/java/design-patterns-gang-of-four/

http://en.wikibooks.org/wiki/Computer_Science_Design_Patterns
http://fr.wikibooks.org/wiki/Patrons_de_conception (en français mais moins complet)