Projet Logiciel Transversal

Federico MONTES DE OCA – Billy UM

Figure 1 - Exemple du jeu Dofus (source : dofus.com)

Table des matières

1. Présentation Générale 2
1.1 Archétype 2
1.2 Règles du jeu 2
1.3 Ressources 3
2. Description et conception des états 4
2.1 Description des états 4
2.1.1 Etat éléments fixes 4
2.1.2 Etat éléments mobiles 5
2.1.2 Etat général 5
2.2 Conception logicielle 6

1. Présentation Générale

1.1 Archétype

L'objectif de ce projet est de réaliser un jeu de type Dofus. A l'origine, Dofus est un jeu de rôle en ligne massivement multijoueur (MMORPG) développé et édité par Ankama, dans lequel les joueurs incarnent un ou plusieurs types de personnages. Comme les autres MMORPG, le joueur peut faire évoluer son personnage en augmentant son niveau, en personnalisant son équipement ou bien encore en choisissant sa classe de personnage. Selon sa classe, le joueur aura alors un gameplay différent. Cependant, notre projet se focalisera en particulier à créer un jeu reposant sur son système de combat. En effet, il s'agira d'un jeu de combat entre personnages se déroulant tour par tour sur des terrains variés. Un exemple du système de combat est présenté Figure 1.

1.2 Règles du jeu

Deux équipes s'affrontent. Dans chacune des équipes, il y aura 3 personnages de type différent maximum. Le but du jeu est alors de vaincre l'ennemi adverse en éliminant tous ses personnages. Chaque tour, chaque joueur aura la possibilité de faire avancer un personnage et le faire attaquer. Le déplacement du personnage sera régi selon des cases prédéfinis sur un terrain. En effet, pour avancer, le personnage devra se déplacer selon des cases proposées. Quant au système de combat, il se déroulera tour par tour. Durant ce tour, le joueur choisira le personnage qu'il veut jouer. Il décidera à la fois de son déplacement et du coup porté à un personnage adverse.

Types de personnage

Le joueur disposera de trois types de personnage :

- Un personnage qui attaque à distance,
- Un mage qui utilise des sorts,
- Un personnage qui attaque au corps à corps.

Modes de jeu

Deux modes de jeu seront proposés :

- Seul, contre l'IA.
- Multiplayer, en ligne, une personne contre une autre personne.

1.3 Ressources

Des TileSets complets ont été trouvés sur le site : https://kenney.nl.
Avec les éléments de la Figure 3, on peut assembler des personnages à partir d'une base.

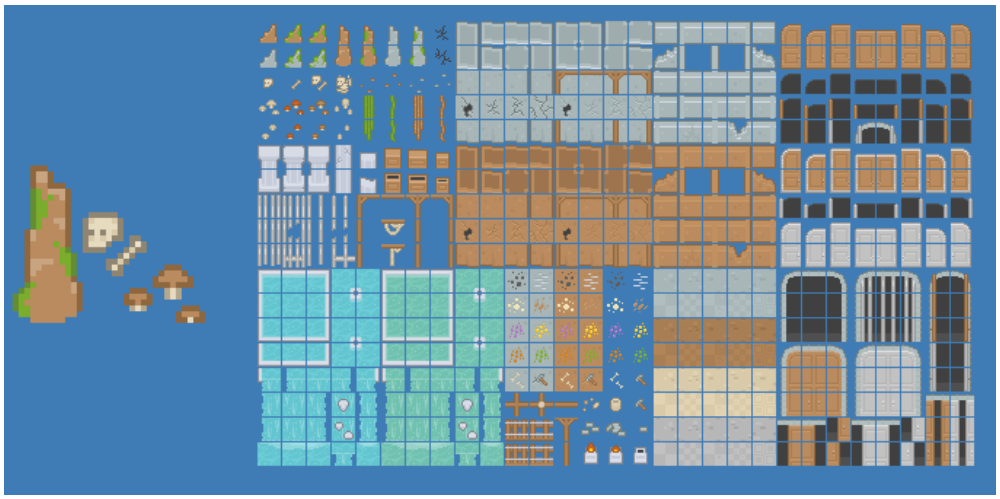

FIGURE 2 -Texture pour les terrains et les décors

FIGURE 3 - Texture pour les personnages

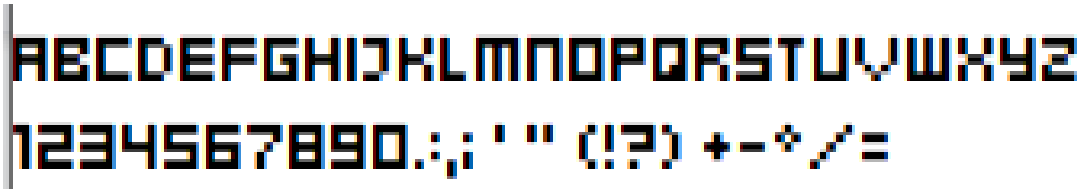

FIGURE 4 - Typographie (Kenney Mini Square v1.0)

2. Description et conception des états

2.1 Description des états

Un état du jeu est formé par un ensemble d'éléments fixes sur le terrain et un ensemble d'éléments mobiles. Tous les éléments possèdent les propriétés suivantes :

- Coordonnées (x, y) dans la grille
- Nom

2.1.1 Etat éléments fixes

Le terrain est formé par une grille d'éléments nommé « MapCell ». La taille de cette grille est fixée au démarrage du niveau. Les types de cases sont :

1. Cases « ObstacleMapCell ». Les cases « ObstacleMapCell » sont des éléments infranchissables pour les éléments mobiles. Le choix de la texture est purement esthétique, et n'a pas d'influence sur l'évolution du jeu. On considère les types de cases suivants :
   - Les obstacles « ROCK », pour les rochers.
   - Les obstacles « TREE », pour les arbres.
   - Les obstacles « WATER », pour l'eau.
   - Les obstacles « FIRE », pour le feu.

2. Cases « SpaceMapCell ». Les cases « SpaceMapCell » sont les éléments franchissables par les éléments mobiles. On considère les types de cases « SpaceMapCell » suivants :
   - Les espaces « GRASS » pour désigner l'herbe.
   - Les espaces « SAND », pour désigner le sable.
   - Les espaces « CONCRETE », pour désigner le béton.

   De plus, certaines cases « SpaceMapCell » ont la possibilité d'améliorer les statistiques des personnages. Ils vont par exemple pouvoir augmenter la santé des personnages ou bien leurs attaques.

2.1.2 Etat éléments mobiles

Deux types d'éléments mobiles peuvent être déplacés par le joueur :

1. Elément mobile « Character » qui représente un personnage. Cet élément est dirigé par le joueur. Chaque « Character » possède des statistiques propres à sa classe. On lui associe ainsi un niveau de santé "heath", un niveau d'attaque "attack", un niveau de mana "mana" et un niveau de défense "defense". Il se distingue également par son caractère "boosted" si le personnage est boosté, par son champ d'attaque et ses types de mouvements.

   À « Character » est également associée une propriété nommée « CharacterStatusID », qui peut prendre les valeurs suivantes :
   - Statut « SELECTED » : cas où le « Character » est sélectionné.
   - Statut « AVAILABLE » : cas où le « Character » est disponible.
   - Statut « WAITING » : cas où le « Character » est en attente.
   - Statut « DEATH » : cas où le « Character » est mort.
   - Statut « TARGET » : cas où le « Character » est visé.

   « Character » possède enfin une propriété nommée « CharacterTypeD » pour distinguer le type de personnage, et qui peut prendre les valeurs suivantes :
   - Type « STRENGHT » : cas où le « Character » est un personnage orienté corp à corp.
   - Type « DISTANCE » : cas où le « Character » est un personnage orienté attaque à distance.
   - Type « MAGICIAN » : cas où le « Character » est un personnage orienté magie.

2. Elément mobile " Cursor " qui représente le curseur pointé. Ce " Cursor" permet de sélectionner un personnage. "Cursor" dispose d'une position repérée par ses coordonnées (x, y).

2.1.2 Etat général

A l'ensemble des éléments statiques et mobiles, nous rajoutons les propriétés suivantes :

- "turn" qui indique le nombre de tours.
- "end" qui indique la fin du jeu.
- "win" qui indique que le joueur a gagné.
- "loose" qui indique que le joueur a perdu.

2.2 Conception logicielle

Le diagramme des classes pour les états est présenté en Figure 11. Son architecture est fondée sur le Polymorphisme par sous-typage dont la classe “Element” est la classe mère. Toute la hiérarchie des classes filles “Element” permettent de représenter les différentes catégories et types d’élément.

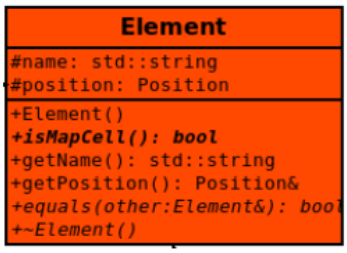

FIGURE 5 - Classe “Element”

On peut distinguer les classes filles qui héritent directement de la classe “Element” :

- La classe “Character” est la classe qui va permettre la création de nos trois personnages. Elle contient toutes les informations nécessaires à leur implémentation. À chaque personnage est associé des statistiques propres à lui par une relation de composition entre la classe “Character” et la classe “Stats”. Dans notre cas, on considère que des statistiques ne peuvent exister sans personnage. On associe également par relation de composition une énumération “CharacterStatusID” décrivant le statut du personnage, ainsi qu’une énumération “CharacterTypeID” exposant sa classe de personnage.

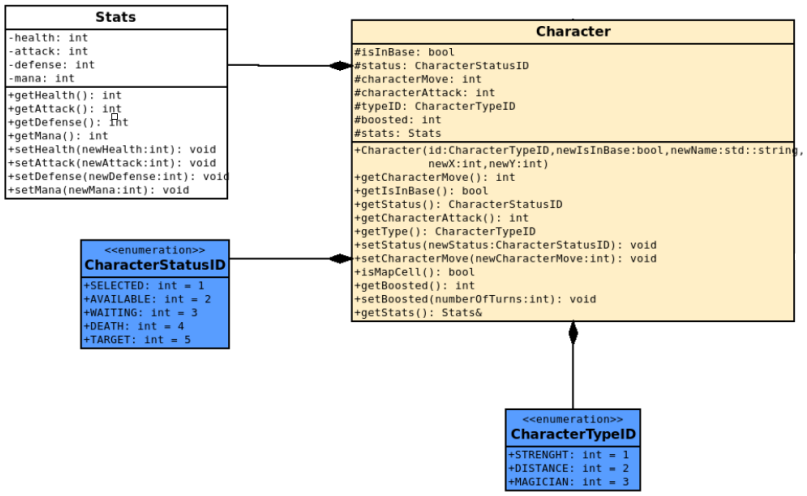

FIGURE 6 - Bloc “Character”

- La classe “Cursor” est la classe qui va permettre de représenter le curseur pointé et va permettre de sélectionner les personnages

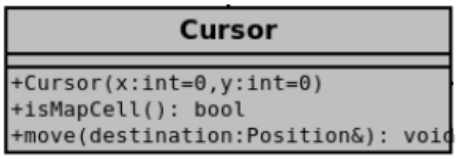

FIGURE 7 - Classe “Cursor”

- La classe “MapCell” établit également une relation d’héritage. Ses deux sous-classes “SpaceMapCell” et “ObstacleMapCell” sont des spécialisations de la classe “MapCell”. Leur rôle est la création du terrain :

    - “SpaceMapCell” représente les cases où l’on peut se déplacer. Une énumération “SpaceMapCellID” par composition lui est associée pour représenter les différents types d’espace.

    - “ObstacleMapCell” représente les obstacles. Une énumération “ObstacleMapCellID” par composition lui est associée pour représenter les différents types d’obstacle.

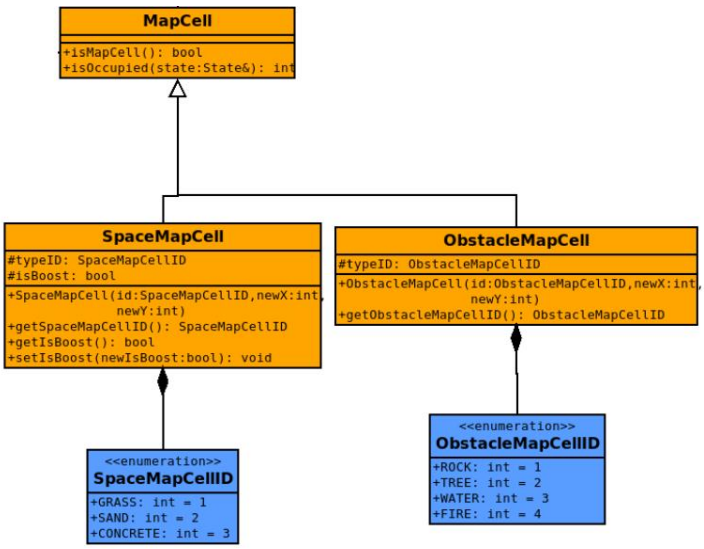

FIGURE 8 - Bloc “MapCell”

Ensuite, pour décrire tous les éléments de l’état de notre jeu, la classe “State” est le conteneur de nos ensembles d’éléments. Elle contient un vecteur “map” à deux dimensions de pointeurs d’éléments de “MapCell” qui forme le terrain de notre jeu, ainsi qu’un vecteur “characters” à une seule dimension de pointeurs des personnages de “Character”. Elle possède également des booléens pour déterminer la fin du jeu, la victoire et la défaite.

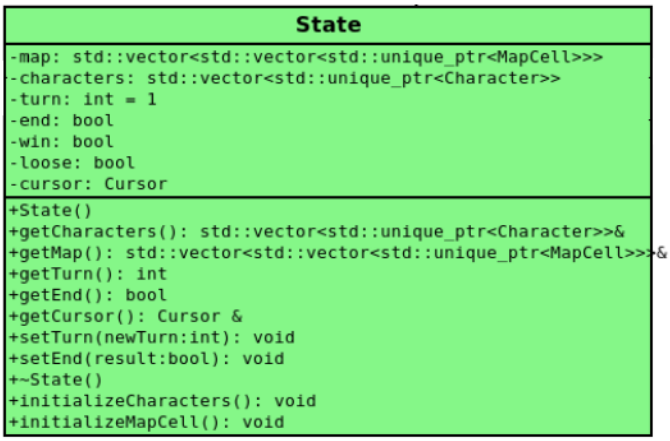

FIGURE 9 - Class “State”

On utilise le design pattern Observable pour signaler les changements d’états. La classe “State” hérite de la classe Observable. Lorsque la classe “State” subit un changement, l’observateur “IObserver” notifie l’évènement correspondant au changement appliqué.

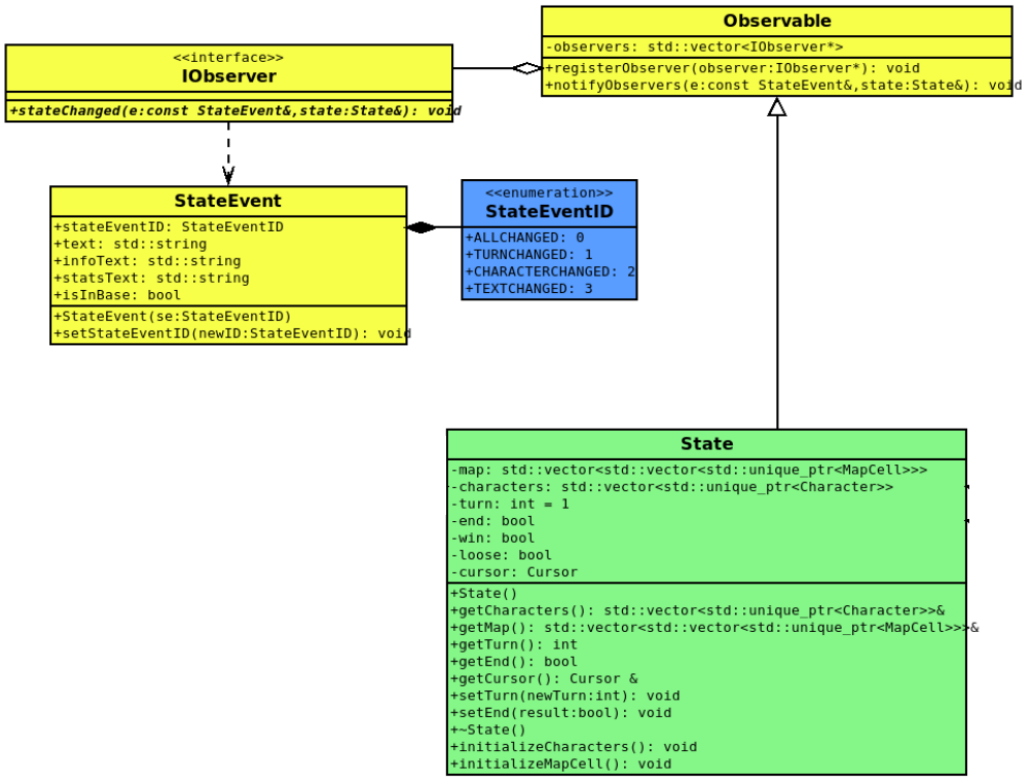

FIGURE 10 - Pattern Observable

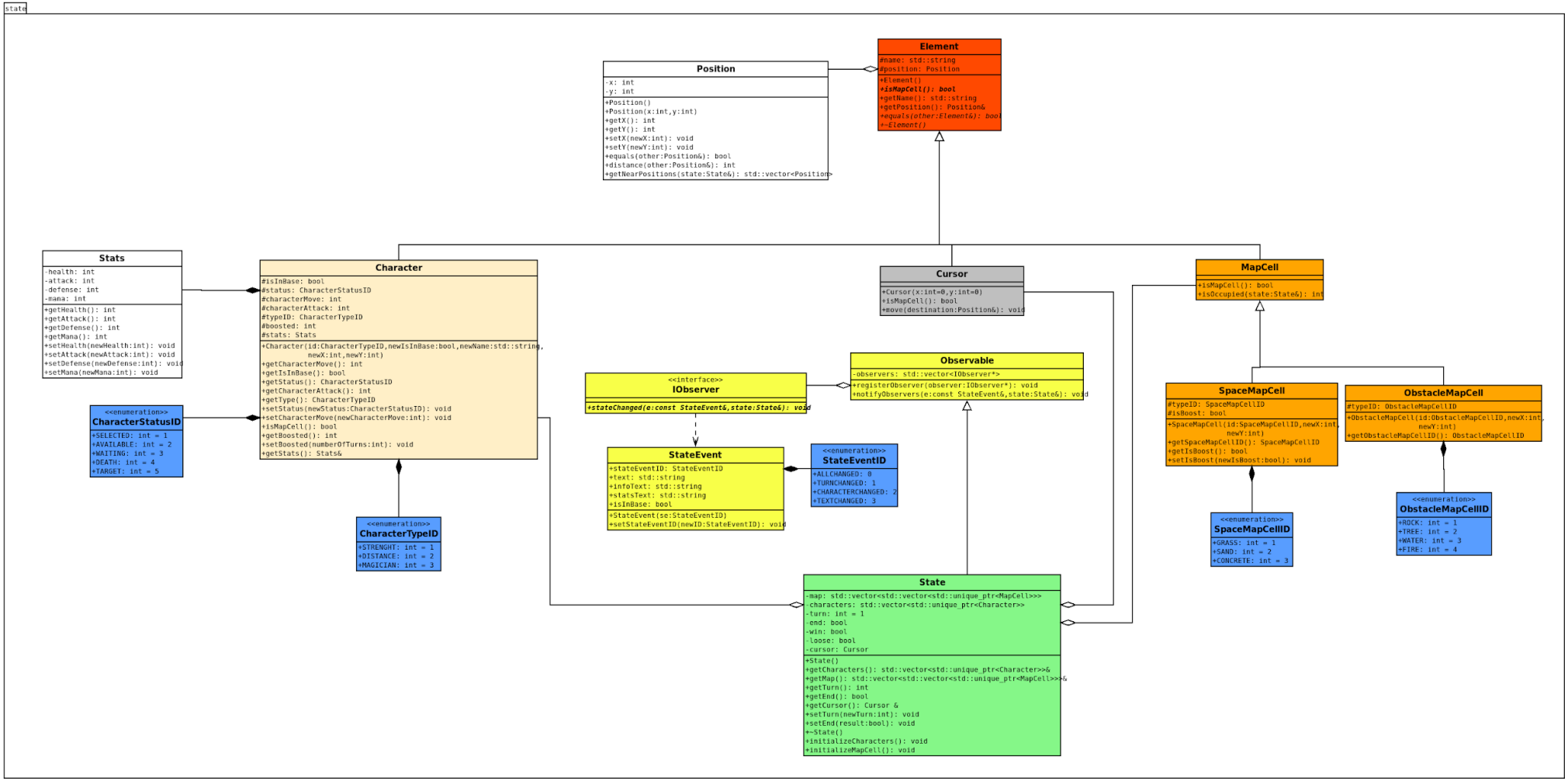

FIGURE 11 - Diagramme des classes