PROJET LOGICIEL TRANSVERSAL:
« CotCot WarFarm »

Jimmy HENNION - Antoine NOURRISSON

Table des matières

1/ Description du jeu

1.1/ Description Générale

CotCot WarFarm est un jeu s'inspirant de Worms Armaggedon (version 2D). Au lieu d'assister à une guerre entre des vers de terre, ce sont des poules qui s'affrontent par équipe. Elles peuvent, tout comme dans Worms, utiliser diverses armes telles que les grenades, les roquettes, les pistolets,…

Il y a toujours la notion de gravité terrestre sans inclure la force du vent. Le but est d'être la dernière équipe possédant des poules en vie. Le déroulement se fait au tour par tour. Les actions possibles à chaque tour sont: se déplacer, sauter, tirer avec une seule arme, utiliser des objets spéciaux… Chaque tour dure 30s maximum ou s'arrête lorsque le joueur a tiré avec son arme. Le décor est indestructible.

Il y aura en tout une capacité de quatre équipes au maximum sur le terrain, donc il peut y avoir jusqu'à quatre joueurs sur une partie. Il est possible, lorsque moins de quatre joueurs jouent, de rajouter des I.A., toujours jusqu'à un maximum de quatre équipes sur le terrain.
Les poules meurent lorsque:
- elles perdent tous leurs points de vie
- elles sont projetées hors de l'écran et se noient dans l'eau (effectivement, une poule ne peut ni voler ni nager!)

1.2/ Textures employées

Voici les différentes textures qui sont employées pour le projet CotCot:

Illustration n°1 : Textures des terrains

Illustration n°2 : Texture des personnages

Comme il est possible de le remarquer, il y a différentes couleurs de poules, et ces couleurs permettent de distinguer l'équipe. Il y a donc une couleur par équipe. De plus, certaines textures sur cette image ne sont pas utilisées, telles que les derrières des poules…

Illustration n°3 : Textures des armes

Ici toute une panoplie d'arme se propose pour équiper les poules, cependant seulement une petite portion d'entre elles seront utilisées.

2/ Description et conception des états

2.1/ Description des états

Un état du jeu est formé par un ensemble d'éléments fixes (l'environnement) et un ensemble d'éléments mobiles (les poules, les armes). Tous les éléments possèdent les propriétés suivantes :

- Coordonnées (x,y) dans la grille
- Identifiant de type d'élément : ce nombre indique la nature de l'élément (ie classe)

2.1.1/ États d'éléments fixes

L'environnement est formée par une grille d'éléments nommée « cases ». La taille de cette grille est fixée au démarrage du niveau. Les types de cases sont:

Cases « Terrain ». Les cases « Terrain » 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.

Cases « Espace ». Les cases « espace » sont des éléments franchissables par les éléments mobiles. On considère les types de cases « espace » suivants :

- Les espaces « vides »
- Les espaces « eau », qui contiennent une texture eau et qui permettent de définir des lieux où les poules peuvent mourir.
- Les espaces « extra », qui contiennent une texture en arrière plan (pour faire joli !).
- Les espaces « départ », qui définissent les positions initiales possibles pour les poules.
- Les espaces « caisse », qui définissent les positions où peuvent apparaître une caisse d'armes.

2.1.2/ États d'éléments mobiles

Les éléments mobiles possèdent une direction (aucune, droite, gauche, haut, bas), une vitesse et une position. Ils sont dirigés par un joueur ou une IA.

Éléments mobiles « Poule ». Cet élément est dirigé par commande de la propriété de direction (aucune, droite et gauche). Ces éléments possèdents deux propriétés. D'une part, on utilise « la couleur » pour différencier les équipes de poules . D'autre part, on utilise une propriété que l'on nommera « status », et qui peut prendre les valeurs suivantes :

- Status « Alive » : cas le plus courant, la poule est en vie et peut agir.
- Status « Hurt » : cas où la poule est blessée et perd des points de vie.
- Status « Dead » : cas où la poule est morte et ne peut plus agir.

Éléments mobiles « Arme ». Cet élément est dirigé par commande de la propriété de direction (aucune, haut et bas). On utilise la propriété « status » qui prendra les valeurs suivantes :

- Status « Visible » : cas où l'arme est sortie, la poule est immobile et s'apprête à tirer.
- Status « Invisible » : cas où l'arme est dans la poche, la poule bouge ou c'est à l'équipe adverse d'agir.

2.1.3/ État général

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

- Compteur MortSubite : compte le nombre de « tic » de l'horloge globale depuis le début de la partie. A un nombre choisi de « tic », nous passons dans l'état mort subite pour accélérer la fin de la partie. L'état mort subite réduit la vie de toutes les poules encore vivantes à 1 point et provoque la montée de l'eau.
- Compteur Tour : compte 30 secondes. Un tour dure au maximum 30 secondes.
- Compteur Poule : compte le nombre de poules vivantes par équipe.
- Compteur équipe : compte le nombre de points de vie par équipe.

2.2/ Conception Logiciel

Nous pouvons mettre en évidence les groupes de classes suivants (diagramme des classes présenté en Illustration n°4) :

Classe Éléments. Toute la hiérarchie des classes filles d'*Eléments* permettent de représenter les différentes catégories et types d'éléments. Pour faire de l'introspection, nous avons opter pour des méthodes comme *isStatic()* qui indiquent la classe d'un objet.

Fabrique d'éléments. Dans le but de pouvoir fabriquer facilement des instances d'Element, nous utilisons la classe *ElementFactory*. Cette classe, qui suit le patron de conception **Abstract Factory**, permet de créer n'importe quelle instance non abstraite à partir d'un caractère. L'idée est de l'utiliser, en autres, pour créer un niveau à partir d'un fichier texte. Par exemple, chaque texture correspondant à un type de terrain est identifiée par une chaîne de quatre caractères (la chaîne de caractère « PHEF » identifie la texture représentant un carré de pierre aux angles droits et ayant de l'herbe).

Dans le but de faciliter l'enregistrement des différentes instanciation possible, nous avons créé le couple de classes *AelementAlloc* et *ElementAlloc<E,ID>*. La première est une classe abstraite dont le seul but est de renvoyer une nouvelle instance. La seconde sert à créer des implantations de la première pour une classe et une propriété d'identification particulière.

Conteneurs d'éléments. Viennent ensuite les classes *State*, *ElementList* et *ElementGrid* qui permettent de contenir des ensembles d'éléments. *ElementList* contient une liste d'éléments, et *ElementGrid* étends ce conteneur pour lui ajouter des fonctions permettant de gérer une grille. Enfin, la classe State est le conteneur principal, avec une grille pour le niveau, et une liste pour les poules et les armes.

2.3/ Conception Logiciel : extension pour le moteur de jeu

Au sein des différentes classes d'*Element*, ainsi que leurs conteneurs comme *ElementList* et *State*, nous avons ajouté des méthodes de clonage et de comparaison :

- Méthodes *clone()* : elles renvoie une copie complète (les sous-éléments sont également copiés) de l'instance.
- Méthodes *equals()* : elles permettent de comparer l'instance actuelle avec une autre, et renvoie true si les deux instances ont un contenu identique

Voici le diagramme des classes pour représenter les états du jeu :

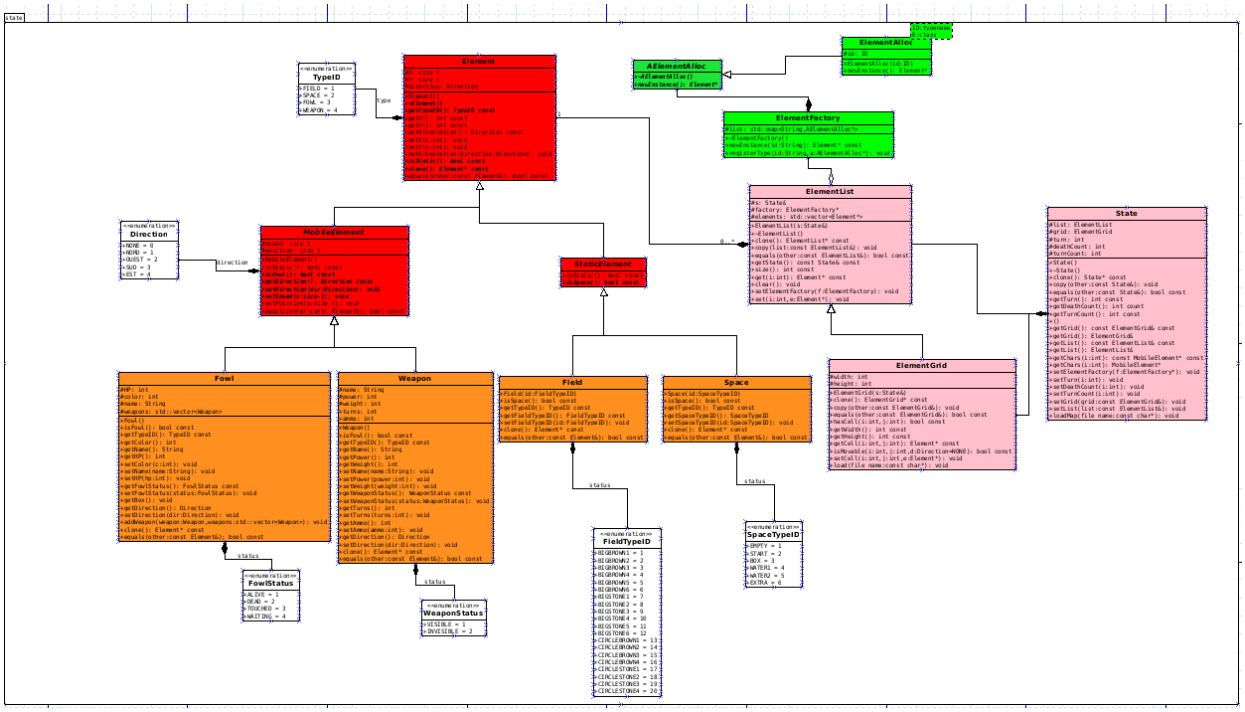

Illustration n°4 : Diagramme des classes pour l'état du jeu

En rouge sont les classes abstraites permettant de distinguer les éléments mobiles des éléments statiques. Les classes en orange sont ces éléments, et elles possèdent toutes les méthodes essentielles pour la configuration des personnages et décors. Les classes roses et vertes forment un pattern Factory, afin de pouvoir générer de manière plus simple nos éléments mobiles et statiques. Il est envisagé de créer un pattern Observer par la suite pour pouvoir communiquer avec la partie rendu du jeu. Ce dernier n'est donc pas encore visible sur le diagramme ci-dessus.

2.4/ Ressources

Les niveaux sont codés en fichier texte, puis traduits en instance d'éléments grâce à la factory. Voici un exemple basique (uniquement cases «terrains») du contenu d’un fichier texte et de son rendu en textures (Illustration n°5) :

```
 PHEpPHEFPHEb
P-EF                                                PHEdPHEq
P-EF                                                      P-EF
P-EFPHEFPHEcPHE?PHEFPHEcPHE?PHEFPHEcPHE?PHEFP-EF
```

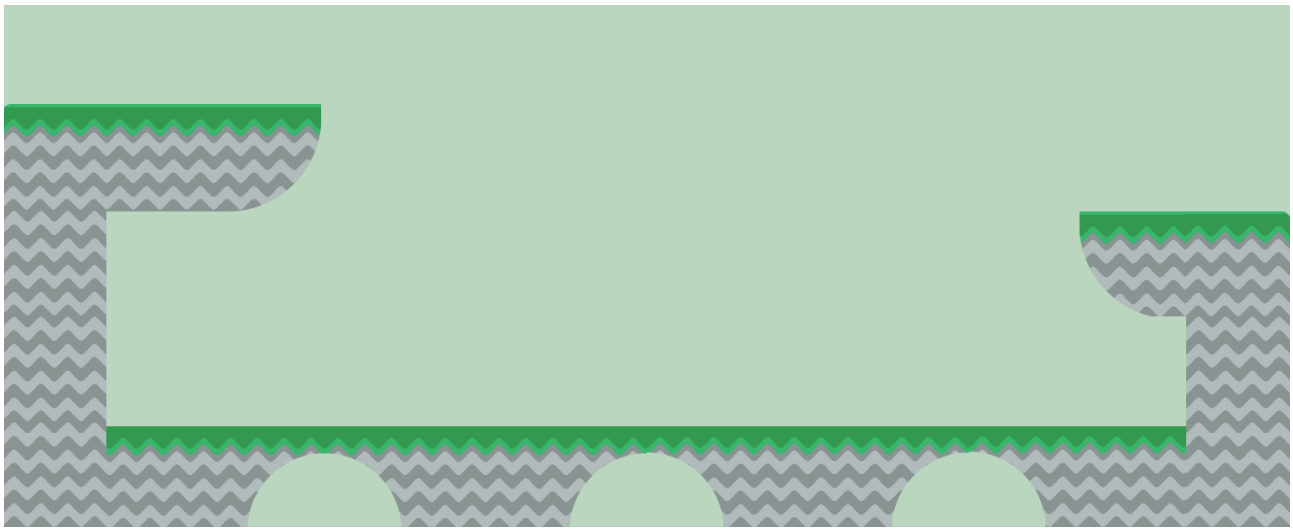

Illustration n°5 : Exemple de map