INFO 0402

Bomberman

CHEMIN Pierre & HADID Hocine | 09/04/2022
Dirigé par **Emilien BOURDY**

TABLE DES MATIERES

1. Présentation

Dans le cadre du module d'INFO0402, nous avons dû réaliser le jeu du Bomberman en C++. Nous verrons dans ce rapport différents points, de la modélisation à la concrétisation de ce projet en passant par les difficultés rencontrées.

Le jeu du Bomberman est un jeu dans lequel un Bomberman (le joueur) est sur une map au milieu d'ennemies et d'items. L'objectif est d'arriver sur une case précise, la « cible » du jeu une fois que tous les ennemis ont été éliminés. Une autre difficulté du jeu, outre les ennemis, sont les murs présents sur la map mais destructible par le joueur à l'aide de bombes. Ces mêmes bombes serviront à éliminer les ennemis.

2. Modélisation

Dans cette partie nous verrons toute la modélisation du projet du Bomberman, la réflexion qui a été faite pour arriver au résultat présenté.

A. Diagramme de classe

Pour mieux organiser et se répartir les tâches de développement de notre projet du Bomberman, nous avons réalisé un diagramme de classe permettant ainsi de mieux se projeter sur le développement du jeu. Le diagramme de classe complet se situe en annexe.

Nous pouvons voir que l'organisation que nous avons choisie est de répartir toutes nos classes en « package », permettant ainsi d'avoir une architecture des dossiers plus propre et lisible.

1) Le package engine

Dans ce package, nous aurons la classe SystemGame (qui permet de gérer une partie de jeux) ainsi que tout ce qui peut servir de manière globale au jeu.

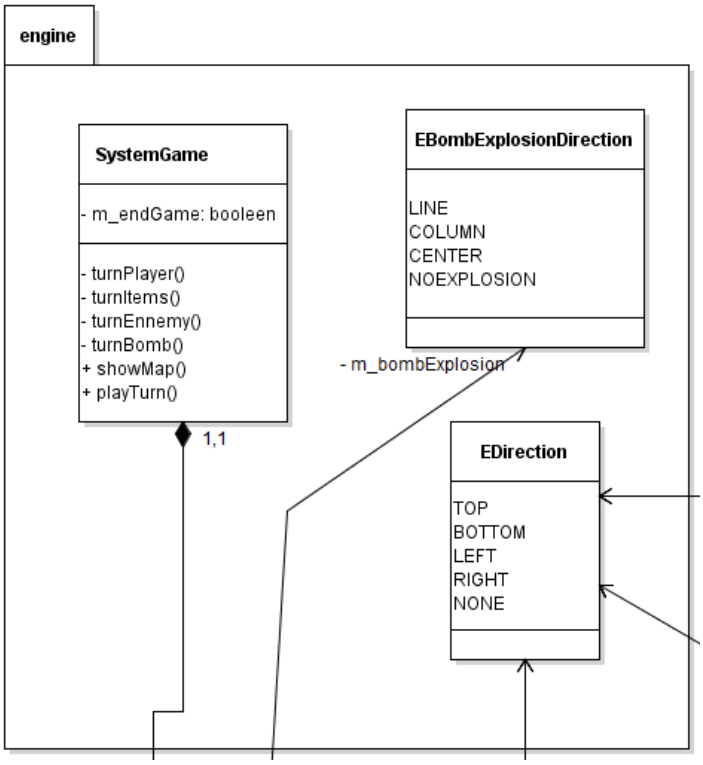

Figure 1 - Diagramme de classe du package engine

Le package engine contient donc la classe SystemGame qui dispose seulement de l'attribut privé m_endGame, un booléen qui permet de savoir si la partie est finit ou non. Pour ce qui est des fonctions et méthodes, nous disposerons des méthodes privées permettant de faire jouer le joueur, les items, les

ennemies et les bombes. Nous disposerons également des méthodes publics showMap permettant d'afficher la map et playTurn qui permet de jouer un tour de jeu.

Nous pouvons également retrouver deux énumérations, une permettant de savoir comment se situe l'explosion sur une case et l'autre permettant de connaitre la direction dans laquelle se déplace un Personnage ou une flèche.

2) Le package Items

Le package Items va contenir toutes les classes permettant de gérer les items présents dans le jeu du Bomberman.

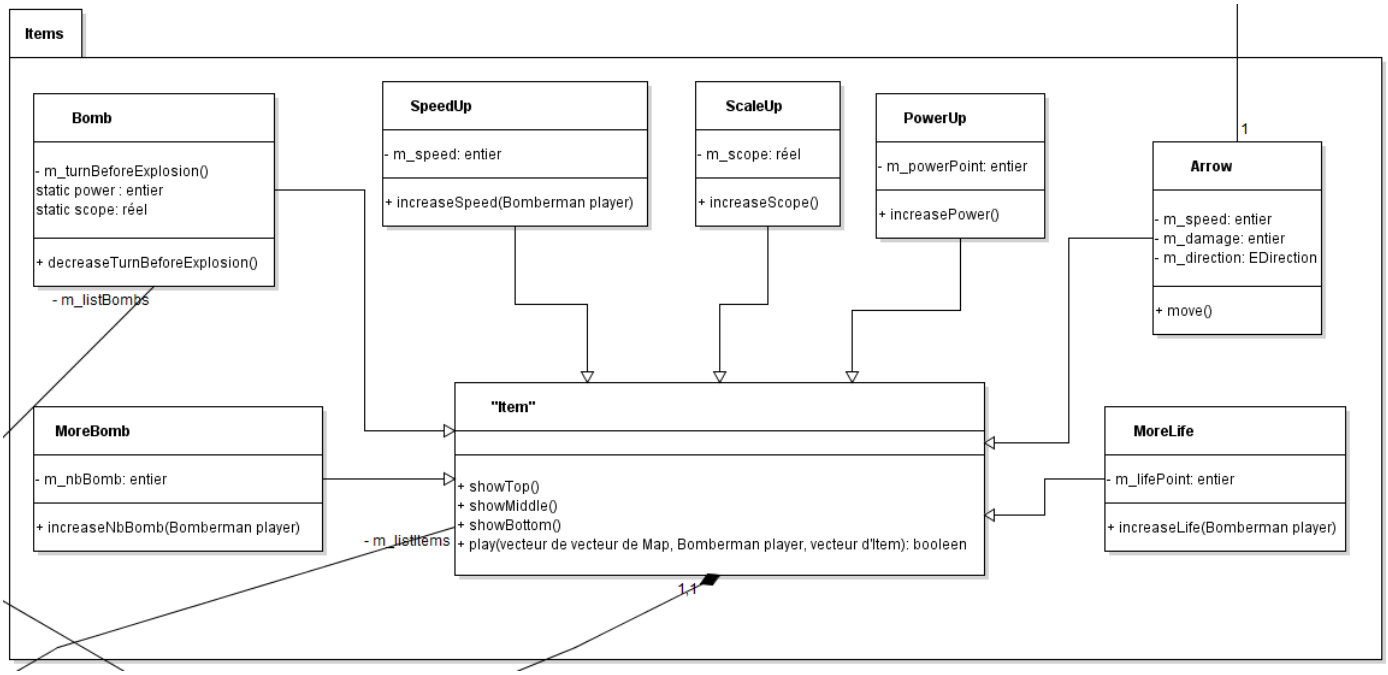

Figure 2 - Diagramme de classe du package Items

La classe « principal » de ce package est la classe abstraite Item. En effet, c'est depuis cette classe que l'on va créer tous nos items. Un item possède donc une position. La position sera une composition étant donné que si l'on supprime notre item cela ne sert à rien de conserver la position. De plus, un item possède les méthodes public showTop, showMiddle et showBottom permettant d'afficher notre item dans une case de 3 caractères de haut ainsi que la méthode play permettant de jouer un item. Cette méthode sera notre méthode virtuelle pure puisque notre classe Item est une classe abstraite. Nous ne pouvons donc pas définir de manière globale comment un Item jouera.

La classe MoreBomb hérite de la classe Item. Cette classe permet de créer des items qui permettent de rajouter des bombes au Bomberman. Pour cela elle possède un attribut correspondant au nombre de bombes à rajouter et une méthode qui rajoute le nombre de bombes au Bomberman.

La classe SpeedUp hérite de la classe Item. Cette classe permet de créer des items qui permettent de rajouter de la vitesse au Bomberman. Pour cela elle possède un attribut correspondant à la vitesse à rajouter et une méthode qui rajoute la vitesse au Bomberman.

La classe PowerUp hérite de la classe Item. Cette classe permet de créer des items qui permettent de rajouter de la puissance aux bombes. Pour cela elle possède un attribut correspondant à la puissance à rajouter aux bombes et une méthode qui rajoute la puissance aux bombes.

La classe MoreLife hérite de la classe Item. Cette classe permet de créer des items qui permettent de rajouter de la vie au Bomberman. Pour cela elle possède un attribut correspondant aux points de vie à rajouter au Bomberman et une méthode qui rajoute le nombre de point de vie au Bomberman.

Ensuite nous aurons la classe Bomb qui permet de gérer les bombes. Cette classe hérite de la classe Item et possède un attribut privé turnBeforeExplosion qui indique le nombre de tour restant avant l'explosion de la bombe. La classe Bomb possède également deux attributs de classe, un pour la puissance d'une bombe et un pour la portée d'une bombe. L'unique méthode publique decreaseTurnBeforeExplosion qui permet de décrémenter le nombre de tour restant avant l'explosion de la bombe.

Enfin, la classe Arrow hérite également de la classe Item et permets de gérer les flèches lancées par le Bomberman. Il y a 3 attributs privées, un pour la vitesse de la flèche, un autre pour les dégâts infligés par la flèche et un pour la direction dans laquelle se dirige la flèche. La classe Arrow possède une méthode publique move qui permet de déplacer la flèche dans la direction souhaitée.

3) Le package Persos

Le package Persos va contenir toutes les classes permettant de gérer les entités qui seront présentes sur la map.

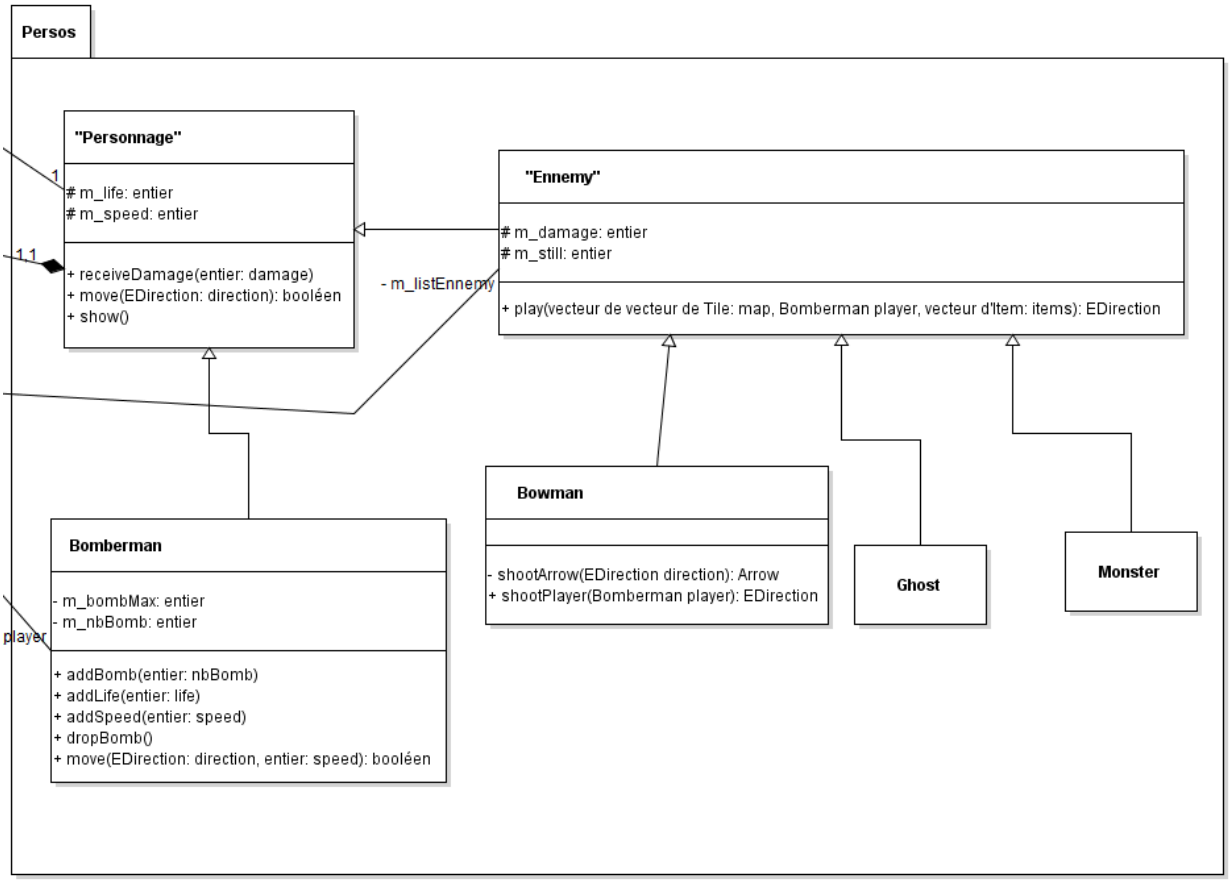

Figure 3 - Diagramme de classe du package Persos

Ici aussi nous retrouverons une classe « principale » qui est la classe Personnage, il s'agit d'une classe abstraite dont vont hériter toutes nos entités présentes sur la Map. Cette classe possède deux attributs protégés, un pour la vie et un autre pour la vitesse du Personnage. Pour ce qui est des méthodes et fonctions, nous aurons une fonction receiveDamage permettant d'infliger des dégâts passés en paramètre au Personnage, une fonction move permettant de déplacer le Personnage dans la direction passée en paramètre et retourne un booléen si le déplacement a été effectué ou non. Enfin la méthode show qui est notre méthode virtuelle pure permettra d'afficher le Personnage sur la Map.

En ce qui concerne le Bomberman, personnage principal du jeu, il est représenté par la classe Bomberman qui hérite de la classe Personnage. Un Bomberman possède deux attributs, un indiquant le nombre de bombe maximal que peut posséder le Bomberman et un indiquant le nombre de bombes possédées par le Bomberman. En ce qui concerne les fonctions et méthodes, nous avons la méthode

addBomb qui prend en paramètre le nombre de bombes à rajouter au Bomberman, la méthode addLife qui prend en paramètre le nombre de point de vie à rajouter, la méthode addSpeed qui prend en paramètre la vitesse à rajouter. Tandis que la méthode dropBomb permet de retirer une bombe au Bomberman. De plus, le Bomberman possède une surcharge de la méthode move prenant un paramètre supplémentaire qui est la vitesse à laquelle il doit aller lors de son déplacement.

Ensuite, la classe abstraite Ennemy va permettre de gérer quant à elle tous les ennemis de la Map. Elle possède alors les attributs m_damage et m_still, le premier indiquant les dégâts infligés par un ennemie et le deuxième le nombre de tour durant lesquelles l'ennemie ne peut pas jouer.

Enfin, nous disposerons des classes Ghost et Monster qui permettent de préciser le type d'ennemie ainsi que de redéfinir leur façon de jouer. La classe Bowman quant à elle définit un autre type d'ennemie qui peut lancer des flèches en direction du joueur, pour cela elle dispose des fonctions shootArrow et shootPlayer, la première permettant de lancer une flèche en direction du joueur et retournant la flèche lancée, la deuxième permettant de savoir si le joueur peut être visé par le Bowman et retourne la direction dans laquelle tiré la flèche.

4) Le package Map

Le package Map contiendra toutes nos classes se rapportant à la gestion de la Map et ce qu'elle peut posséder et qui ne peut être inclus dans un autre package.

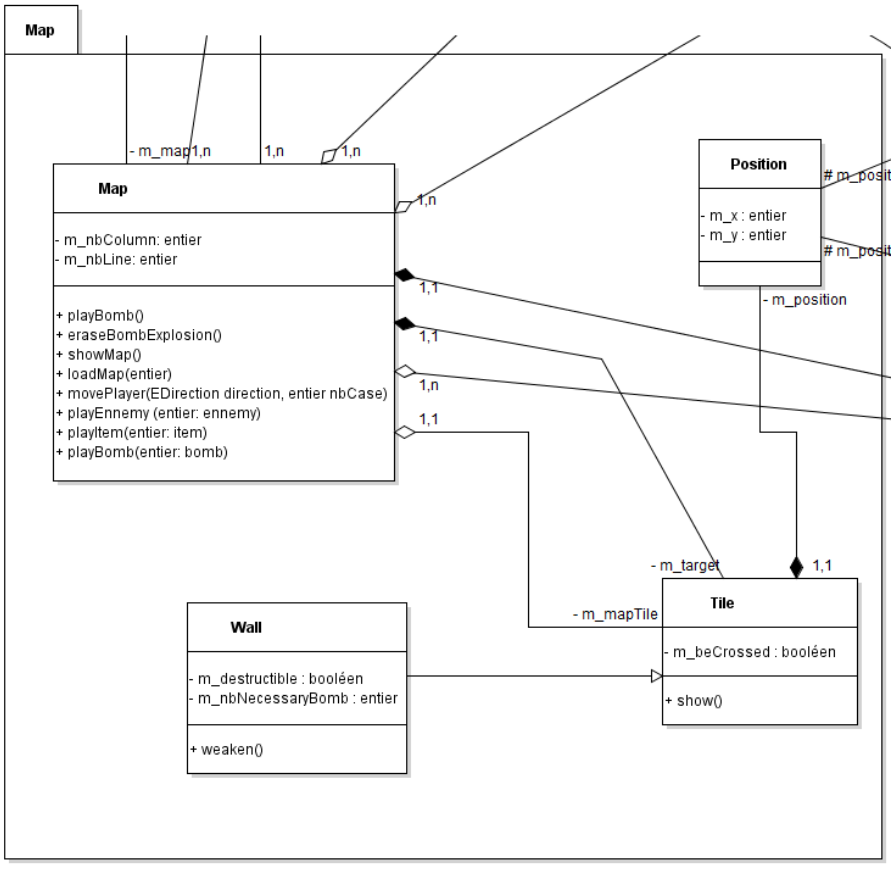

Figure 4 - Diagramme de classe du package Map

Dans ce package nous retrouverons la classe Position, permettant de définir la position de tous les éléments sur la Map. La Map étant en 2D, la position se calcule selon un axe X et Y qui seront nos attributs de la classe Position.

Ensuite, la classe Tile pour la définir simplement il s'agit d'une case de la Map, elle possède donc un attribut m_beCrossed permettant de savoir si la case peut être traversé ou si elle est occupée et est donc non traversable. Elle possède également une méthode show permettant de l'afficher, il s'agit d'un espace.

Quant à la classe Wall, elle hérite de la classe Tile, elle permet de gérer les murs de la Map qui ne sont franchissables que par un Ghost. Certains murs peuvent être détruit par le Bomberman et d'autres non, cela est représenté par l'attribut m_destructible et l'attribut m_nbNecessaryBomb qui permet de savoir combien de bombes sont nécessaire et cela indépendamment de leur puissance. La méthode weaken permet ainsi d'affaiblir le mur.

Enfin la classe Map est la classe qui nous permettra de gérer toute la Map ainsi que les entités, items et bombes présentes dessus. Pour ce qui est de sa représentation, il s'agira d'une Map en 2D avec pour taille un nombre de ligne et de colonne représentés par les attributs m_nbLine et m_nbColumn. Notre classe Map va également posséder plusieurs compositions et agrégations. Tout d'abord les compositions : Il s'agit du joueur et de l'objectif de la Map. Pour ce qui est des agrégations, elles seront surtout dû au fait qu'il ne s'agira pas d'une composition en C++ mais d'une agrégation interne. Pour cela, un vecteur de vecteur de Tile, un vecteur d'Ennemy, un vecteur de Bomb et un vecteur d'Item qui seront des agrégations. Etant donné qu'il est impossible de connaître leur taille à l'avance et pour des raisons de polymorphisme expliqué plus tard, il s'agira d'agrégation interne mais nous y reviendrons. Pour ce qui est des méthodes, nous aurons toutes les méthodes nécessaires pour faire jouer chaque entités, items et bombes, une méthode pour charger une map et une méthode pour afficher la map dans la console.

B. Diagramme de cas d'utilisation

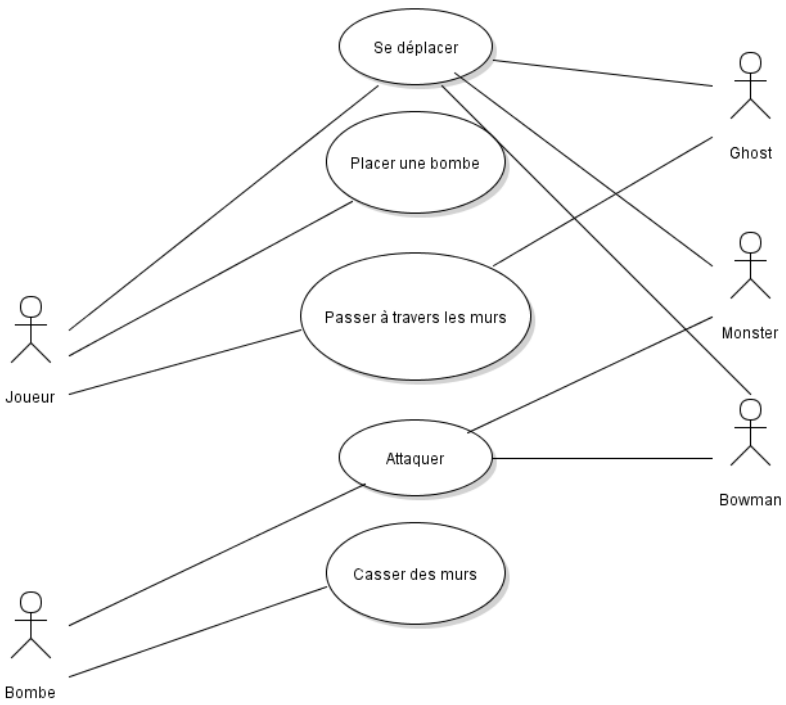

Figure 5 : Diagramme de cas d'utilisation

Avec ce diagramme des cas d'utilisation, nous pouvons constater qu'un joueur peut effectuer 3 actions, c'est-à-dire, il peut se déplacer, placer une bombe ainsi que passez à travers les murs.

Pour ce qui est de la Bombe, cette dernière peut attaquer ainsi que casser des murs. Pour le Monster et le Bowman, ils possèdent les mêmes actions, en effet, ils peuvent tous les deux attaquer et se déplacer. Tandis que le Ghost, peut se déplacer et passer à travers les murs mais il ne peut pas attaquer.

Et enfin pour ce qui des Mur, ils peuvent seulement faire apparaitre des items.

3. Développement du jeu

A. Agrégation ou composition dans la classe Map ?

L'une des classes les plus importantes et qui a demandé plus de travail et de réflexion par rapport aux autres est la classe Map. En effet, étant donné que c'est cette classe qui se charge de la gestion de tout ce qui se passe sur la map du jeu, elle possède beaucoup d'attributs et de méthodes par rapport aux autres.

Une des principales problématiques de cette classe fût la question : agrégation ou composition ? En effet, que choisir pour chaque attribut, puisque tous ne « vivront » que si la Map vit. Pour cela le choix le plus simple qui s'offrait à nous en C++ était d'en faire des agrégations pour les objets dont nous ne savions le nombre qui allait être implémenté. En effet, en C++ nous avons la possibilité de faire des agrégations internes, donc tant que notre objet Map vivra, ces agrégations vivront et pourront même être récupéré ailleurs. De plus, pour notre méthode qui va afficher la Map, comme nous stockons nos ennemies dans un vecteur d'Ennemy (classe abstraite) nous devons pouvoir appeler la bonne méthode show de ces ennemies. Ainsi pour utiliser au mieux le polymorphisme qui nécessite des pointeurs, l'agrégation interne était le meilleur moyen de faire.

B. Le polymorphisme

Comme nous l'avons vu, pour avoir du polymorphisme nous avons dû choisir pour plusieurs éléments une agrégation interne. En effet le polymorphisme est très important, notamment pour l'affichage de la map. Lors de l'affichage il faut que la méthode show de la classe Map appelle la bonne méthode d'affichage de chacun de nos éléments. Par exemple, étant donné que tous nos ennemies et qu'importent leur type sont stockés dans un vecteur d'Ennemy, il faut que lorsque l'on appelle la méthode show de la classe Ennemy l'affichage se fasse en fonction du type de l'ennemie, pour cela nous utilisons le polymorphisme en rendant la méthode show présente dans la classe Ennemy virtuelle (héritage de Personnage d'où provient la méthode show) et en la redéfinissant dans les classes héritant de la classe Ennemy. Ce même principe est utilisé pour les Wall et les Items.

4. Le fonctionnement du jeu

A. L'architecture des dossiers et fichiers

Comme nous l'avons dans la partie de modélisation, notre jeu du Bomberman est organisé en « package ». Etant donné que cette notion n'existe pas en C++, nous avons divisé notre projet dans plusieurs sous-dossiers pour en faciliter la compréhension et la lecture. Nous disposerons alors d'un dossier « engine » contenant la classe System et des éléments qui seront globalement nécessaire au projet comme l'énumération de direction, le dossier « Items » contenant tout ce qui concerne les items, le dossier « Map » qui contient tout ce qui est utile à la Map comme la classe Tile qui symbolise une case de la map et enfin le dossier « Persos » permettant de regrouper tous les fichiers concernant les entités présentes sur une Map.

Nous retrouverons cette subdivision en sous-dossier dans le dossier « include » et « src » correspondant respectivement aux fichiers « .h » et « .cpp » et se situant dans le dossier « src ». A ce même niveau, il y aura le fichier ProgBomberman permettant de lancer une partie de Bomberman à l'aide d'un menu, le fichier Makefile permettant de créer l'exécutable.

Enfin, le dossier « resources » contiendra un dossier « map » regroupant les différentes map pouvant être chargé.

B. Création de l'exécutable

Pour pouvoir jouer au jeu, il va falloir compiler notre programme à l'aide du fichier makefile présent dans le dossier en utilisant la commande « make ». Le makefile créera alors le fichier exécutable intitulé

« Bomberman ». Il suffira alors de lancer cet exécutable pour pouvoir jouer sur le Bomberman en console en utilisant la commande « ./Bomberman ».

C. Lancement d'une partie

Pour pouvoir lancer une partie de Bomberman une fois la commande « ./Bomberman » tapé, il vous faudra comme indiqué par le menu tapé le chiffre « 2 » et indiqué le numéro de la Map à charger (une évolution du projet serait de pouvoir afficher la liste des maps disponibles).

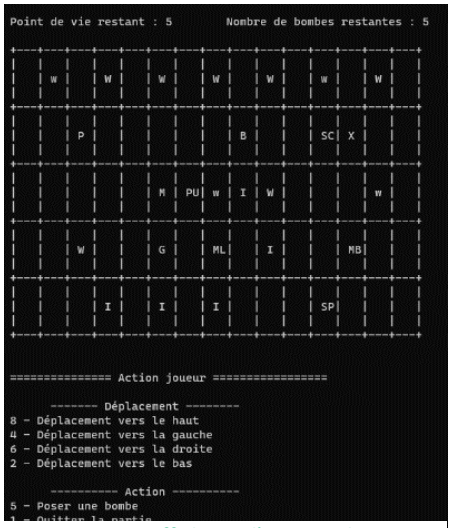

Figure 6 - Affichage d'un tour de jeu

Une fois le numéro de la map renseignée, le jeu se lance alors et l'affichage de la map se fait.

A chaque tour, le jeu s'affiche comme tel, le nombre de point de vie du joueur restant en haut à gauche, le nombre de bombes restantes en haut à droite. Ensuite vient l'affichage de la map.

Enfin, la liste des actions possibles par le joueur est alors proposée et il est invité au joueur de saisir son choix.

5. Difficultés rencontrées

A. L'affichage d'une Map

L'affichage d'une Map ne fût pas évident à réaliser. En effet, nos cases devant être de trois caractères par trois caractères, il ne fût pas simple au début de parcourir tous les vecteurs contenant les murs, les ennemies, les items et les bombes pour les afficher correctement sur la Map. Pour cela, nous parcourons notre vecteur de vecteur de Tile, chaque « case » de ce vecteur est alors parcouru trois fois (car la map possède une taille de ligne de trois) affichant ainsi au premier tour le haut de la case, au deuxième le milieu et au troisième le bas de la case. Grâce au polymorphisme la bonne méthode d'affichage pour chaque entités, mur et bombe est appelée.

B. La gestion de l'explosion des bombes

Une autre difficulté rencontrée lors du développement fût la gestion de l'explosion des bombes. En effet, lors de l'explosion des bombes il faut gérer quelles entités sont affectées par l'explosion ainsi que l'affichage de l'explosion. Pour ces raisons nous ne gérons que l'explosion individuelles de bombes, il est malheureusement impossible de réaliser des explosions en chaîne.

Pour gérer l'affectation des entités par l'explosion et son affichage nous nous aidons d'un vecteur de vecteur de l'énumération BombExplosionDirection. Grâce à cette énumération, nous pouvons savoir dans quelle direction on se situe sur chaque case par rapport à l'explosion ou s'il l'explosion ne concerne pas la case. Ainsi à l'aide de ce vecteur de vecteur, nous pouvons déterminer si une entité est touché par le bombe ou non et pour l'affichage le fait de savoir si on doit afficher l'explosion en ligne ou en colonne est simple et donné par l'énumération.

7. Conclusion

Pour conclure, ce projet fut très intéressant à mener à son terme et nous a demandé pas mal de temps.

Certes des choses pourraient être amélioré (un projet n'est jamais parfait) et bon nombre de fonctionnalités peuvent être rajouté. Cependant notre jeu Bomberman n'en reste pas moins fonctionnel et très jolie !

8. Annexes

A. Diagramme de classe globale

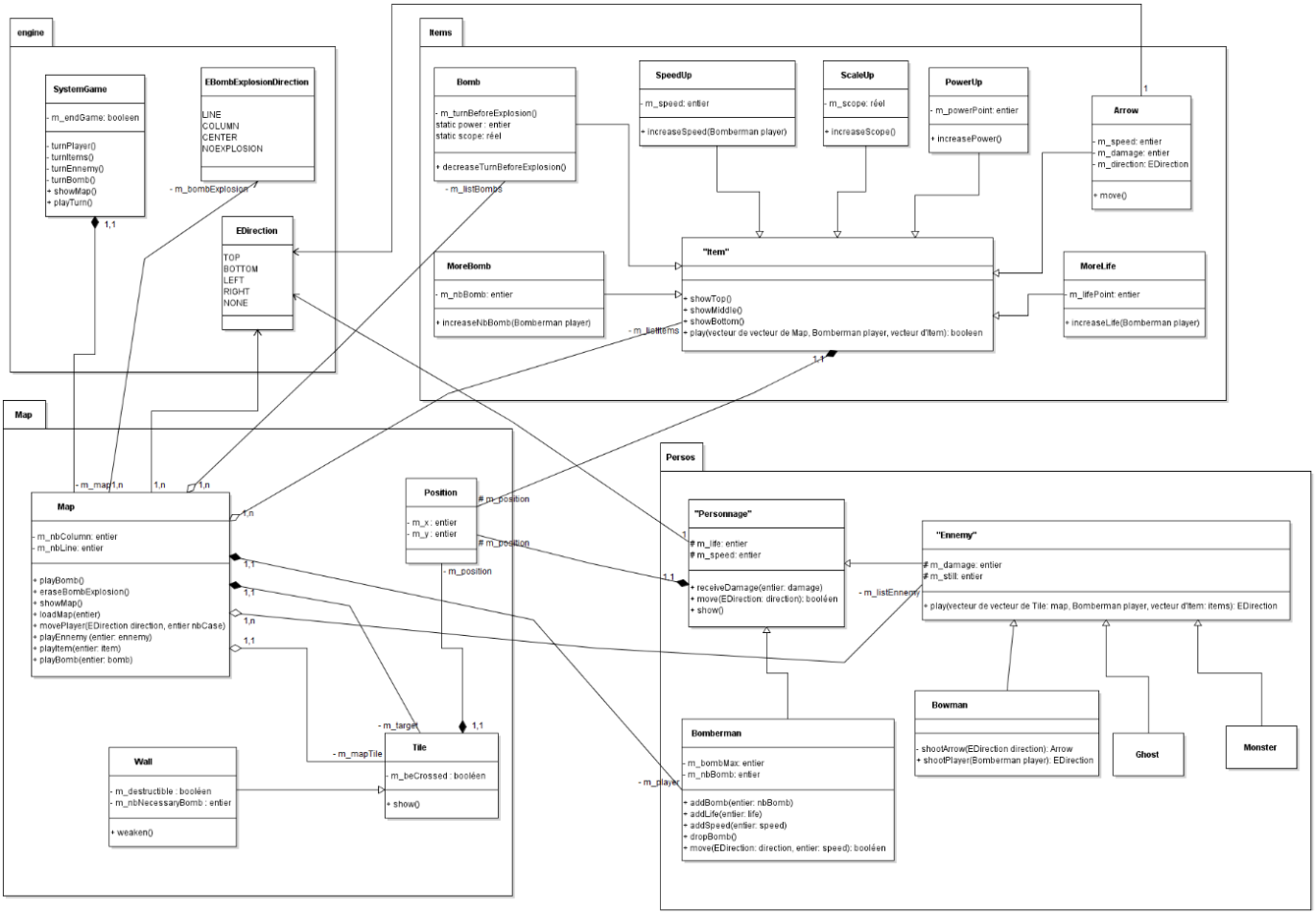

Figure 7 - Diagramme de classe globale