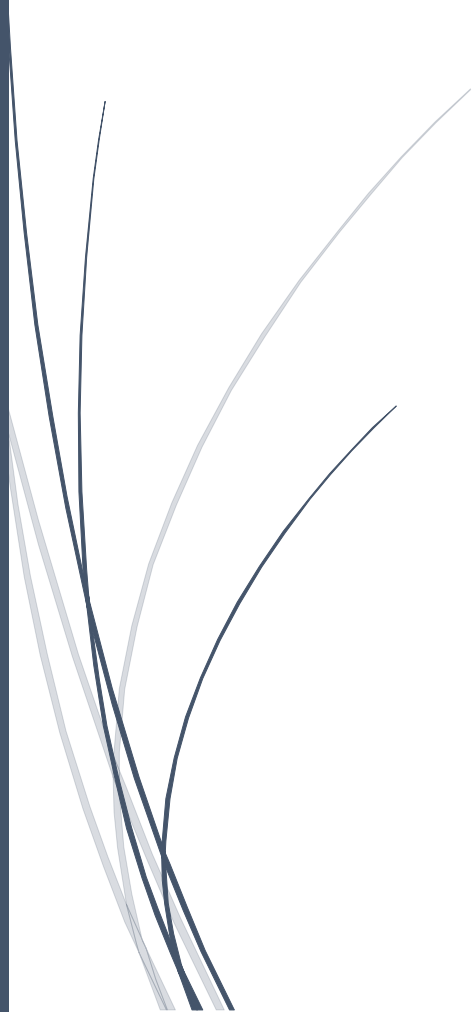
10/04/2022

Rapport projet Bomberman

DARVILLE Killian – SINET Théo – S4F4



Killian.darville@etudiant.univ-reims.fr
theo.sinet@etudiant.univ-reims.fr
UNIVERSITE DE REIMS CHAMPAGNE ARDENNES

Sommaire

Conception du projet

Présentation du projet

Ce projet consiste à développer le jeu Bomberman en C++ dans un terminal. Nous avons développé ce jeu en suivant les règles de base :

- Le but du jeu est de détruire tous les ennemis, puis d'atteindre l'objectif
- Le joueur ne peut, de base, déposer qu'une seule bombe à la fois.
    - Il doit attendre que la bombe ait explosé avant de poser une nouvelle bombe
- Il y a trois types d'ennemis disponibles
    1. Monster qui sont des ennemis de base.
    2. Ghost qui sont des ennemis qui peuvent traverser les murs.
    3. Bowman qui sont des ennemis pouvant attaquer à distance.
- Différents Item peuvent se trouver sur la carte afin d'améliorer les capacités du joueur. Seul le joueur récupère automatiquement un Item en allant sur sa case. Les différents Item sont:
    1. MoreLife augmente le nombre de points de vie de notre Bomberman
    2. PowerUp augmente la puissance des bombes
    3. MoreBomb augmente le nombre de bombes
    4. SpeedUp augmente la vitesse du Bomberman
    5. ScaleUp augmente la portée des bombes
- Les Item sont invincibles et ne disparaissent que lorsque le joueur le récupère
- Lorsqu'une bombe explose, elle touche les cases voisines horizontalement et verticalement sur une certaine portée. La portée n'est bloquée que par les murs
- Les bombes infligent des dégâts aussi bien aux ennemis, qu'au joueur
- Si une bombe touche une autre bombe, cette dernière explose aussi
- Enfin, certains murs peuvent être invincibles.

Description formelle du projet

Joueur est une classe qui a pour attributs priver :

1. hp qui est un entier
2. speed qui est un entier
3. position qui est un entier
4. nbBombMax qui est un entier
5. nbBombRestante qui est un entier
6. bombe qui est un pointeur de bombe

Joueur a pour méthodes publiques :

1. seDeplacer qui a pour argument un caractère et un entier et qui ne retourne rien.
2. poserBombe qui n'a pas d'argument et qui ne retourne rien.

Mob est une classe qui a pour attributs protéger :

1. hp qui est un entier
2. speed qui est un entier
3. damage qui est un entier
4. position qui est un entier
5. tourAttente qui est un entier
6. mort qui est un booléen

Mob a pour méthodes protéger:

1. getline qui a pour argument trois entiers et qui ne retourne un entier.
2. getline qui a pour argument deux entiers et qui ne retourne un entier.

Mob a pour méthodes publiques:

1. seDeplacer qui a pour argument un caractère et deux entiers et qui ne retourne rien.
2. attaquer qui a pour argument deux entiers et qui retourne un entier.
3. IsBowman qui ne prend pas d'argument et retourne un booléen.
4. PeutTirer qui prend en paramètre trois entiers et une chaînes de caractère et qui retourne un caractère
5. tirer qui prend en paramètre trois entiers et une chaînes de caractère et qui retourne un caractère

Monster est une classe qui hérite de Mob et qui a pour méthodes publiques :

1. isBowman qui ne prend pas de paramètre et retourne un booléen

Ghost est une classe qui hérite de Mob et qui a pour méthodes publiques :

1. isBowman qui ne prend pas de paramètre et retourne un booléen

Bowman est une classe qui hérite de Mob et qui a pour attributs priver :

1. nbFleche qui est un entier
2. arrow qui est un pointeur de Arrow

Bowman a pour méthodes publiques :

1. isBowman qui ne prend pas de paramètre et retourne un booléen
2. PeutTirer qui prend en paramètre trois entiers et une chaînes de caractère et qui retourne un caractère
3. tirer qui prend en paramètre trois entiers et une chaînes de caractère et qui retourne un caractère

Arrow est une classe qui a pour attributs priver :

1. degat qui est un entier
2. degat qui est un entier

3. touche qui est un booléen
4. direction qui est un caractère

Arrow a pour méthodes publiques :

1. seDeplacer qui prend en paramètre un entier et ne retourne rien.

Bombe est une classe qui a pour attributs priver :

1. puissance qui est un entier
2. portee qui est un entier
3. position qui est un entier
4. tourExplosion qui est un entier

Bombe a pour méthodes publique:

1. toucherPersonnage qui n'a pas d'argument et qui retourne un entier.
2. getline qui a pour argument deux entiers et qui retourne un entier.

Item est une classe qui a pour attributs priver :

1. position qui est un entier

Item a pour méthodes publique:

1. ajouterVie qui prend un Joueur en paramètre et ne retourne rien.
2. ajouterBombe qui prend un Joueur en paramètre et ne retourne rien.
3. augmenterVitesse qui prend un Joueur en paramètre et ne retourne rien.
4. augmenterPuissance qui prend un Joueur en paramètre et ne retourne rien.
5. augmenterPortee qui prend un Joueur en paramètre et ne retourne rien.

MoreLife est une classe qui hérite de Item et qui a pour méthodes publiques :

1. ajouterVie qui prend un Joueur en paramètre et ne retourne rien.

MoreBombe est une classe qui hérite de Item et qui a pour méthodes publiques :

1. ajouterBombe qui prend un Joueur en paramètre et ne retourne rien.

PowerUp est une classe qui hérite de Item et qui a pour méthodes publiques :

1. augmenterPuissance qui prend un Joueur en paramètre et ne retourne rien.

ScaleUp est une classe qui hérite de Item et qui a pour méthodes publiques :

1. augmenterPortee qui prend un Joueur en paramètre et ne retourne rien.

SpeedUp est une classe qui hérite de Item et qui a pour méthodes publiques :

1. augmenterVitesse qui prend un Joueur en paramètre et ne retourne rien.

Wall est une classe qui a pour attributs priver :

1. position qui est un entier
2. hp qui est un entier

Objectif est une classe qui a pour attributs priver :

3. position qui est un entier
4. hp qui est un entier
5. sortieValide qui est un booléen

Map est une classe qui a pour attributs priver :

1. nbligne qui est un entier
2. nbColonne qui est un entier
3. nbCaractere qui est un entier
4. tab qui est un tableau de chaîne de caractère

5. nomFichier qui est une chaîne de caractère.
6. emoji qui est un booléen
7. monFlux qui est un flux de stream
8. porteePossibleBombe qui est un tableau de quatres entiers
9. wall qui est un pointeur de Wall
10. nbWallMap qui est un entier
11. objectif qui est un pointeur d'objectif

Bombe a pour méthodes priver:

1. recupPorteeMaxBombe qui a pour argument deux entiers et un caractère et qui retourne un entier.
2. getline qui a pour argument trois entiers et qui retourne un entier.
3. getColonne qui a pour argument deux entiers et qui retourne un entier.
4. FlecheBloquer qui a pour argument un entier et un caractère et qui retourne un booléen.
5. RetirerFleche qui prend en paramètre un entier et qui ne retourne rien.

Bombe a pour méthodes publique:

1. affichageMap qui prend cinq entiers en paramètre et qui ne retourne rien.
2. ActualiserMap qui ne prend pas de paramètre et qui ne retourne rien.
3. LectureMap qui ne prend pas de paramètre et qui ne retourne rien.
4. InfoMap qui ne prend pas de paramètre et qui ne retourne rien.
5. AfficherPersoObjet qui prend en paramètre un entier et un caractère et qui ne retourne rien.
6. gestionDeplacement qui prend en paramètre deux entiers et qui ne retourne rien.
7. explosionBombe qui prend en paramètre deux entiers et qui ne retourne rien.
8. retirerExplosionBombe qui prend en paramètre un entier et qui ne retourne rien.
9. verifJoueurBloquer qui prend en paramètre un entier et qui retourne un booléen.
10. deplacementFleche qui prend en paramètre un entier et un caractère et qui ne retourne rien.
11. compteMurs qui ne prend pas de paramètre et qui retourne un entier.
12. allouMurs qui ne prend pas de paramètre et qui ne retourne rien.
13. positionObjectif qui ne prend pas de paramètre et qui ne retourne rien.

System est une classe qui a pour attributs priver :

1. map qui est un pointeur de Map
1. joueur qui est un pointeur de Joueur
1. mob qui est un pointeur de Mob
1. item qui est un pointeur d'Item
1. nbMobMap qui est un entier
1. map qui est un pointeur de Wall
1. objectif qui est un pointeur d'Objectif

System a pour méthodes priver:

1. joueurMort qui ne prend pas de paramètre et qui retourne un booléen.
2. mobMort qui ne prend pas de paramètre et qui retourne un booléen.
3. joueurItemFlecheMonstre qui prend en paramètre un entier et qui ne retourne rien.
4. gestionFlecheMob qui prend en paramètre un entier et qui ne retourne rien.
5. gestionFlecheJoueur qui prend en paramètre un entier et qui ne retourne rien.
6. gestionMoreLife qui prend en paramètre un entier et un booléen et qui ne retourne rien.
7. gestionMoreBomb qui prend en paramètre un entier et un booléen et qui ne retourne rien.
8. gestionPowerUp qui prend en paramètre un entier et un booléen et qui ne retourne rien.
9. gestionSpeedUp qui prend en paramètre un entier et un booléen et qui ne retourne rien.
10. gestionScaleUp qui prend en paramètre un entier et un booléen et qui ne retourne rien.
11. conversionDeplacementArrow qui prend en paramètre un caractère et qui retourne un entier.
12. conversionDirectionArrow qui prend en paramètre un caractère et qui retourne un caractère.
13. bombeInfligerDegats qui prend en paramètre un entier et une bombe et qui ne retourne rien.

14. mortParBombe qui prend en paramètre un entier et une bombe et qui ne retourne rien.
15. degatMurs qui prend en paramètre un entier et une bombe et qui ne retourne rien.
16. gagner qui prend ne prend pas de paramètre et qui retourne un booleén .

System a pour méthodes public:

1. afficherCarte qui prend en paramètre cinq entier et qui ne retourne rien.
2. demanderAction qui ne prend pas de paramètre et qui retourne un booléen.
3. deplacerMonstres qui prend en paramètre un entier et qui ne retourne rien.
4. exploserBombe qui prend en paramètre une bombe et qui ne retourne rien.
5. enleverExplosionBombe qui prend en paramètre un entier et qui ne retourne rien.
6. resetBomb qui prend en paramètre une Bombe et qui ne retourne rien.
7. actualiserCarte qui ne prend pas de paramètre et qui ne retourne rien.
8. verifFuturPosition qui prend en paramètre deux entiers, un caractère et une chaine de caractère et qui retourne un booléen.
9. nombreMobMap qui ne prend pas de paramètre et qui retourne un entier.
10. alloueMob qui ne prend pas de paramètre et qui ne retourne rien.
11. nombreItemMap qui ne prend pas de paramètre et qui retourne un entier.
12. alloueItem qui ne prend pas de paramètre et qui ne retourne rien.
13. alloueBomb qui ne prend pas de paramètre et qui ne retourne rien.
14. lirePositionJoueur qui ne prend pas de paramètre et qui ne retourne rien.
15. jouer qui prend en paramètre une chaine de caractère et qui ne retourne rien.

Nos diagrammes

Diagramme de classe

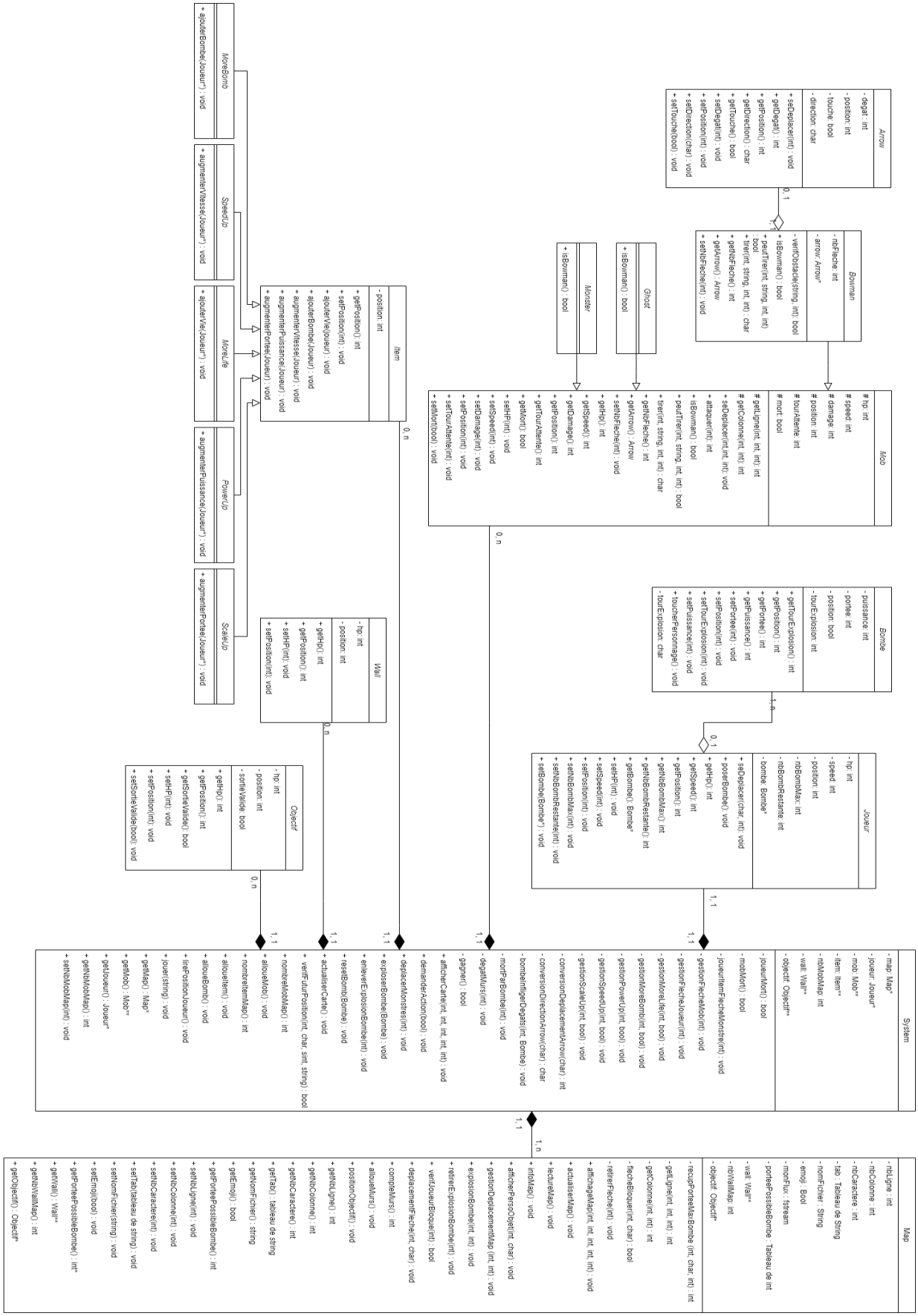


Diagramme des cas d'utilisations

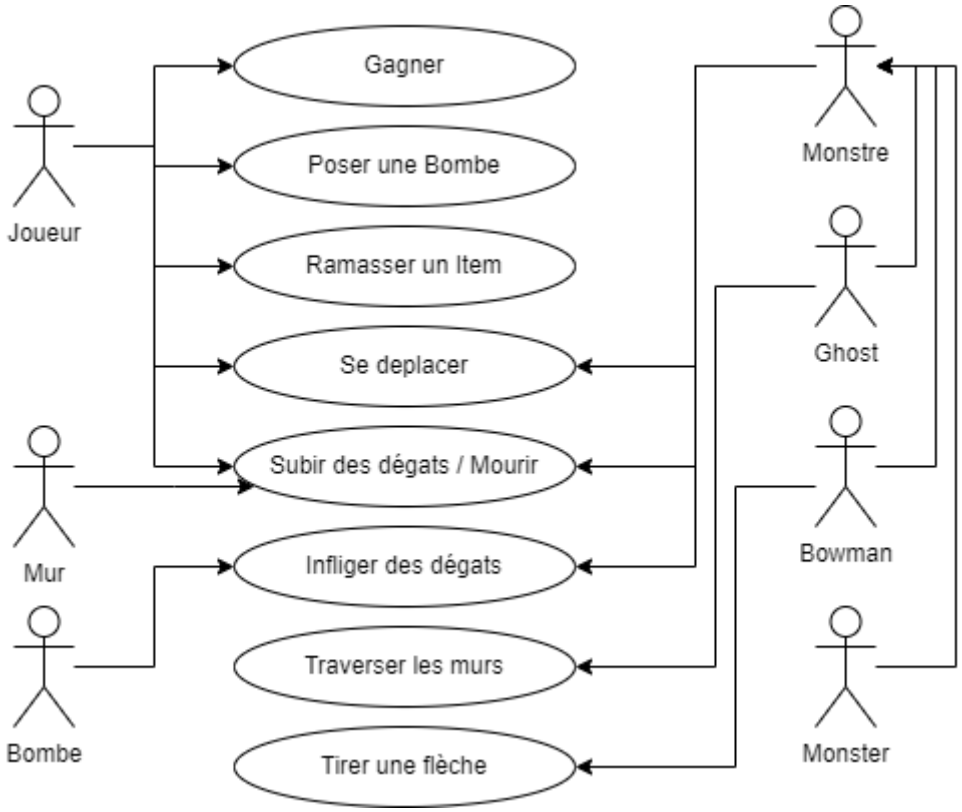


Diagramme d'activité

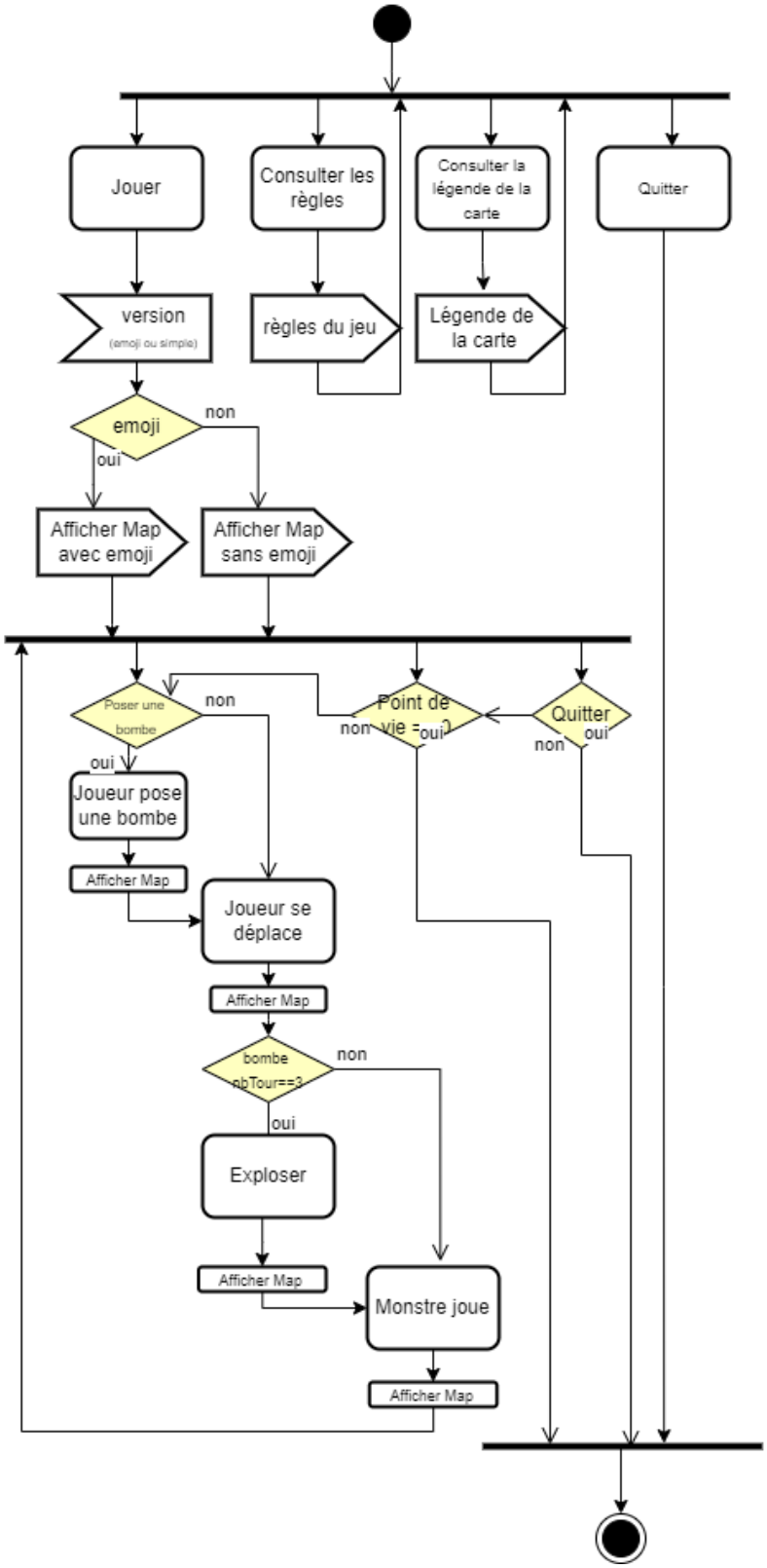
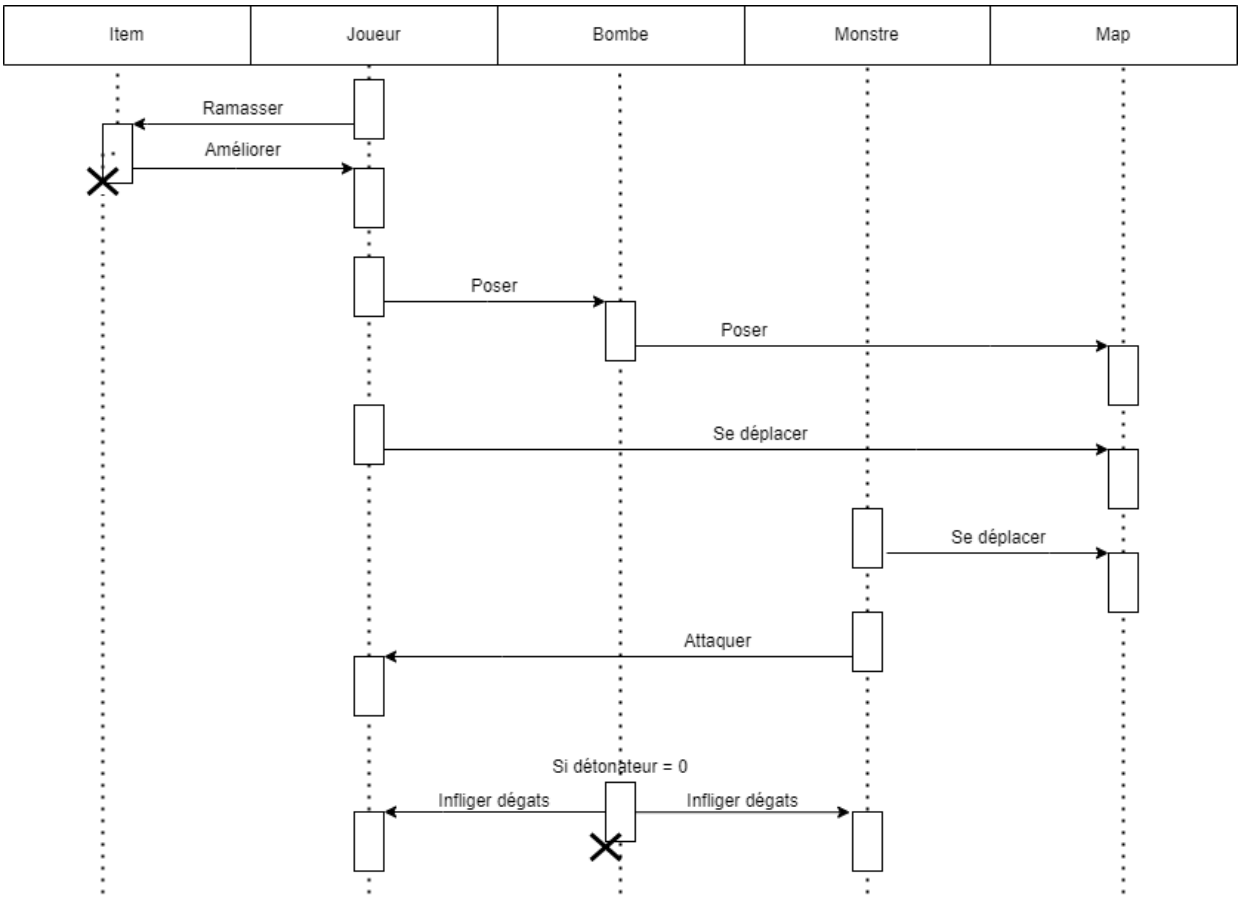


Diagramme de séquence



Le développement

Concepts de la programmation orienté objet et du C++ non triviaux

Nous allons traiter cette partie fichier par fichier.

Bowman.cpp :

peutTirer utilise *verifObstacle* pour savoir si, lorsque que le bowman est sur la même ligne ou colonne que le joueur, il va rencontrer un autre monstre, un mur.

Joueur.cpp

poserBombe permet de poser une bombe. Elle vérifie si la bombe que l'on consulte (puisque le joueur à un tableau de bombe) est déjà placé ou non. Si une bombe n'a pas été positionné on la pose et on réduit de un le nombre de bombe restante au joueur.

Map.cpp

affichageMap sert à afficher la carte ainsi que le nombre de points de vie, de bombe maximum, de vitesse, de portée et puissance des bombes. Cette grande méthode gère tous les affichages et utilise aussi l'attribut « emoji » de sa classe pour donner un affichage plus esthétique avec des emojis. La version emoji ne fonctionne pas sur tous les terminaux.

lectureMap ouvre et lis le fichier texte de la carte puis la stocke dans un tableau de string.

actualiserMap avait pour but de réécrire le tableau de string précédent dans le fichier texte afin de mettre en place un système de sauvegarde. Toutefois elle n'est pas utilisé puisque nous avons manqué de temps pour mettre en place la fonctionnalité.

infoMap lis le fichier texte de la carte pour initialiser les valeurs des attributs de la classe Map.

afficherPersoObjet modifie une case du tableau contenant la carte pour y afficher une flèche ou une bombe.

gestionDeplacementMap déplace le joueur ou les monstres dans le tableau de la carte. On vérifie la futur position de l'entité pour afficher correctement les différentes entités qui peuvent se trouver dans la case de la prochaine position. Cette fonction permet aussi de séparer, au moment d'un déplacement, plusieurs entités qui étaient sur la même case .

getLigne calcule la ligne de l'entité passée en paramètre par rapport au tableau à une dimension de la carte. Cette fonction sert pour connaitre la ligne de la bombe afin de savoir plus tard, dans une autre fonction, si l'explosion de la bombe dépasse de la ligne

getColonne calcul la colonne de l'entité passée en paramètre par rapport au tableau à une dimension de la carte. Ici cette fonction sert pour connaitre la colonne de la bombe afin de savoir plus tard si l'explosion de la bombe dépasse de la colonne

recupPorteeBombe teste la portée possible des bombes dans les quatre directions. On y utilise *getLigne* et *getColonne* et on regarde si l'explosion de sort pas de la carte ou si elle ne rencontre pas un mur. Si l'un de ces cas se produit, on limite la portée de la bombe à une portée réellement possible. La vérification se fait pour dans les quatre directions et sera récupérer dans un tableau de quatre cases dans une autre fonction.

explosionBombe permet d'afficher les traits de l'explosion selon la bonne portée possible récupéré grâce à la fonction *recupPorteeBombe*. Chaque case du tableau contient la portée d'une des directions pour afficher correctement de chaque côtés de la bombe.

retirerExplosionBombe utilise l'attribut correspondant au tableau contenant la portée dans les quatre directions pour retirer ce qu'il faut concernant les traits de l'explosion.

verifJoueurBloque vérifie si le joueur n'est pas bloqué dans les quatre directions. S'il l'est, on retournera l'information pour que l'on puisse lui faire passer son tour.

deplacementFleche fait avancer la flèche dans le tableau de la carte si la flèche n'est pas bloqué par un monstre, joueur ou mur.

flecheBloquer permet de retourner si la flèche est bloqué par un monstre, joueur, un mur ou encore si elle ne sort pas de la carte. Si elle est bloquée, la fonction en appelle une autre pour faire disparaitre la flèche.

Mob.cpp

getLigne permet d'abord de récupérer la ligne du monstre et du joueur par rapport au tableau de la carte. Le calcul est simple, il vérifie si la position passé se trouve entre les deux extrémités d'une ligne, et si oui on retourne la ligne. Le premier usage de cette fonction concerne l'algorithme permettant de faire se déplacer les monstres. Ensuite, cette fonction est réutilisé pour déplacer les flèches des bowman.

getColonne permet de récupérer la colonne du monstre et du joueur par rapport au tableau de la carte. Le calcul correspond simplement à un modulo de la position par le nombre de colonne du tableau. Le premier usage de cette fonction concerne l'algorithme permettant de faire se déplacer les monstres. Ensuite, cette fonction est réutilisé pour déplacer les flèches des bowman.

MoreBomb.cpp

ajouterBombe ne fait rien de compliqué mais il y a une petite particularité du fait que lorsqu'un augment le nombre de bombe, on réalloue la taille du tableau de bombe du joueur. Ainsi, on perd toutes gestion des bombes qui étaient posés avant de passé par cette fonction. De ce fait on stock avant de réallouer les informations des bombes dans un variable temporaire pour ensuite, une fois réalloué, recopier dans le nouveau tableau les anciennes bombes.

System.cpp

joueurItemFlecheMonstre cette grande fonction vérifie chaque cas possible de rencontrer sur une case entre ces quatre entités que sont le joueur, les items, les flèches et les monstres. Par exemple, lorsqu'il y a un item, le joueur doit pouvoir le récupérer en plus de se prendre des dégâts si un monstre ou une flèche arrivent ou sont présents dans la case de l'item. La vérification se fait par rapport au tableau de la carte. L'affichage est géré en conséquence de l'événement qui se passe dans la case. Ainsi, si le joueur ou un monstre meurt, il disparait de la case et donc par la même occasion de la carte.

gestionFlecheJoueur permet premièrement d'infliger les dégâts de la flèche au joueur qui se la prend. Ensuite, elle vérifie aussi qu'elle bowman à tirer cette flèche afin de le bloquer pendant trois tours

gestionFlecheMob à le même usage que *gestionFlecheJoueur* mais pour les monstres.

gestionMorelife, *gestionMoreBomb*, *gestionSpeedUp*, *gestionPowerUp*, *gestionScaleUp* vérifient toutes les cinq si le joueur à bien marché sur l'item en vérifiant leurs position. Si c'est le cas, on applique le bonus correspondant à l'item.

exploserBombe permet d'abord d'afficher le caractère de la bombe qui explose en plus de ce qu'il pouvait y avoir sur la case au moment de l'explosion. Ensuite, on fait l'affichage des traits de la portée de la bombe puis, on inflige des dégâts à tout ce que se trouve dans la portée des quatre directions.

degatMurs change l'état du mur en préservant le fait qu'il puisse y avoir un ghost en vie dedans.

mortParBombe fait disparaitre l'entité morte suite à l'explosion de l'affichage en préservant l'affichage suite à la disparition de l'entité.

verifFuturPosition vérifie si l'entité peut se déplacer dans la case dans laquelle il veut se rendre. Par exemple, si le joueur avance dans un mur, on lui bloque l'accès et il rejoue.

conversionDeplacementArrow : convertie l'attribut de direction d'une flèche en une unité de déplacement pour faire bouger la flèche.

jouer permet d'allouer tout ce qu'il nécessite de l'être et instaure le tour par tour du jeu.

Documentation Doxygen

Nous avons fait la documentation, nous vous invitons à consulter le pdf ci-joint.