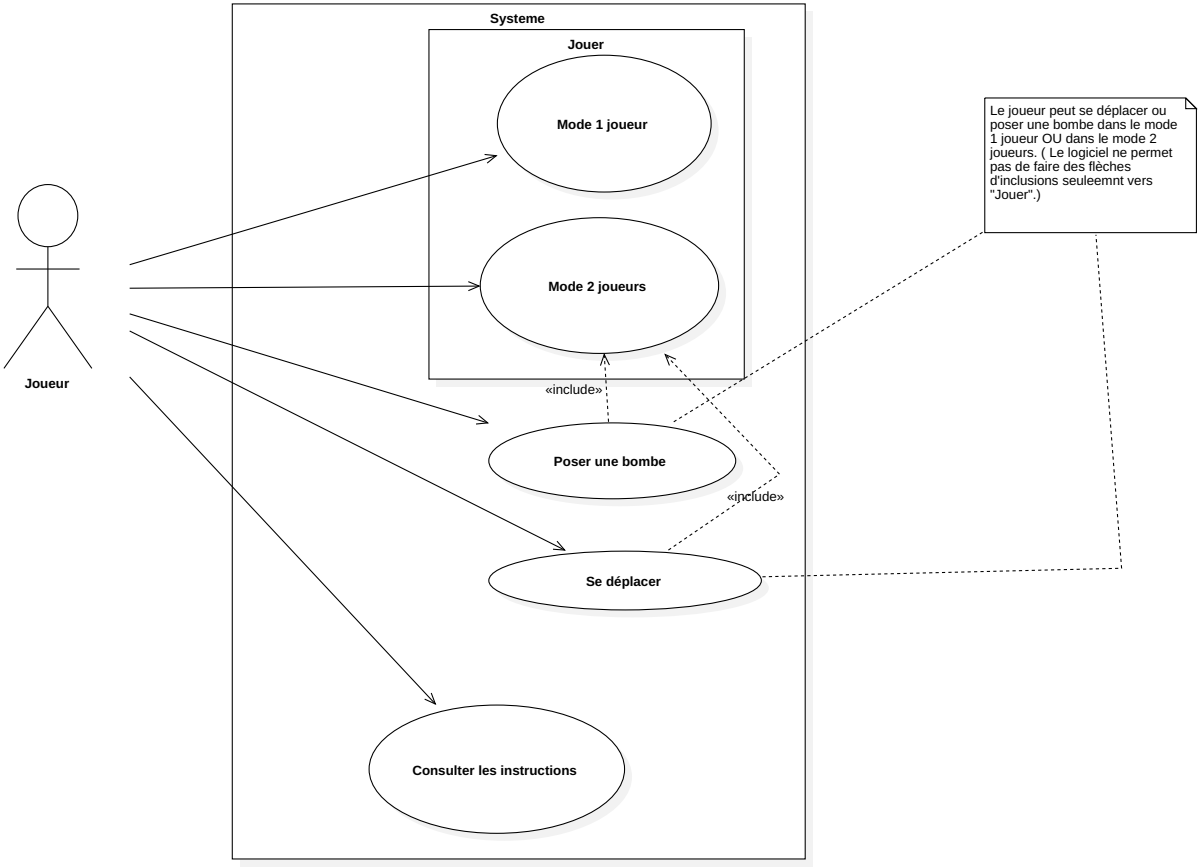
Contexte

De nos jours, les applications Android ne cessent d'évoluer, proposant des jeux avec des graphismes de plus en plus évolué. Nous avons décidé de retourner à un classique : Bomberman. Ce dernier étant destiné aux personnes de toutes âge, il est jouable par tous.

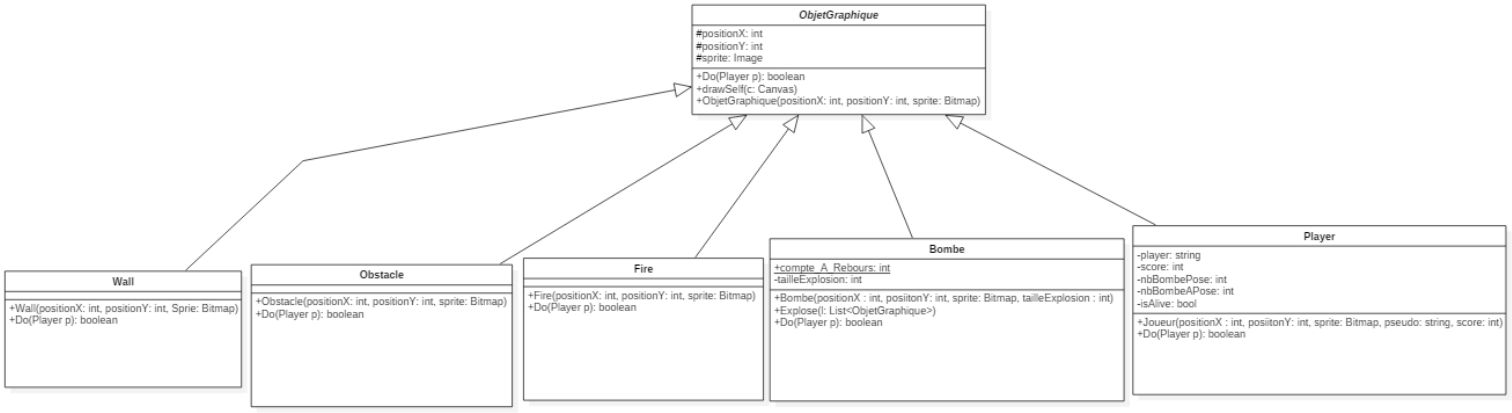
Ainsi, le joueur incarne un poseur de bombes, le but étant de faire exploser les adversaires ou ennemis pour gagner. Notre jeu propose un mode "Un joueur" et un mode "Deux joueurs" où deux joueurs s'affrontent en Bluetooth, le but étant d'exploser son adversaire. Le joueur peut exploser les murs de type obstacle mais pas les murs solides. Les bombes explosent au bout d'un certain temps. Mais attention, il est impératif de s'écarter avant que celles-ci explosent. En effet, le feu produit par ces bombes est très dangereux et pourrait vous faire perdre la partie !!!

Model::UseCaseDiagram1

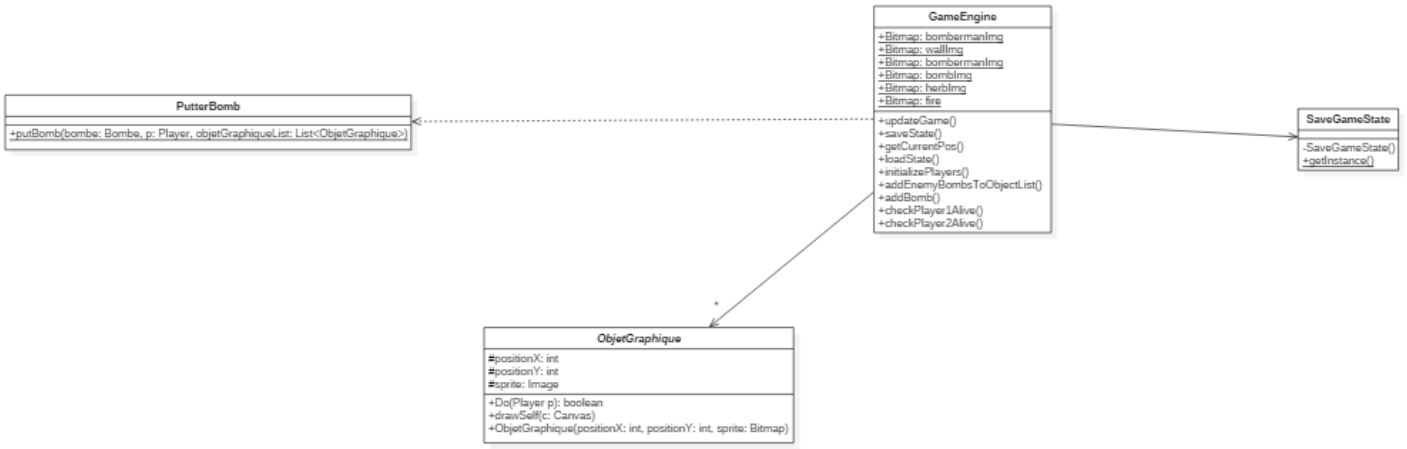


CONCEPTION

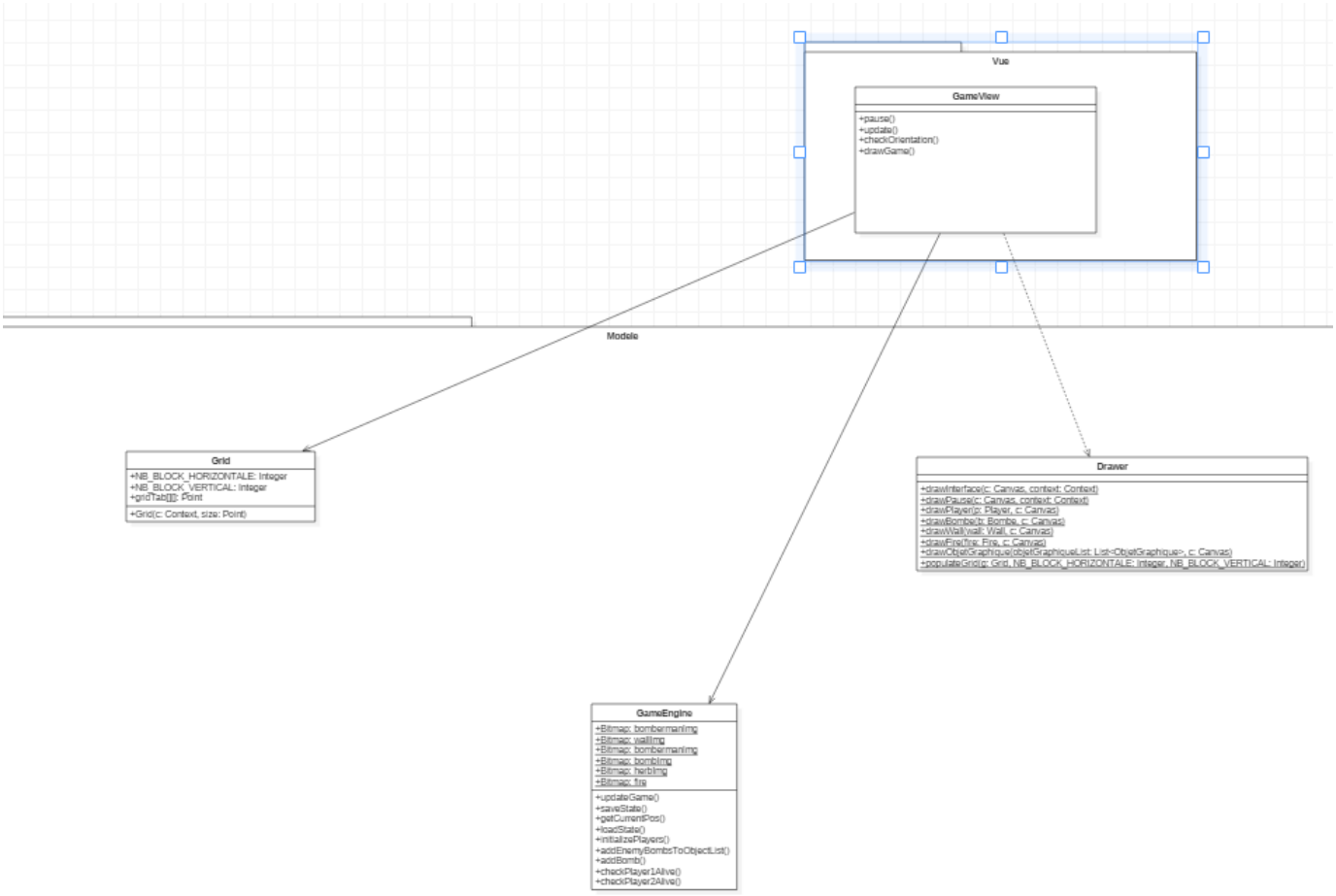
Description du diagramme de classe :



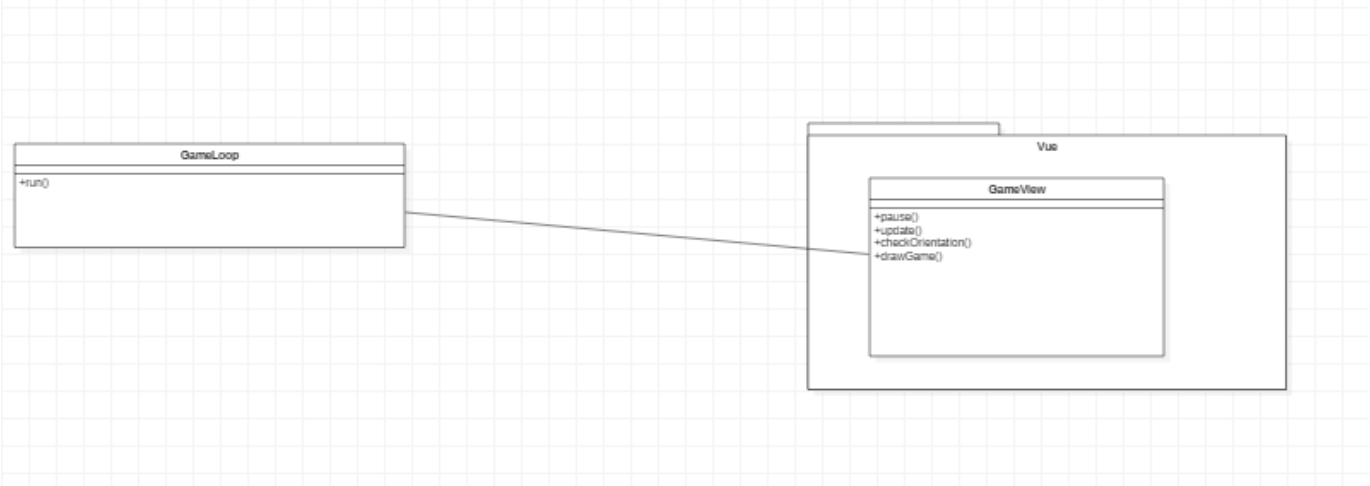
Cette partie du diagramme décrit les éléments graphiques de notre projet. Notre application possède donc cinq objets graphique différents : un mur, un obstacle, le feu, une bombe et un joueur. Ces derniers possèdent chacun un position X et Y ainsi qu'un sprite qui va définir son apparence graphique. La classe ObjetGraphique est une classe abstraite, non-instanciable. Elle possède notamment la méthode Do qui nous redéfinissons dans chacune de ses filles, définissant leur comportement lorsque Bomberman (le personnage) est en collision avec. Elle renvoie un booléen qui est à true lorsque le héros possède une collision avec l'objet (c'est-à-dire qu'il est bloqué quand ils se touchent) et false quand il n'en a pas (il peut traverser l'objet).



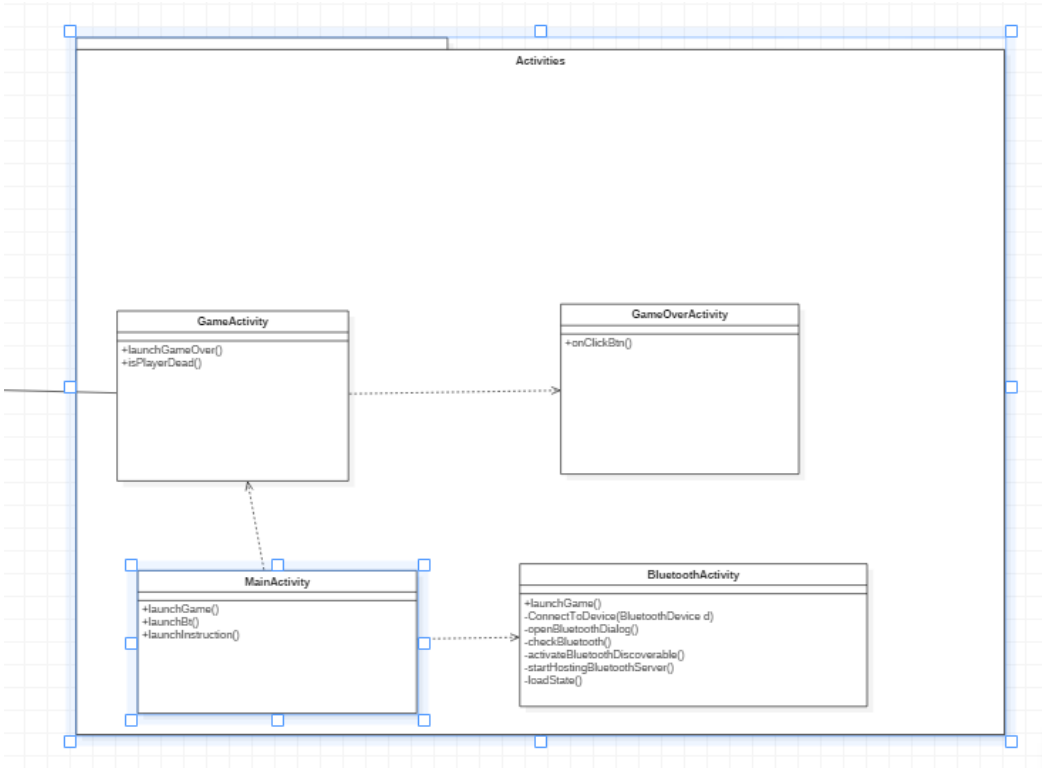
La classe GameEngine constitue notre moteur de jeu. C'est elle qui fait l'intermédiaire entre le modèle et la vue. Elle possède une liste d'ObjetGraphique représentant tous les objets graphiques à représenter dans notre Grille. Elle appelle la classe SaveGameState qui est chargé d'effectuer la sauvegarde légère de notre application. La classe GameEngine initialise tous les éléments à placer de notre jeu. Lorq'elle reçoit l'événement « Bombe à poser », elle va appeler PutterBombe afin d'insérer la bombe au sein de la liste d'objetGraphique (tout en vérifiant que le joueur n'a pas poser plus de X bombes).



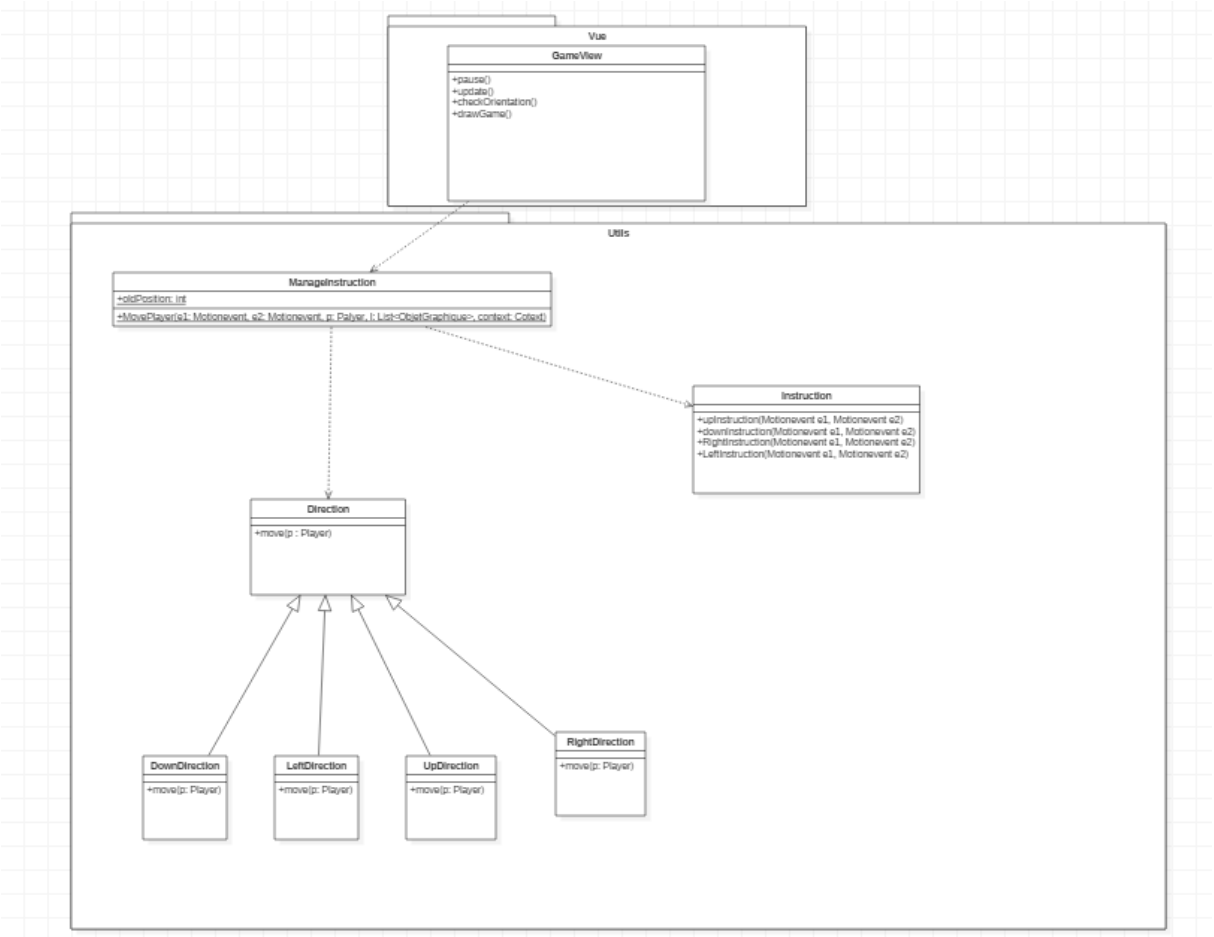
La classe GameView représentée ci-dessus constitue notre vue principale lorsque le joueur joue à notre jeu. Elle possède une Grid implémentée par nous-même. Les constantes NB_BLOCK_HORIZONTALE et NB_BLOCK_VERTICAL sont des constantes qui définissent respectivement le nombre de block horizontal et vertical. Ainsi, nous pouvons placer très facilement un objet graphique dans notre vue en spécifiant sa position verticale et horizontal. La fonctionnalité principale de la classe Drawer est de dessiner chaque élément de notre grille avec le sprite qui convient. Toutes ses méthodes sont statiques afin que la classe GameView n'est pas besoin d'instancier Drawer lorsque l'on veut dessiner un élément graphique.



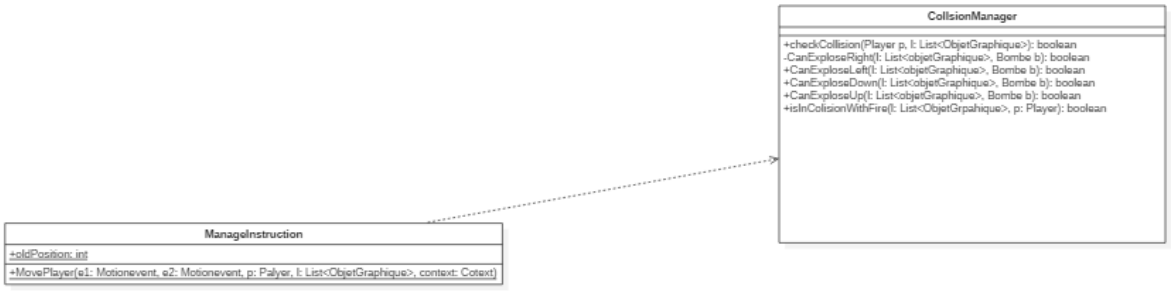
La classe GameLoop constitue la boucle de notre jeu. Elle met à joueur la vue en fonction du modele 60 fois par seconde. La flèche entre la GameLoop et la GameView est bidirectionnelle puisque GameLoop a besoin de connaître la vue afin de la mettre à jour et GameView a besoin de connaitre la boucle afin de la mettre en pause lorsque notre téléphone n'est plus en mode paysage par exemple (checkOrientation).



Voici le package Activity de notre application. L'activité principale (MainActivity) appelle GameActivity et Bluetooth activité lorsqu'elle reçoit l'événement correspondant à telle ou telle activité. GameActivity appelle GameOverActivity lorsque un des joueurs est mort.

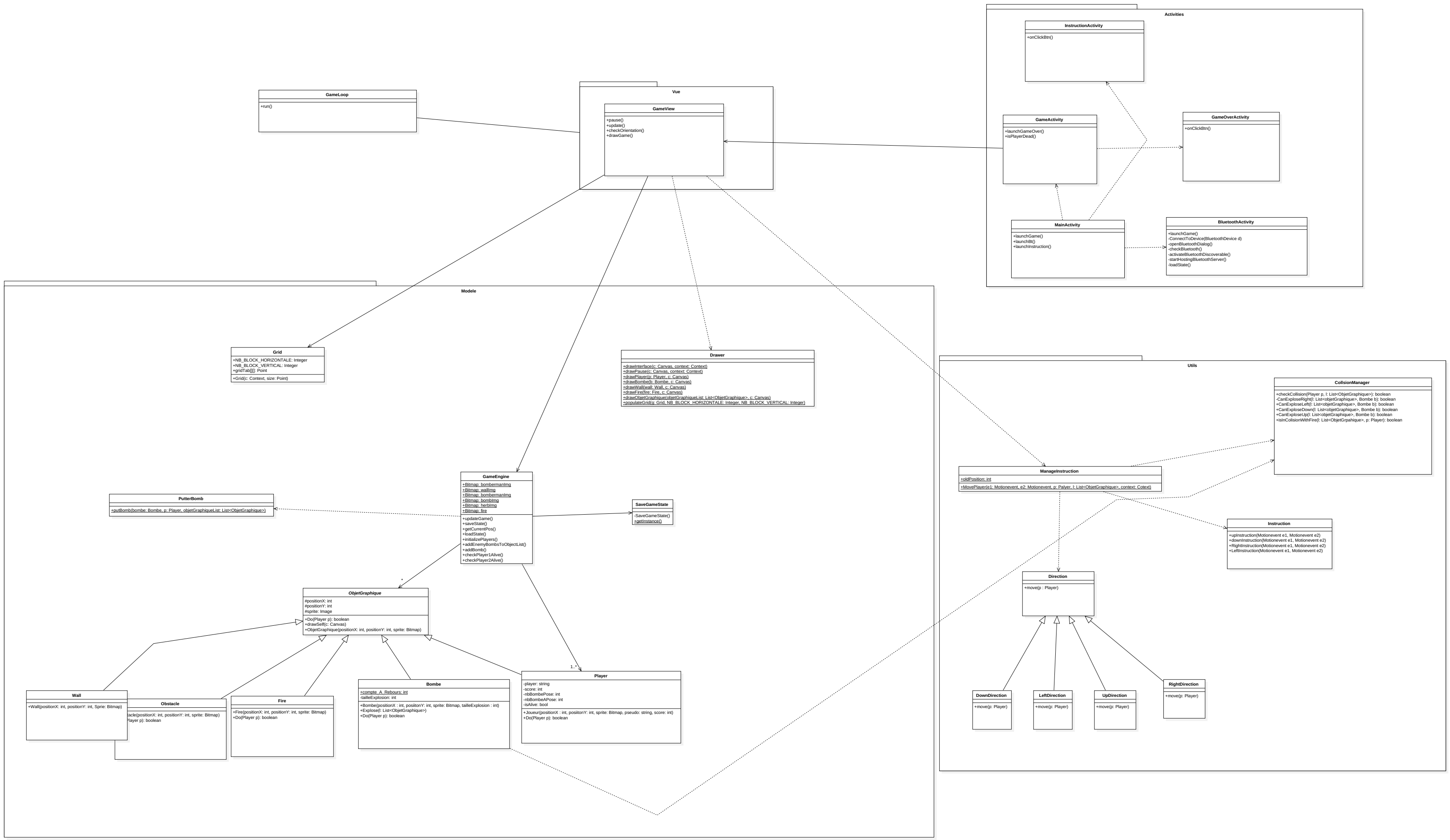


Comme nous pouvons le voir dans le diagramme ci-dessus, la classe ManageInstruction va gérer les mouvements du personnage en fonction de ce que l'utilisateur va effectuer. La classe direction possède 4 filles correspondant chacune d'elle à une direction différente. ManageInstruction va les appeler en fonction de l'instruction définit par la classe Instruction. ManageInstruction va donc relayer cette information à GameView afin qu'il effectue la mise à jour de la vue.



Comme nous pouvons le voir ci-dessus, la classe ManageInstruction va demander à la méthode statique checkCollision si le personnage est en collision avec un objet graphique. Si c'est le cas, il n'effectue pas l'instruction demandé par l'utilisateur.

Model::Main


Vue
GameView
+pause()
+update()
+checkOrientation()
+drawGame()
GameLoop
+run()
Activities
InstructionActivity
+onClickBtn()
GameActivity
+launchGameOver()
+isPlayerDead()
GameOverActivity
+onClickBtn()
MainActivity
+launchGame()
+launchBt()
+launchInstruction()
BluetoothActivity
+launchGame()
-ConnectToDevice(BluetoothDevice d)
-openBluetoothDialog()
-checkBluetooth()
-activateBluetoothDiscoverable()
-startHostingBluetoothServer()
-loadState()
Modele
Grid
+NB_BLOCK_HORIZONTALE: Integer
+NB_BLOCK_VERTICAL: Integer
+gridTab[][]: Point
+Grid(c: Context, size: Point)
Drawer
+drawInterface(c: Canvas, context: Context)
+drawPause(c: Canvas, context: Context)
+drawPlayer(p: Player, c: Canvas)
+drawBombe(b: Bombe, c: Canvas)
+drawWall(wall: Wall, c: Canvas)
+drawFire(fire: Fire, c: Canvas)
+drawObjetGraphique(objetGraphiqueList: List<ObjetGraphique>, c: Canvas)
+populateGrid(g: Grid, NB_BLOCK_HORIZONTALE: Integer, NB_BLOCK_VERTICAL: Integer)
PutterBomb
+putBomb(bombe: Bombe, p: Player, objetGraphiqueList: List<ObjetGraphique>)
GameEngine
+Bitmap: bombermanImg
+Bitmap: wallImg
+Bitmap: bombermanImg
+Bitmap: bombImg
+Bitmap: herbImg
+Bitmap: fire
+updateGame()
+saveState()
+getCurrentPos()
+loadState()
+initializePlayers()
+addEnemyBombsToObjectList()
+addBomb()
+checkPlayer1Alive()
+checkPlayer2Alive()
SaveGameState
-SaveGameState()
+getInstance()
*
ObjetGraphique
#positionX: int
#positionY: int
#sprite: Image
+Do(Player p): boolean
+drawSelf(c: Canvas)
+ObjetGraphique(positionX: int, positionY: int, sprite: Bitmap)
1..*
Wall
+Wall(positionX: int, positionY: int, Sprie: Bitmap)
Obstacle
tacle(positionX: int, positionY: int, sprite: Bitmap)
Player p): boolean
Fire
+Fire(positionX: int, positionY: int, sprite: Bitmap)
+Do(Player p): boolean
Bombe
+compte_A_Rebours: int
-tailleExplosion: int
+Bombe(positionX : int, posiitonY: int, sprite: Bitmap, tailleExplosion : int)
+Explose(l: List<ObjetGraphique>)
+Do(Player p): boolean
Player
-player: string
-score: int
-nbBombePose: int
-nbBombeAPose: int
-isAlive: bool
+Joueur(positionX : int, posiitonY: int, sprite: Bitmap, pseudo: string, score: int)
+Do(Player p): boolean
Utils
CollsionManager
+checkCollision(Player p, l: List<ObjetGraphique>): boolean
-CanExploseRight(l: List<objetGraphique>, Bombe b): boolean
+CanExploseLeft(l: List<objetGraphique>, Bombe b): boolean
+CanExploseDown(l: List<objetGraphique>, Bombe b): boolean
+CanExploseUp(l: List<objetGraphique>, Bombe b): boolean
+isInColisionWithFire(l: List<ObjetGrpahique>, p: Player): boolean
ManageInstruction
+oldPosition: int
+MovePlayer(e1: Motionevent, e2: Motionevent, p: Palyer, l: List<ObjetGraphique>, context: Cotext)
Instruction
+upInstruction(Motionevent e1, Motionevent e2)
+downInstruction(Motionevent e1, Motionevent e2)
+RightInstruction(Motionevent e1, Motionevent e2)
+LeftInstruction(Motionevent e1, Motionevent e2)
Direction
+move(p : Player)
DownDirection
+move(p: Player)
LeftDirection
+move(p: Player)
UpDirection
+move(p: Player)
RightDirection
+move(p: Player)