Dossier du Projet

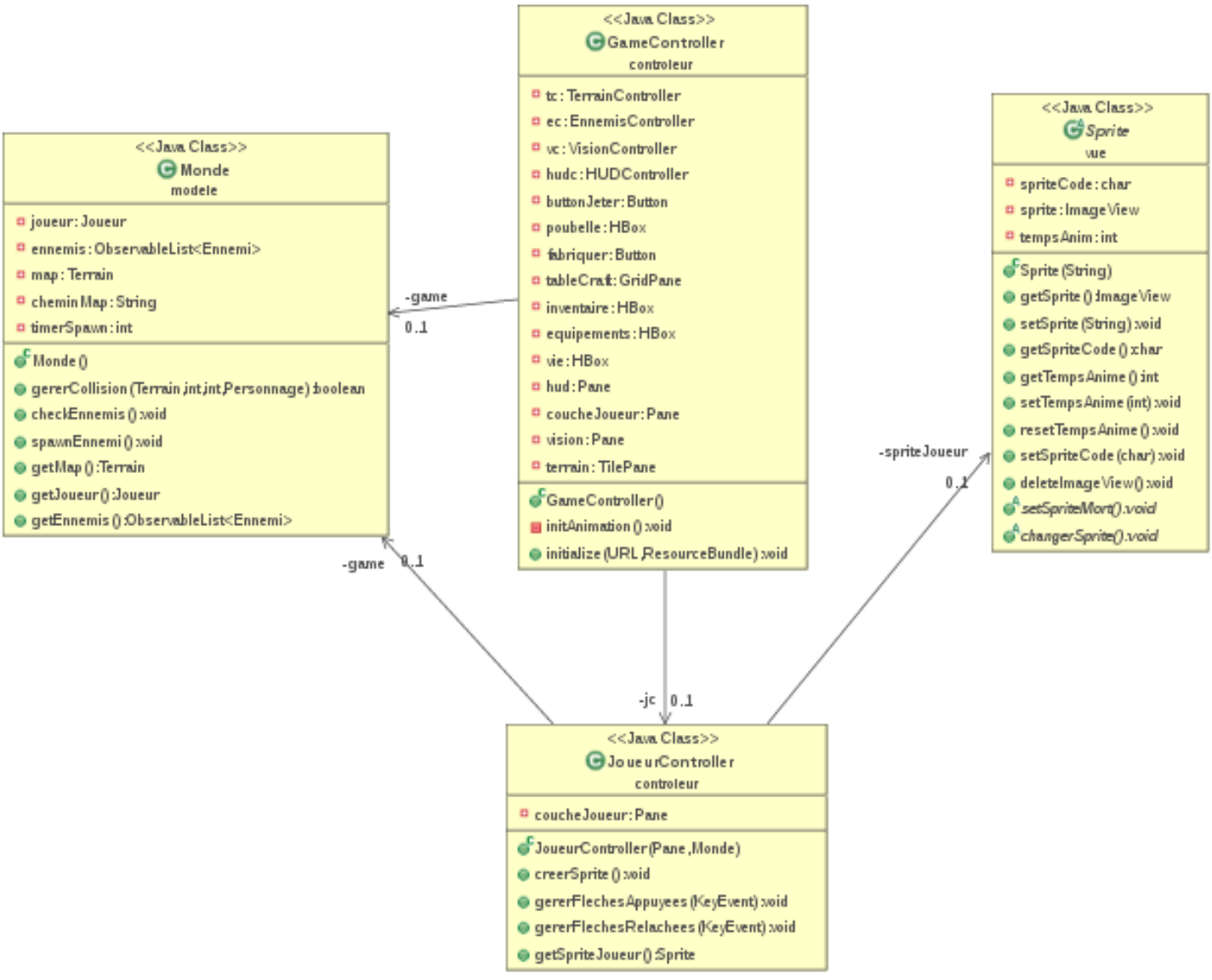
GROUPE : **CHASTEL Quentin, BRUNEAU Oscar, RYNKIEWICZ Paul**

1:Documents pour CPOO

Architecture (5/60 points)

Architecture MVC :

L'architecture MVC du jeu est la suivante. Le GameController (qui regroupe plusieurs controleurs comme le JoueurController) fait le lien entre le monde qui est la classe principale du modèle et la vue avec les Sprites, et d'autres éléments javaFX.

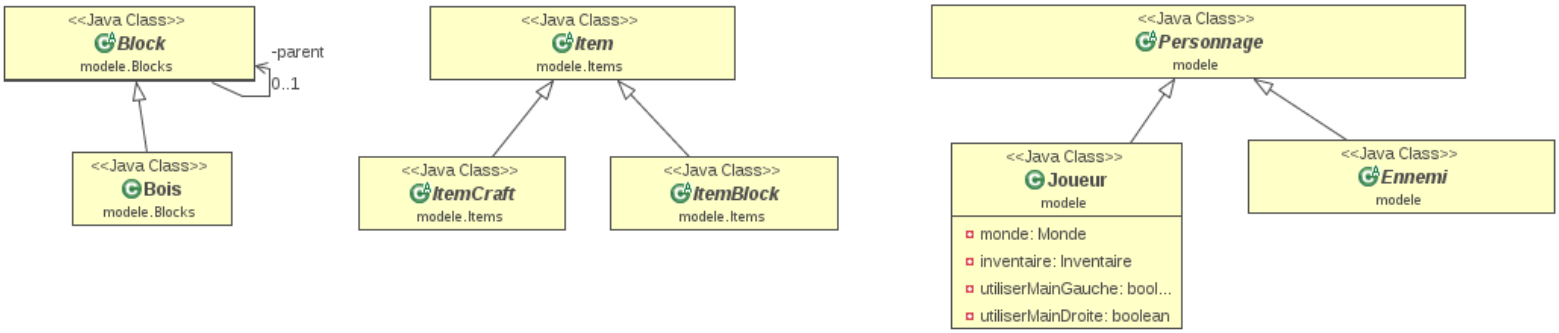


Les briques élémentaires du modèle :

Les blocks et les items sont des objets distincts.
Les blocks seront tangibles, affichés sur le terrain et ramassables par le joueur. Les blocks se différencient entre eux par leur résistance, certains n'ont pas de collisions, et car ils ont des items correspondants différents.

Les items sont stockables dans l'inventaire, peuvent être utilisés pour le craft et sont intangibles.

Certains items correspondent à des blocks mais sont différents. D'une part il se pose, pour les blocks, le problème des collisions qui ne se pose pas avec les items. Les items, de leurs côté ont une quantité.
Les items sont divisés en itemBlock et itemCraft car les itemCraft n'ont pas de représentation sur le terrain. Un itemBlock connaît sont block correspondant via un attribut (char).Lorsqu'il est utilisé, le block correspondant se pose.
Les ennemis et le joueur sont des personnages, mais sont séparés. En effet, le joueur à un inventaire et doit pouvoir utiliser des items contrairement aux ennemis.



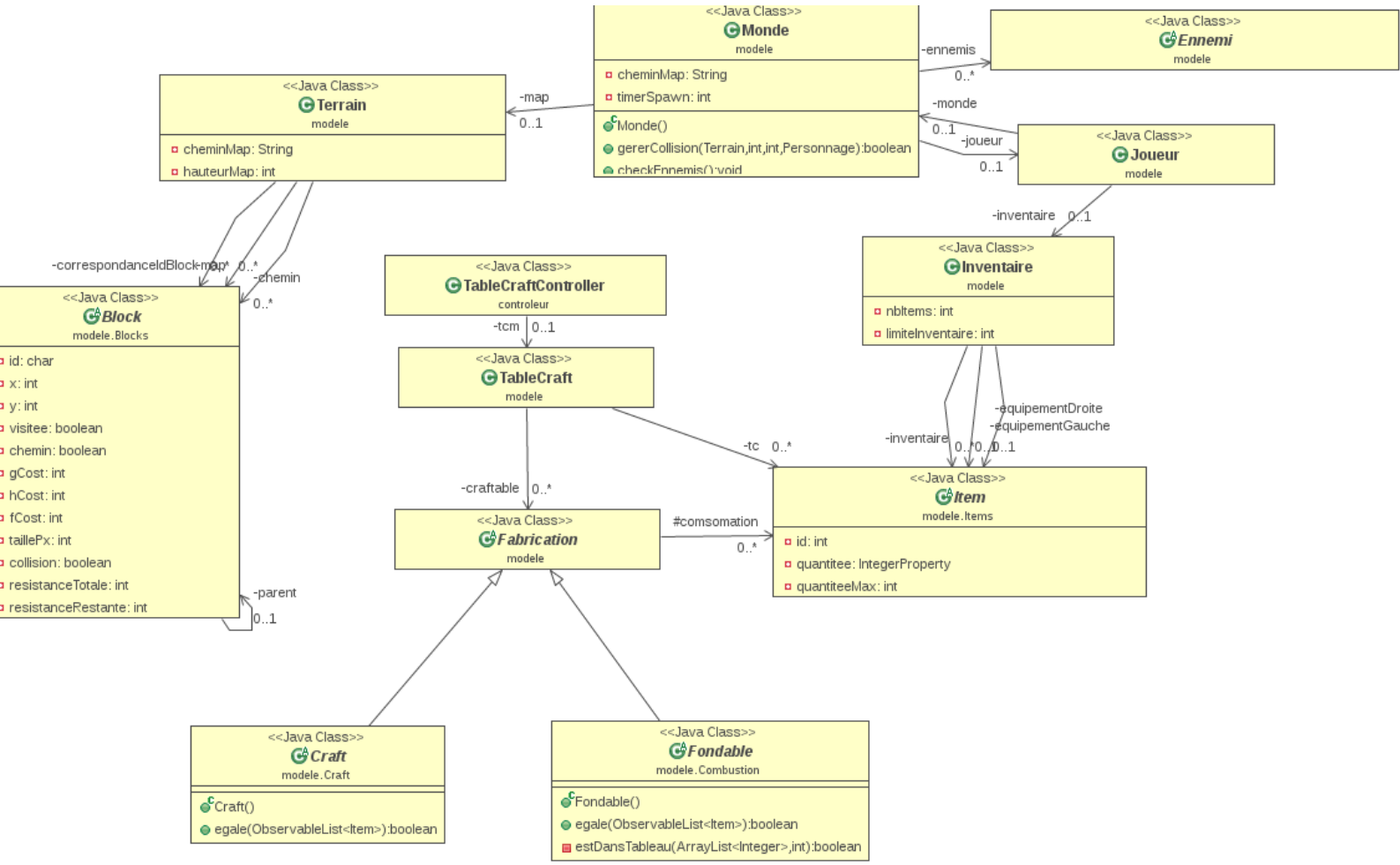
Comment sont-elles liées entre elles ?

Nous avons regoupé tous les éléments du jeu dans la classe Monde. Le monde possède un terrain qui contient une ObservableList de blocks.

Il possède également un joueur . Et le joueur connaît le monde car il agit sur ce dernier (en modifiant le terrain par ex).

Le joueur a un inventaire. Les items peuvent être stockés dans son inventaire. La table de craft prend une ArrayList de Fabrication qui désigne les items consommés et leur quantité lors du craft ou de la fonte.

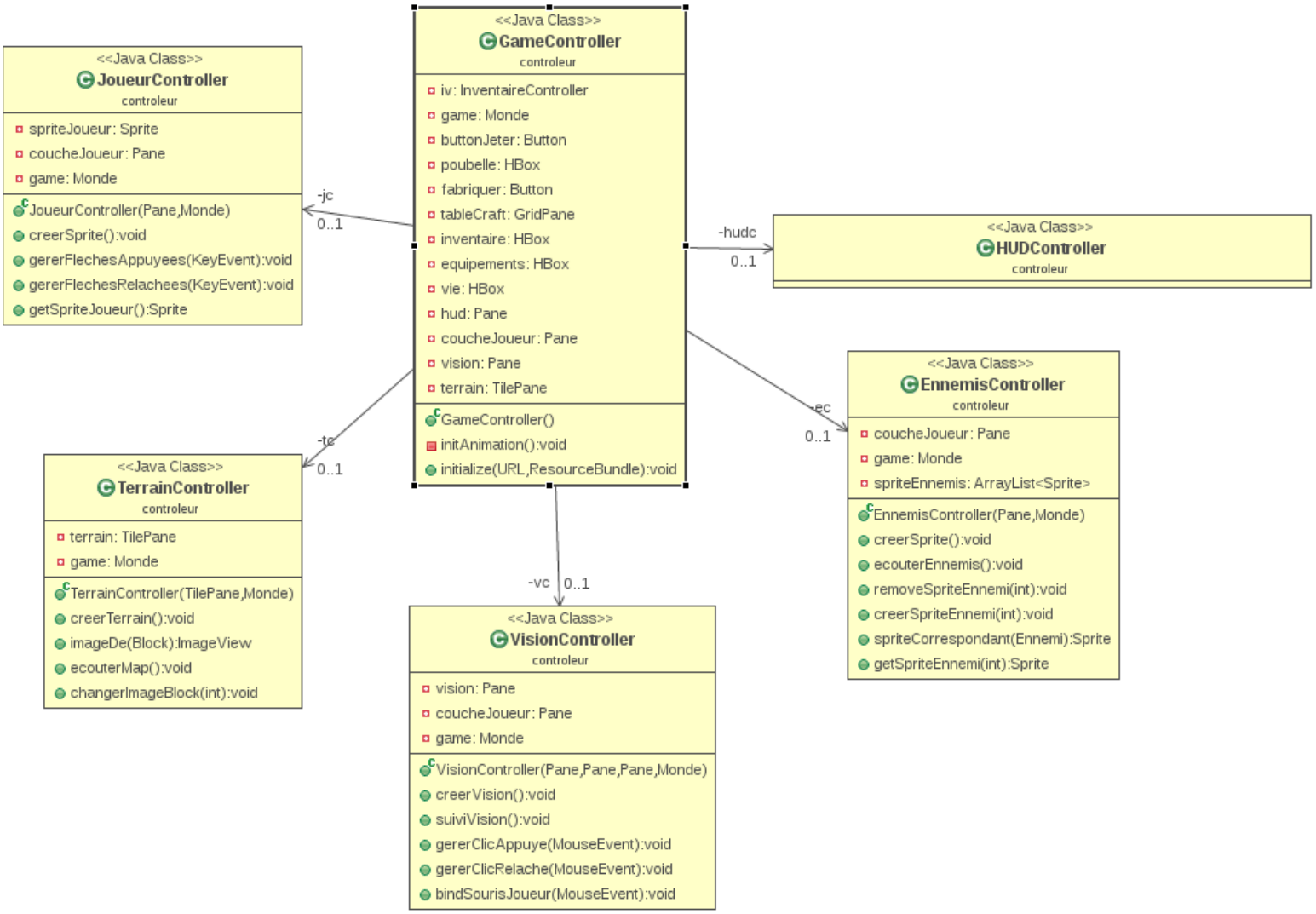
Craft et Fondable sont des sous classe de Fabrication, car ils consomment tous les deux des items pour en fabriquer un autre, mais ils sont différents car un schéma doit être reproduit de manière identique à chaque fois dans Craft, tendis que la position des items n'a pas d'importance dans Fondable.



Les controleurs :

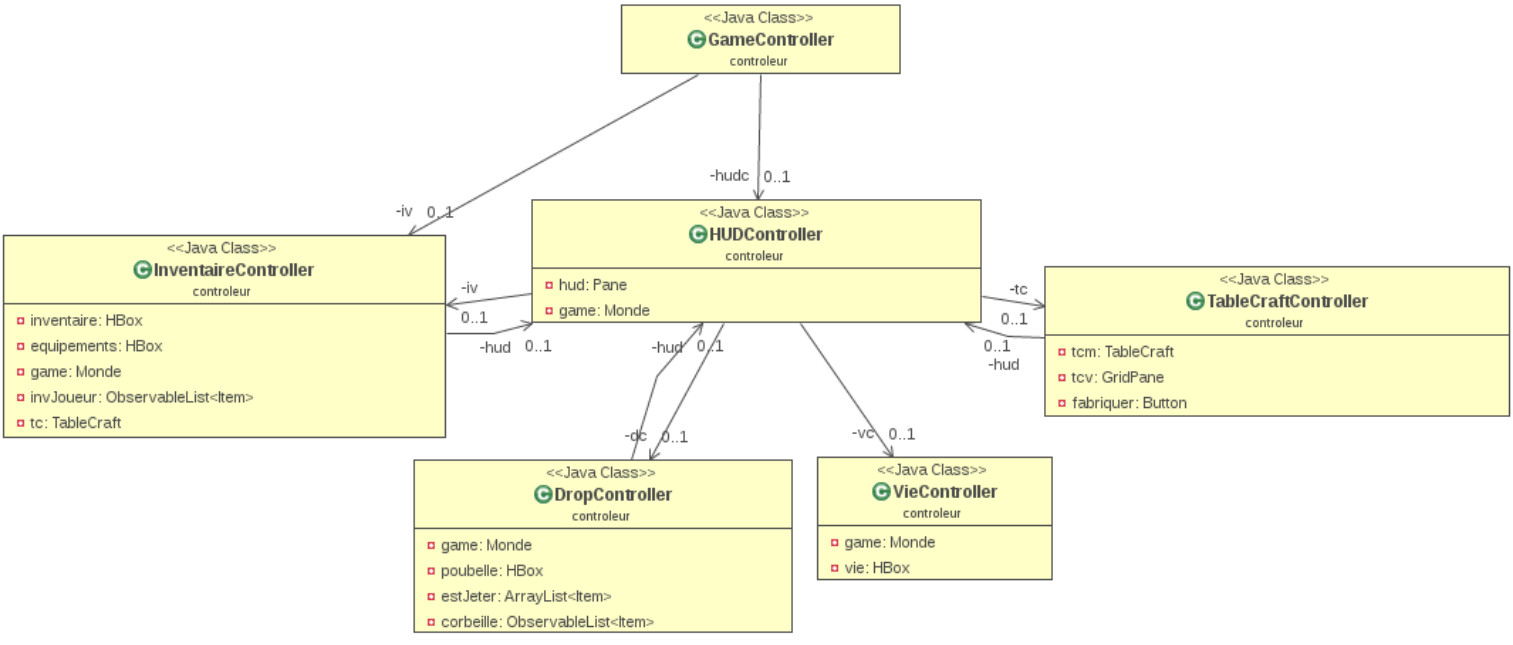
Les controleurs sont séparés pour plus de clarté. Les différents controleurs gèrent différents liens entre le modèle et la vue, ils sont des objets initilalisés dans un controleur principal qui contient la procédure initialize() ainsi que la game loop du jeu.

	Modèle	Vue	Events
GameController	-game loop -initialise un monde	-initialise les controleurs avec en paramètre les Nodes construits avec SceneBuilder	-pas de gestion d'évenement
JoueurController	-permet au joueur de se déplacer via le clavier	-fait bouger le sprite du joueur en fonction des coordonnées du joueur dans le modèle et le supprime lorsqu'il meurt	-gère les touches appuyées et relachées (zqsd)
VisionController	-permet de donner des coordonnées à la cible du joueur (qui lui serviront à attaquer, miner…) et d'utiliser ses items	-fait bouger une pane en fonction des déplacements du joueur (suivi de la vision)	-gère le clic appuyé et relaché -bind des coordonnées de la souris et de la cible du joueur lorsqu'il y a drag de la souris
EnnemisController	-observe les actions des ennemis du modèle	-initialise une liste de sprites et fait correspondre chacun d'entre eux à un ennemi du modèle -modifie les sprites et les fait se deplacer en fonction des actions des ennemis dans le modèle	-écoute si des ennemis apparaissent ou meurent et fait disparaître ou apparaître le sprite correspondant en conséquence
TerrainController	-observe le terrain du modèle	-crée le terrain du côté vue en faisant correspondre chaque block du modèle à une image dans la vue	-écoute les modifications d'une ObservableList de Blocks et modifie les images de la vue en conséquence

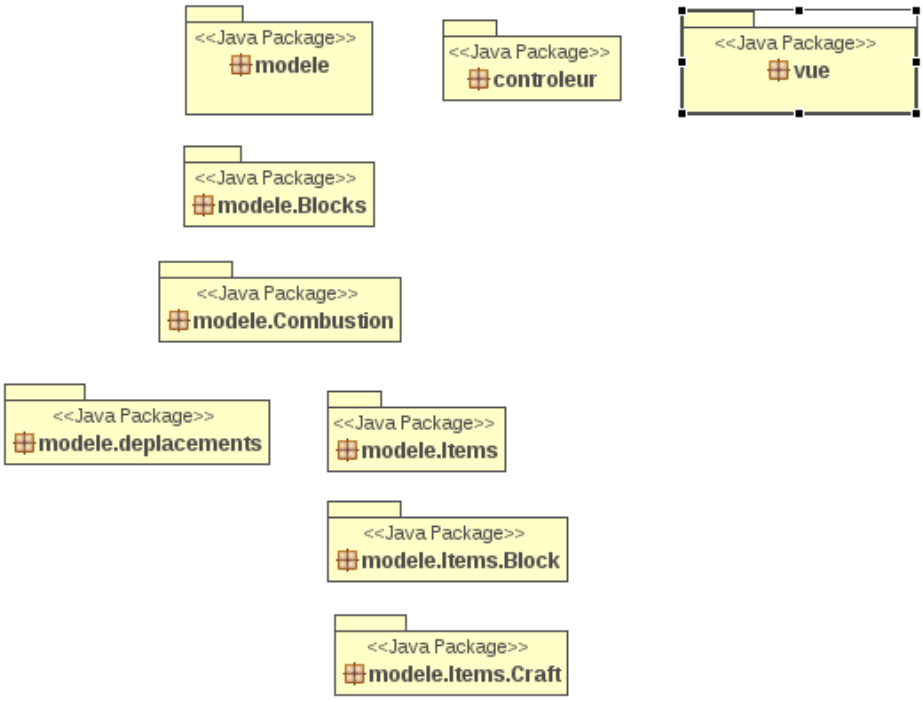


Les controleurs (HUD):

Le HUDController est lui-même divisé en plusieurs controleurs qui s'occupent de l'affichage du craft, de la vie, des cases de drop, de l'inventaire.



Les packages :



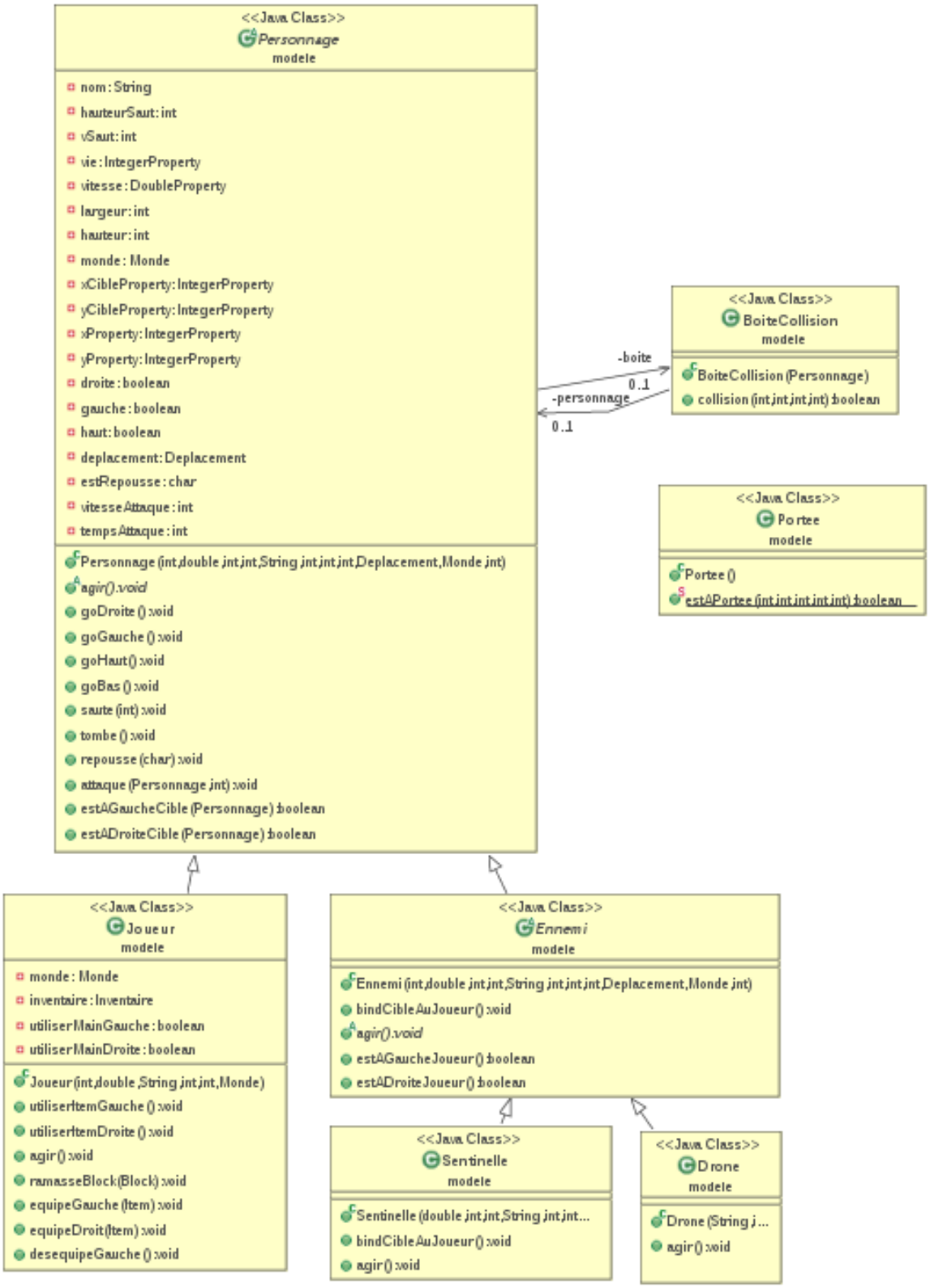
Détails : diagrammes de classe (5/60points)

Le diagramme des personnages :

Personnage est une classe abstraite. Tous les personnages auront une méthode agir() qui gèrera les actions des personnages en temps réél car elle sera appelée dans la game loop. Les déplacements et les attaques sont gérés de la même manière pour tous les personnages. Ces méthodes sont donc directement codées dans Personnage .

En revanche ils pourront avoir différents modes de déplacement. Deux personnages pourraient avoir le même mode de déplacement. C'est la raison pour laquelle ce dernier n'est pas une méthode mais est un attribut de Personnage.
Tous les personnages ont accès à la classe Portée. Portée est une classe statique.

Le Joueur est un personnage qui pourra en plus utiliser des items. Les ennemis devront se placer par rapport au joueur. Les sentinelles et les drones sont des ennemis qui ont pour cible le joueur, mais qui ont des modes de déplacement différents.

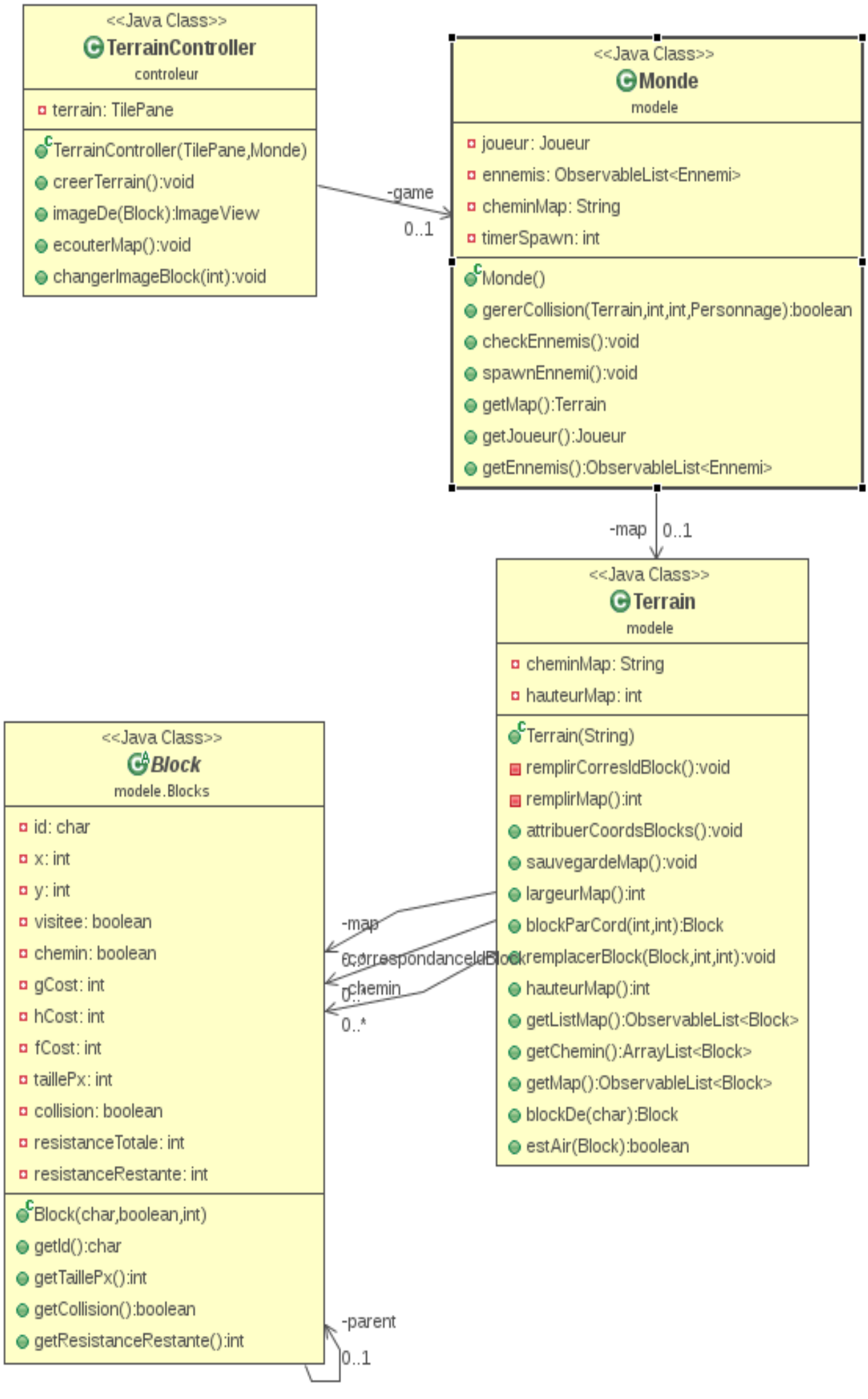

<<Java Class>>
Personnage
modele
nom : String
hauteurSaut : int
vSaut : int
vie : IntegerProperty
vitesse : DoubleProperty
largeur : int
hauteur : int
monde : Monde
xCibleProperty : IntegerProperty
yCibleProperty : IntegerProperty
xProperty : IntegerProperty
yProperty : IntegerProperty
droite : boolean
gauche : boolean
haut : boolean
deplacement : Deplacement
estRepousse : char
vitesseAttaque : int
tempsAttaque : int
Personnage (int,double,int,int,String,int,int,int,Deplacement,Monde,int)
agir().void
goDroite () :void
goGauche () :void
goHaut () :void
goBas () :void
saute (int) :void
tombe () :void
repousse (char) :void
attaque (Personnage,int) :void
estAGaucheCible (Personnage) :boolean
estADroiteCible (Personnage) :boolean
<<Java Class>>
BoiteCollision
modele
BoiteCollision (Personnage)
collision (int,int,int,int) :boolean
-boite
0..1
-personnage
0..1
<<Java Class>>
Portee
modele
Portee ()
estAPortee (int,int,int,int,int) :boolean
<<Java Class>>
Joueur
modele
monde : Monde
inventaire : Inventaire
utiliserMainGauche : boolean
utiliserMainDroite : boolean
Joueur (int,double,String,int,int,Monde)
utiliserItemGauche () :void
utiliserItemDroite () :void
agir () :void
ramasseBlock (Block) :void
equipeGauche (Item) :void
equipeDroit (Item) :void
desequipeGauche () :void
<<Java Class>>
Ennemi
modele
Ennemi (int,double,int,int,String,int,int,int,Deplacement,Monde,int)
bindCibleAuJoueur () :void
agir().void
estAGaucheJoueur () :boolean
estADroiteJoueur () :boolean
<<Java Class>>
Sentinelle
modele
Sentinelle (double,int,int,String,int,int...
bindCibleAuJoueur () :void
agir () :void
<<Java Class>>
Drone
modele
Drone (String,i...
agir () :void

Le diagramme du terrain (vue et modèle) :

Le terrain un attribut ObservableList de blocks.
Cette liste est remplie à partir de la lecture d'un fichier .csv grâce à une méthode (blockDe(charactère)) qui fait correspondre à chaque caractère un block dans la classe Terrain.

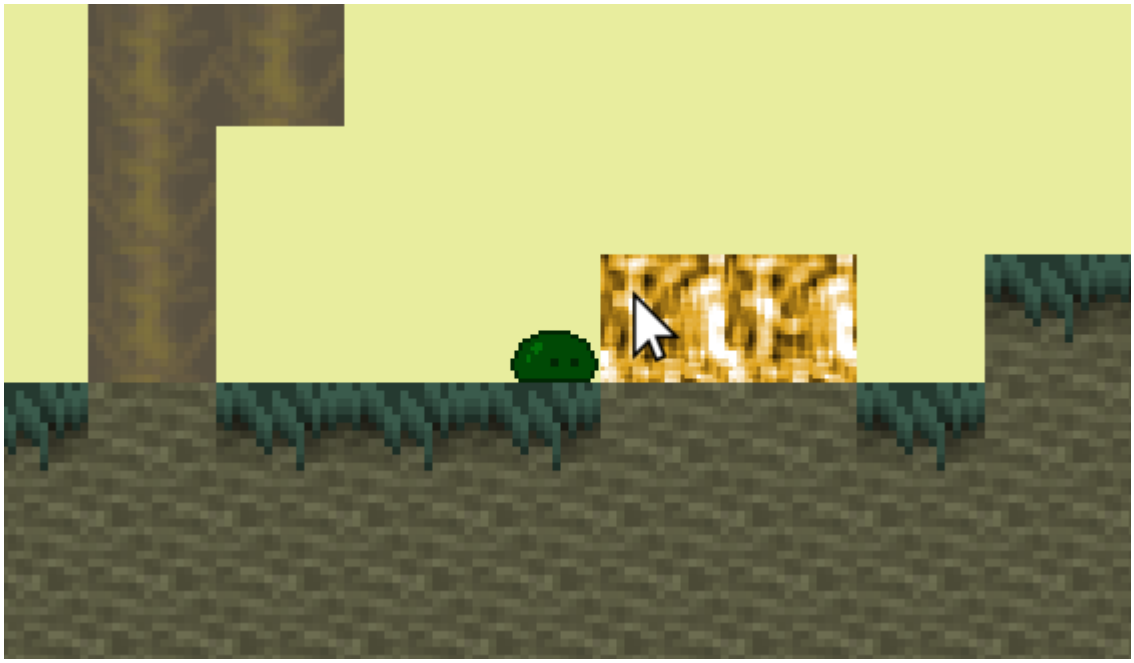
Le TerrainController fait, de son côté, correspondre à chaque block une image via la méthode imageDe(block). Cette méthode est appelée lors de la construction du terrain côté vue. La liste des block du terrain est observable car on souhaite écouter ses changements pour remplacer l'image du block modifié dans la vue. Il existe une méthode remplacerBlock(block qui remplace, x, y) dans Terrain qui remplace un block par un autre (lorsque l'on construit ou mine dans le jeu).

Le listener va écouter ce changement avec ecouterMap() dans le controleur et changer l'image pour qu'elle corresponde au nouveau block côté vue en appelant changerImageBlock(index).


<<Java Class>>
TerrainController
controleur
terrain: TilePane
TerrainController(TilePane,Monde)
creerTerrain():void
imageDe(Block):ImageView
ecouterMap():void
changerImageBlock(int):void
-game
0..1
<<Java Class>>
Monde
modele
joueur: Joueur
ennemis: ObservableList<Ennemi>
cheminMap: String
timerSpawn: int
Monde()
gererCollision(Terrain,int,int,Personnage):boolean
checkEnnemis():void
spawnEnnemi():void
getMap():Terrain
getJoueur():Joueur
getEnnemis():ObservableList<Ennemi>
-map
0..1
<<Java Class>>
Terrain
modele
cheminMap: String
hauteurMap: int
Terrain(String)
remplirCorresIdBlock():void
remplirMap():int
attribuerCoordsBlocks():void
sauvegardeMap():void
largeurMap():int
blockParCord(int,int):Block
remplacerBlock(Block,int,int):void
hauteurMap():int
getListMap():ObservableList<Block>
getChemin():ArrayList<Block>
getMap():ObservableList<Block>
blockDe(char):Block
estAir(Block):boolean
-map
correspondanceIdBlock
-chemin
0..*
<<Java Class>>
Block
modele.Blocks
id: char
x: int
y: int
visitee: boolean
chemin: boolean
gCost: int
hCost: int
fCost: int
taillePx: int
collision: boolean
resistanceTotale: int
resistanceRestante: int
Block(char,boolean,int)
getId():char
getTaillePx():int
getCollision():boolean
getResistanceRestante():int
-parent
0..1

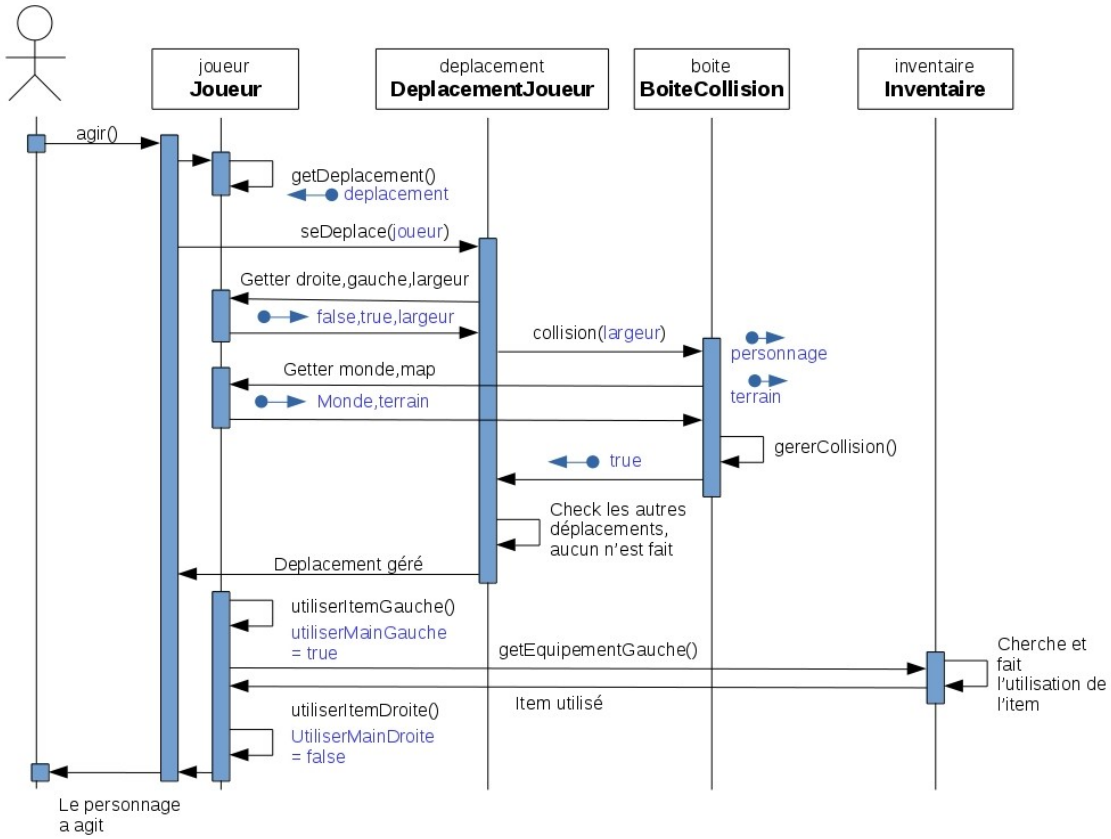
Diagrammes de séquence (5/60 points)

Nous avons choisi de représenter la méthode “agir()”, soit la méthode chargée de gérer les déplacements et les actions de clics du personnage à chaque GameLoop.
La situation suivante a été choisie :



Le joueur essaie de se déplacer à droite mais rencontre un bloc de déchet sur son chemin. Il clique sur ce même bloc pendant son déplacement afin de le détruire.

Voici l’exécution de la méthode agir dans cette configuration :



A chaque GameLoop, la méthode agir() du joueur est appelée. Celle-ci se compose de 3 actions, la première servant à déplacer le joueur, et les deux autres à utiliser ses deux “mains”.

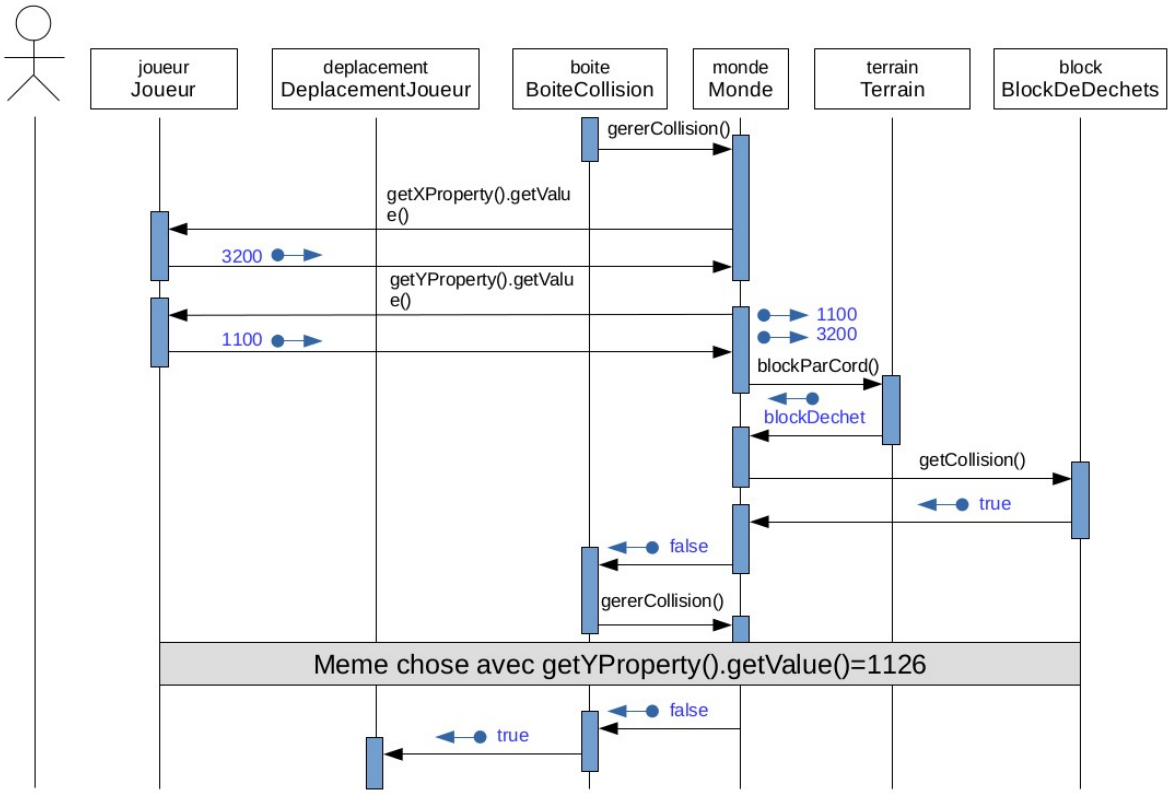
Tout d’abord, la méthode seDeplace() du DeplacementJoueur va devoir vérifier quelle direction veut être prise par le joueur (gauche, droite, haut) en se reportant aux booléens correspondant.

Lorsque l’une des directions est à “true”, que le joueur veut se déplacer vers celle-ci, la méthode collision() de BoiteCollision va s’occuper de gérer les collisions (nous verrons plus bas comment cela fonctionne).

Une fois les déplacements effectués, la méthode agir() va d’abord regarder si le joueur veut utiliser son item de gauche(), et l’inventaire s’occupera d’en faire l’utilisation si oui.

De même pour l’item de droite.

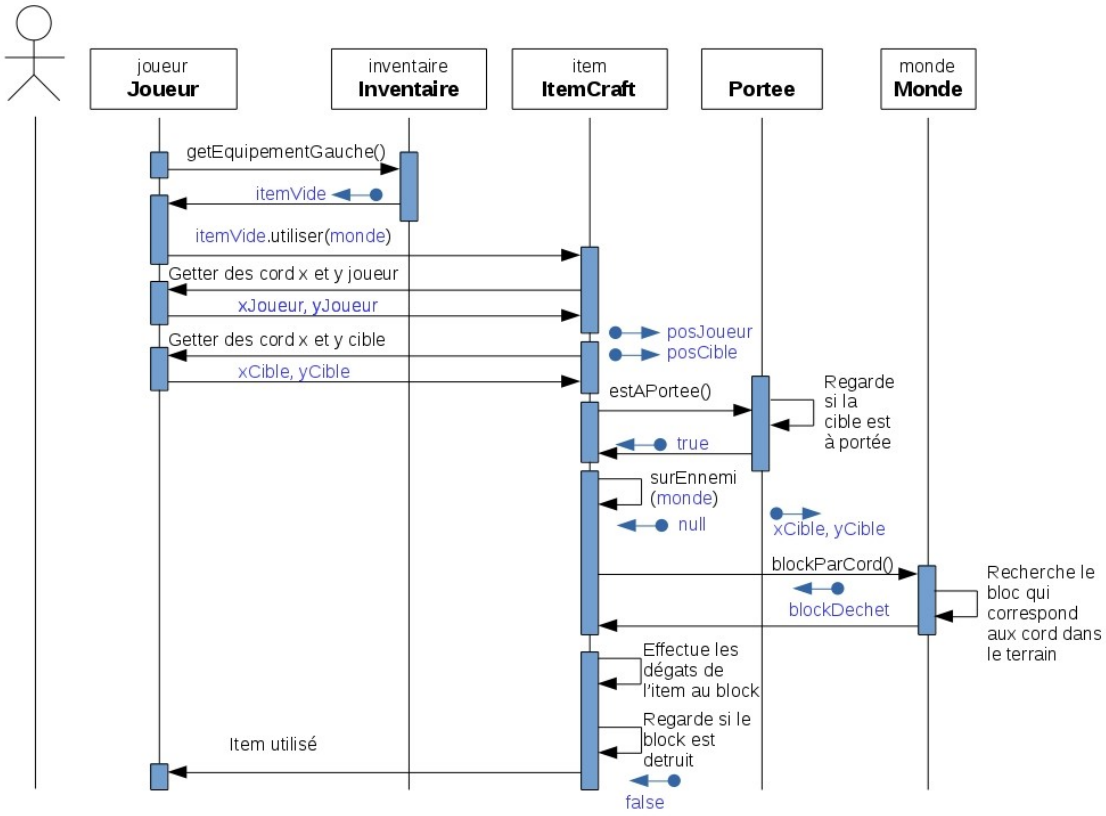
Détails de “gererCollision()” pour le déplacement à droite, avec la collision contre le bloc de déchet :



Ici donc, nous regardons pour le déplacement de droite, qui est à “true”, il est possible de l’effectuer. La boite va d’abord demander au monde de gérer les collisions via les coordonnées du point haut droit du joueur. (Puisque déplacement à droite)
Une fois ces coordonnées récupérées, le monde va chercher le bloc qui correspond à ces coordonnées et récupérer sa collision.
Ici le bloc de déchet a une collision, et retourne donc “true”.
La méthode gererCollision() retourne donc “false”, c’est à dire qu’il est impossible de se déplacer par là, pour le point haut droit.

Il faut donc vérifier que le point bas droit est lui aussi bloqué, ce qui mène aux mêmes étapes avec les coordonnées du point bas droite du joueur.
La méthode collision() qui avait appelée gererCollision() retourne “true”, signifiant qu’il y a bien des collisions et qu’il est donc impossible d’aller à droite.

Détails de l’utilisation de l’item gauche sur le bloc de déchet



L’item de gauche veut être utilisé comme en témoigne le booléen “utiliserMainGauche=true” du premier diagramme.
Le joueur va donc demander à son inventaire l’item de gauche, qui est ici un itemVide, soit l’item “neutre” qui correspond à une place vacante dans l’inventaire ou la “main” du joueur dans les emplacements d’équipement. C’est donc un ItemCraft.
Elle appelle ensuite la méthode “utiliser()” de cet item.
L’item va vérifier via la classe Portée qu’il est à portée de la cible sur laquelle le joueur a cliqué.
Ici c’est le cas, et il va d’abord falloir vérifier qu’il ne clic pas sur un ennemi via la fonction “surEnnemi()” de la classe abstraite Item.
Ce n’est donc pas le cas, il ne lui reste qu’à récupérer le bloc sur lequel il clique avec la fonction blockParCord() du Monde.
Le bloc de déchet est récupéré et les dégats de l’itemVide s’applique.
Ne reste qu’à vérifier si le bloc est détruit, ce n’est pas le cas et la fonction s’arrête donc là.

Structures de données :

Vous pourrez retrouver une HashMap, correspondant au critère des structures de données, dans la classe «Terrain». Malheureusement elle n’est pas utilisé dans le Sprint4 pour causes de problèmes décris dans la classe en question.

Exception :

Vous trouverez un «trycatch»dans le package controleur, dans la classe «Dropcontroller». L’exception se lance dans “`removeItemInt`” qui renvoit l’indice dans l’inventaire d’un item contenue dans la corbeille mais lorsqu’il ne le sont pas on crée l’exception “`ItemInexistantException`” qui est recuperée dans la classe “`jeterItemButton`”.

Algo :

La classe «DeplacementAStar» dans le package modele.deplacements, est basée l’algorithme «A*» et est surtout detaillée dans la methode “`AStar`”.

Document utilisateur (10 points/20)

Dans un jeu post-apocalyptique, vous contrôlez un blob naissant de déchets radioactifs, traqué par les humains et leur technologie, qui cherchent à vous détruire.
Vous êtes capable d'absorber toutes sortes de matériaux afin de créer des objets, grâce au système de crafting, qui vous serviront à progresser dans ce monde dangereux et regorgeant de ressources. Vous aller pouvoir modifier votre environnement en plaçant et détruisant des blocs, tout en vous déplaçant sur le terrain.
Vous disposez donc de toutes les ressources nécessaires à survivre face à ces terribles ennemis, il ne vous reste qu'à vous imposer dans ce monde ravagé.

Déplacements :

Votre personnage est capable de se déplacer dans les directions **Haut (Sauter)**, **Gauche** et **Droite**, sachant que la gravité le tire naturellement vers le **Bas**.
Voici les touches servant aux déplacements :

Haut : z
Gauche : q
Droite : d

Equiper un objet de l'inventaire :

Votre personnage dispose d'un inventaire dans lequel sont stockés tous les objets qu'il va ramasser sur le terrain. Il est également capable d'en équiper un dans chaque main. Il dispose en effet d'un "**Equipement de droite**" et d'un "**Equipement de gauche**".
Voici les touches servant à équiper les objets :

Equiper à droite : clique droit
Equiper à gauche : clique gauche

Deplacer des items :

Votre personnage est également capable de déplacer les objets de son inventaire afin de les placer dans la **Table de Craft** ou simplement dans sa **Poubelle**. Cette dernière lui servira à

jeter certains objets qu'il ne veut pas garder dans son inventaire, ils seront alors totalement détruits.
Voici les touches servant à équiper les objets :

Clic gauche + drag'n drop

Creuser et poser des blocs :

Il est évidemment possible de détuire les blocs et de poser ceux que vous avez équipé dans l'un de vos emplacement d'équipement. Pour ce faire, il vous faudra cliquer sur la map à une distance de 1 à 2 blocks autour de vous.
Voici les touches servant à creuser et à poser les blocs :

Clic correspondant au bloc équipé : pose le bloc
Clic correspondant à un objet équipé ou vide : creuse

Frapper un ennemi :

Pour frapper un ennemi, il s'agit de la même action que pour creuser. Il vous faudra donc obligatoirement utiliser un équipement autre qu'un bloc, puis cliquer sur un ennemi à l'écran pour le frapper.
Voici les touches servant à creuser et à poser les blocs :

Clic correspondant à un objet équipé ou vide : attaque

Différents crafts :

La fabrication se répartie en 2 catégories, le crafting et la fonderie. La base de la fabrication est du drag'n drop depuis l' inventaire vers la Table de Craft.
Le crafting est basé sur des schémas qui doivent etre parfaitement identiques à ce qui suit :

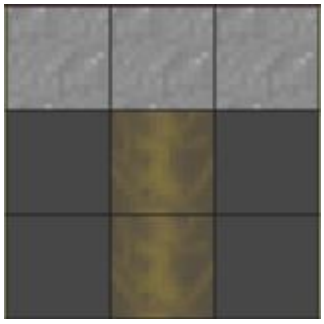
<u>Arme :</u>

Barre en Metal :

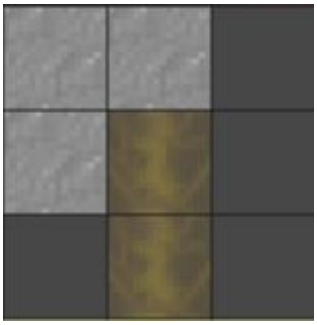
Lance Pierre :

Outil :

Pioche :



Hache :



La fonderie est juste basée sur une liste d'items contenant un minerai (actuellement minerai de fer) et un combustible (actuellement minerai de radium) qui doivent être placés dans la table de craft peu importe l'ordre ou place :

Lingot de Metal :

Item : A partir du craft on peut différencier deux catégories :

Les objets : Ils sont créés par le joueur via le craft :

-Outils : Ils sont plus efficace sur certain bloc que sur d'autre. Par exemple la pioche cassera plus facilement des minerais ou de la pierre là ou a la hache qui cassera plus facilement du bois.

-Arme : Elles servent à attaquer les ennemis. Soit elle augmente drastiquement les dégâts, comme la barre en métal, soit il permettent d’attaquer à distance, comme le lance pierre (non implenté).

Les ressources : Elles permettent de fabriquer des objets ou d’autre ressource, proviennent du terrain.

-Les ressources tangibles : Ce sont les blocs que nous pouvont ramasser et poser sur le terrain.

-Les ressources intangibles : Exemple parlant : Le **fil**. Récupérable sur le terrain aléatoirement en cassant un bloc de déchet. Ne peut être crafté ni posé sur le terrain.