Dossier du Projet

GROUPE : LAY ELODIE, TOIHIR YOA, BEN MOUSSA MOHAMMED
Ce fichier reprend la description des documents demandés pour le projet, ainsi que quelques renseignements que vous devez fournir.
Vous devez le compléter et le mettre à disposition sur git.

Tous les diagrammes devront être fournis avec un texte de présentation et des explications.

1. Documents pour CPOO

Architecture(5/60 points)

Dans le cadre de notre projet, nous avons utilisé l'architecture MVC (Model View Controller -> Modèle Vue Contrôleur). Cela permet de créer un affichage dépendant du modèle et permet une cohésion complète entre le modèle et la vue et une interaction entre eux via le contrôleur.

Nous avons ensuite optés pour une architecture avec des superclasses. Par exemple, dans le cas du personnage principale, nous avons décidés d'inclure dans une superclasse nommé "Personnage" tous les types d'objets "vivants" et dynamiques de notre projet (par exemple Carolina, Ennemi, Boss Final, etc). Ces objets partagent pas mal d'attributs et de méthodes, ce qui nous a mis d'accord sur comment nous allions organiser nos classes. Concernant l'inventaire, il est constitué d'une liste Observable d'Objets où Objet est une super classe possédant multiples sous classes : tout d'abord Arme qui est une super classe de Épée et Arc, ensuite nous avons Pioche, Établi et Fer.

Pour notre contrôleur, il est rempli de fonctions liant la vue et le modèle (comme les déplacements du personnage par exemple). Il possède une gameloop qui lui permet de gérer la gravité de notre joueur ainsi que les déplacements de notre ennemi et les différents cas possibles (ex: si le personnage meurt, si l'ennemi meurt, etc). La map est générée dans l'Initialize et est constituée :

- d'un panel (Pane) principal de la map
- d'un panel (Pane) permettant d'afficher le ciel (avec un fond.png)
- d'un panel (Pane) réservé à l'affichage du sol (les tuiles étant générées dans l'Initialize)
- de 2 ImageView pour l'affichage d'une image png représentant le fond de l'inventaire
- et de 6 ImageView (avec un identifiant différent) pour chaque cases de l'inventaire

On s'intéresse particulièrement aux événements :

- Nous disposons d'un listener sur la liste Observable de l'Inventaire. Chaque fois qu'un objet est ajouté à la liste de l'inventaire, le contrôleur modifie la vue et ajoute les éléments ajoutés à la liste dans les ImageView appropriées. De même pour les éléments supprimés. Chaque fois qu'un objet est retiré de la liste de l'inventaire coté modèle, le listener va le repérer côté vue et enlever l'objet de l'ImageView.

- Nous disposons également d'un Event Handler qui gère l'interaction du joueur et du jeu au niveau des clics de souris (que ce soit par rapport au fait de miner/déposer un bloc ou par rapport à l'inventaire).

Détails : diagrammes de classe (5/60 points)

DIAGRAMME DE PACKAGE

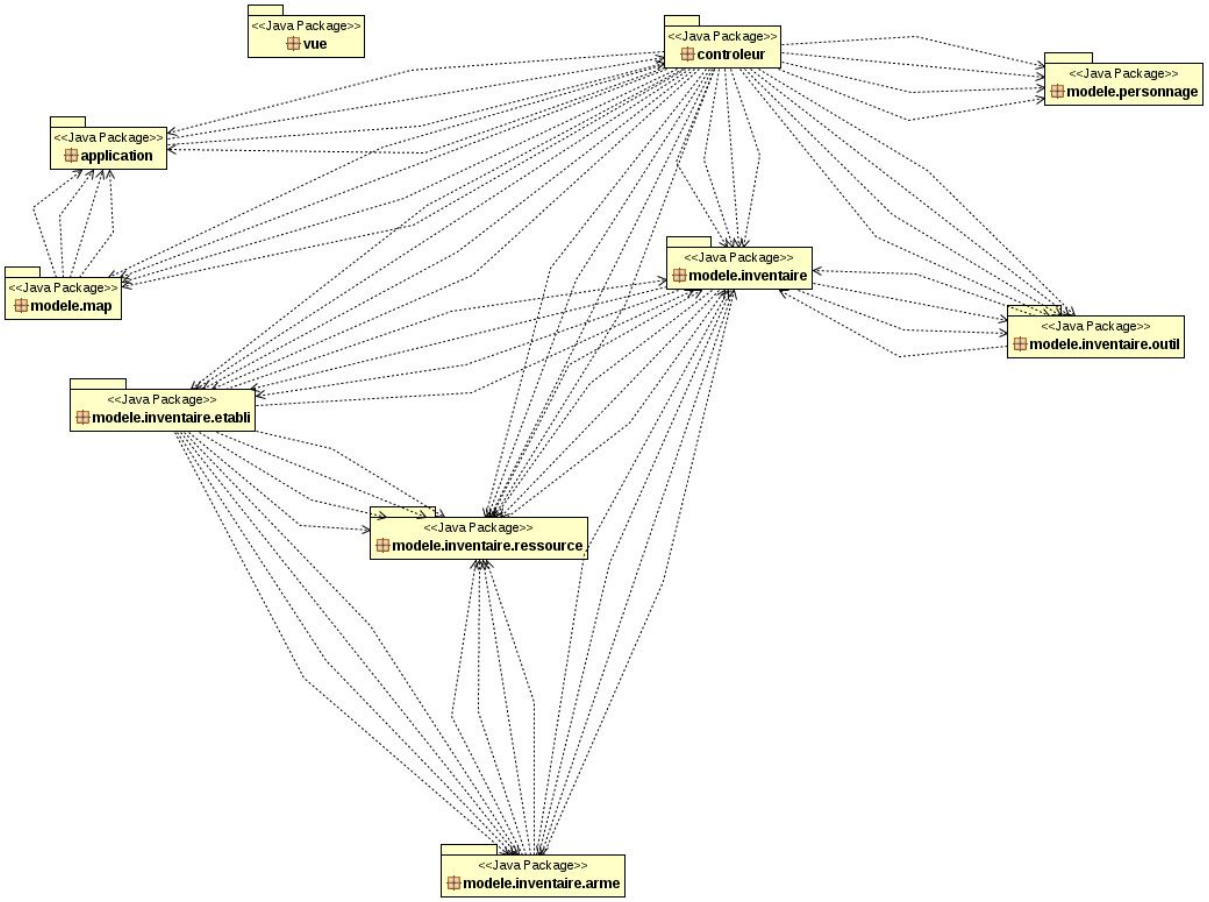
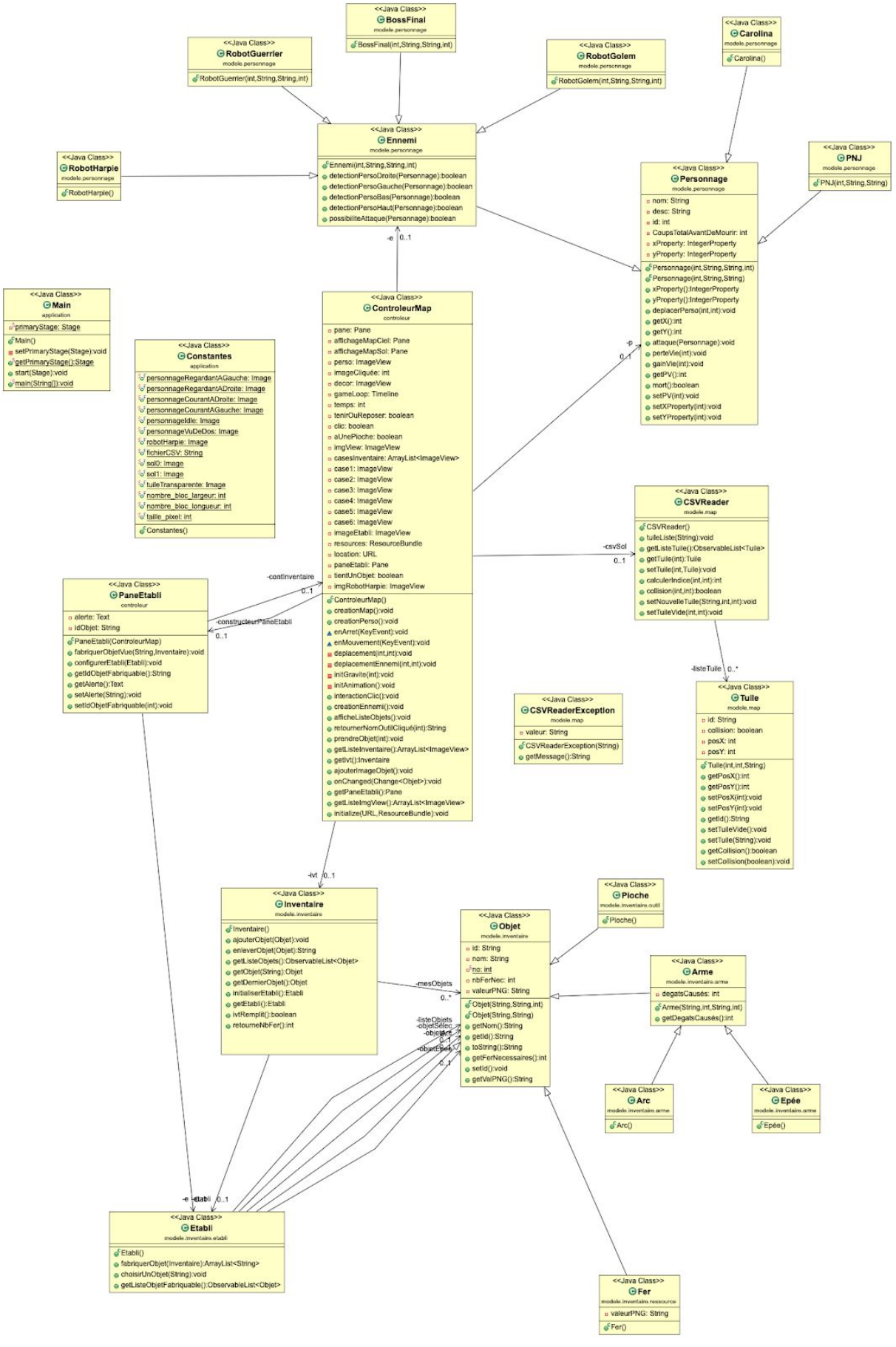


DIAGRAMME ENTIER DU JEU


BossFinal
modele.personnage
BossFinal(int,String,String,int)
RobotGuerrier
modele.personnage
RobotGuerrier(int,String,String,int)
RobotGolem
modele.personnage
RobotGolem(int,String,String,int)
Carolina
modele.personnage
Carolina()
Ennemi
modele.personnage
Ennemi(int,String,String,int)
detectionPersoDroite(Personnage):boolean
detectionPersoGauche(Personnage):boolean
detectionPersoBas(Personnage):boolean
detectionPersoHaut(Personnage):boolean
possibiliteAttaque(Personnage):boolean
RobotHarpie
modele.personnage
RobotHarpie()
PNJ
modele.personnage
PNJ(int,String,String)
Personnage
modele.personnage
nom: String
desc: String
id: int
CoupsTotalAvantDeMourir: int
xProperty: IntegerProperty
yProperty: IntegerProperty
Personnage(int,String,String,int)
Personnage(int,String,String)
xProperty():IntegerProperty
yProperty():IntegerProperty
deplacerPerso(int,int):void
getX():int
getY():int
attaque(Personnage):void
perteVie(int):void
gainVie(int):void
getPV():int
mort():boolean
setPV(int):void
setXProperty(int):void
setYProperty(int):void
-e
0..1
-p
0..1
Main
application
primaryStage: Stage
Main()
setPrimaryStage(Stage):void
getPrimaryStage():Stage
start(Stage):void
main(String[]):void
Constantes
application
personnageRegardantAGauche: Image
personnageRegardantADroite: Image
personnageCourantADroite: Image
personnageCourantAGauche: Image
personnageIdle: Image
personnageVuDeDos: Image
robotHarpie: Image
fichierCSV: String
sol0: Image
sol1: Image
tuileTransparente: Image
nombre_bloc_largeur: int
nombre_bloc_longueur: int
taille_pixel: int
Constantes()
ControleurMap
controleur
pane: Pane
affichageMapCiel: Pane
affichageMapSol: Pane
perso: ImageView
imageCliquée: int
decor: ImageView
gameLoop: Timeline
temps: int
tenirOuReposer: boolean
clic: boolean
aUnePioche: boolean
imgView: ImageView
casesInventaire: ArrayList<ImageView>
case1: ImageView
case2: ImageView
case3: ImageView
case4: ImageView
case5: ImageView
case6: ImageView
imageEtabli: ImageView
resources: ResourceBundle
location: URL
paneEtabli: Pane
tientUnObjet: boolean
imgRobotHarpie: ImageView
ControleurMap()
creationMap():void
creationPerso():void
enArret(KeyEvent):void
enMouvement(KeyEvent):void
deplacement(int,int):void
deplacementEnnemi(int,int):void
initGravite(int):void
initAnimation():void
interactionClic():void
creationEnnemi():void
afficheListeObjets():void
retournerNomOutilCliqué(int):String
prendreObjet(int):void
getListeInventaire():ArrayList<ImageView>
getIvt():Inventaire
ajouterImageObjet():void
onChanged(Change<Objet>):void
getPaneEtabli():Pane
getListeImgView():ArrayList<ImageView>
initialize(URL,ResourceBundle):void
-csvSol
0..1
-contInventaire
0..1
-constructeurPaneEtabli
0..1
PaneEtabli
controleur
alerte: Text
idObjet: String
PaneEtabli(ControleurMap)
fabriquerObjetVue(String,Inventaire):void
configurerEtabli(Etabli):void
getIdObjetFabriquable():String
getAlerte():Text
setAlerte(String):void
setIdObjetFabriquable(int):void
CSVReader
modele.map
CSVReader()
tuileListe(String):void
getListeTuile():ObservableList<Tuile>
getTuile(int):Tuile
setTuile(int,Tuile):void
calculerIndice(int,int):int
collision(int,int):boolean
setNouvelleTuile(String,int,int):void
setTuileVide(int,int):void
-listeTuile
0..*
CSVReaderException
modele.map
valeur: String
CSVReaderException(String)
getMessage():String
Tuile
modele.map
id: String
collision: boolean
posX: int
posY: int
Tuile(int,int,String)
getPosX():int
getPosY():int
setPosX(int):void
setPosY(int):void
getId():String
setTuileVide():void
setTuile(String):void
getCollision():boolean
setCollision(boolean):void
-ivt
0..1
Inventaire
modele.inventaire
Inventaire()
ajouterObjet(Objet):void
enleverObjet(Objet):String
getListeObjets():ObservableList<Objet>
getObjet(String):Objet
getDernierObjet():Objet
initialiserEtabli():Etabli
getEtabli():Etabli
ivtRemplit():boolean
retourneNbFer():int
-mesObjets
0..*
-listeObjets
-objetSélec
0..1
-objetEtabli
0..1
Objet
modele.inventaire
id: String
nom: String
no: int
nbFerNec: int
valeurPNG: String
Objet(String,String,int)
Objet(String,String)
getNom():String
getId():String
toString():String
getFerNecessaires():int
setId():void
getValPNG():String
Pioche
modele.inventaire.outil
Pioche()
Arme
modele.inventaire.arme
degatsCausés: int
Arme(String,int,String,int)
getDegatsCausés():int
Arc
modele.inventaire.arme
Arc()
Epée
modele.inventaire.arme
Epée()
-e
-etabli
0..1
Etabli
modele.inventaire.etabli
Etabli()
fabriquerObjet(Inventaire):ArrayList<String>
choisirUnObjet(String):void
getListeObjetFabriquable():ObservableList<Objet>
Fer
modele.inventaire.ressource
valeurPNG: String
Fer()

Ce diagramme représente l'ensemble de notre jeu.

1) Inventaire : Inventaire est la liste d'objets possédés par le joueur durant le jeu.

2) Établi : Établi est un objet dans l'inventaire qui nous sert à crafter des armes quand on le souhaite. Il interagit avec la classe Fer et avec l'Inventaire lui même pour fabriquer les objets.

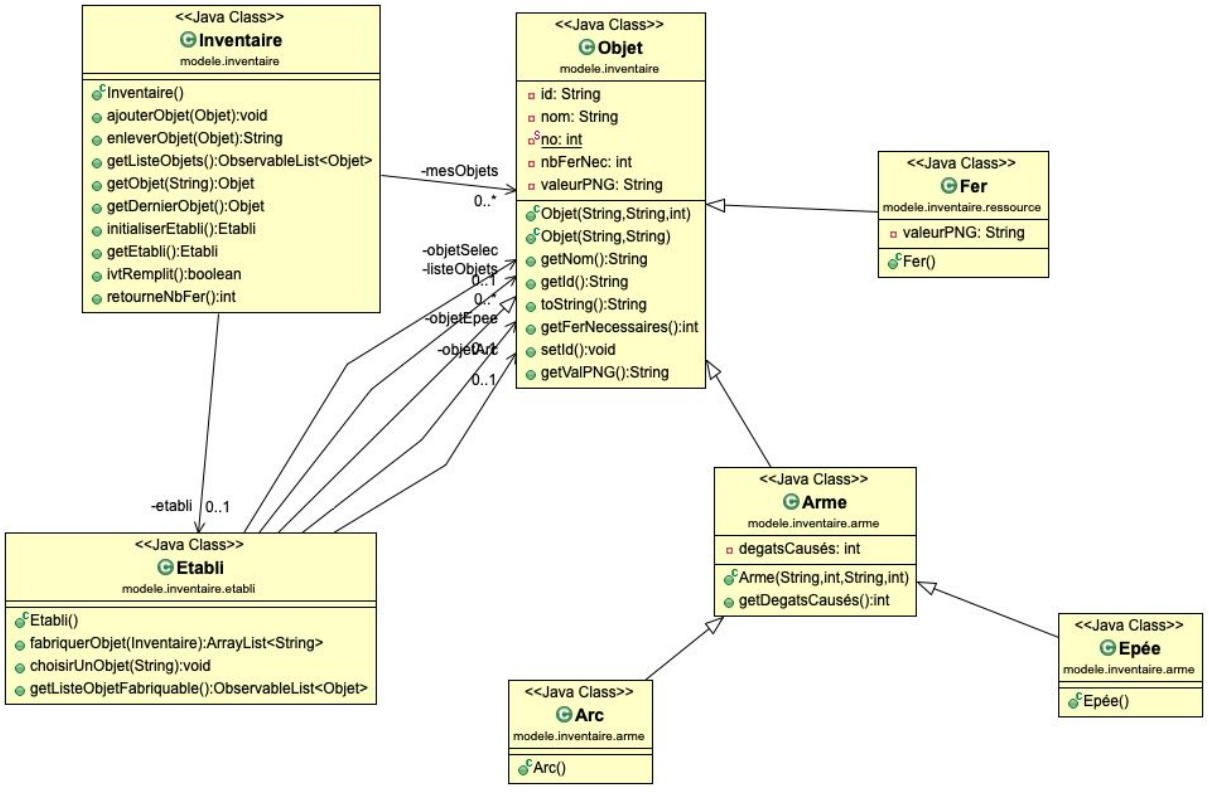
3) Objet : c'est la super-classe qui organise l'ensemble des objets qui appartiennent à notre jeu (pioche, épée, arc, fer et établi)

4) CSVReader : CSVReader affiche la matrice des différentes tuiles de la map à afficher grâce au fichier CSV généré par Tiled et génère des tuiles ainsi que leurs coordonnées x et y et les places dans une ObservableList de Tuiles.

5) Tuile : Tuile permet de gérer une tuile à la fois. Elle permet par exemple connaître ses coordonnées x y, savoir si la tuile est traversable ou bien de changer son id.

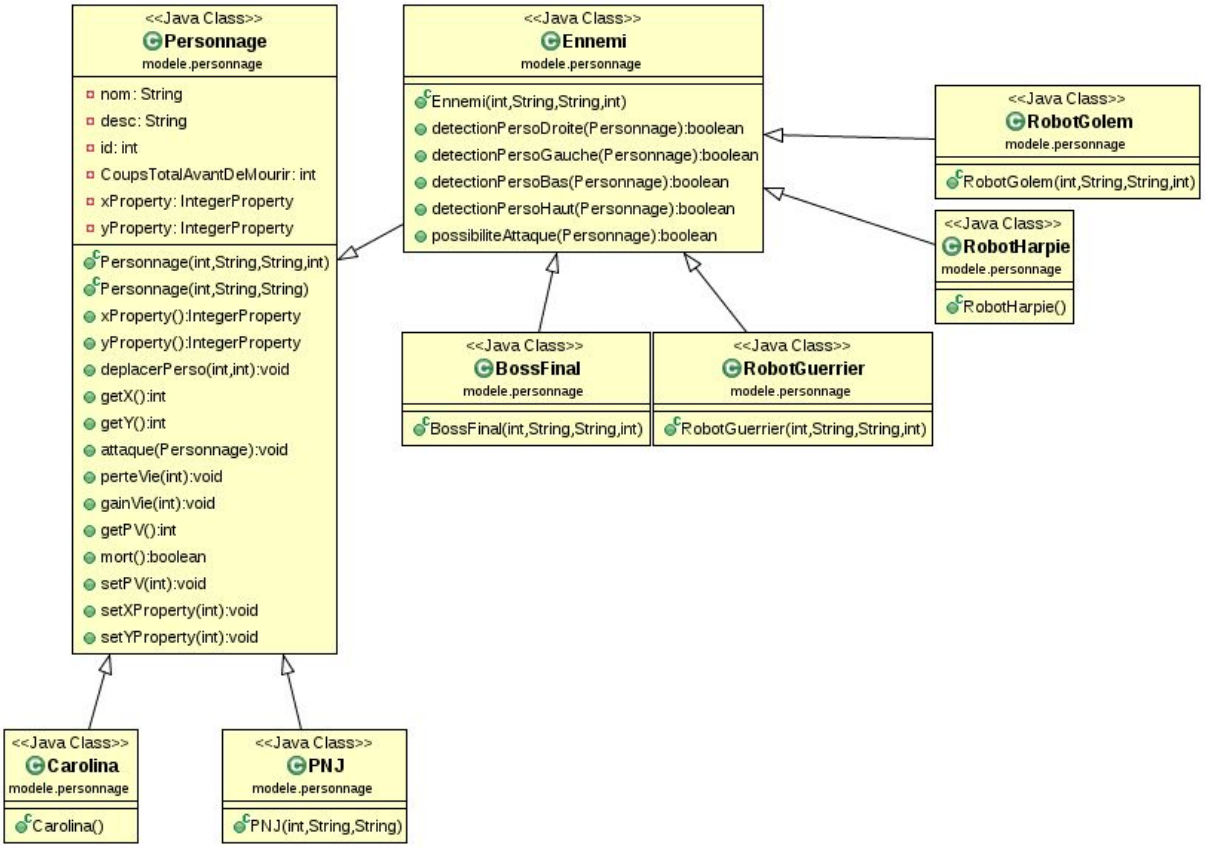
6) Personnage : Personnage est la superclasse qui gère l'ensemble des objets "vivants" et dynamiques de notre jeu (personnage principale et ennemis essentiellement).

DIAGRAMME DE LA GESTION DE L'INVENTAIRE + CRAFTING



La classe principale de notre notre modèle est Inventaire. Elle permet de pouvoir gérer l'Inventaire présent dans notre jeu ainsi que le crafting de l'établi. Nous avons séparé l'Inventaire par thématique : un package pour les armes, un autre pour l'outil pioche, un autre pour l'établi et un dernier pour les ressources. La classe objet est dans le package de l'inventaire car ce sont les objets qui sont bien présents dans notre inventaire. Les armes, la pioche, l'établi et le fer sont des sous classes de Objet. Objet est présent dans Inventaire sous forme d'une liste pour y stocker les objets présents dedans.

DIAGRAMME DE LA GESTION DES PERSONNAGES

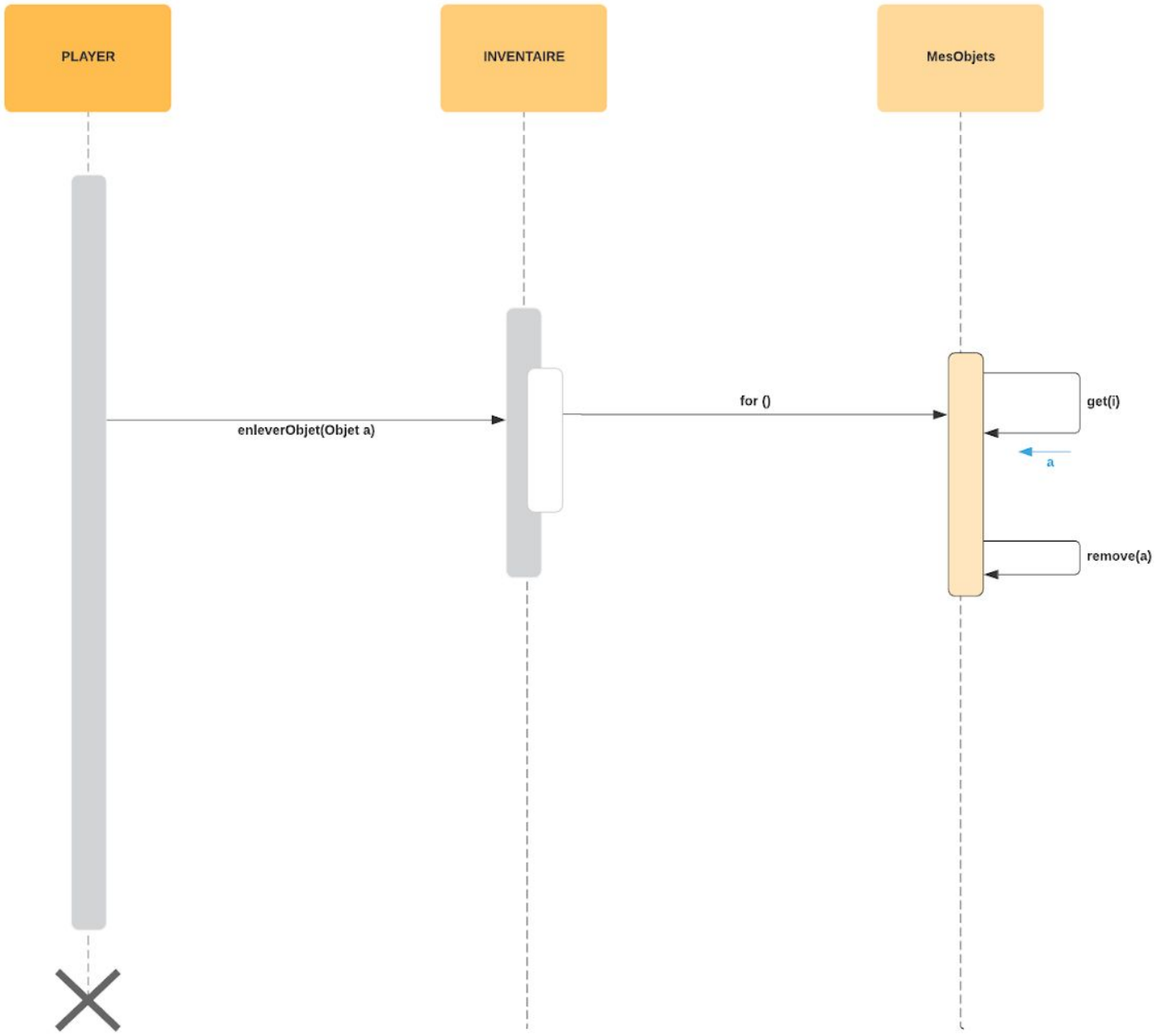


La superclasse Personnage permet de gérer tous les personnages de notre jeu. La superclasse Personnage contient une sous-classe Ennemi qui est elle-même une superclasse qui contient 4 sous-classes : RobotGolem, RobotHarpie, RobotGuerrier et BossFinal. La superclasse Personnage contient également une sous-classe Carolina (qui est le personnage principale) et ainsi qu'une sous-classe PNJ.

Diagrammes de séquence (5/60 points)

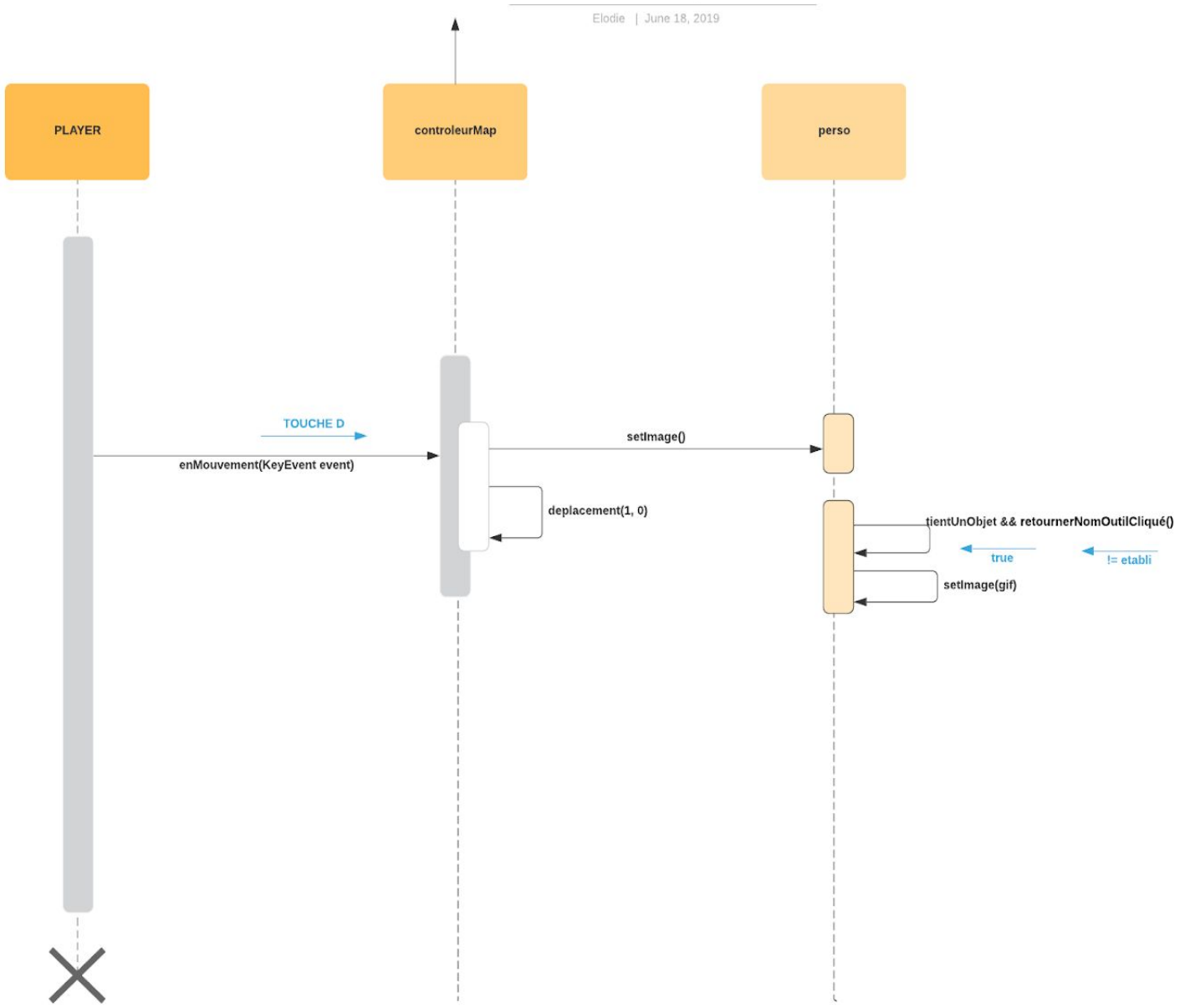
→ enleverObjet() : enlever un objet de la liste d'objets de l'inventaire (gestion inventaire)

↳ est appelée quand une ressource est utilisée et doit être enlevée de la liste



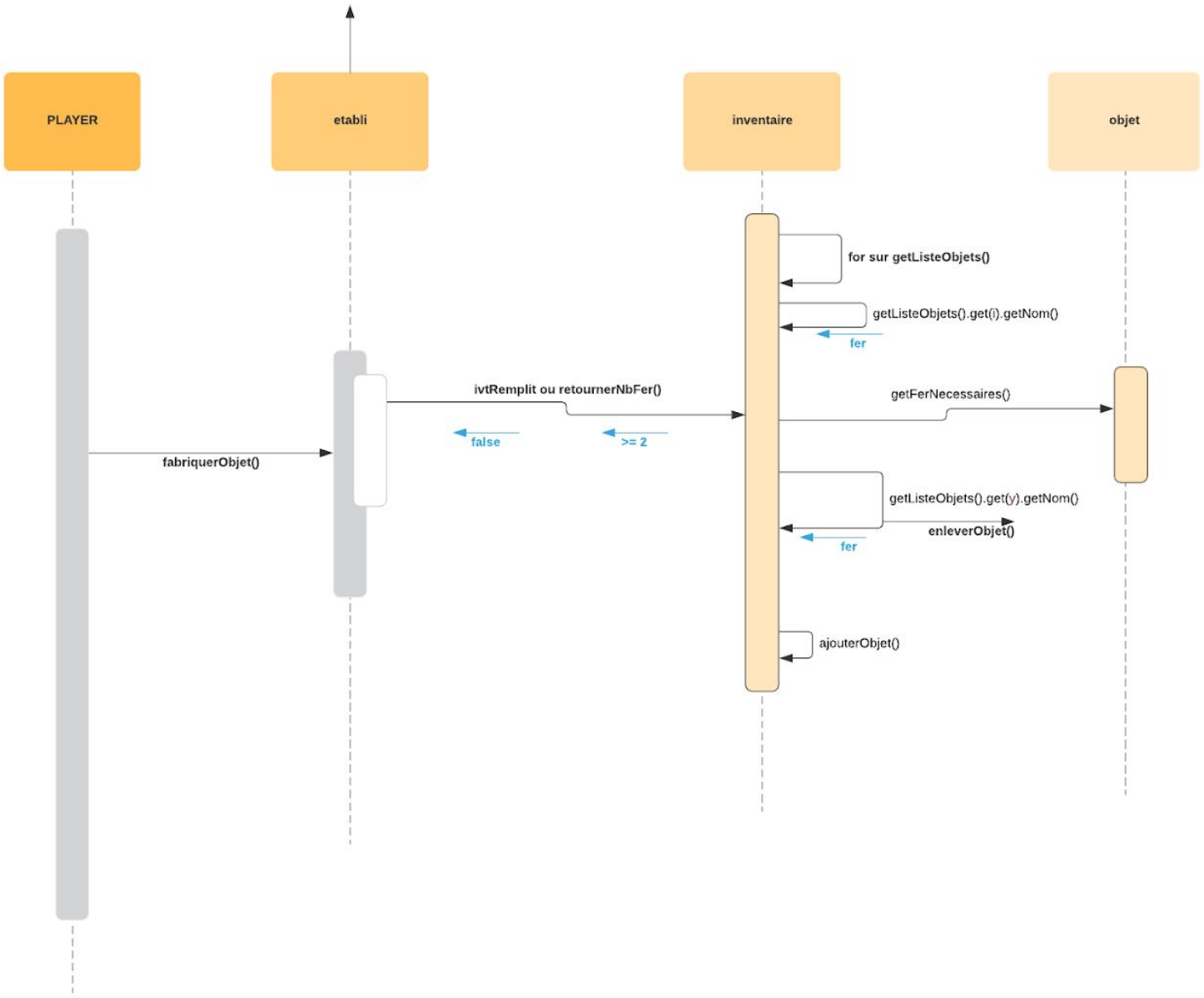
→ enMouvement() : déplacement du personnage

↳ est appelée quand le joueur appuie sur la touche D du clavier pour faire avancer son personnage



→ fabriquerObjet() : crafting d'un objet

↳ est appelée quand le joueur souhaite crafter un objet pour l'ajouter dans son inventaire



Structures de données :

Nous avons utilisé les structures ObservableList afin de pouvoir interagir avec la vue lorsque nous modifions les tableaux (afin de changer l'affichage par exemple). Nous avons également utilisé BufferedReader pour lire la map d'un fichier csv.

Exception :

Dans le main, il y a une exception au lancement du jeu pour n'importe quelle exception trouvée.
Dans le CSVReader, il y a plusieurs exceptions :

- FileNotFoundException (afin de vérifier si le fichier entré est trouvable)
- IOException (afin de gérer une erreur d'input/output)
- CSVReaderException (exception créée par Mohammed qui permet de lancer une exception lorsqu'il y a une erreur dans le fichier csv de la map)

Dans ControleurMap, il y a les mêmes exceptions que dans CSVReader car on utilise souvent cette classe.

Algo :

Nous avons un algorithme qui permet à un ennemi de se déplacer vers le personnage principale et de vérifier en fonction de sa position et de la position du personnage principale si il est en mesure de l'attaquer (détection via une portée et possibilité d'attaque si même position). Nous avons aussi un algorithme qui, en fonction des coordonnées x y de la tuile, permet de savoir si la tuile est traversable ou non (si il y a une collision).

2. Documents pour gestion de projet

Document utilisateur (10 points/20)

I. Description du jeu

Notre jeu s'intitule *Last Resort*. Ce jeu nous met dans la peau d'une jeune fille nommée Carolina qui, pour une raison ou une autre, se réveille en l'an 2 147 483 647. Son but devient donc tout de suite très clair : Carolina doit retourner à l'époque dans laquelle elle appartient. (C'est-à-dire en 2019 !)

Pour se faire, elle devra survivre dans cette dimension hallucinante où toute forme de vie a disparu et où la Terre est peuplé par des robots. Le terrain (et le décor) sur lequel Carolina se trouve est exclusivement composé de fer que Carolina pourra crafter avec sa pioche afin qu'elle puisse récolter du fer et se construire des armes pour qu'elle puisse se protéger des différents monstres/ennemis qu'elle pourrait rencontrer dans son aventure…

Le but de ce jeu (et donc l'objectif de Carolina) est de terrasser le boss final afin que l'on/Carolina puisse retourner à notre/son époque.

Carolina va spawner au début du niveau avec une pioche dans son inventaire. Elle pourra l'utiliser pour crafter une épée afin qu'elle puisse attaquer les ennemis. Il n'est pas impossible qu'elle puisse aussi builder un abri…

La difficulté de ce jeu est qu'il est, malgré son aspect futuriste/post-apocalyptique, plutôt réaliste. En effet, la particularité de *Last Resort* (en anglais "Dernier Recours") est la mortalité de son héros. Et oui ! Carolina n'a qu'une seule vie. Ce qui veut dire que si notre pauvre Carolina subit une fin tragique à la suite de son aventure extrêmement difficile, elle mourra pour de bon...
Mais ne vous en faites pas ! On pourra toujours recommencer l'aventure en lançant une nouvelle partie ! (Bien que toute notre progression sera effacée…)

II. Mécaniques du jeu

Déplacement :

- Avancer vers la droite : **touche D** du clavier
- Avancer vers la gauche : **touche Q** du clavier
- Aller en bas : **touche S** du clavier
- Aller en haut : **touche Z** du clavier

Interaction avec les blocs :

- Miner un bloc : **clic gauche** sur le bloc (à condition d'avoir une pioche en main et d'être proche du bloc que vous voulez miner)
- Poser un bloc : **clic droit** sur la map (à condition d'avoir un bloc dans son inventaire, d'être proche de l'endroit où on veut poser le bloc et qu'il n'y est pas de vide en dessous)

Barre d'inventaire :

Le joueur possède une barre d'inventaire se remplissant avec les objets qu'il collecte. Il peut interagir avec celui-ci en utilisant la souris :

- Prendre un outil de l'inventaire en main : **clic gauche** sur un outil de l'inventaire
- Crafter un outil : **clic gauche** sur l'établi dans l'inventaire puis re clic gauche sur le bouton correspondant à l'outil souhaité (si on a le nombre adéquat de fer)

Crafting :
Le personnage peut utiliser les ressources dans son inventaire en cliquant sur l'établi comme dit plus tôt. Les objets "craftables" sont :

ÉPÉE :
- Prix en crafting : 3 fer
- Permet d'affronter les ennemis en combat rapproché

ARC :

- Prix en crafting : 2 fer
- Permet d'affronter les ennemis en combat à longue distance

Combattre :
Lorsque le personnage et l'ennemi sont à la même position, le personnage se mettra automatiquement à combattre.

Subir des dégâts :
Lorsque le personnage et l'ennemi sont à la même position, le personnage se mettra à subir des dégâts de la part de l'ennemi.

Déplacement des ennemis:
Si ils sont à portée, les ennemis se dirigeront automatiquement vers le joueur.

Mort du joueur :
Le joueur meurt s'il n'a plus de PV (PV = 0). En cas de mort, le jeu se fermera automatiquement avec un message disant que le joueur a perdu.