LAST LIGHT

PROJET ZELDA-LIKE

KERROUÉ Sébastien – HAMELIN Malcom – BERTHOUT Martin

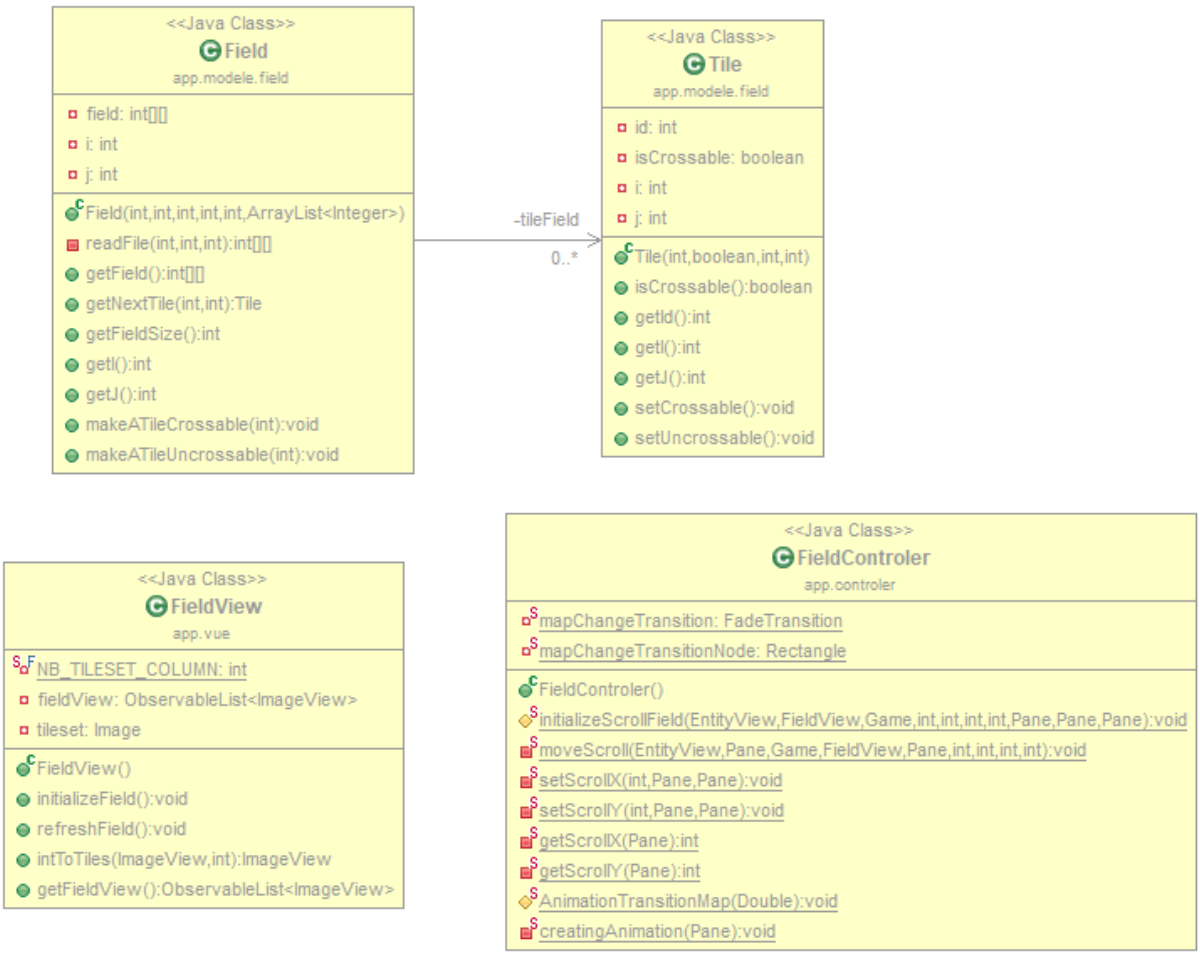
Sommaire

PARTIE 1

DOCUMENTS POUR CPOO

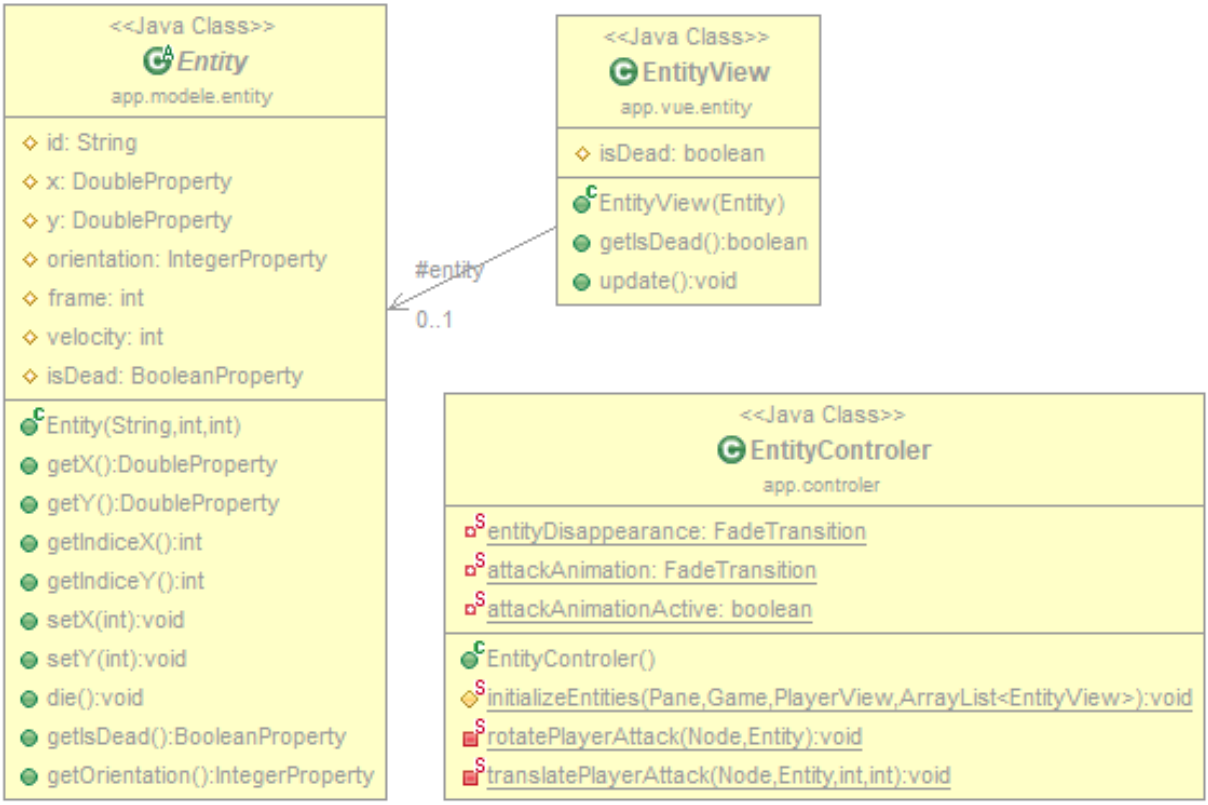
1. *Architecture*

A – Choix principaux d'architecture



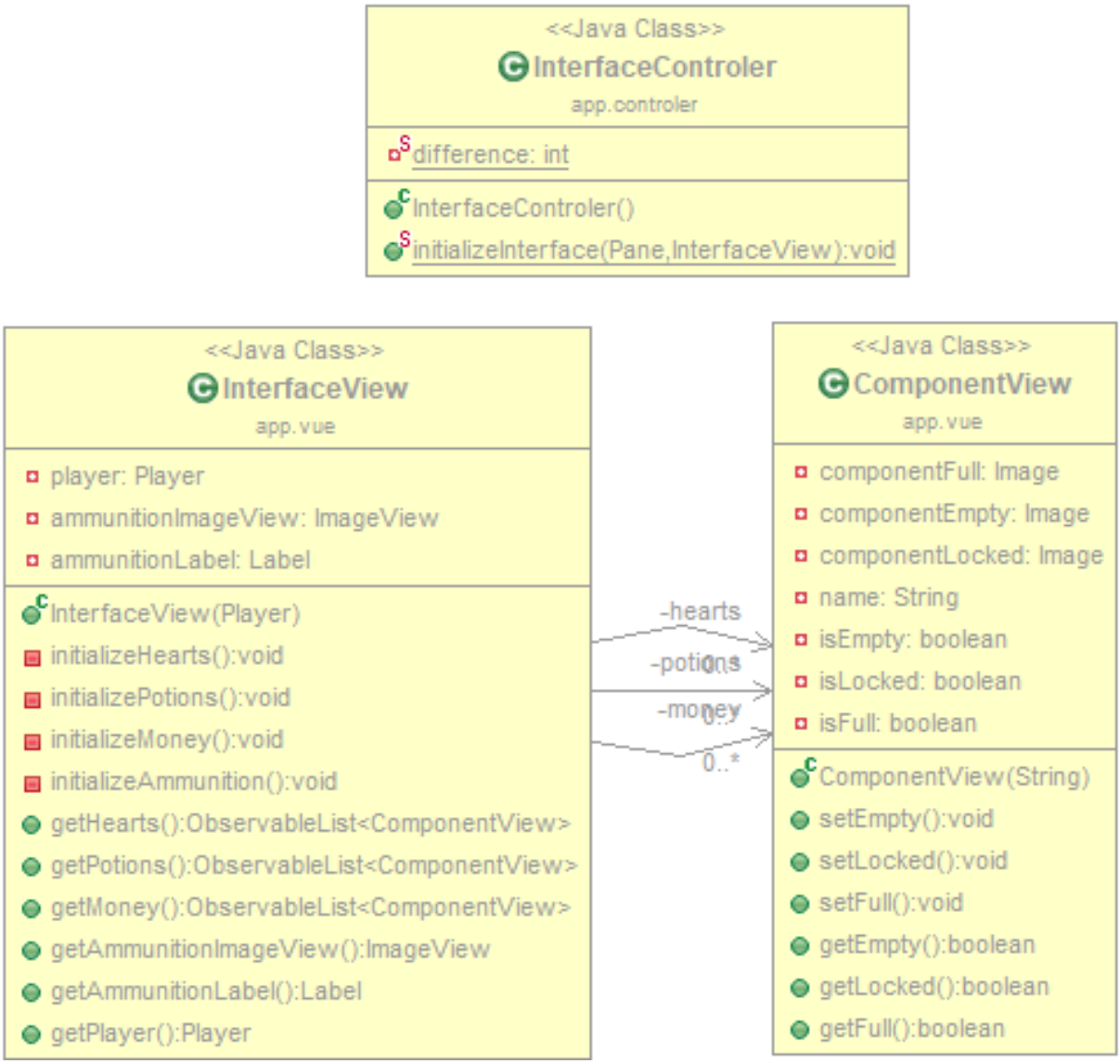
Pour poser la base de ce jeu il était nécessaire de créer un support visuel pour la carte. Ici, la classe « Field » fais office de terrain pour le jeu, est instancié dans la classe qui instancie une partie, « Game » et ajouté à un objet de type « Pane » afin d'apparaître visuellement aux yeux de l'utilisateur. Il a été décidé de faire contenir à la classe « Field » un tableau d'entier *(correspondant à la valeur des cases, et à leur position dans le tileset créé à l'aide du logiciel Tiled)*, ainsi qu'un tableau d'instances de la classe « Tile » grâce auxquelles il est possible de découper le terrain en tuiles régulières, permettant de connaître la position sur la carte, la valeur et s'il était possible de traverser la case voisine.

Afin de faire apparaître cette carte visuellement, il a été choisi d'instancier autant d' « ImageView » de l'unique « Image » du tileset qu'il y a de case dans le tableau de chaque map, afin de recadrer, grâce aux valeurs des cases, sur la bonne tuile de ce tileset, et ainsi ne pas avoir à recharger d'images lors du changement de carte mais de simplement recadrer à nouveau sur la bonne tuile.



Ensuite, il a été question d'ajouter le personnage, ainsi que d'autres entités *(ennemis, objets)*. Pour ce faire, un deuxième objet de type « Pane » a été utilisé afin de stocker toutes les instances d'entités. Ce conteneur a été créé grâce à SceneBuilder et est donc exporter depuis ce logiciel en FXML. Il est donc possible grâce à ça de récupérer les inputs de l'utilisateur via une méthode « onKeyPressed » appliquée à cet objet « Pane ».

Chaque entité à de fait un équivalent de classe dans le package Vue, et instanciée dans le package Controler, dont les caractéristiques sont liées *(« bindés »)* à leur équivalent du modèle *(les coordonnées dans le modèle par exemple représentent la position en pixel de la contrepartie visuelle de cette entité)*.

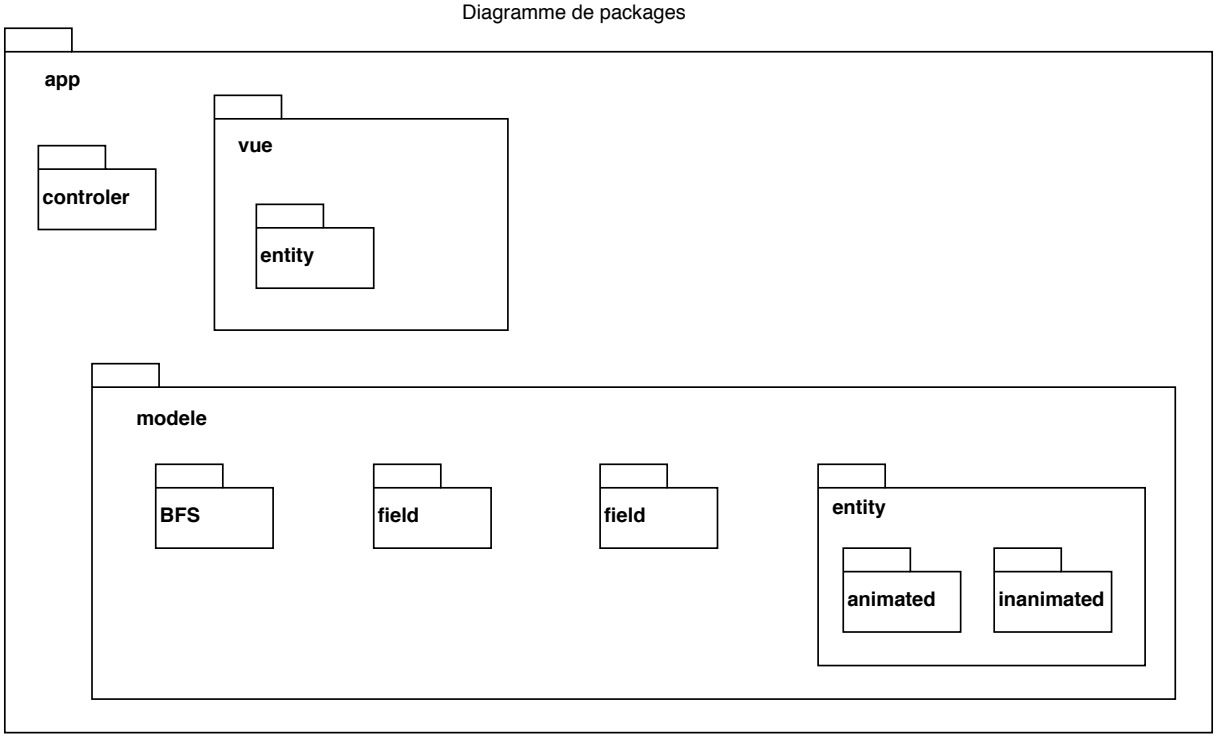


Un troisième objet « Pane » a été nécessaire afin de faire connaître au joueur ses caractéristiques en toute circonstances. Il est composé d'objets écoutant seulement les caractéristiques du joueur. L'objet « ComponentView » a été créé afin de donner trois états possibles aux cœurs *(points de vie)*, aux potions et aux pièces : plein, vide ou à débloquer. Quand un objet « ComponentView » est créé *(au lancement du jeu donc),* il charge ses trois états possibles et en change en fonction des caractéristiques du joueur grâce à des écouteurs *(« listeners »).*

Enfin, un quatrième et dernier « Pane » est utilisé et ce dans le seul intérêt de l'expérience utilisateur et sans impact sur le modèle, afin d'avoir un écran de pause, contenant une image détaillant les contrôles du jeu et permettant de quitter proprement sa session *(ou bien de la reprendre).*

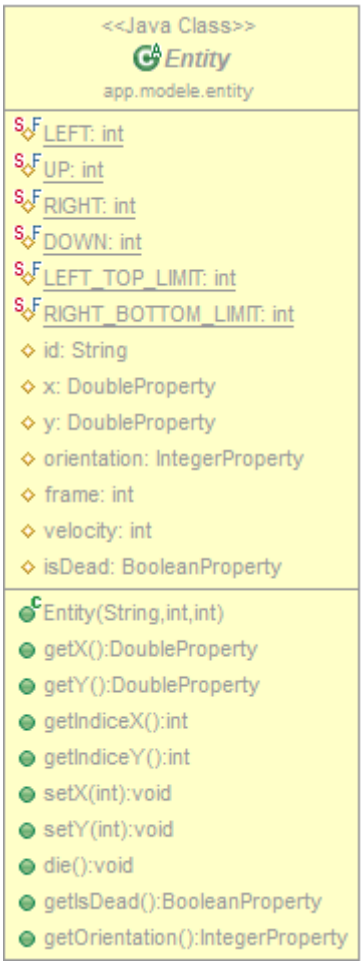
B – Choix principaux d'architecture

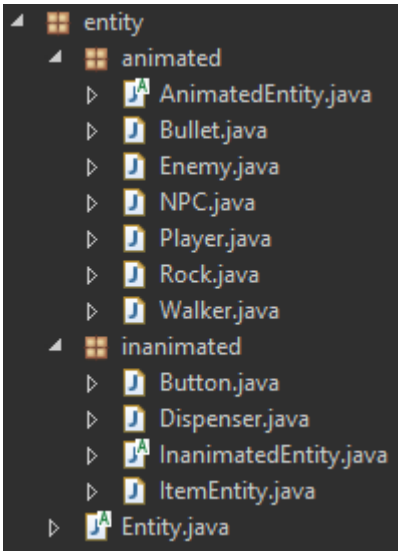
Diagramme de packages



2. *Diagrammes de classe*

A – Classes « Entity »





Le choix a été fait de considérer comme entité tout ce qui devait apparaître visuellement à l'écran pour le joueur du terrain de jeu. Ici, cette classe abstraite se divise donc en deux sous classes :

- La classe « AnimatedEntity » gérant toutes les entités étant amenées à se déplacer ou à être déplaçables.
- La classe « InanimatedEntity » s'occupant de toutes les entités ne se déplaçant pas.

La classe « AnimatedEntity » se divise donc par la suite en sous classes représentant différents objets, personnages et ennemis :

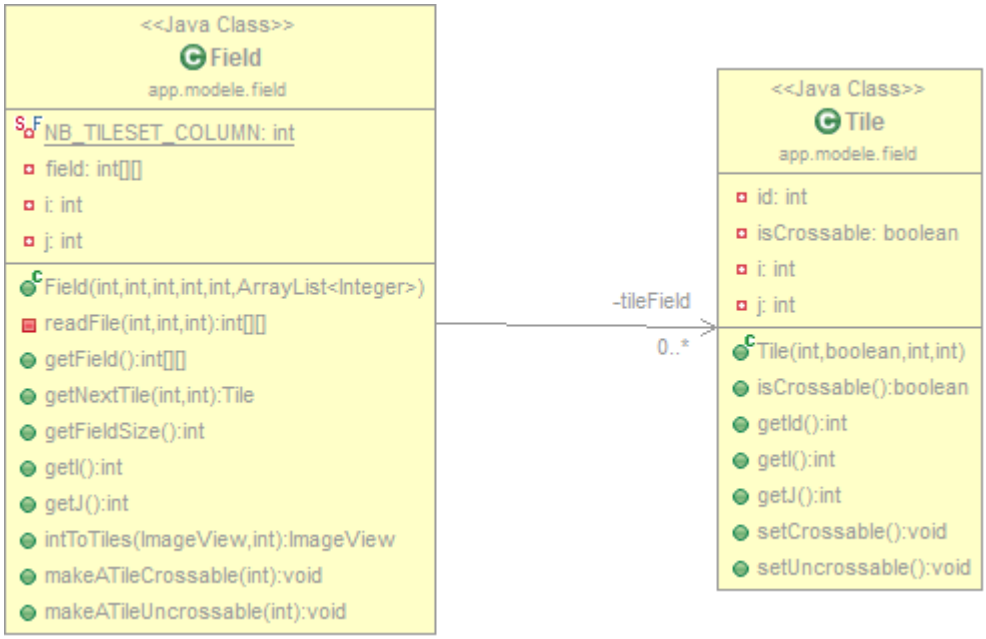
- La classe « Player » représente le joueur qui est un personnage doté de paramètres nécessitant une gestion particulière *(l'exécution de méthodes après que certains inputs aient été détectés par le contrôleur par exemple).*

- La classe « Enemy » représente une superclasse regroupant tous les types d'ennemis *(entités amenées à faire des dégâts au joueur),* tels la classe « Walker », représentant les ennemis basiques faisant des dégâts au corps à corps.
- La classe « NPC » (*Non Playable Character),* représentant tous les personnages autres que le joueur et les ennemis.
- La classe « Bullet », permettant d'instancier chacun des projectiles générés par une action du joueur ou d'un ennemi.
- La classe « Rock » définissant un objet déplaçable par le joueur.

La classe « InanimatedEntity » elle se divise en différentes sous classes permettant de définir des objets avec lesquels le joueur pourra interagir :

- La classe « Button » est un objet sur lequel le joueur appuie afin d'interagir avec une autre instance d' « InanimatedEntity ».
- La classe « Dispenser » représente un distributeur avec lequel le joueur interagit et qui affiche une boîte de dialogue adaptée à la situation du joueur.
- La classe « ItemEntity » permets de représenter les objets n'ayant pas de comportement particulier et ne nécessitant pas une sous classe dédiée.

B – Classes « Field » et « Tile »



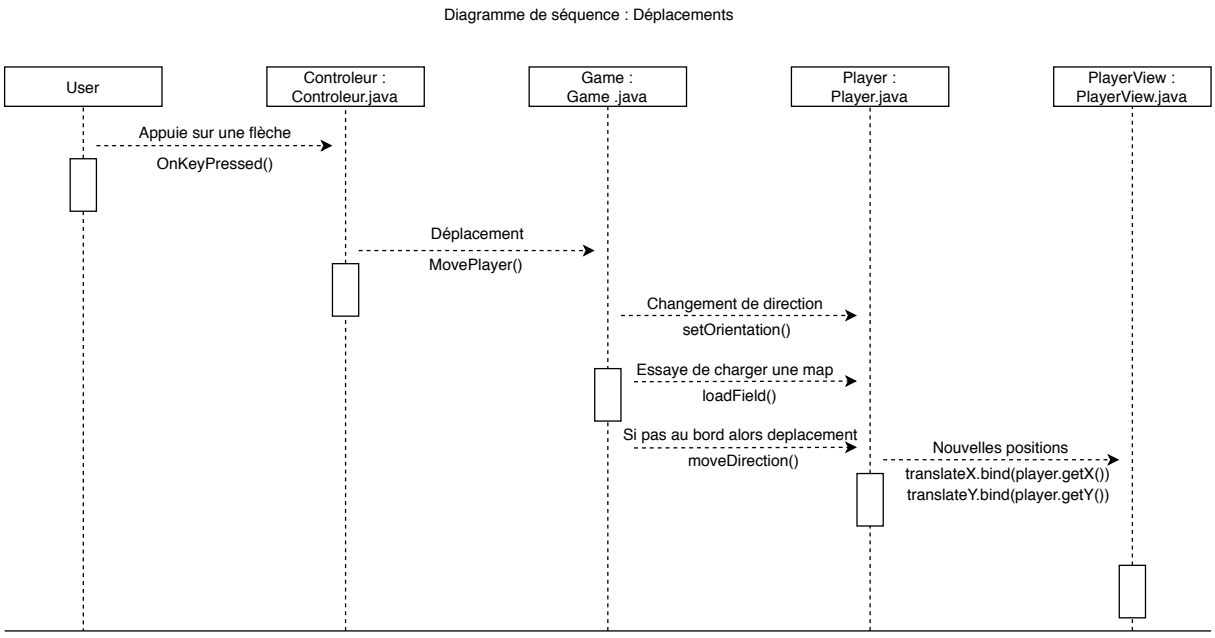
Chaque carte sur laquelle le joueur se déplace est une instance de la classe Field, qui contient elle-même un tableau composé de Tile représentant chacune des tuiles du plateau de jeu. Chaque instance de Field contient donc un tableau de valeurs entières représentant

l'identifiant de chaque tuile, ainsi que sa position sur la carte particulière permettant de situer chaque instance de field par rapport aux autres instances de field, position qui est représentée grâce à un indice « i » et un indice « j » correspondant à l'emplacement dans un tableau d'entier.

La classe « Field » contient des méthodes permettant de récupérer un élément de son tableau de « Tile » afin de connaître la valeur de son booléen « isCrossable » permettant de savoir si une tuile est traversable par une instance de la classe « Entity ». Elle possède aussi une méthode permettant de rendre toutes les tuiles d'un identifiant donné traversable, afin de permettre le passage d'entités sur ces tuiles lorsque certaines actions sont effectuées par le joueur. Par équivalence, la classe « Tile » contient des méthodes permettant de mettre ce booléen à vrai ou à faux.

3. Diagrammes de séquence

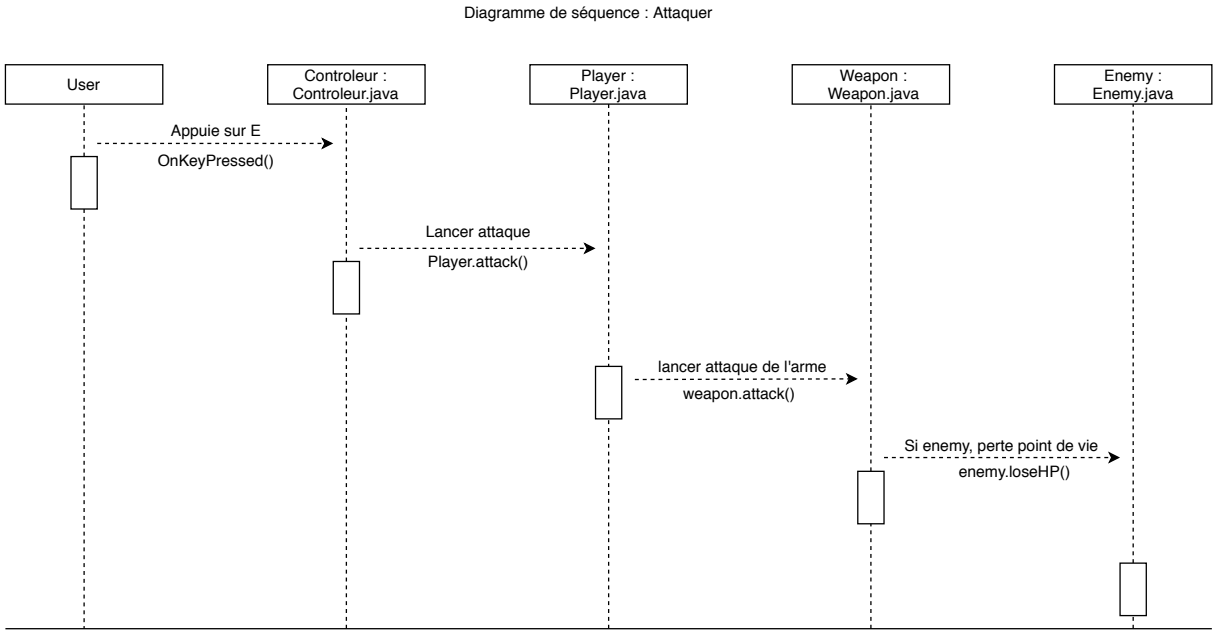
A – Déplacements



Lorsque l'utilisateur appuie sur les flèches du clavier, le contrôleur récupère l'information sous forme de « KeyEvent », via la méthode « onKeyPressed(KeyEvent k) ». Peu importe la flèche, il appelle la méthode « movePlayer(KeyEvent k) » de l'instance de « Game » qui elle-même effectue des vérifications : elle vérifie sur le jouer est au bord de la map, auquel cas elle essaie de charger la carte qui est de ce côté. S'il n'y a pas de map, alors le joueur est au bord et ne

peut pas de toutes façons pas aller de ce côté, et s'il y en a une, alors elle est chargée et les caractéristiques du joueur sont changée afin de le faire au bon endroit. S'il n'est pas au bord, alors la méthode « move*Direction*() » *(avec Direction changeant selon la flèche appuyé)* de l'instance de « Player » est appelé et cette méthode s'occupe de vérifier si la case visée est occupée et si elle ne l'est pas, alors les coordonnées du joueur changent. Enfin, la liaison (le binding) sur les coordonnées de « Player » par « PlayerView » change l'emplacement visuel de l'image du joueur.

B – Attaquer



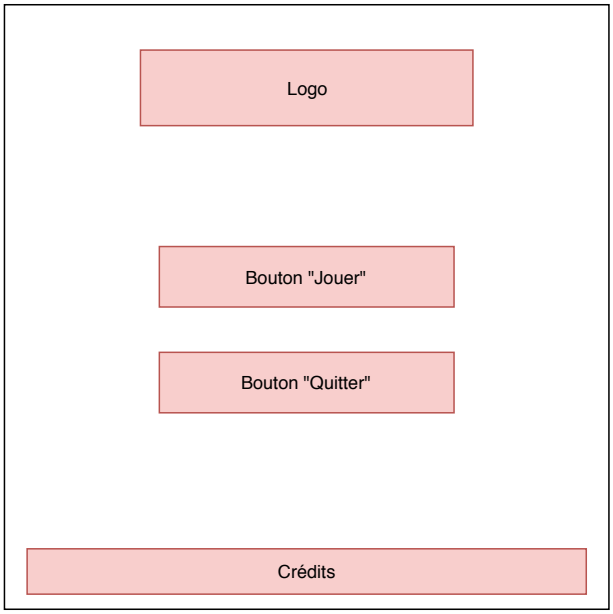
Lorsque l'utilisateur appuie sur la touche pour attaquer *(ici « E »)*, le contrôleur récupère l'information sous forme de "KeyEvent", via la méthode « onKeyPressed (KeyEvent k) », qui elle appelle la méthode « attack () » de l'instance de « Player », si le joueur n'a interagi avec rien *(c'est à dire qu'aucune entité inanimée doté d'une interaction ne se trouve devant lui).* Cette méthode appelle la méthode "attack ()" de l'arme active du joueur s'il en possède une *(sinon n'appelle rien).* Cette méthode "attack ()" de l'arme active vérifie s'il y a un ennemi à portée, et si c'est le cas, lui inflige les dégâts à l'aide de la méthode « enemy.loseHP(weapon.getAttack()) ».

PARTIE 2

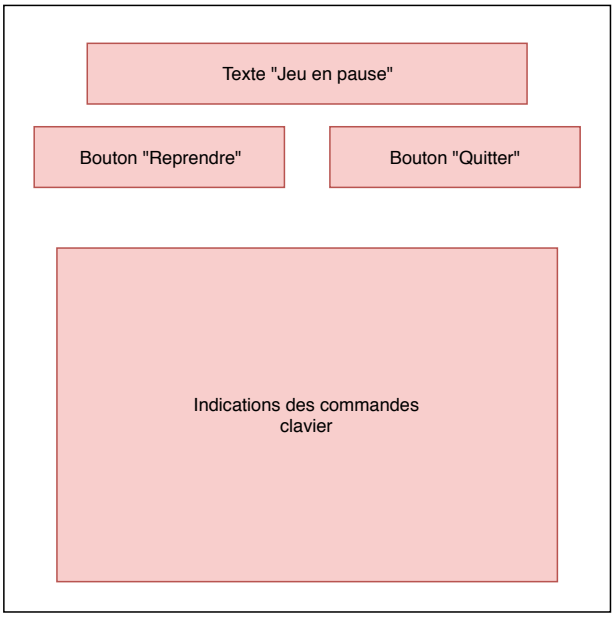
DOCUMENTS POUR IHM

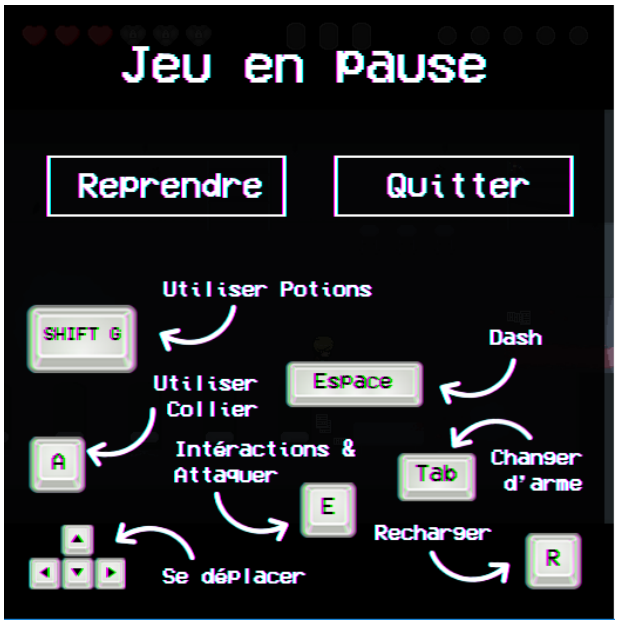
1. *Maquettes*

A – Menu de lancement

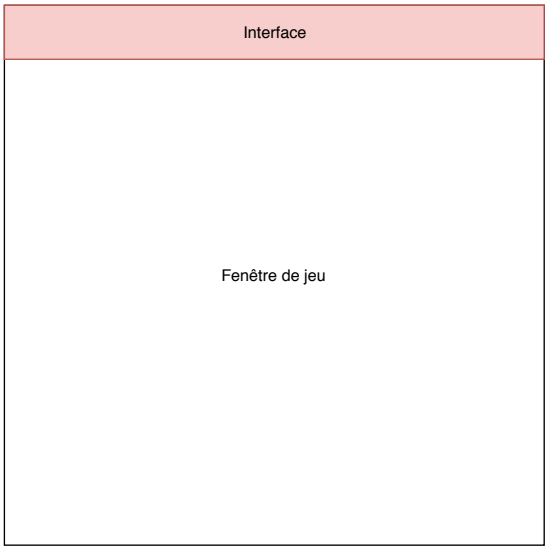


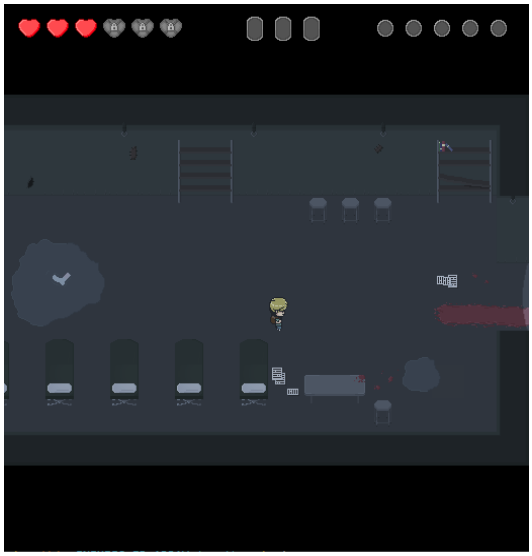
B – Menu de pause

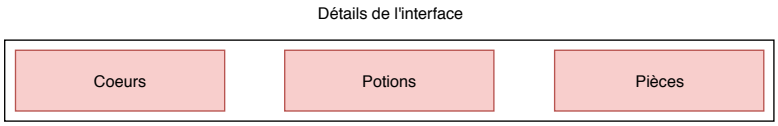




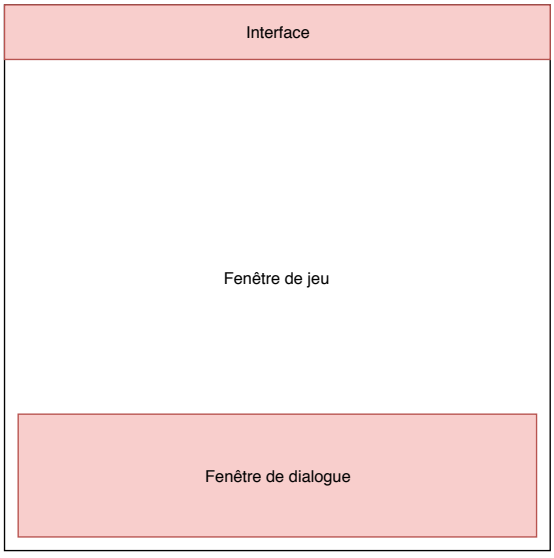
C – Écran classique de jeu







D – Écran de jeu lors d'un dialogue

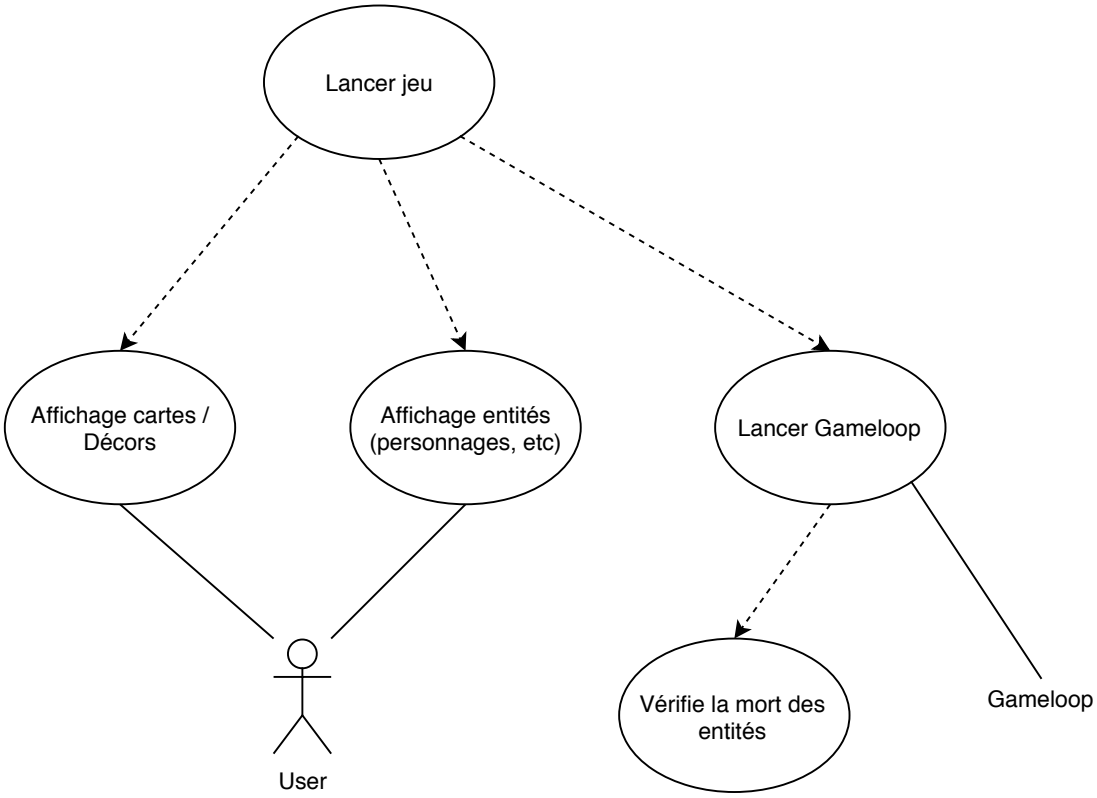


PARTIE 3

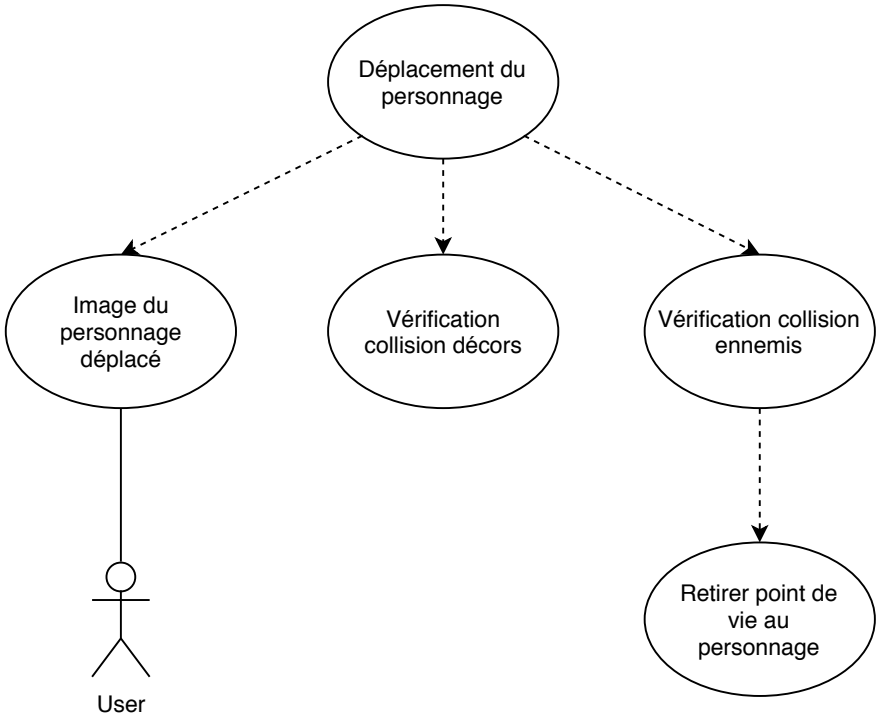
DOCUMENTS POUR GESTION DE PROJET

1. *Cas d'utilisation*

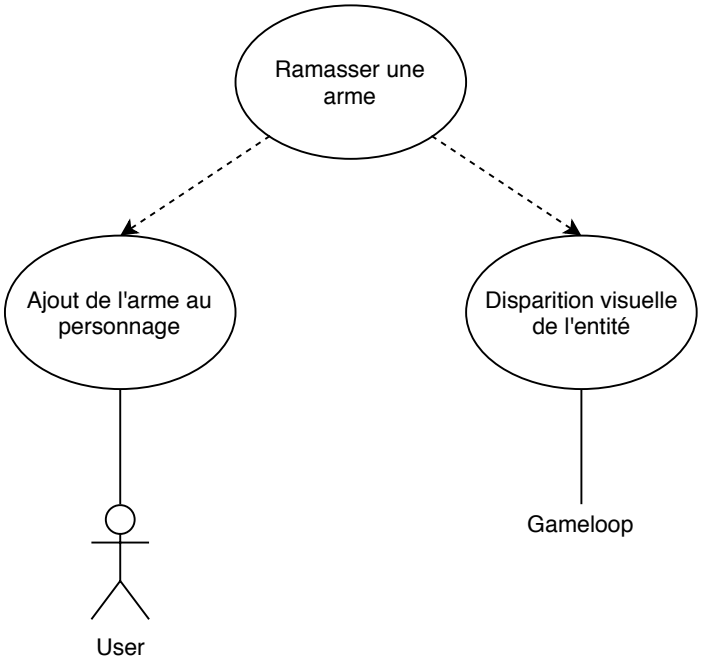
A – Lancement du jeu



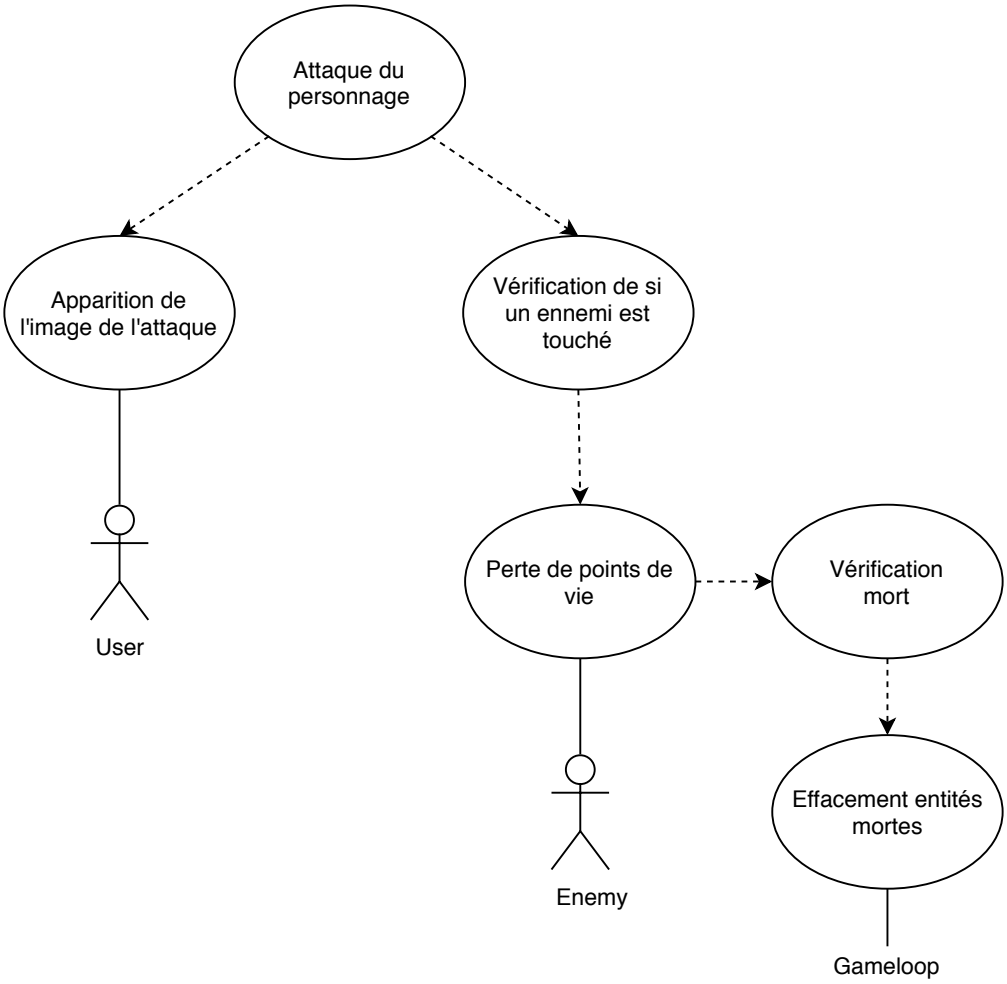
B – Déplacement du personnage



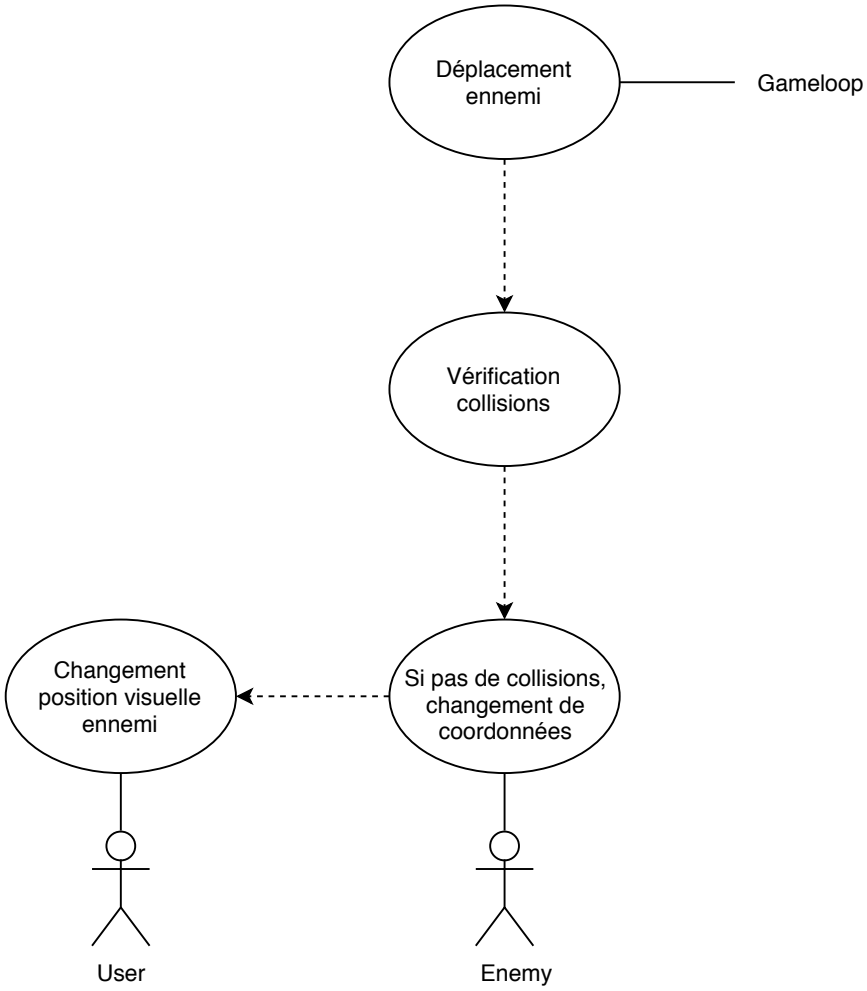
C – Ramasser un objet



D – Attaquer



E – Déplacement ennemi



2. *Cahier d'initialisation*

A – Scénario

Le joueur se réveille seul dans une tente militaire médicale sans savoir comment il s'est retrouvé là ni ce qu'il se passe. Il trouve une **lampe torche** à côté. Plus loin il rencontre un premier monstre. Il trouve ensuite un cadavre avec un journal sur lui. Celui-ci indique que les ennemis sont sensibles à la lumière et qu'allumer le grand phare serait une solution.

En continuant son chemin, il arrive à une intersection avec trois chemins. L'un est inaccessible à cause d'un trou dans le sol. Le deuxième est bloqué par un PNJ *(Personnage Non-Joueur)* qui ne veut pas laisser passer le joueur. Seul le troisième chemin est disponible. Le joueur arrive

donc au phare au bout de ce chemin. Cependant, pour accéder au phare, il faut traverser un mur spécial et le joueur ne peut pas entrer.

En revenant en arrière, le PNJ qui bloquait un des chemins est mort. Au bout de ce chemin, le joueur trouve les **bottes** après avoir passé une pièce avec des **ennemis non-craintifs**. Le joueur se rend compte qu'il est impossible de passer sans éteindre la lumière car un PNJ y rentre avant lui et meurt.

Le joueur retourne en arrière et se dirige donc vers le dernier chemin qui est maintenant accessible. Au début du chemin, il voit un PNJ qui se fait attaquer dans un endroit inaccessible. Le PNJ lui dit de prendre son **boomerang** qui est à côté du joueur pour le sauver. Le joueur conserve le boomerang. Au bout il trouve un magasin avec des boutons à activer dans un certain ordre pour entrer. A l'intérieure il trouve un cadavre avec dessus le **collier** d'intangibilité ainsi qu'une notice. Juste après la lumière s'allume et des **ennemis non-craintifs** l'empêchent de revenir en arrière. Il doit donc utiliser son collier pour passer à travers un mur spécial. Il sort du magasin et se rend au phare.

Lorsque le joueur entre dans le phare, il doit déjouer plusieurs énigmes au cours desquelles il aura l'occasion de récupérer le **taser** et vaincre différents monstres afin d'atteindre le sommet du phare où il rencontre le boss. Une fois le boss vaincu, il allume le phare.

B – Mécaniques

- Shift gauche : Boire une potion
- Barre espace : dash
- Touche A : Utiliser collier
- Touche E : Interactions & Attaquer
- Tabulation : Changer d'arme
- Touche R : Recharger
- Flèches directionnelles : se déplacer

C – Armes

Lampe torche :

- Il s'agit de l'arme trouvée par le joueur au début du jeu
- Distance d'attaque : une case
- Attaque au corps à corps
- Dégâts faibles

Boomerang lumineux :

- Il s'agit de la seconde arme trouvée par le joueur
- Distance d'attaque : trois cases
- Attaque à distance *(mouvement d'aller-retour comme un boomerang)*
- Dégâts faibles
- Permet d'actionner certains boutons / objets à distance

Taser :

- Il s'agit de la dernière arme trouvée par le joueur
- Distance d'attaque : une ligne
- Attaque à distance
- Dégâts élevés
- Consomme des munitions

D – Objets

Objets permanents :

- Chaussures de courses
  - Octroi un dash
  - Permet de sauter au-dessus des trous
  - Possède un temps de rechargement (1s)
- Collier
  - Permet de devenir intangible
  - Permet de passer à travers les ennemis et certains murs spéciaux
  - Possède un temps de rechargement (5s)

Objets consommables :

- Potion
  - Peut s'acheter auprès des distributeurs avec deux pièces
  - Remplit la barre de vie
  - Peut se trouver
- Argent
  - Peut s'obtenir en tuant des ennemis
  - Permet d'acheter des potions auprès des distributeurs
- Cœurs :
  - Accroit de 1 le nombre de vie maximale

Objets du décor :

- Bouton
  - Peut s'actionner pour libérer certains passages
- Distributeur
  - Sont des points de sauvegarde
  - Proposent des potions à la vente
- Murs spéciaux
  - Peuvent être traversés lorsque le joueur est intangible
- Rochers
  - Bloque le passage
  - Peut être déplacé

E – Ennemis

Rôdeur :

- Attaque au corps à corps
- Se déplace vers le joueur en marchant

Volant :

- Attaque à distance
- Se déplace en volant

Rôdeur blessé :

- Attaque à distance
- Ne se déplace pas, est immobile

Tanky :

- Attaque au corps à corps
- Sensible uniquement dans le dos
- Se déplace vers le joueur en marchant

Ennemi non-craintif

- Attaque au corps à corps
- N'est pas sensible
- Saute sur le joueur et lui inflige de lourds dégâts lorsque la pièce est allumée
- Inoffensif lorsque la pièce est éteinte

F – Donjon

Le donjon est représenté par le phare que le joueur doit allumer. Il pourra accéder au donjon lorsqu'il aura trouvé les bottes et le collier. Le donjon est composé de plusieurs salles. Certaines sont remplies d'ennemis à tuer, d'autres d'énigmes.

G– Boss

Le boss final est un monstre qui a absorbé l'ampoule du phare à l'intérieur de lui. Il est constitué de 3 phases.

Phase 1 :

- Lance des projectiles sur le joueur
- Se déplace horizontalement face au joueur

Phase 2 :

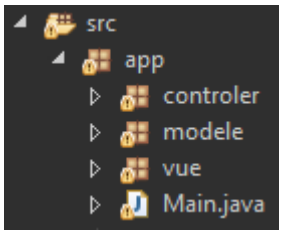
- La phase 2 se déclenche lorsqu'il ne reste que 50% de ses PV au boss
- Reprends les mécaniques de la phase 1
- Occasionnellement, fonce sur le joueur pour l'écraser

Phase 3 :

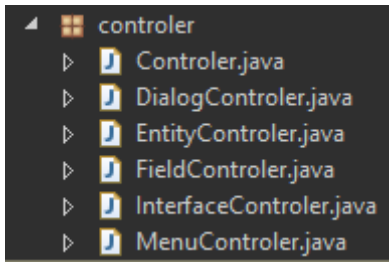
- La phase 3 se déclenche lorsqu'il ne reste que 25% de ses PV au boss
- Devient insensible tant que le joueur n'a pas désactivé deux générateurs qui se présenteront à lui
- Reprends les mécaniques des phases 1 et 2
- Occasionnellement, tire un laser qui balaie la zone du joueur avec l'ampoule de son torse.

3. Cahier des spécifications fonctionnelles

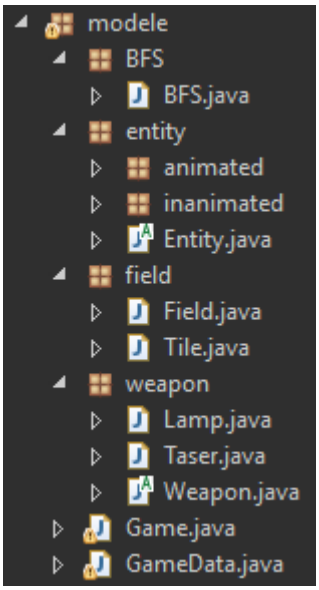
A – Arborescence

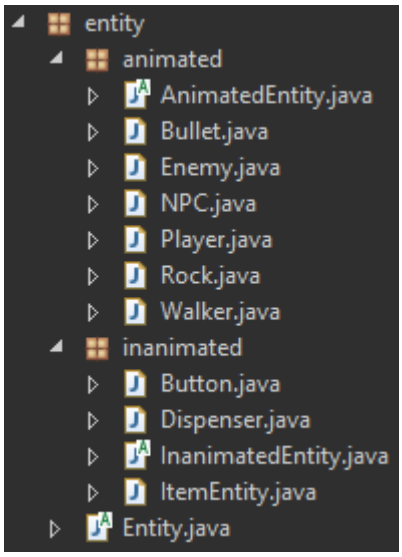


Ce projet est réalisé à partir de la structure MVC *(Modèle-Vue-Contrôleur),* où le modèle est indépendant, le contrôle appelle les changements dans le modèle et la vue représente graphiquement les données du modèle.



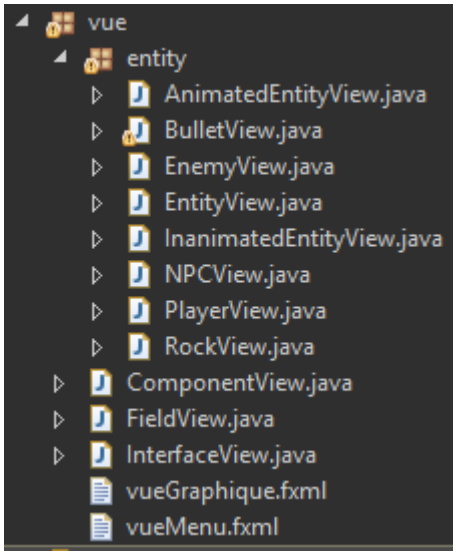
Le contrôleur est doté de plusieurs classes afin de gérer toutes les parties du modèle, ainsi que de convertir les entrées du clavier et de la souris en actions. Ici, par exemple, la classe « EntityControler » permets d'initialiser l'apparition des entités générées dans le modèle tout en liant à l'aide de listeners les changements du modèle à leur contrepartie visuelle *(le mouvement de l'image lorsque les coordonnées du personnage changent, la disparition d'ennemis lorsqu'ils meurent, etc).* La classe principale « Controler » récupère les inputs de l'utilisateur et les transforme en actions dans le modèle.





Le modèle est ainsi composé de différents types d'objets :

- La classe principale est la classe « Game », qui est instanciée lors du lancement et permet la création d'une partie de jeu et de la gestion de ce qui la compose *(terrain, entités, armes et BFS).* Sa classe complémentaire « GameData » contient ses constantes les plus utilisées.
- La classe « BFS » est utilisée afin de trouver les chemins les plus courts en fonction d'un point donné.
- La classe « Entitiy » (et ses sous-classes) permettent de représenter chacune des entité (personnage, ennemis, objets) présentes dans le jeu.
- La classe « Field » permet de gérer le terrain et contient un tableau d'entiers représentant ce dernier ainsi qu'un tableau de « Tile » permettant de savoir si une case est traversable ou non. La classe « Tile » est elle-même composée de sa position dans le tableau de « Tile » ainsi que son ID numérique et l'information de si elle est traversable ou non.
- La classe « Weapon » permet d'ajouter et de gérer les différents types d'arme que peut posséder le joueur.



La vue permet de représenter les contreparties visuelles des éléments du modèle afin d'offrir une interface graphique au joueur. Elle permet aussi, grâce aux informations récupérées dans le modèle, de créer des animations (donner l'impression d'un mouvement de jambe lors du mouvement d'un personnage par exemple). Pour ce faire, il est important de créer une architecture cohérente au modèle afin de donner un comportement visuel similaire aux éléments qui se comportent de manière similaire dans le modèle, et de prévoir l'inverse, en créant des classes différentes lorsque cela est nécessaire.

Les éléments nouveaux de la partie vue par rapport au modèle étant les classes « InterfaceView » et « ComponentView », permettant de représenter visuellement les informations que le joueur à besoin de connaître à tout moment sur ses propres caractéristiques. Ici, la classe « InterfaceView » est composé de différents objets « ComponentView » représentant chacun un élément défini dans « InterfaceView » afin d'afficher, par exemple, chaque cœur *(représentant le nombre de points de vie du personnage),* ou objet en sa possession sur l'écran de jeu.

B – Spécifications fonctionnelles

Déplacement du personnage :

Cas	Résultat
Se déplacer sur une case vide	Déplacement possible, changement des coordonnées du personnage
Se déplacer sur une case avec un ennemi	Déplacement impossible, le personnage garde les mêmes coordonnées mais change de direction
Se déplacer en bordure de carte	Essaie de charger la map qui est de ce côté si elle existe, sinon déplacement impossible mais changement de direction
Se déplacer sur une case avec un objet	Déplacement impossible, le personnage garde les mêmes coordonnées mais change de direction

Attaque du personnage :

Cas	Résultat
Attaquer une case vide	Recherche d'une entité avec laquelle interagir, sinon, essaie de trouver un ennemi sur la case en face de lui, déclenche l'animation de l'arme
Attaquer une case avec un ennemi	Recherche d'une entité avec laquelle interagir, sinon, essaie de trouver un ennemi sur la case en face de lui, lui enlève des points de vie, égaux au montant de l'attaque de l'arme, s'il existe, déclenche l'animation

	de l'arme
Attaquer une case non vide sans ennemi	Recherche d'une entité avec laquelle interagir, ne déclenche pas l'animation de l'arme
Attaquer en bordure de carte	Recherche d'une entité avec laquelle interagir, recherche d'un ennemi à attaquer, déclenche l'animation de l'arme

Utilisation du collier d'intangibilité :

Cas	Résultat
Utiliser le collier quand on ne le possède pas	Lance la méthode d'utilisation du collier, vérifie s'il est possédé, ce n'est pas le cas donc effet non déclenché
Utiliser le collier lorsqu'on le possède et qu'il est disponible	Lance la méthode d'utilisation du collier, rends la case spéciale traversable par le joueur, provoque une transparence visuelle sur le joueur, et remets à leurs états initiaux la case et le joueur une fois que l'animation est finie
Utiliser le collier lorsqu'on le possède et qu'on est en train de l'utiliser	Lance la méthode d'utilisation du collier, vérifie si on est en train de l'utiliser, ce qui est le cas donc ne déclenche pas l'effet
Utiliser le collier lorsqu'on le possède et qu'il est en train de se recharger	Lance la méthode d'utilisation du collier, vérifie s'il est disponible, ce qui n'est pas le cas donc ne déclenche pas l'effet

Utilisation des bottes (dash) :

Cas	Résultat
Utiliser les bottes alors qu'on ne les possède pas	Lance la méthode d'utilisation des bottes, vérifie si on les possède, ce qui n'est pas le cas donc la méthode ne se déclenche pas
Utiliser les bottes alors qu'on les possède et que les deux cases en face sont vides	Lance la méthode d'utilisation des bottes, vérifie qu'on les possède, vérifie si les cases sont libres, donc effectue le dash complet

Utiliser les bottes alors qu'on les possède et qu'un obstacle (décor) est présent sur l'une des deux cases	Lance la méthode d'utilisation des bottes, vérifie qu'on les possède, vérifie l'espace libre disponible devant, effectue le dash sur cette distance

Utiliser potion :

Cas	Résultat
Utiliser une potion alors que les points de vie sont au maximum	Lance la méthode pour utiliser une potion, vérifie si les points de vie sont au maximum, alors ne l'utilise pas
Utiliser une potion alors que les points de vie ne sont pas au maximum mais > 0	Lance la méthode pour utiliser une potion, vérifie si les points de vie sont au maximum et > 0, alors rends tous les points de vie au joueur
Utiliser une potion alors que les points de vie sont <= 0	Lance la méthode pour utiliser une potion, vérifie si les points de vie sont au maximum et > 0, alors ne l'utilise pas

Utiliser argent :

Cas	Résultat
Utiliser des pièces alors que l'on n'en possède pas ou pas assez	Lance la méthode utilisant des pièces, vérifie si le joueur en possède suffisamment, n'effectue donc pas l'action
Utiliser des pièces alors que l'on en possède assez et que l'on ne possède pas le maximum de l'objet acheté	Lance la méthode utilisant des pièces, vérifie si le joueur en possède suffisamment et si le joueur a le maximum de l'objet voulu, effectue l'action
Utiliser des pièces alors que l'on en possède assez et que l'on possède le maximum de l'objet acheté	Lance la méthode utilisant des pièces, vérifie si le joueur en possède suffisamment et si le joueur a le maximum de l'objet voulu, n'effectue pas l'action

Se faire attaquer (entité animée) :

Cas	Résultat
Perdre des points de vie mais en avoir toujours > 0	Les points de vie sont enlevés, mais l'entité reste (dans le cas du joueur, les cœurs disparaissent visuellement proportionnellement aux points de vie perdus)
Perdre des points de vie mais en avoir == 0	Les points de vie sont enlevés, l'entité est supprimée (dans le cas du joueur, le jeu est terminé)
Perdre des points de vie mais en avoir < 0	Les points de vie sont enlevés, l'entité est supprimée (dans le cas du joueur, le jeu est terminé)