Rapport Général Itération 3

7elda

Groupe GH-4 /Année 2022

CRUVELLIER Baptiste
EL YESSEFI Mohamed
LABISTE Lilian
MUSSARD Cassandra
NICOBAHARAYE Brice-nathan
NOUY Tess
WU Christophe

Table des matières

1 Introduction

Nous réalisons un jeu reprenant le principe des premières versions de Zelda. C'est un jeu d'action et d'aventure vu en troisième personne depuis le ciel. Le personnage (nommé Nicobaharaye) que l'on contrôle peut se déplacer en 2D sur une carte verticalement et horizontalement. Il doit vaincre des ennemis en utilisant des flèches.

2 Fonctionnalités

Pour la première itération nous avons réalisé les fonctionnalités suivantes :
- Réalisation de la carte.
- Déplacement du joueur par l'utilisateur via les touches (Q, Z, S, et D) du clavier.
- Réalisation du menu principal.
- Implantation d'un ennemi représenté par une case noire qui se déplace sur la carte.
- Réalisation de la collision (nous avons presque fini d'implanter cette classe).
- Réalisation d'une première salle.

Pour la deuxième itération nous avons réalisé les fonctionnalités suivantes :
-Réalisation de 3 nouvelles salles (avec 2 graphismes différents correspondant aux différentes pièces du jeu (magasin, salles où se cachent les clés et salle où se situe la princesse)).
-Passage d'une salle à l'autre.
-Réalisation d'un inventaire contenant les objets du joueur.
-Construction de la barre de vie du joueur, visible en haut à gauche de l'écran.
-Possibilité pour le joueur de lancer des flèches.
-Construction de la collision des flèches avec l'ennemi et réalisation de la disparition de l'ennemi lorsque celui-ci est tué.
-Réalisation d'une fenêtre intermédiaire entre le menu principal et le jeu pour expliquer l'objectif du jeu.
-Insertion d'une musique.

Pour la dernière itération nous avons effectué le travail suivant :
- Changement de la position du joueur lorsque l'on change de salle,
- Collision entre les différentes entités (personnage et ennemis, personnage et princesse ou druide).
- Dégâts des ennemis sur le personnage (perte de vie).
- Réalisation du magasin (achat d'objets).
- Affichage d'un message Game Over lorsque le personnage n'a plus de vie.
- Affichage des dialogues entre les différentes personnages du jeu.
- Possibilité de récupérer des objets.
- Possibilité de sauvegarder la partie.
- Possibilité de récupérer des clés pour ouvrir la salle du Boss et de la Princesse.
- Possibilité de mettre en pause le jeu.
- Attaque du boss.
- Blocage des portes des salles du boss et de la princesse tant que l'utilisateur n'a pas récupéré les clés associées.
- Réglage de la collision entre les flèches et les portes des salles.
- Faire tourner les pièces et les ramasser.
- Ajout d'un logo personnalisé quand on lance l'application.
- Affichage de la map (pas finalisé).

Les fonctionnalités qui n'ont pas pu être réalisées pendant les 3 itérations sont :
- Possibilité pour l'utilisateur de choisir un niveau de difficulté (ennemis plus robustes au combat).
- Combat du joueur avec une épée.
- Utilisation de potions et de bouclier pour rendre plus robuste le joueur.
- Attaque des petits ennemis dans les salles.
- Réalisation de différents ennemis.
- Dialogue avec le boss pour savoir où se situe la princesse et comment la libérer.
- Personnalisation de l'apparence du personnage principal.

3 Découpage de l'application

Nous avons découpé notre application en 3 paquetages.
Nous avons tout d'abord un paquetage "Main" où nous avons implanté le menu principal. Dans ce paquetage nous avons :
- La classe jeu qui se charge d'afficher les différentes entités (joueur, ennemis), d'actualiser les positions des entités sur la map.
- La classe ControleClavier qui permet de récupérer des informations tapées sur le clavier par l'utilisateur,
- La classe Collision qui permet d'éviter qu'un joueur ou un ennemi ne rentre dans les murs ou les obstacles.
- La classe Inventaire qui permet d'ajouter et de retirer des objets dans l'inventaire et d'associer les textures aux objets.
- Les classes InventairePleinException et InventaireVideException.
- La classe Main qui permet de lancer le jeu.
- Les classes CreationSauvegarde, LectureSauvegarde, MaxSauvegardeException et MenuSauvegarde qui permettent de sauvegarder la partie.
- La classe MapEntier qui permet d'afficher la map en entier.
- La classe NewMenuPrincipal qui permet la création du menu.
- la classe GUI qui permet d'afficher à l'écran l'inventaire, les dialogues entre les personnages, le magasin, le message Game Over.

Ensuite nous avons un second paquetage ("Map") qui permet de définir la carte dans laquelle le joueur va pouvoir se déplacer. Dans ce paquetage nous avons définit :
- Une classe Map.
-Une interface Salle que les classes SalleClassique, SalleClassiquepassage, SalleClassique3, Salleboss, Magasin, SalleSecrete et SallePrincesse doivent réaliser. Chaque classe réalisant cette interface permet de définir la salle où le personnage se situe et d'associer des nombres (d'un fichier .txt) à des textures. Elle permet aussi de dessiner à l'écran cette salle.
-Une classe Importationsalle qui permet d'obtenir la salle dans laquelle se trouve le personnage.

Finalement, nous avons un dernier paquetage ("Entite") qui permet d'implanter les différentes entités du jeu, comme le joueur principal ou les ennemis. Les différentes classes composant ce paquetage sont : - Les classes Boss, Druide, Ennemi, Ennemi1, Joueur et Princesse qui correspondent aux différents personnages présents dans le jeu et qui implémentent l'interface Entite.
- La classe Fleche qui permet la gestion des flèches.
- La classe Vie qui permet la gestion de la vie.
- La classe CollisionObjet qui traite la collision entre les objets et le joueur.
- Les classes ListeObjet, Objet et ObjetCollectable qui permettent la gestion des objets.

4 Méthodes Agiles

4.1 TD1

Nous avons réalisé un Pitch lors du 1er TD pour mieux définir notre application. Voici les informations importantes que nous avons noté :

Pour : les amateurs de jeu d'aventure.

Qui souhaitent : redécouvrir les jeux rétro.

Notre produit est : un jeu vidéo se nommant 7elda.

Qui : reprend les codes des premiers Zelda avec des mécaniques plus actuelles.

A la différence des : nouveaux jeux d'aventures.

Il permet de : retrouver cette ambiance simpliste (par les graphismes en pixel-art, la musique) qui est centré sur les combats et les énigmes.

Ensuite nous avons aussi effectué la charte du produit :

Vision produit

Pour les amateurs de jeu d'aventure qui veulent redécouvrir les jeux rétro, notre produit est un jeu vidéo se nommant 7elda qui reprend les codes des premiers Zelda avec des mécaniques plus actuels à la différence des nouveaux jeux d'aventures qui permet de retrouver cette ambiance simpliste (par les graphismes en pixel-art, la musique) qui est centré sur les combats et les énigmes.

Clients/ Utilisateurs

Les clients sont nos professeurs et les utilisateurs sont nos collègues et nos professeurs.

Jalons : dates ou fonctionnalités

9 avril : rendu itération 1 : rapport individuel + code + manuel utilisateur+ rapport général

23 avril : rendu itération 2 : rapport individuel + code + manuel utilisateur + rapport général

21 mai : rendu final : rapport individuel + code + manuel utilisateur +rapport général

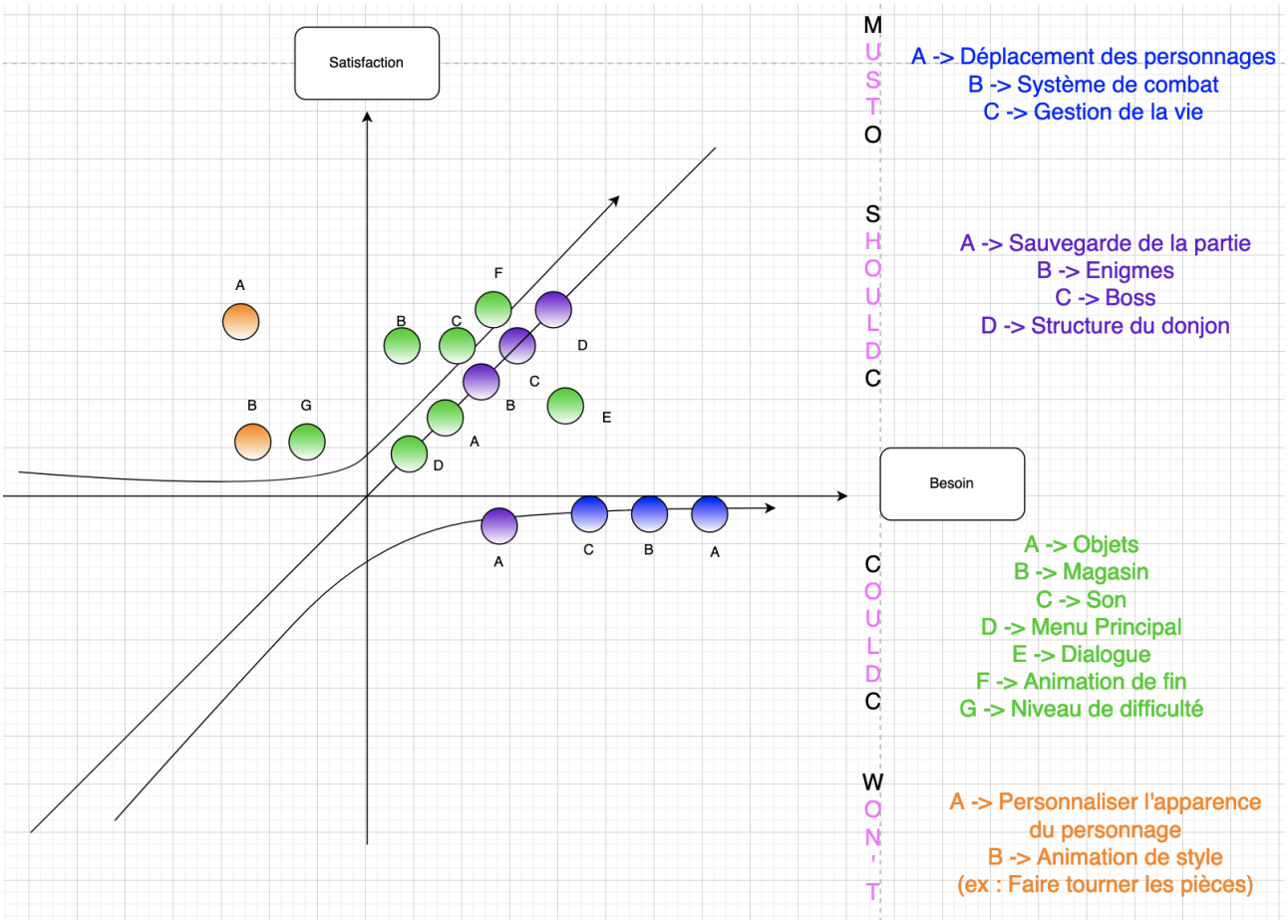
31 mai : oral

Risques :

- Organisation en équipe,
- Non réalisation de certaines fonctionnalités.

4.2 TD2

4.2.1 Graphique



4.2.2 Users Stories

1. En tant qu'utilisateur je veux pouvoir déplacer mon joueur afin de pouvoir jouer.

2. En tant qu'utilisateur je veux pouvoir avoir accès à un système de combat afin de pouvoir jouer aux jeux vidéos.

3. En tant qu'utilisateur je veux pouvoir sauvegarder la partie afin de pouvoir la reprendre à l'endroit où je me suis arrêté.

4. En tant qu'utilisateur je veux pouvoir récupérer des objets afin de pouvoir augmenter mes chances de gagner.

5. En tant qu'utilisateur je veux pouvoir avoir accès à un magasin afin de pouvoir acheter des objets qui augmenteront mes chances de gagner.

6. En tant qu'utilisateur je veux pouvoir avoir accès à un inventaire afin de pouvoir stocker mes objets.

7. En tant qu'utilisateur je veux pouvoir me déplacer d'une salle à l'autre afin de pouvoir affronter différents ennemis, le boss et sauver la princesse.

8. En tant qu'utilisateur je veux pouvoir faire des dégâts avec mes armes afin de pouvoir tuer les ennemis.

9. En tant qu'utilisateur je veux pouvoir voir ma barre de vie afin de me tenir informer sur la vie de mon personnage.

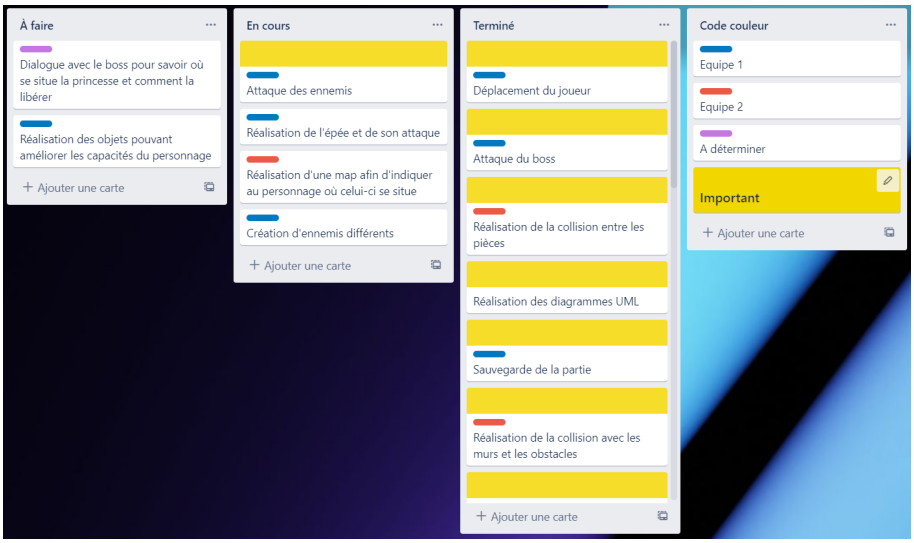
10. En tant qu'utilisateur je veux pouvoir voir afficher la carte contenant toutes les salles afin de savoir où je dois me diriger.

11. En tant qu'utilisateur je veux pouvoir choisir le niveau de difficulté du jeu afin de me challenger.

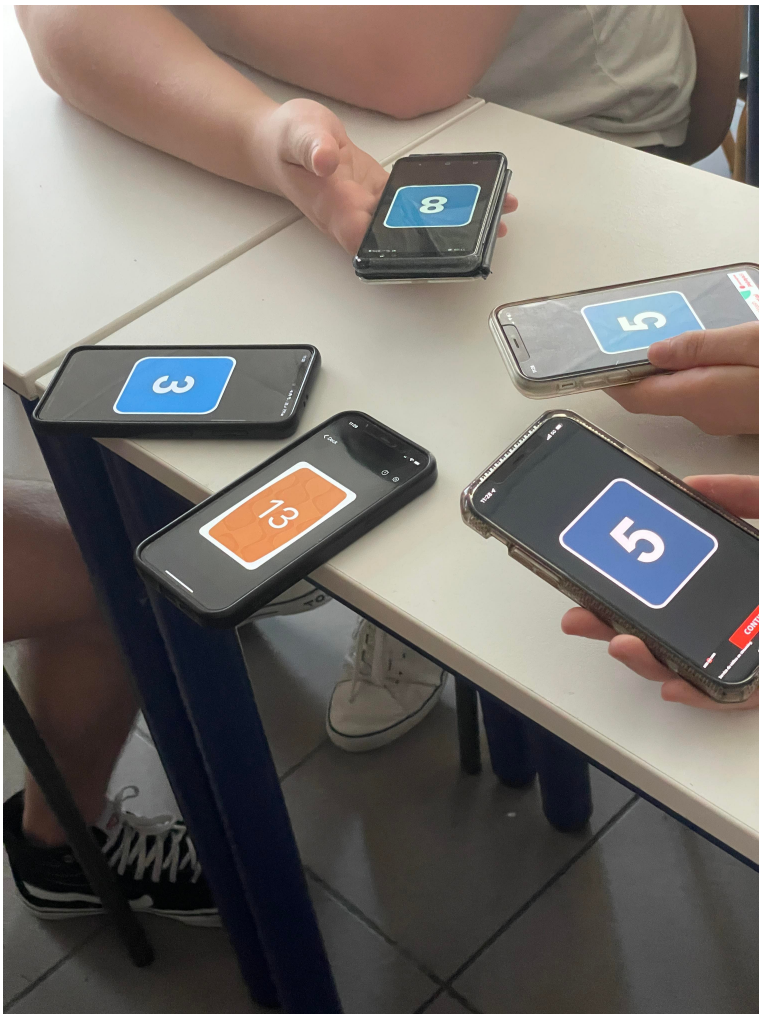
12. En tant qu'utilisateur je veux pouvoir entendre une musique en même temps que je joue afin de me divertir.

4.3 TD3

Lors du TD3 de méthodes agiles, nous nous sommes inscrits sur Trello afin de mieux répartir les taches et afin de se rendre compte plus facilement de ce qu'il nous restait à faire.
Voici le résultat :



Pour la dernière itération, à la suite de ce TD, nous avons aussi décidé de mettre en place ce que nous avions appris. Nous avons utilisé l'application Scrum Poker, et, pour chaque fonctionnalité que nous avions décidé de réaliser pour cette itération, nous avons fait un vote concernant sa difficulté.
Pour les fonctionnalités suivantes voici le résultat du vote :
- Changement de la position du joueur lorsque l'on change de salle Vote = 3
- Collision entre les différentes entités (personnage et ennemis, personnage et princesse ou druide) Vote = 5.
- Dégâts des ennemis sur le personnage (perte de vie) Vote = 3.
- Réalisation du magasin (achat d'objets) Vote = 2.
- Affichage d'un message Game Over lorsque le personnage n'a plus de vie Vote = 2.
- Affichage des dialogues entre les différentes personnages du jeu Vote = 2.
- Possibilité de récupérer des objets Vote = 3.
- Possibilité de sauvegarder la partie Vote = 8.
- Possibilité de récupérer des clés pour ouvrir la salle du Boss et de la Princesse Vote = 1.
- Possibilité de mettre en pause le jeu. Vote = 1
- Attaque du boss Vote = 5.
- Blocage des portes des salles du boss et de la princesse tant que l'utilisateur n'a pas récupéré les clés associées Vote = 1.
- Réglage de la collision entre les flèches et les portes des salles Vote = 1.
- Faire tourner les pièces et les ramasser Vote = 3.
- Ajout d'un logo personnalisé quand on lance l'application Vote = 2.
- Affichage de la map (pas finalisé) Vote = 5.



Nous avions estimé un total de 47 pour l'ensemble des fonctionnalités de l'itération 3. Nous avons fait ce même vote sur les fonctionnalités de l'itération 2 qui étaient déjà terminées pour avoir une idée de notre capacité à réaliser l'ensemble des fonctionnalités listées pour l'itération 3. Pour l'itération 2 nous avions un total de 39. Nous en avons donc conclu qu'il serait compliqué de réaliser l'ensemble des fonctionnalités choisies pour l'itération 3. Cependant, comme nous avions 2 semaines de vacances, nous avons eu plus de temps pour tout réaliser et nous avons réussi à faire à peu près tout ce que nous avions listés.

5 Les principaux choix de conception et de réalisation

Lorsque nous avons commencé à réfléchir au projet, deux paquetages principaux nous ont semblé nécessaires : la réalisation de la map et la création des entités. Nous nous sommes donc répartis par équipe autour de ces deux paquetages. Le dernier paquetage est un paquetage commun aux deux équipes.

☐ Pour le paquetage Main

Pour la collision avec les murs, nous avons créé des booléens. Ces booléens seront vrais lorsqu'il y a une collision avec certaines textures comme les murs des différentes salles. Ils seront faux lorsque le personnage peut traverser certaines textures comme par exemple le sol ou les portes.

Pour gérer la sauvegarde, nous avons créé un fichier texte qui est lui-même dans un dossier sauvegarde. Ce fichier contient une liste de données qui est connue par le lecteur de sauvegarde. Nous avons choisi de limiter le nombre de sauvegarde à trois. Le principe de cette sauvegarde est que nous lançons tout d'abord le jeu "neuf", sans les paramètres changés. Ensuite nous changeons tout les paramètres du jeu. Et enfin, nous affichons le jeu.

Nous avons aussi définit une classe GUI qui gère les graphismes visibles par l'utilisateur. C'est ici que nous affichons l'inventaire, les dialogues avec les personnages, le message Game Over et le magasin.

Pour continuer, nous avons aussi réalisé une classe NewMenuPrincipal qui se charge d'afficher à l'écran le menu permettant de lancer une partie ou de reprendre une partie sauvegardée. Pour cela, nous avons décidé d'utiliser Graphics2D et Swing de Java pour afficher des boutons avec des Jpanel. Ces boutons sont actifs grâce à ActionListener.

La réunion de toutes les classes et la gestion de la mise à jour se fait dans la classe Jeu. C'est donc dans cette classe que nous changeons les salles, que nous affichons le joueur, les ennemis, le boss, le druide ou encore la princesse.

☐ Pour le paquetage Map

Pour le paquetage Map, nous avons commencé par choisir la forme de la map et les différentes salles. Nous avons décidé de construire 4 types de salles différentes. Pour cela, nous avons utilisé le logiciel *Gimp* afin de réaliser les différentes textures. Nous avons ensuite réalisé une classe pour chaque map. Dans cette classe, nous avons créé un tableau dans lequel chaque case est composé d'une texture. Pour avoir le graphisme, nous avons utilisé un fichier texte dans lequel nous mettons une suite d'entiers. Un entier correspond à un bloc de 32x32 pixels. Il y a donc autant d'entiers que de bloc présents dans la salle. Les nombres des cases représentent les numéros de textures rentrés dans le tableau de la classe salle associée.

☐ Pour le paquetage Entités

Nous allons maintenant vous expliquer comment nous avons décidé de créer les ennemis. Les ennemis doivent avoir un comportement fonctionnel, c'est-à-dire qu'ils doivent avancer dans une direction aléatoire pendant environ une seconde. Ils doivent ensuite s'arrêter pendant environ une seconde. Lorsque le personnage les touchent, les ennemis doivent perdre de la vie. Lorsque le joueur touche plusieurs fois l'ennemi, celui-ci doit mourir. Enfin, lorsque les deux entités sont en contact, le joueur doit perdre de la vie. Pour cela, nous utilisons une variable (entier) update qui s'incrémente à chaque mise à jour et dont la valeur définie ce que va faire l'ennemi.

Pour le boss, nous avons décidé que celui-ci devait attaquer toute les 10 secondes environ. Il doit choisir son attaque aléatoirement parmi 2 choix : salve de flèches ou salve rapide de flèches.

Il faut donc qu'il fasse perdre de la vie au personnage lorsque des flèches le touche ou lorsque celui-ci rentre en contact avec le boss. Enfin,le boss doit mourir lorsqu'il a perdu tout ses points de vie.

Nous avons donc décidé de créer une classe ennemi et une classe boss pour factoriser le code et pour pouvoir créer une liste d'ennemis sur laquelle nous pouvons itérer les mises à jour/affichages.

Pour la gestion des objets, nous avons créé une classe ListeObjet qui contient tout les objets du jeu. Nous avons décidé que l'objet collectable implémente Entité. Pour traiter la collision entre l'objet et le joueur, nous avons créé la classe CollisionObjet.

Pour la création des flèches, nous avons élaboré un tableau de 25 flèches correspondant au nombre de flèches émises par la même entité qui puissent apparaître simultanément. Ensuite, nous updatons chaque flèche se trouvant dans le tableau. Dès qu'une flèche rencontre un obstacle, son emplacement dans le tableau devient null. On peut donc en accueillir une nouvelle.

5.1 Les problèmes rencontrés

□ Pour le paquetage Main

Lors de l'implantation du menu principal, un membre du groupe avait réalisé une première version du menu en swing mais un autre membre du groupe a voulu améliorer ce menu en utilisant Graphics2D. En utilisant Graphics2D nous avons rencontré des problèmes car ce paquetage est beaucoup plus compliqué à utiliser et à comprendre que Swing. En effet, l'un des premiers problèmes rencontré à été le fait que le menu ouvrait le jeu mais n'avait pas accès aux touches du clavier.
De plus, il a fallu adapter les fenêtres que nous avions réalisées en Swing en Graphics2D. Le plus gros problème a été le passage de la première fenêtre (où l'on voit les différents boutons Jouer, Sauvegarde, ...) à la deuxième fenêtre expliquant les règles du jeu. Finalement, nous avons rencontré exactement les mêmes difficultés pour le passage du boutons Sauvegarde au menu où l'on peut choisir la sauvegarde pour lancer la partie.

Pour la sauvegarde, nous avons tenté d'utiliser les classes ObjectInputStream et ObjectOutputStream. Cependant, la classe BufferedImage que nous utilisons pour changer nos textures n'est pas Serializable donc pas enregistrable. En effet, pour qu'une classe soit enregistrable, il faut que tout les attributs de la classe le soient aussi.

Nous avions aussi eu un problème lorsque nous lancions les flèches : si les flèches étaient tirées sur les portes, nous changions de salle.

Pour la collision entre le personnage et les différentes textures de la salle, nous avons eu du mal à vérifier s'il y avait collision ou non. Le joueur pouvait se bloquer alors qu'il n'y avait pas de collision.

Pour finir, nous avons rencontré un problème pour afficher la carte des salles avec la position du joueur sur celle-ci. Nous avons réalisé cette classe séparément du reste du jeu et nous avons des difficultés à l'implanter. En effet, nous avons réussi à afficher la carte des différentes salles et le point localisant le personnage mais en implantant cette solution dans le reste du jeu et en appuyant sur le bouton M pour ouvrir la map rien ne se passe.

☐ Pour le paquetage Map

Lors de la deuxième itération, nous avons rencontré quelques difficultés. En effet, au tout début, nous avions réussi à changer une fois de salle mais nous ne parvenions pas à rechanger de salle ou à revenir dans l'ancienne salle. De plus, lorsque le personnage changeait de salle, il fallait pouvoir revenir dans la salle initiale. Le problème était que lorsque le personnage apparaissait dans une nouvelle salle, on détectait une collision avec la porte et on retournait immédiatement dans la salle initiale.

De plus, lorsque le personnage changeait de salle, il fallait qu'il change de position, il ne devait pas garder la position qu'il avait en passant la porte. Nous avons eu beaucoup de mal à effectuer ce point car, si l'on ne faisait que mettre des fonctions changeant la position du personnage lorsque nous changions de salle, la fonction se répétant indéfiniment et le personnage ne pouvait plus avancer.

☐ Pour le paquetage Entités

Lorsque nous avons voulu gérer la gestion de la vie, nous avons rencontré un problème : lorsque le joueur touchait un ennemi, il perdait toute sa vie en moins d'une seconde. Il fallait donc mettre en place un temps d'invincibilité.

Pour les mouvements du joueur, nous avons aussi rencontré un problème similaire à la gestion de la vie : dès que nous voulions déplacer le joueur, celui-ci disparaissait. Le personnage avançait d'une certain nombre de pixel par frame et donc finissait très rapidement par se retrouver en dehors de la fenêtre.

Pour la gestion des coeurs, nous avons commencé par faire une mauvaise implantation : nous avions 4 coeurs fixes et indépendants les uns des autres. Nous n'avions donc que la possibilité de perdre un demi coeur.

Pour la création des ennemis, nous avons eu des difficultés à invoquer des flèches depuis celui-ci. Nous avons aussi eu du mal à faire la différence entre les flèches tirées par l'ennemi et les flèches lancées par le joueur. En effet, au début, le boss se prenait ses propres flèches et mourrait. Les autres ennemis pouvaient aussi être touchés par le boss.

Nous avons aussi eu du mal à gérer la collision entre ennemis et murs, entre joueur et ennemis, et flèches et ennemis. Nous devions rajouter dans chaque entité une Hitbox qui alourdissait énormément les attributs (il y avait 4 points donc 8 coordonnées). C'était aussi compliqué de régler précisément la Hitbox de chaque entité sur la map.

☐ Les problèmes générales

Nous avons eu beaucoup de difficultés à assembler les parties de chacun. En effet, tout le monde touchait aux mêmes classes et c'était donc compliqué de s'y retrouver.

Notre logiciel est vraiment lent et bug beaucoup, sûrement du au fait que nous avons importé beaucoup de textures différentes.

5.2 Les solutions apportées

□ Pour le paquetage Main

Pour le menu principal, la solution apportée pour passer de la fenêtre du menu à la fenêtre de la sauvegarde a été de créer une nouvelle classe MenuSauvegarde que nous appelons directement dans la classe NewMenuPrincipal. Ensuite, pour pouvoir passer de la fenêtre du menu principal à la fenêtre qui explique les règles du jeu, il a fallu dans la méthode qui active le bouton Jouer implanter une nouvelle fenêtre contenant l'image souhaitée. Ensuite, pour régler le problème lié à l'accès du clavier, il a fallu utiliser la méthode grabFocus() pour pouvoir être focusable par l'utilisateur.
Pour la sauvegarde, nous avons décidé d'écrire les informations à sauvegarder dans un fichier texte. Lors de la recharge du jeu, nous lisons les différentes informations dans ce fichier.

Pour éviter que le joueur se bloque à n'importe quel endroit, nous avons créer les variables coin1 et coin2 qui prennent en comptent la position du personnage et qui rajoutent un certain entier pour pouvoir obtenir la position des extrémités du joueur.

Pour éviter que nous changions de salle lorsque nous lancions des flèches, nous avons testé dans la classe Collision si, en plus de la collision, l'entité entraînant cette collision était bien le joueur.

Nous n'avons pas encore réussi à trouver la solution pour afficher la map en entière et pour localiser le personnage.

□ Pour le paquetage Map

Pour le changement de salles, nous avons créé différents booléens dans les différentes classes Salles. Il y a autant de booléens que de changement de salles. Ensuite, nous vérifions si le booléen est vrai dans la classe collision. Si le booléen est vrai, nous avons créé deux autres booléens. Le premier booléen permet le changement de salle et le second booléen permet le changement de position du personnage.

□ Pour le paquetage Entités

Pour la gestion de la vie et du temps d'invincibilité, nous avons mis en place un compteur de frame qui permet une perte de vie à partir d'une certaine valeur.

Pour le mouvement du joueur, nous avons fait la même chose que pour la gestion de la vie. C'est-à-dire que nous avons placé un compteur de frame qui permet un mouvement à partir d'une certaine valeur.

Pour la gestion des coeurs, nous avons effectué un changement total d'implémentation : nous avons lié les coeurs les uns avec les autres. Pour savoir combien de coeur afficher, nous utilisons un double qui change de valeur à chaque dégât.

Pour la gestion des flèches, nous avons du faire la différence entre flèches ennemis et flèches alliées. Pour cela, nous avons utilisé lors de la création d'une flèche un booléen qui permet de vérifier lors de la collision qui a envoyé la flèche.

Pour la création de la Hitbox, nous avons conçu une méthode permettant d'afficher celle-ci avec des traits rouges. nous avons aussi fait que chaque personnage possède un constructeur.

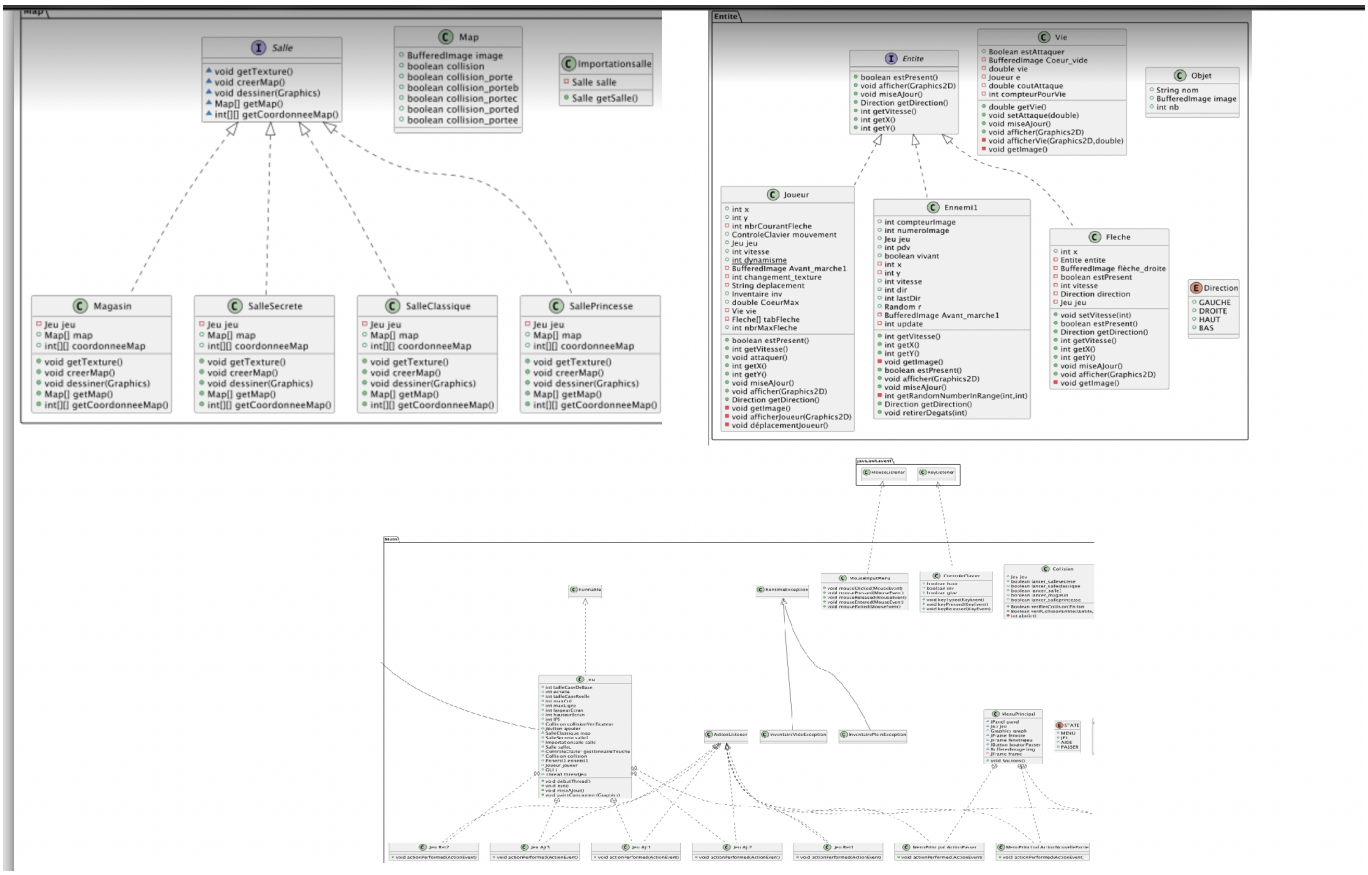
☐ Les solutions générales

Pour l'assemblage du jeu, nous avons décidé que chacun envoie les classes qu'il avait modifié en rajoutant des commentaires pour que nous ne soyons plus obligé de décortiquer le code ligne par ligne.

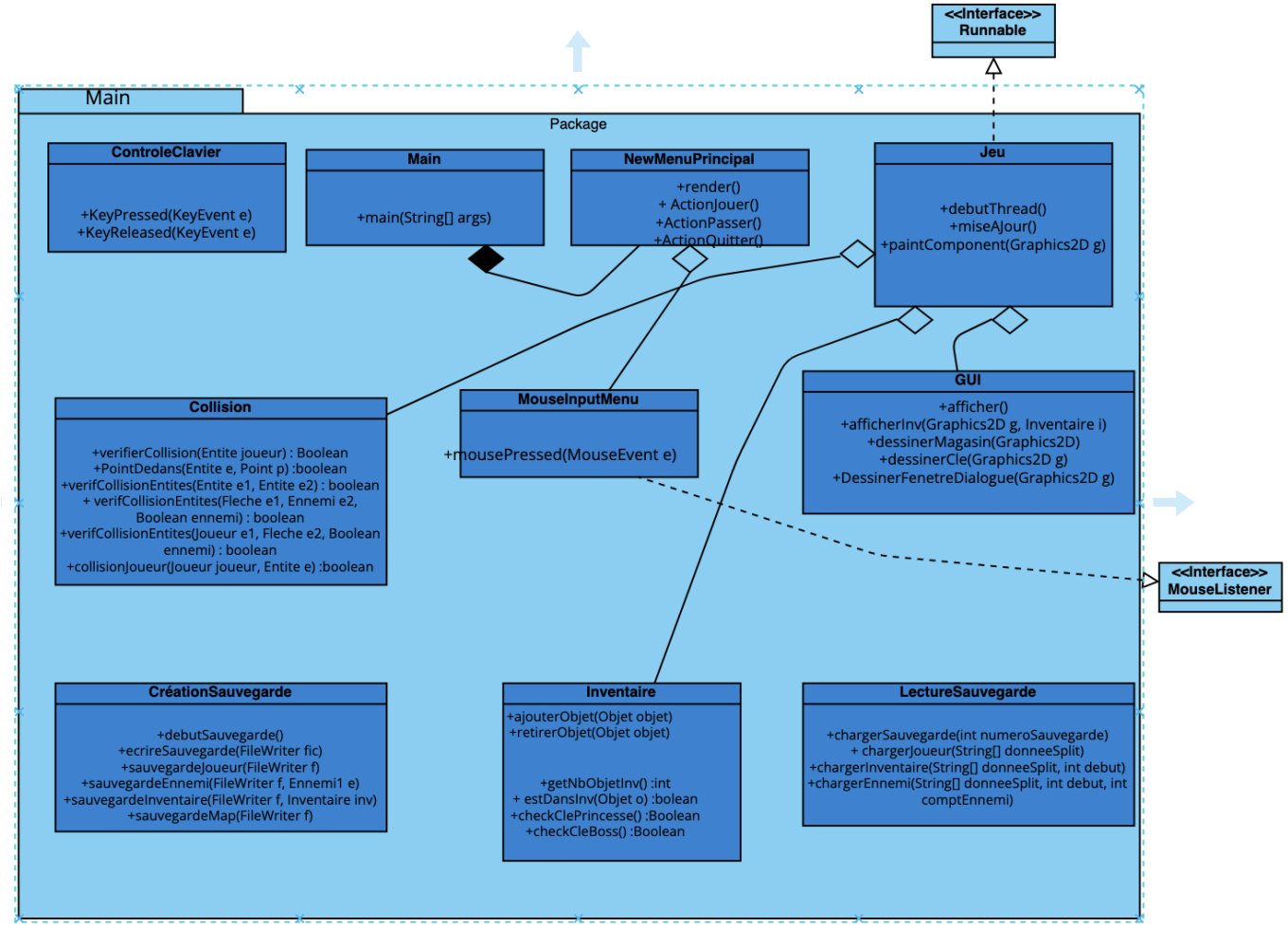
Pour le fait que le jeu bug beaucoup, cela c'est amélioré. Nous ne saurons pas trop dire pourquoi...

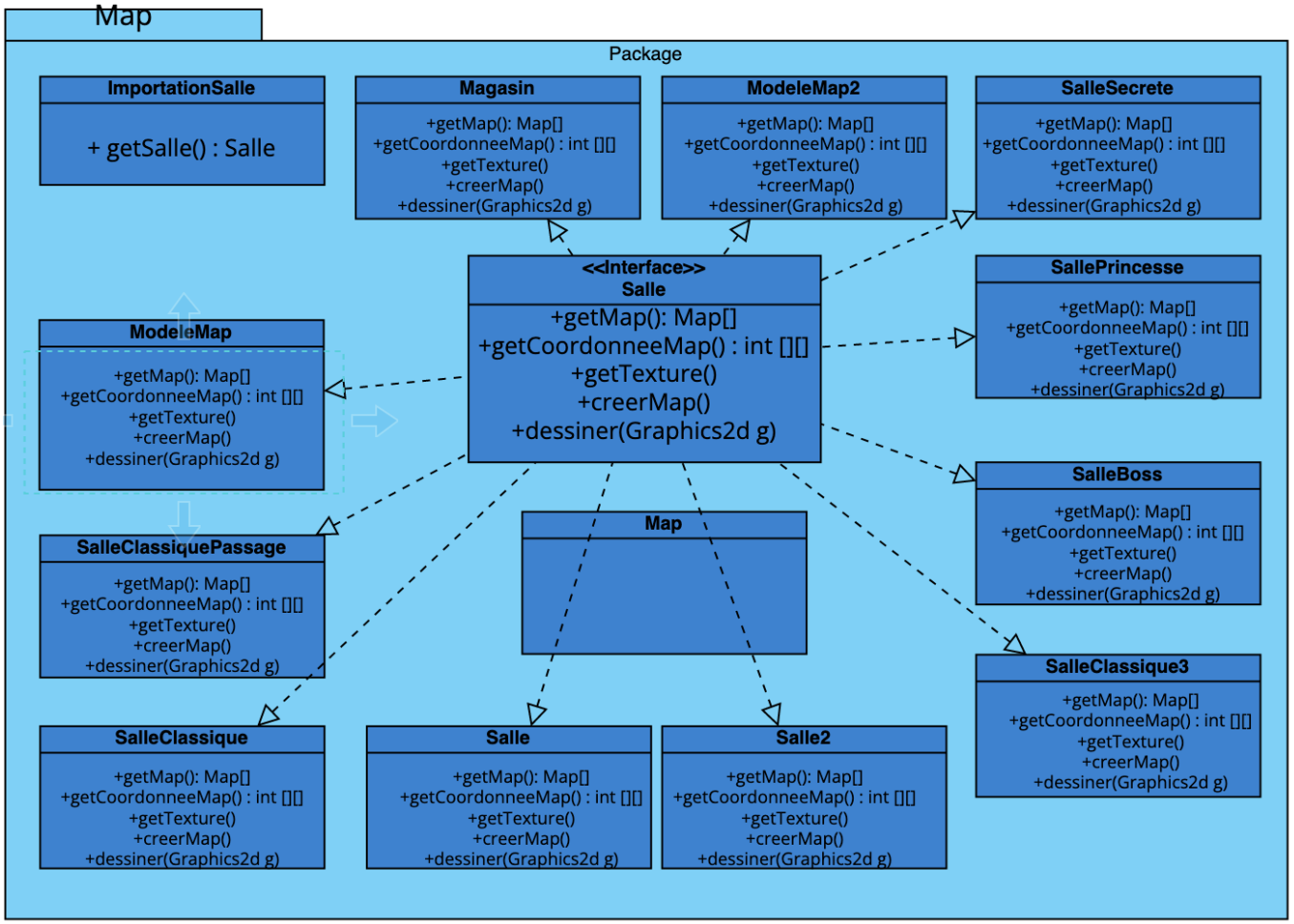
6 Diagrammes UML

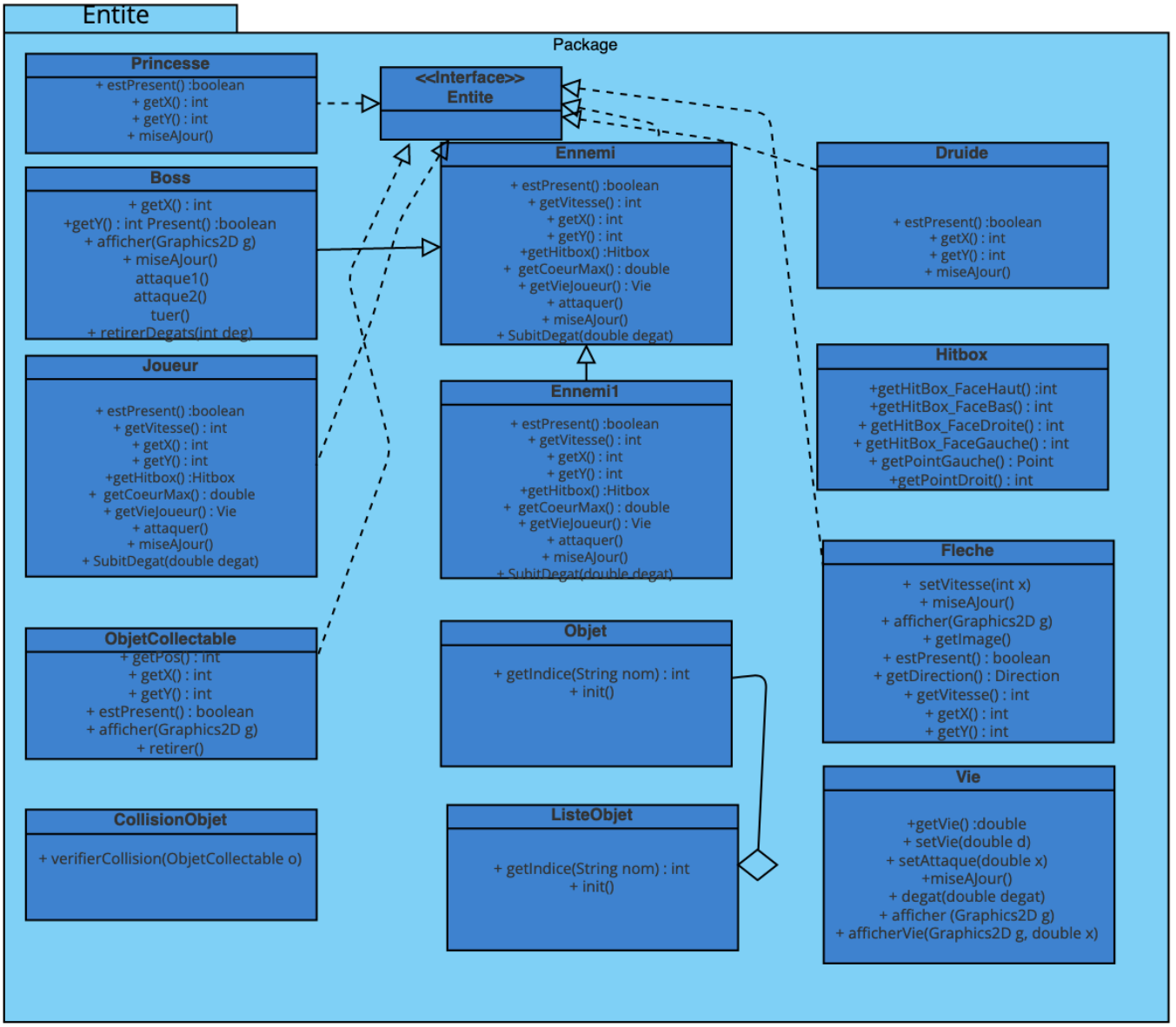
6.1 Diagramme de classe



Par paquetage :

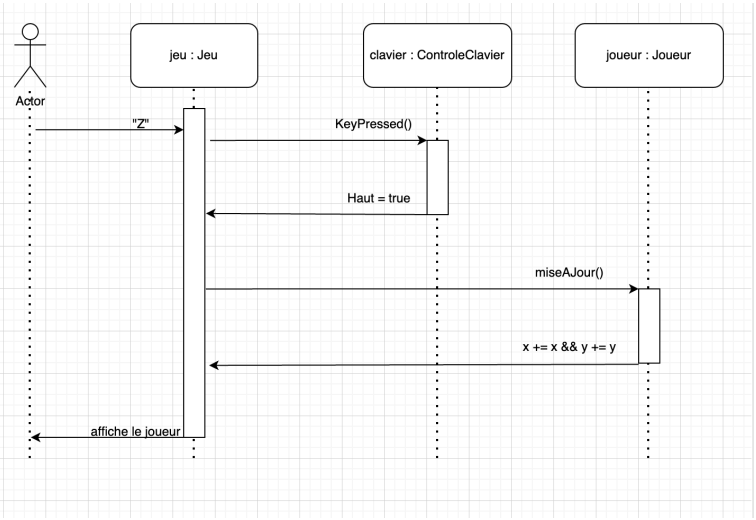

<<Interface>>
Runnable
Main
Package
ControleClavier
+KeyPressed(KeyEvent e)
+KeyReleased(KeyEvent e)
Main
+main(String[] args)
NewMenuPrincipal
+render()
+ ActionJouer()
+ActionPasser()
+ActionQuitter()
Jeu
+debutThread()
+miseAJour()
+paintComponent(Graphics2D g)
Collision
+verifierCollision(Entite joueur) : Boolean
+PointDedans(Entite e, Point p) :boolean
+verifCollisionEntites(Entite e1, Entite e2) : boolean
+ verifCollisionEntites(Fleche e1, Ennemi e2, Boolean ennemi) : boolean
+verifCollisionEntites(Joueur e1, Fleche e2, Boolean ennemi) : boolean
+collisionJoueur(Joueur joueur, Entite e) :boolean
MouseInputMenu
+mousePressed(MouseEvent e)
GUI
+afficher()
+afficherInv(Graphics2D g, Inventaire i)
+dessinerMagasin(Graphics2D)
+dessinerCle(Graphics2D g)
+DessinerFenetreDialogue(Graphics2D g)
<<Interface>>
MouseListener
CréationSauvegarde
+debutSauvegarde()
+ecrireSauvegarde(FileWriter fic)
+sauvegardeJoueur(FileWriter f)
+sauvegardeEnnemi(FileWriter f, Ennemi1 e)
+sauvegardeInventaire(FileWriter f, Inventaire inv)
+sauvegardeMap(FileWriter f)
Inventaire
+ajouterObjet(Objet objet)
+retirerObjet(Objet objet)
+getNbObjetInv() :int
+ estDansInv(Objet o) :bolean
+checkClePrincesse() :Boolean
+checkCleBoss() :Boolean
LectureSauvegarde
+chargerSauvegarde(int numeroSauvegarde)
+ chargerJoueur(String[] donneeSplit)
+chargerInventaire(String[] donneeSplit, int debut)
+chargerEnnemi(String[] donneeSplit, int debut, int comptEnnemi)


Map
Package
ImportationSalle
+ getSalle() : Salle
Magasin
+getMap(): Map[]
+getCoordonneeMap() : int [][]
+getTexture()
+creerMap()
+dessiner(Graphics2d g)
ModeleMap2
+getMap(): Map[]
+getCoordonneeMap() : int [][]
+getTexture()
+creerMap()
+dessiner(Graphics2d g)
SalleSecrete
+getMap(): Map[]
+getCoordonneeMap() : int [][]
+getTexture()
+creerMap()
+dessiner(Graphics2d g)
<<Interface>>
Salle
+getMap(): Map[]
+getCoordonneeMap() : int [][]
+getTexture()
+creerMap()
+dessiner(Graphics2d g)
SallePrincesse
+getMap(): Map[]
+getCoordonneeMap() : int [][]
+getTexture()
+creerMap()
+dessiner(Graphics2d g)
ModeleMap
+getMap(): Map[]
+getCoordonneeMap() : int [][]
+getTexture()
+creerMap()
+dessiner(Graphics2d g)
SalleBoss
+getMap(): Map[]
+getCoordonneeMap() : int [][]
+getTexture()
+creerMap()
+dessiner(Graphics2d g)
Map
SalleClassiquePassage
+getMap(): Map[]
+getCoordonneeMap() : int [][]
+getTexture()
+creerMap()
+dessiner(Graphics2d g)
SalleClassique3
+getMap(): Map[]
+getCoordonneeMap() : int [][]
+getTexture()
+creerMap()
+dessiner(Graphics2d g)
SalleClassique
+getMap(): Map[]
+getCoordonneeMap() : int [][]
+getTexture()
+creerMap()
+dessiner(Graphics2d g)
Salle
+getMap(): Map[]
+getCoordonneeMap() : int [][]
+getTexture()
+creerMap()
+dessiner(Graphics2d g)
Salle2
+getMap(): Map[]
+getCoordonneeMap() : int [][]
+getTexture()
+creerMap()
+dessiner(Graphics2d g)

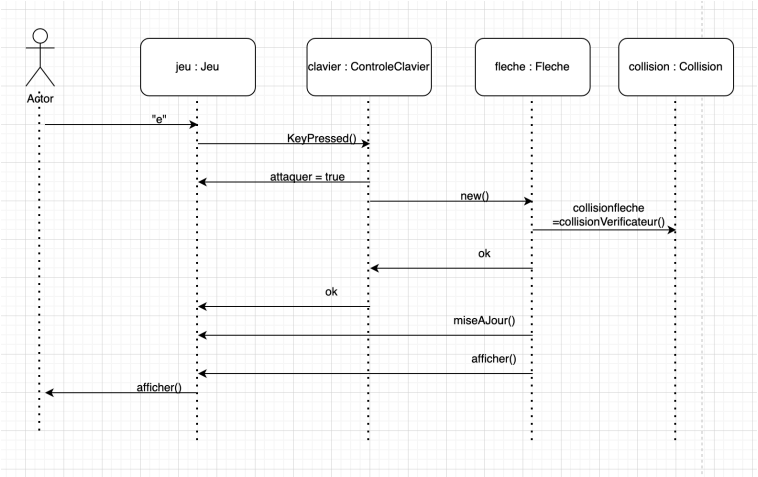

Entite
Package
Princesse
+ estPresent() :boolean
+ getX() : int
+ getY() : int
+ miseAJour()
<<Interface>>
Entite
Boss
+ getX() : int
+getY() : int Present() :boolean
+ afficher(Graphics2D g)
+ miseAJour()
attaque1()
attaque2()
tuer()
+ retirerDegats(int deg)
Ennemi
+ estPresent() :boolean
+ getVitesse() : int
+ getX() : int
+ getY() : int
+getHitbox() :Hitbox
+ getCoeurMax() : double
+ getVieJoueur() : Vie
+ attaquer()
+ miseAJour()
+ SubitDegat(double degat)
Druide
+ estPresent() :boolean
+ getX() : int
+ getY() : int
+ miseAJour()
Joueur
+ estPresent() :boolean
+ getVitesse() : int
+ getX() : int
+ getY() : int
+getHitbox() :Hitbox
+ getCoeurMax() : double
+ getVieJoueur() : Vie
+ attaquer()
+ miseAJour()
+ SubitDegat(double degat)
Ennemi1
+ estPresent() :boolean
+ getVitesse() : int
+ getX() : int
+ getY() : int
+getHitbox() :Hitbox
+ getCoeurMax() : double
+ getVieJoueur() : Vie
+ attaquer()
+ miseAJour()
+ SubitDegat(double degat)
Hitbox
+getHitBox_FaceHaut() :int
+getHitBox_FaceBas() : int
+ getHitBox_FaceDroite() : int
+ getHitBox_FaceGauche() : int
+ getPointGauche() : Point
+getPointDroit() : int
Fleche
+ setVitesse(int x)
+ miseAJour()
+ afficher(Graphics2D g)
+ getImage()
+ estPresent() : boolean
+ getDirection() : Direction
+ getVitesse() : int
+ getX() : int
+ getY() : int
ObjetCollectable
+ getPos() : int
+ getX() : int
+ getY() : int
+ estPresent() : boolean
+ afficher(Graphics2D g)
+ retirer()
Objet
+ getIndice(String nom) : int
+ init()
Vie
+getVie() :double
+ setVie(double d)
+ setAttaque(double x)
+miseAJour()
+ degat(double degat)
+ afficher (Graphics2D g)
+ afficherVie(Graphics2D g, double x)
CollisionObjet
+ verifierCollision(ObjetCollectable o)
ListeObjet
+ getIndice(String nom) : int
+ init()

6.2 Diagramme de séquence

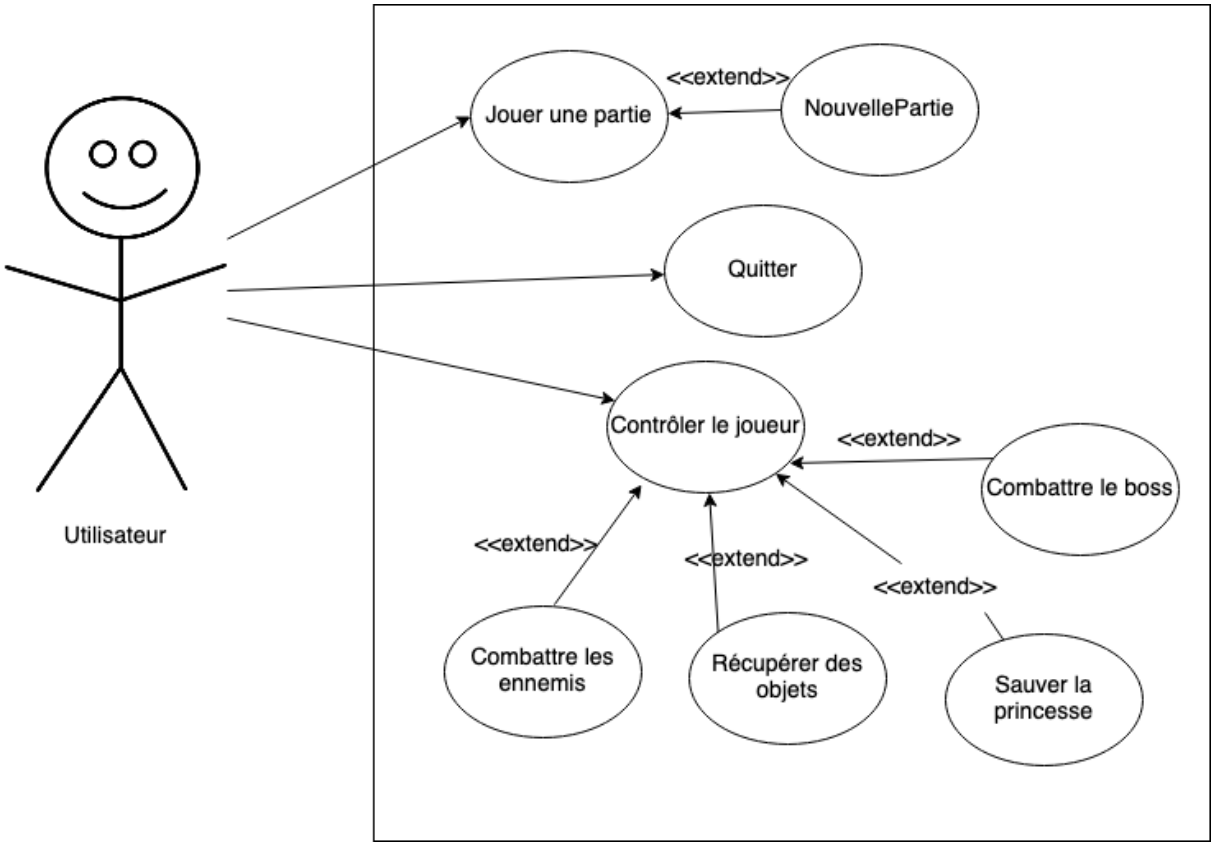
Voici un premier diagramme de séquence associé à l'enfoncement d'une touche du clavier par l'utilisateur.



Maintenant, voici un autre diagramme de séquence qui représente le lancement d'une flèche par l'utilisateur via la touche E du clavier.



6.3 Diagramme de cas d'utilisation



7 Annexe

Par rapport à l'itération 3 nous avons remarqué que lors de notre dépôt nous avions des accents qui empêchaient la bonne exécution du programme, nous avons donc modifié ceci. Nous avons fait de même pour la sauvegarde.
De plus, nous avons rajouté une animation lorsque nous perdons une vie.
Finalement, nous avons aussi rajouté l'accès au bouton Contrôle. Si jamais l'utilisateur cherche de l'aide pour trouver la salle secrète, un indice se trouve sur ce bouton.