



Projet de Java - du 16/05/20 au 31/05/20

CLASSE CYCLE D'INGENIEUR ING1G04

A L'INTENTION DE :

Julien Mercadal et Damien Faugere-Battiato

réalisé par :

Lilian Naretto, Hassan Yazane, Ilham Laatarsi, Sabrina Smail et Corentin
Brillant

Table des matières

I	Introduction	4
II	Le Jeu et ses règles	4
III	Le manuel d'utilisation	5
1	Les différents écrans	5
1.1	L'écran d'accueil	5
1.2	L'écran de jeu	8
2	Le contrôle du personnage en jeu	12
IV	Cahier des charges	13
3	Fiche produit	14
4	Description générale du projet	14
5	Principe du jeu	14
6	Fonctions principales	15
7	Fonctions optionnelles	15
8	Langages et framework	15
8.1	Gestion de versions	15
8.2	Langages	16
8.3	Bibliothèques graphiques	16
8.4	plateforme de développement	16
9	Tâches à faire	16
V	Architecture	17
10	Préambule	17
11	Diagrammes structurels	17
11.1	Description du modèle de données	17
11.2	Description du modèle de contrôle	22
11.2.1	La classe ControleCarte	22
11.2.2	La classe ControleEntite	22
11.3	Description du modèle de présentation	22

12 Diagrammes de comportement	23
12.1 Diagrammes de séquence	23
12.2 Diagramme de cas d'utilisation	26
13 Structure de l'IHM	27
13.1 Description du graphe de scène de l'interface du jeu	28
13.2 Description du graphe de scène du Menu	29
VI Le code : Arborescence des fichiers et choix d'implémentation	30
14 Arborescence des packages	30
15 Logique des packages	31
15.1 Le package Jeu	31
15.2 Le package threadService	31
15.3 Le package contrôle	32
15.4 Le package saveAndLoad	32
15.5 Le package com	32
15.6 Le package affichage	32
16 L'illusion du mouvement : Techniques d'affichage et Algorithmique	32
16.1 Mouvements du héros	32
16.2 Défilement de la carte	32
16.3 Attaque automatique des ennemis	34
VII L'équipe : rôles et organisation	34
17 Méthode de fonctionnement au sein du groupe	34
18 Les outils de communication	36
19 Répartition des tâches	37
VIII Configuration et Ajouts personnalisés	37
20 Ajouter une carte	37
21 Ajouter un personnage	38
IX Conclusion	38
22 Bilan	38
23 Difficultés rencontrées	38

24 Améliorations possibles	39
X Abstract	39
XI Annexes	39

Première partie

Introduction

La réalisation de ce jeu s'inscrit dans le cadre du projet Génie logiciel 2 de fin d'année des étudiants de l'EISTI en 1ère année de cycle ingénieur filière Génie Informatique. Elle consiste à développer un jeu vidéo en utilisant les notions du cours de Java et JavaFX. Nous avons choisi de créer l'un des types de jeux les plus appréciés ; un RPG permettant au personnage d'évoluer en se rapprochant de son objectif. La consigne de base était de reprendre le sujet du parti de Java de l'année 2020 et d'en faire un jeu plus poussé en laissant libre cours à notre imagination, faisant de nous, notre propre client. Le sujet est rappelé à la fin du rapport (partie XI). La consigne a été divulguée de manière officielle de 16/05/2020 à 11h. Le rendu du projet a été fixé au 31/05/2020. Le projet dans son ensemble aura donc duré 16 jours.

Deuxième partie

Le Jeu et ses règles

1. Le jeu sera un rpg qui se déroule dans un grand donjon.
2. La carte du jeu est générée lorsque le joueur lance la partie.
3. Elle est segmentée en plusieurs parties connexes, le joueur découvre la carte au fur et à mesure qu'il ouvre des portes. La carte est construite de telle sorte que lorsqu'on ouvre une porte le niveau des adversaires rencontrés dans la nouvelle partie de la map est plus élevé.
4. L'objectif d'une partie est de détruire successivement des "boss" jusqu'au dernier, le "boss" final.
5. **les actions possibles pour le joueur :** Lors de la partie, le joueur est libre de se déplacer à tout moment si son état le lui permet. Il peut également effectuer les actions suivantes :
 - ramasser un objet à ses pieds
 - jeter un objets
 - attaquer un ennemi rencontré avec une arme qu'il a en main
 - utiliser une potion
 - changer les objets qu'il tient dans sa main
6. Le joueur, comme les monstres et les armes, a un niveau qui peut évoluer avec l'expérience qu'il accumule. Le niveau détermine la vie du joueur. De plus, le joueur ne pourra utiliser une arme que s'il a le niveau suffisant.
7. Les armes ont toutes un niveau lorsqu'on les ramasse et une durabilité. Le joueur peut améliorer ses armes (augmenter leur niveau) en utilisant des ressources.
8. **stockage des items et ressources :** Le joueur est muni de deux inventaires. Un pour stocker les ressources (Fer, Clé, ...) qu'il ramasse et un pour stocker les items (armes,

- potions, ...) qu'il trouve. Attention : Toutefois le nombre d'items est limité. Pour chaque catégorie d'item, le joueur ne pourra en posséder qu'un certain nombre.
- 9. Les ennemis varient en fonction de leur catégorie(boss, normale) et de leur race(Orc,...).
 - 10. Pour attaquer, le joueur peut utiliser ses mains, ses armes, empoisonner ou assommer l'ennemi.
 - 11. Lorsque le joueur gagne un combat contre un ennemi, il peut récupérer ses ressources, armes et items.
 - 12. Chaque "sous boss" correspond à un niveau. Pour en affronter un, il faut avoir récupérer les clefs des sous boss des niveaux inférieurs. Pour affronter le boss final il faut donc posséder toutes les clefs du jeu.
 - 13. Le jeu se finit lorsque le joueur meurt ou qu'il tue le boss final.
 - 14. La carte est constituée d'emplacements spéciaux tels que les enclumes. Ces emplacements ont des effets sur le joueur(render des actions accessibles au joueur, modifier l'état du joueur,etc). Ils sont pour certains, visible par le joueur (emplacement avec de la lave), et pour d'autres, non.
 - 15. Le joueur peut améliorer une arme uniquement lorsqu'il se situe à proximité d'une case enclume et qu'il a les ressources nécessaires à l'amélioration.
 - 16. Le personnage, comme les ennemis, peut rater ses attaques. La probabilité de réussir une attaque dépend de la différence de niveau entre les deux adversaires.
 - 17. Lorsque le joueur réussit son attaque. Il y a toujours une probabilité fixe pour que l'ennemi soit étourdi.
 - 18. optionnel : sauvegarder une partie, barre de faim, mettre en pause.
 - 19. **ramasser des items et ressources** : Lors du déroulement de la partie, le joueur est amené à trouver des objets au sol. Pour les ramasser, il lui suffit de marcher sur la case. Si son sac est plein, les objets restent au sol.
 - 20. **jeter des items** : Le joueur possède un espace de stockage limité pour chaque type d'items du jeu. Il peut donc être amené à se séparer d'un item qu'il juge trop usagé par un nouveau qu'il trouve sur le sol. Pour ce faire, il peut donc jeter son item. Le jet s'opère de la manière suivante. Le joueur tente de jeter l'objet devant lui. Si cela est impossible à cause de la présence d'un obstacle, il essaie successivement de jeter l'objet sur chaque position qui décrit un carré autour de lui jusqu'à ce qu'il y ait une position disponible pour l'item.

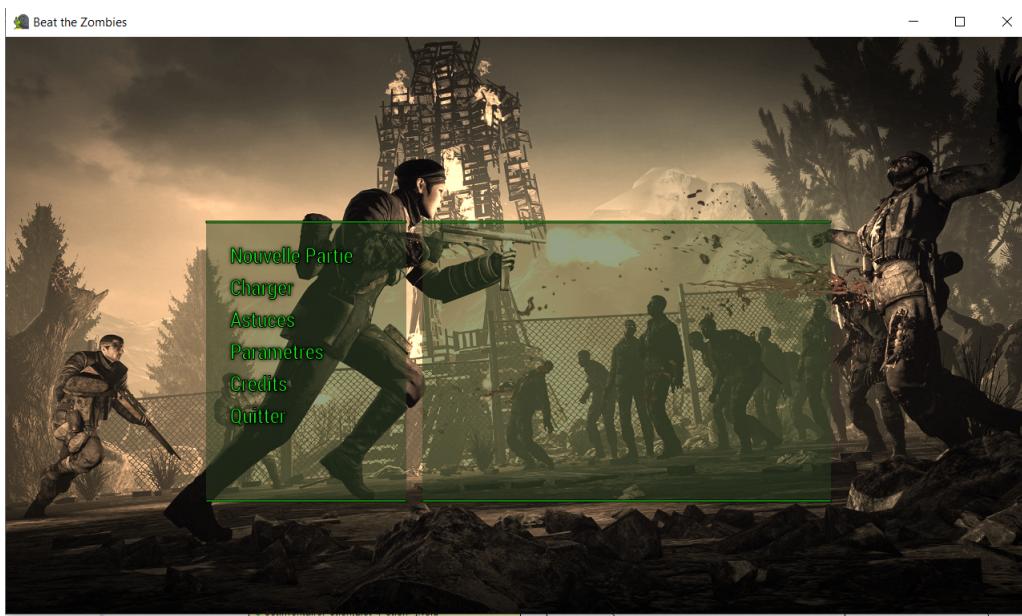
Troisième partie

Le manuel d'utilisation

1 Les différents écrans

1.1 L'écran d'accueil

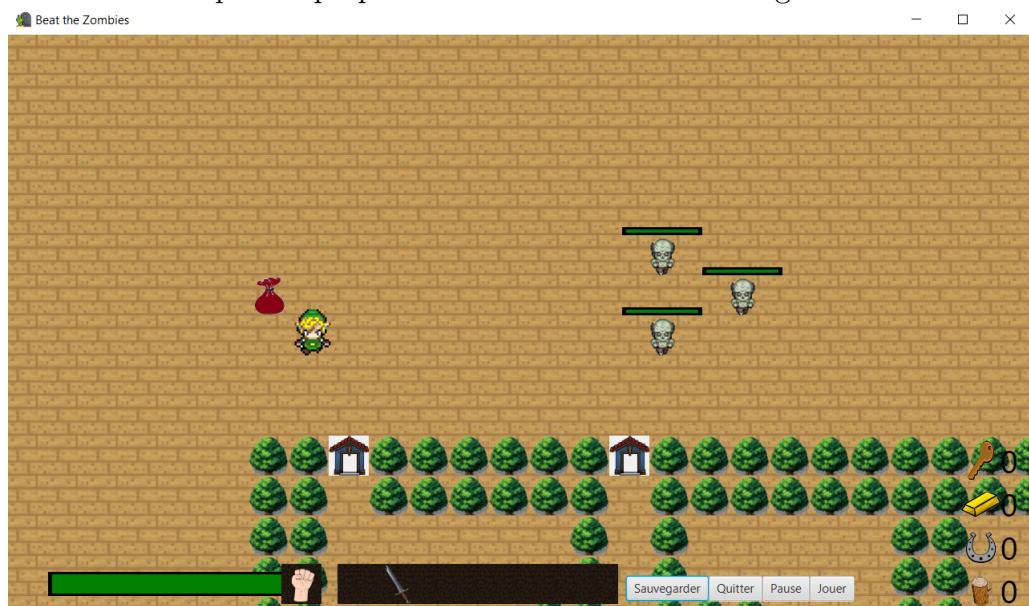
A l'exécution du dossier compressé de l'application se présente l'affichage suivant :



Le menu principal propose donc les différentes options :

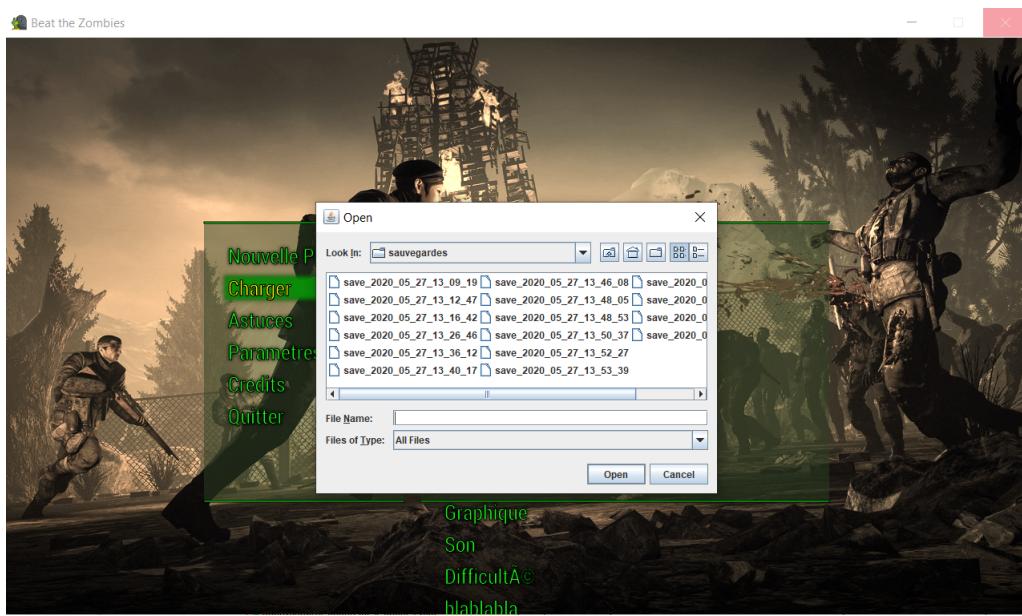
1. Nouvelle Partie

Cette option permet de charger une nouvelle partie et affichera directement à l'écran d'une nouvelle partie qui pourra ressembler à cet affichage :



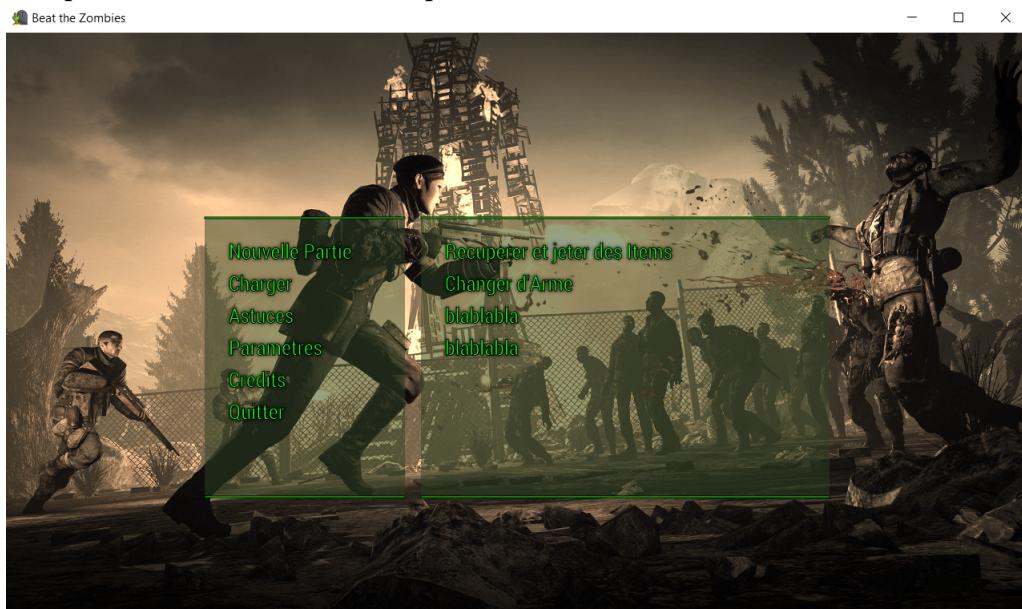
2. Charger

L'option charger permet de charger une partie déjà commencée et sauvegardée dans le dossier intitulé sauvegarde.



3. Astuces

L'option astuces donne quelques informations sur le contenu du jeu comme les ennemis potentiels ou le but de la quête.



4. Crédits

L'option Credits donne des informations sur le développement du jeu

5. Quitter

L'option quitter ferme simplement l'application

1.2 L'écran de jeu

Le joueur peut interagir avec l'écran du jeu via différents éléments cliquables qui sont les suivants :

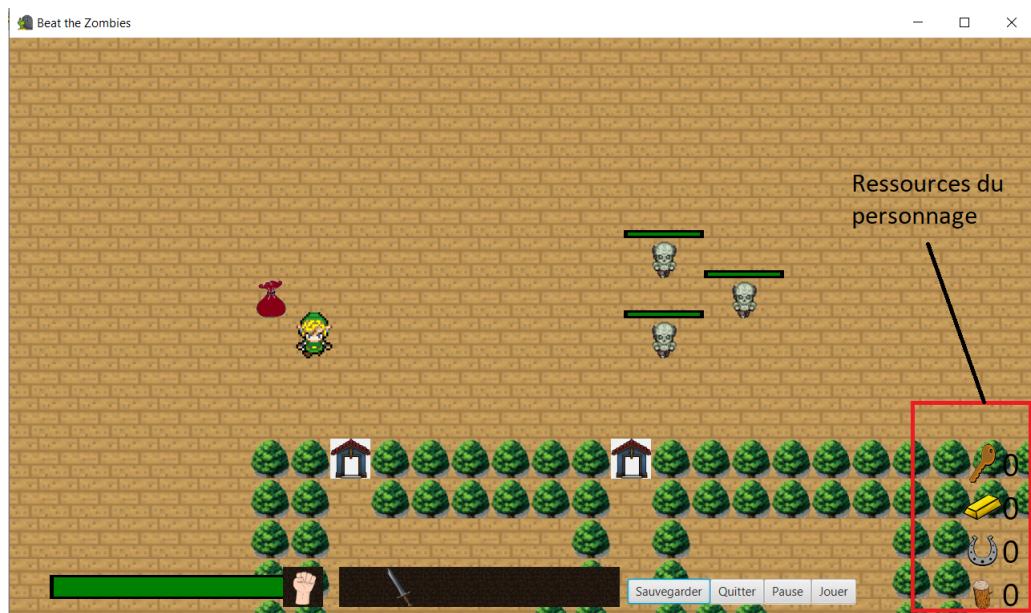
1. Le contrôle de l'inventaire des items du personnage

Le joueur dispose d'une barre cliquable centrée en bas de son écran. Cette barre est constituée de plusieurs cases avec, dans chacune d'elle, un item présent dans le sac du personnage. La case isolée et la plus à gauche est celle qui correspond à l'item que le personnage tient actuellement dans sa main et peut utiliser à l'aide de la touche SPACE. Pour enlever un item de sa main et le ranger dans le sac. Il suffit de cliquer sur la case la plus à gauche, ce qui aura pour effet d'afficher la main du personnage pour signifier qu'il a les mains vides. A l'inverse, pour prendre en main un item du sac il suffit de cliquer sur l'item souhaité dans la barre.



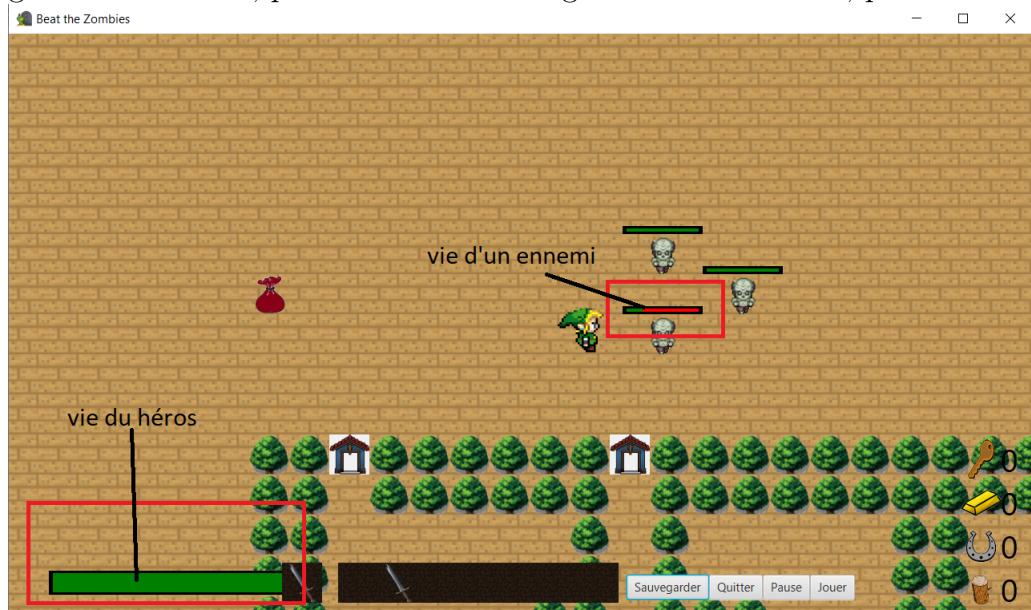
2. Le contrôle des ressources du personnage

L'utilisateur connaît à tout moment les ressources que possèdent le personnage grâce à un affichage en bas à droite de l'écran.



3. Le contrôle de la vie du personnage et des ennemis

L'utilisateur connaît également la vie qui lui reste ainsi que celle de ses ennemis grâce aux barres verte et rouge. Celle du personnage est située en bas à gauche de l'écran et celles des ennemis sont situées chacune au-dessus de leur tête. Plus une entité est gravement blessée, plus la barre sera rouge. Moins elle le sera, plus la barre sera verte.



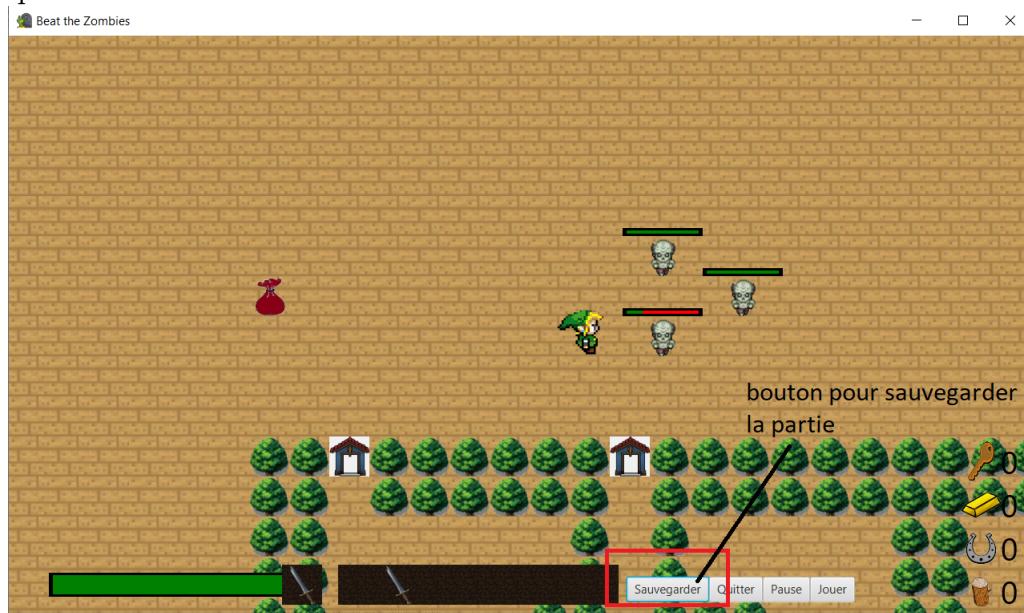
4. Le contrôle du niveau du personnage et de son expérience

Le niveau du joueur ainsi que sa barre d'expérience sont affichées en haut à droite de l'écran.



5. Quitter le jeu

L'utilisateur peut à tout moment quitter en fermant l'application à l'aide du bouton quitter en bas de son écran.



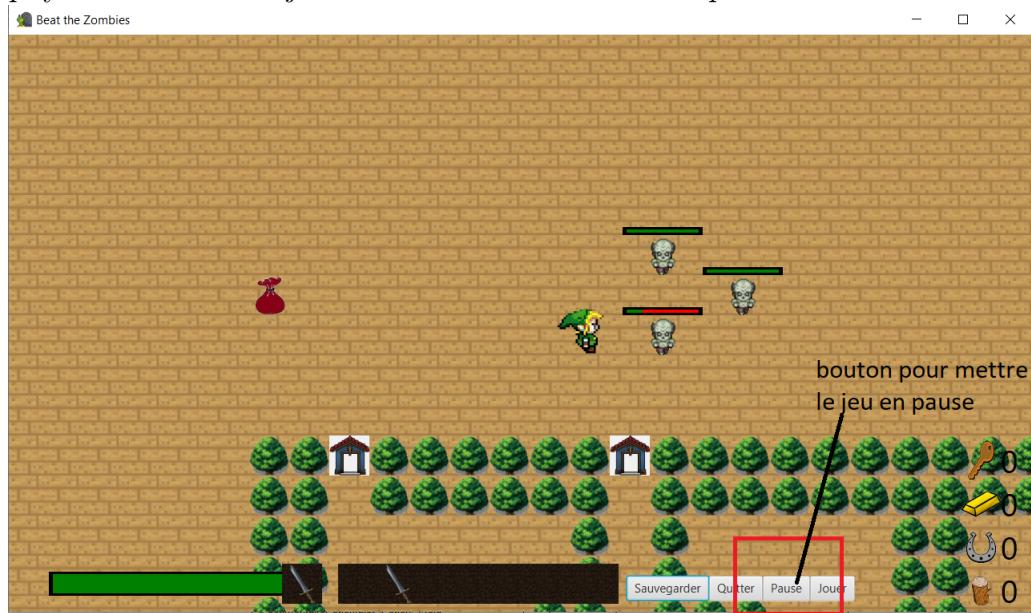
6. Sauvegarder une partie

L'utilisateur peut à tout moment sauvegarder sa partie en cliquant sur le bouton Sauvegarder. Cela enregistrera sa partie sous forme d'un fichier intitulé sauvergarde (date du jour - heure de sauvegarde) et rangé dans le répertoire "sauvergardes".



7. Mettre une partie en pause

L'utilisateur peut à tout moment mettre en pause sa partie. Pour cela il lui suffit de cliquer sur le bouton en bas à droite de l'écran. Pour relancer la partie, il suffira d'appuyer sur le bouton jouer situé à droite du bouton pause.





2 Le contrôle du personnage en jeu

Pendant le déroulement d'une partie, le personnage a la possibilité d'effectuer différentes actions :

1. Se déplacer

Pour se déplacer, le joueur utilise 4 directions :

- (a) - monter : touche Z
- (b) - descendre : touche S
- (c) - aller à gauche : touche Q
- (d) - aller à droite : touche D
- (e) - aller plus vite : touche SHIFT

Le joueur a la possibilité de combiner plusieurs touches afin de se déplacer en diagonale.

2. Attaquer

Pour attaquer l'utilisateur doit presser la touche SPACE. Attention : l'attaque n'aura d'effet que si le joueur attaque dans la direction d'un ennemi et que cet ennemi est à sa portée.

3. Ouvrir une porte

Lorsque le joueur passe à proximité d'une porte, un bouton s'affiche. En cliquant sur ce bouton la porte s'ouvrira, à condition que le personnage possède suffisamment de clés

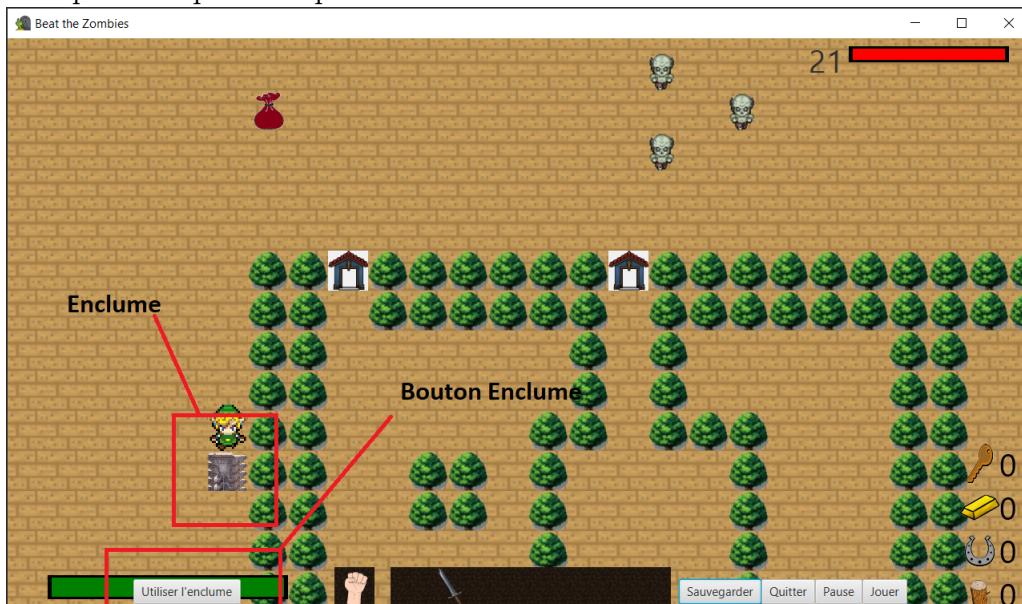
dans ses ressources.



4. Utiliser une enclume

Il existe deux types d'enclume, les enclumes pour réparer vos armes (enclume noire) et les enclumes pour améliorer vos armes (enclume blanche). Dans les deux cas, utiliser une enclume coûte des ressources. Le prix dépend du type d'arme. L'arme qui est améliorée est celle que le joueur a en main.

Pour ce faire, il suffit de cliquer sur le bouton "Utiliser l'enclume" qui s'active à chaque fois que vous passez à proximité d'une enclume.



Quatrième partie

Cahier des charges

3 Fiche produit

Genre : RPG(Role playing game)

Style : Jeu d'action

Plate-forme : PC

Nombre de joueurs : 1

Pegi : ?

4 Description générale du projet

Le jeu est un RPG où le joueur incarne un seul personnage qui évolue en passant d'un niveau à un autre. L'objectif du joueur est de tuer ses ennemis, récupérer leurs clés, ouvrir des portes, collecter des items et améliorer ses armes afin de pouvoir affronter le Boss final.

5 Principe du jeu

- Le joueur lance une partie
 - la carte du jeu est générée
 - le joueur peut se déplacer uniquement sur la partie de la carte correspondant à son niveau.
 - pour passer d'un niveau à un autre il faut ouvrir des portes
 - pour ouvrir une porte il faut récupérer sa clé (une ressource)
 - une clé est possédée par un ennemi, le seul moyen de la récupérer est de le tuer.
 -
- Le joueur peut attaquer ou être attaqué par un ennemi
 - lors d'une attaque, le joueur comme les ennemis ,peut rater un coup. Cette probabilité dépend de leurs niveaux.
 - un ennemi peut posséder plusieurs items (armes, potions ..). Après avoir tué un ennemi, le joueur récupère tous ses items.
 - Le joueur peut améliorer une arme uniquement lorsqu'il se situe à proximité d'une case enclume et qu'il a les ressources nécessaires à l'amélioration.
 - Le joueur, comme les monstres et les armes, a un niveau qui peut évoluer avec l'expérience qu'il accumule. Le niveau détermine la vie joueur. De plus, le joueur ne pourra utiliser une arme que s'il a le niveau suffisant.
- Le joueur peut récupérer ou jeter des items.
 - Sur la carte, à chaque niveau se trouve des items (armes, potions ..) que le joueur ramasse automatiquement en marchant dessus. En outre, le nombre d'item de chaque catégorie que peut posséder un joueur est limité.
 - Les items ramassés sont stockés dans un inventaire.
- Le jeu se termine dans deux cas : lorsque le joueur tue le boss ou meurt.

- Un personnage meurt lorsqu'il n'a plus de vie.

6 Fonctions principales

- **LevelUp()** : en gagnant de l'expérience, le joueur augmente son level
- **AméliorerArme(arame)** : le personnage peut utiliser les ressources qu'il collecte pour améliorer ses armes
- **Jeter()** : Il peut également jeter une arme ou une potion lorsqu'il en a plus besoin
- **Ramasser(item)** : cette fonction permet au joueur de ramasser les items (armes, potions ..) et de les mettre dans un inventaire.
- **Bouger()** : Permet au joueur de se déplacer dans la carte
- **addSomeRessources () , addSomeItems()** : Pour mettre des ressources ou des items dans une case.
- **actualiserInventaire()** : les items ramassés sont stockés dans des inventaires, donc cette fonction permet de mettre à jour ces inventaires après chaque stockage.
- **Utiliser(Entité)** : le joueur et les ennemis sont considérés comme des entités, ils peuvent donc utiliser des armes pour attaquer ou des potions pour gagner une vie, empoisonner ..
- **Jouer()** :
- **save()** :
- **Pause()** : un arrêt temporaire d'une partie du jeu
- **Exit()** : Permet au joueur de sortir du jeu.
- **recommencer()** :
- **GameOver()** : Lorsque le personnage meurt, le jeu s'arrête.

7 Fonctions optionnelles

En plus de ces fonctionnalités principales, nous envisageons d'en faire d'autres qui seront donc optionnelles, en fonction de notre avancement et du temps restant.

- Un menu personnalisé pour le Jeu (Custom Game Menu), Ce dernier facilitera l'expérience de l'utilisateur, et rendra notre jeu plus esthétique.
- Sauvegarder ou Charger une partie
- Plus d'effets sonores et visuels (textures par exemple) pour une meilleure expérience utilisateur.
- Un historique du score en fonction du temps des parties précédentes.

8 Langages et framework

8.1 Gestion de versions

Pour gérer nos codes sources, nous avons choisi Github qui est un logiciel de gestion de versions décentralisé qui permettra aux membres du groupe de voir toutes les modifications faites et de visualiser l'historique des validations.

8.2 Langages

Nous allons réaliser cette application dans le cadre du projet de programmation orientée objet en Java. Nous sommes donc obligé de choisir ce langage pour appliquer l'ensemble des notions vues en cours.

8.3 Bibliothèques graphiques

Afin d'appliquer ce que nous avons vu en cours dans la partie IHM JavaFx, nous allons probablement utiliser la bibliothèque FXGL qui facilite la création d'interfaces graphiques pour les jeux vidéos 2D et 3D, son utilisation sera donc optionnelle en fonction de notre vitesse d'apprentissage.

8.4 plateforme de développement

IntelliJ et Eclipse, ce dernier est l'IDE le plus utilisé dans les entreprises. C'est également l'IDE vu en cours de Java.

9 Tâches à faire

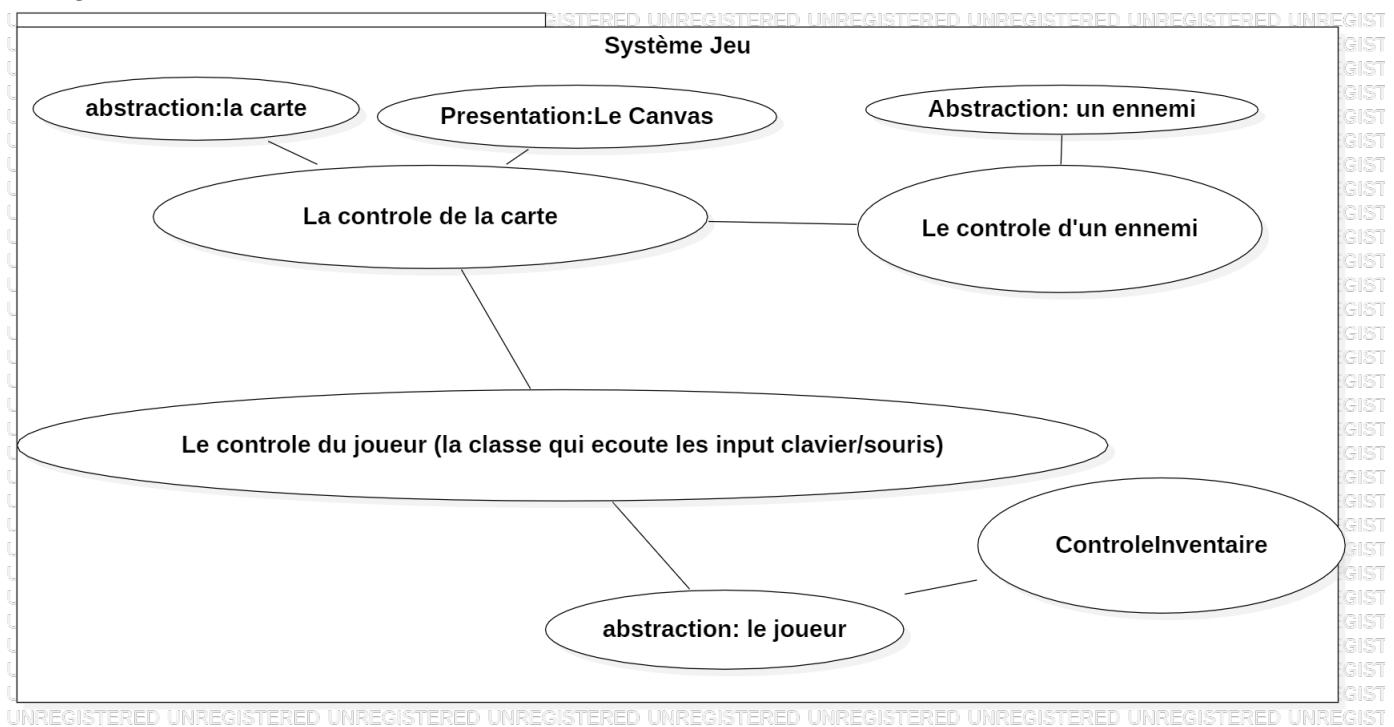
Comme spécifier lors de la présentation de ce projet, nous allons traiter nos tâches en fonction de leur priorité.

- Brainstorming à propos du Jeu, et choix de ce dernier
- Modélisation du jeu en UML
- Comparaison de l'idée initiale et la modélisation, Corrections si nécessaire.
- Elaboration des différents algorithmes et adaptation de notre UML en fonction des erreurs et problèmes rencontrés.
- Vérification des algorithmes (tests d'intégrations)
- En parallèle, travailler sur la bibliothèque FXGL, en cas d'utilisation
- Réalisation de la partie IHM
- Réalisation de quelques ou toutes les fonctions optionnelles en fonction de notre avancement
- Vérification finale
- Faire des tests à l'issue de chaque tâche.

Cinquième partie Architecture

10 Préambule

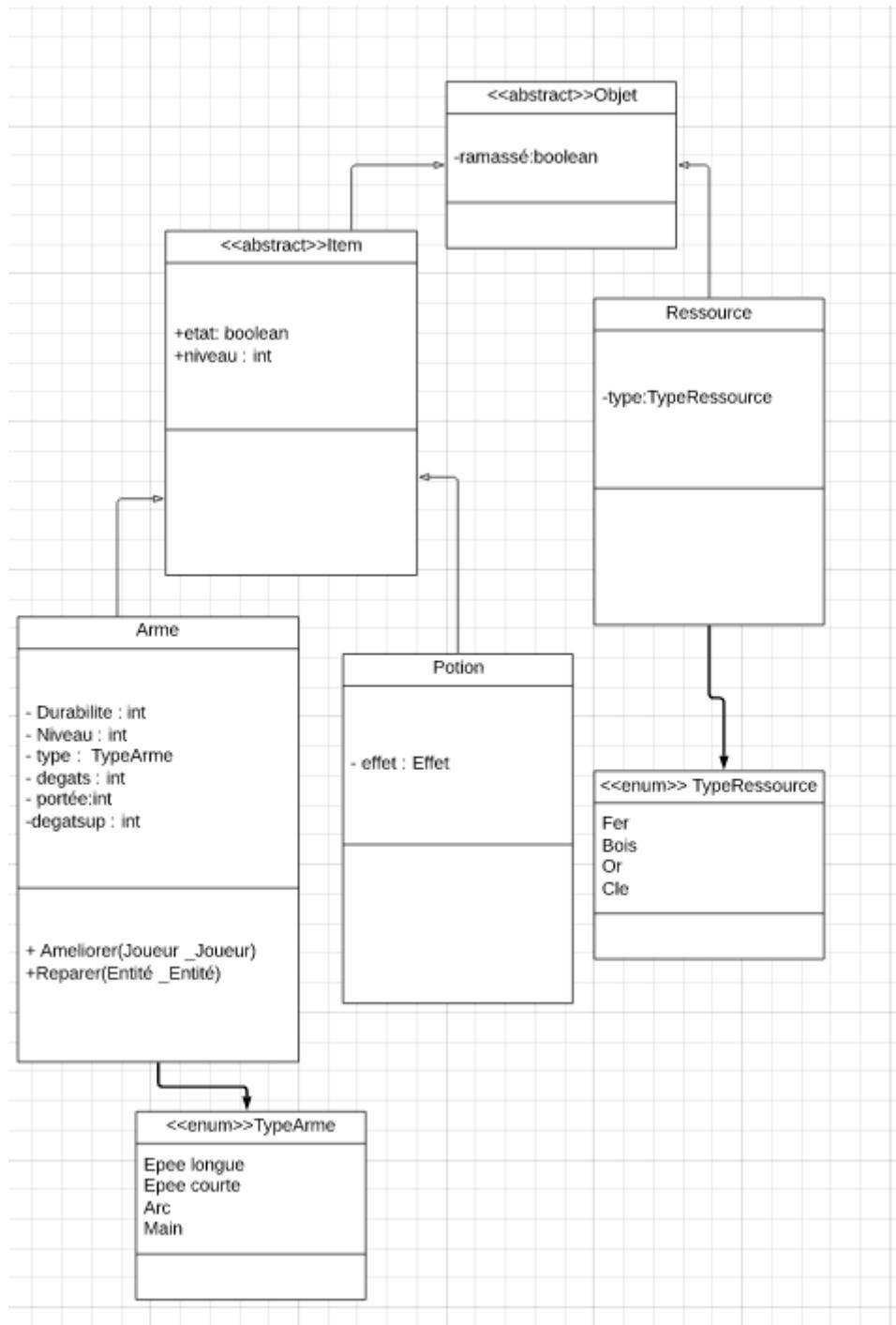
L'application, et plus précisement l'interface du jeu est basée sur la méthode PAC, les différents packages du code sont donc organisés suivant cette logique. Le package Jeu contient principalement les classes du modèle de données du jeu. Le package contrôle contient les différentes classes de contrôle qui font la liaison entre les composants graphiques de l'IHM et les entrées clavier de l'utilisateur et le modèle de données. Pour illustrer une ébauche de cette configuration, nous avons confectionné le schéma suivant à titre indicatif :



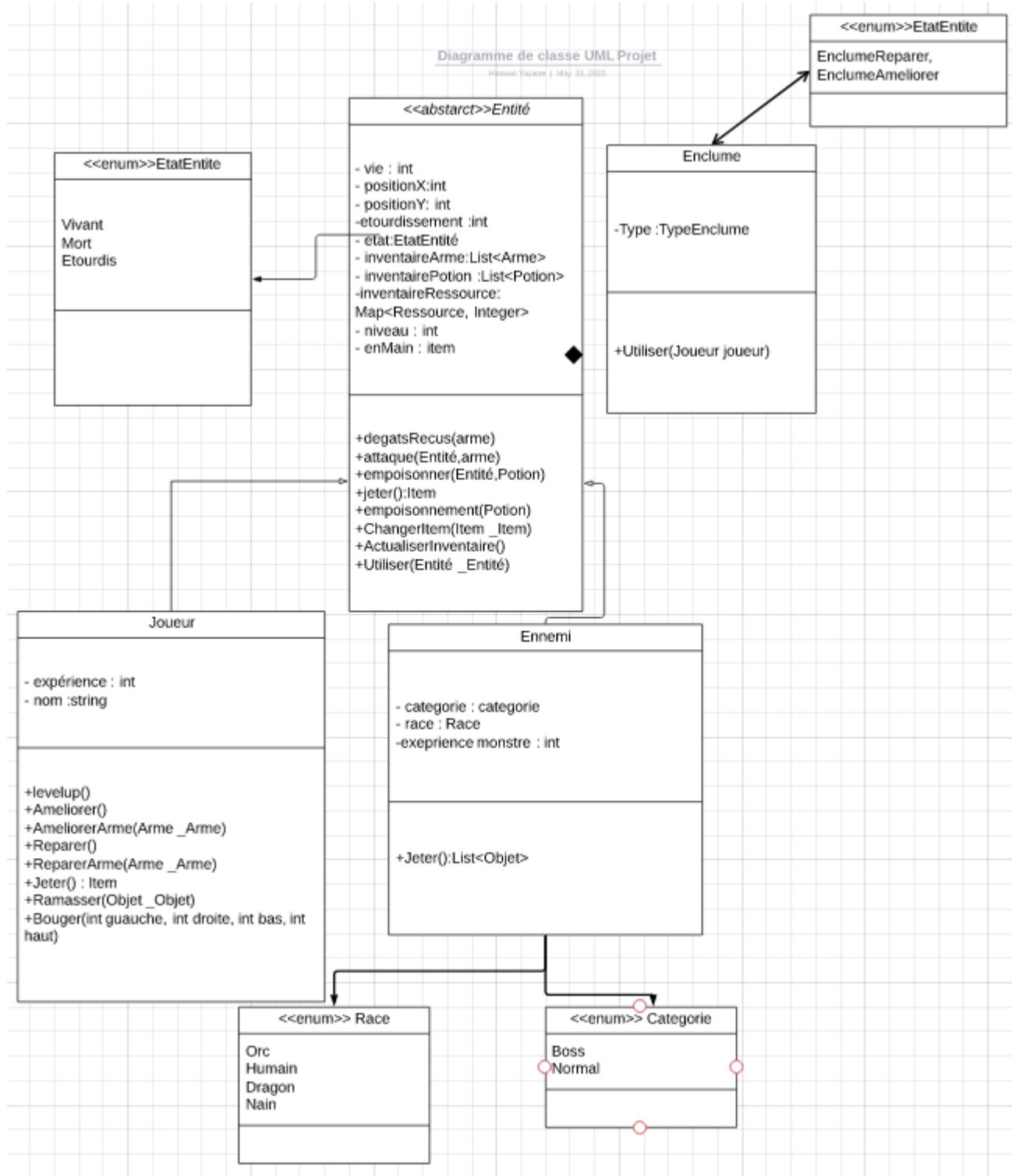
11 Diagrammes structurels

11.1 Description du modèle de données

Partie abstraite



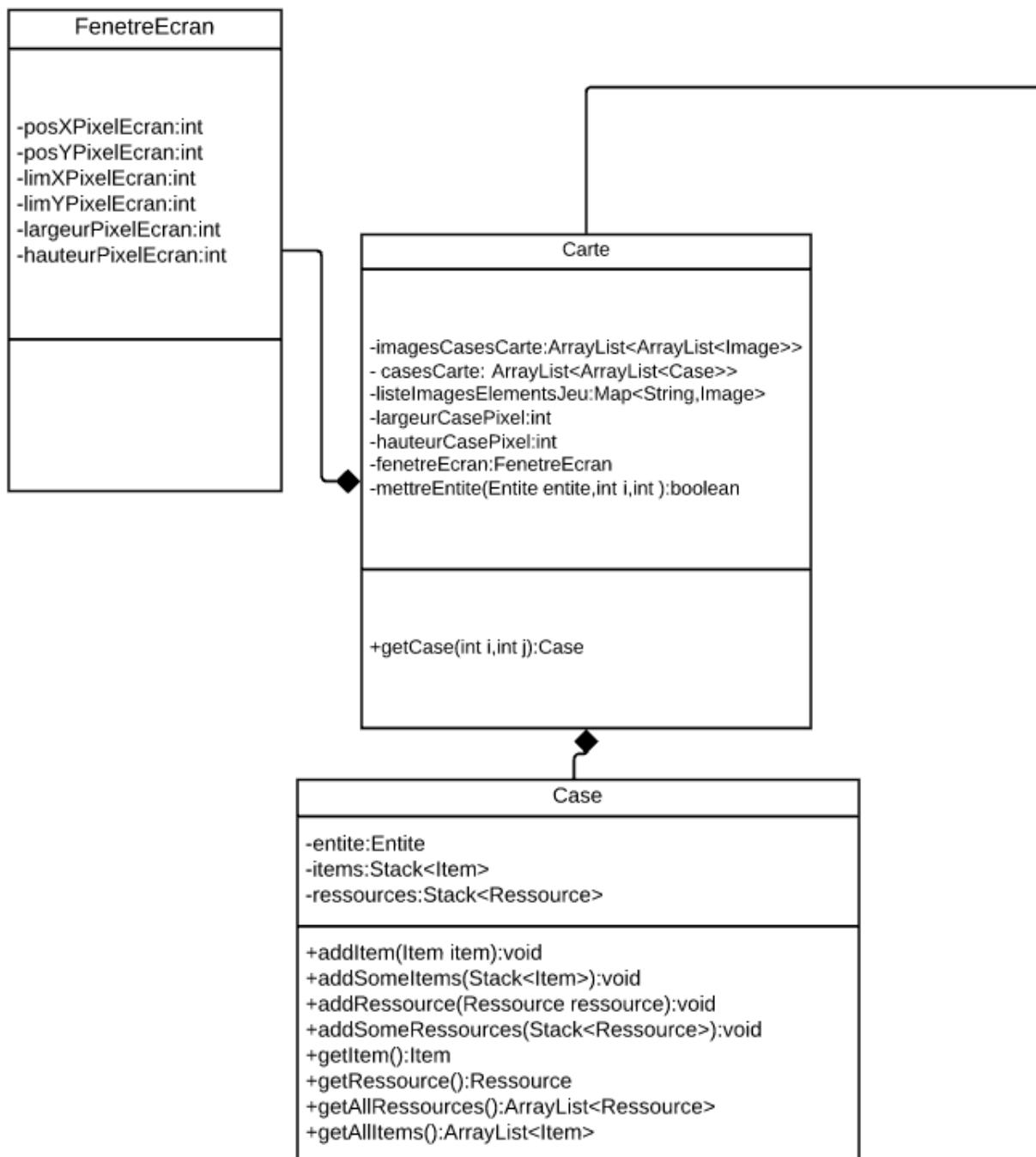
Pour la structure de ces classes, nous avons décidé de créer une classe objet en tant que classe père qui définit si un objet est ramassé ou non. Par la suite nous avons deux types d'objets, les ressources et les items. Les ressources sont les clés et les matériaux qui seront enregistrés dans un HashMap par la suite, dans le jeu. Les items sont des objets utilisables par le joueur ou les ennemis, on y retrouve les potions et les armes. Un item peut se casser, soit après utilisation, comme la potion, soit après la fin de sa durabilité, comme les armes ; il existe différents types d'armes, qui peuvent toutes être améliorées ou réparées, à l'exception des armes du type "main".



Pour la partie joueur et ennemi, nous avons décidé de créer une classe entité père qui regroupe la plupart des fonctions communes du joueur et de l'ennemi, comme recevoir des dégâts, attaquer, changer d'arme, jeter une arme, etc . Une entité peut être étourdie, morte ou vivante, elle possède un niveau et une position ainsi que 3 types d'inventaires, un Hashmap pour les ressources et deux listes pour les armes et les potions ; une entité possède également un item en main, que ce soit potion ou arme. Un joueur peut augmenter de niveau en tuant un monstre, il

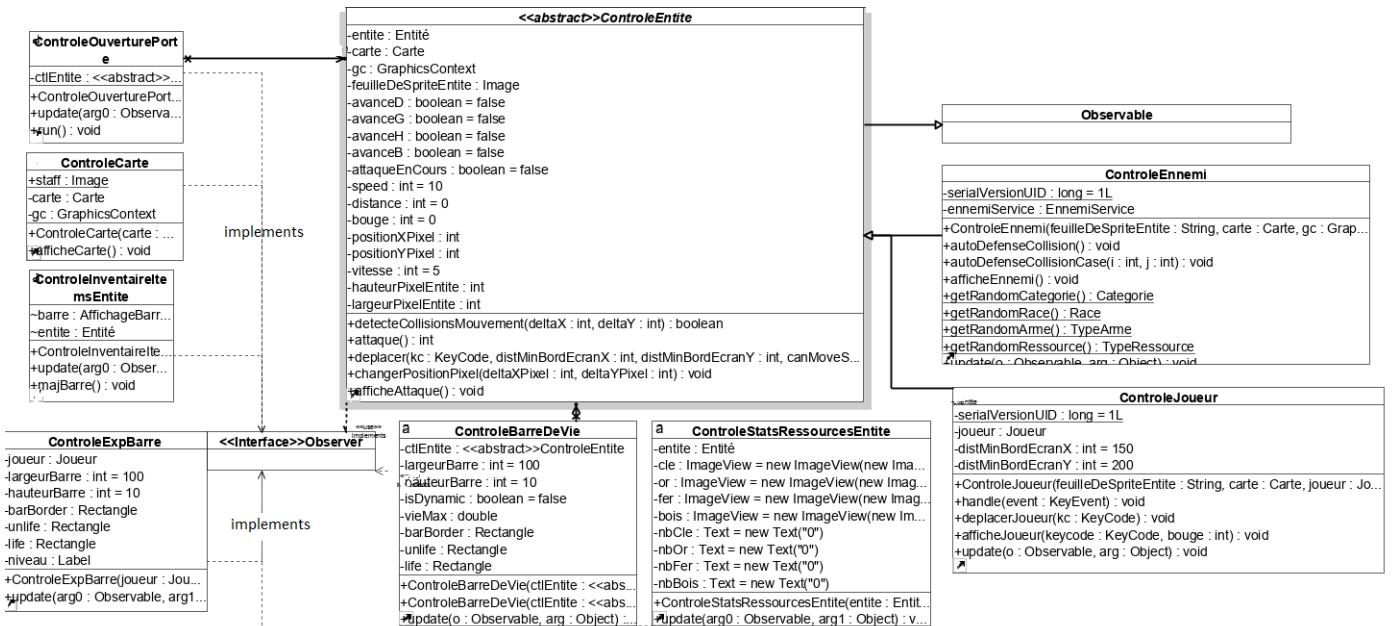
peut également bouger et ramasser des objets, il peut améliorer son arme ou encore la réparer grâce à une enclume. A l'inverse, l'ennemi ne peut donc pas bouger, il possède uniquement une jauge d'expérience qu'il donnera au joueur au moment de sa mort, ainsi que les objets dont il dispose qu'il laissera tomber sur le sol. Un ennemi est du type Boss ou normal et possède différentes races.

La carte est l'élément d'abstraction de la map dans laquelle évolue le joueur. Idéalement, elle peut être assimilée à un échiquier dont les cases (les objets de la classe Case) ne peuvent être occupés uniquement par : soit une entité, soit un obstacle (porte fermée, arbre, mur, enclume,...), soit rien (d'où la variable statique dans le code Case.VIDE). Chaque case possède une liste d'items qui y ont été déposés au début ou pendant le jeu.



11.2 Description du modèle de contrôle

Contrôles



Les contrôleurs ont pour mission de faire la liaison entre le modèle d'échiquier du plateau de jeu, sur lequel les positions sont en coordonnées de case, et le canvas, sur lequel les positions sont en pixels. Ce qui rend les déplacements continus et plus agréables pour l'utilisateur. Nous allons dans les deux sous-parties qui suivent décrire le rôle des deux principaux contrôleurs qui sont le ControleCarte et le ControleEntite.

11.2.1 La classe ControleCarte

Le contrôle carte est principalement chargé d'afficher à l'écran (de dessiner sur le canvas) la partie de la carte où se situe le joueur via la méthode afficherCarte() dont est décrit le fonctionnement en partie VI section 17.2

11.2.2 La classe ControleEntite

Le contrôle entité transforme les positions en coordonnées de case vers des positions en coordonnées pixelisées.

Pour cela, on utilise le principe suivant : L'entité se situe dans la case qui contient le centre de son image. Ainsi, si le centre de l'image (centre en pixels) passe à travers le bord d'une case en pixel, le contrôle change l'entité de case grâce à la fonction mettreEntiteCase(...) de la classe carte.

11.3 Description du modèle de présentation

La présentation des données se faisant à travers les composants graphiques de l'IHM. Celle-ci est décrite plus bas dans le rapport dans la partie V section 13 Structure de l'IHM.

12 Diagrammes de comportement

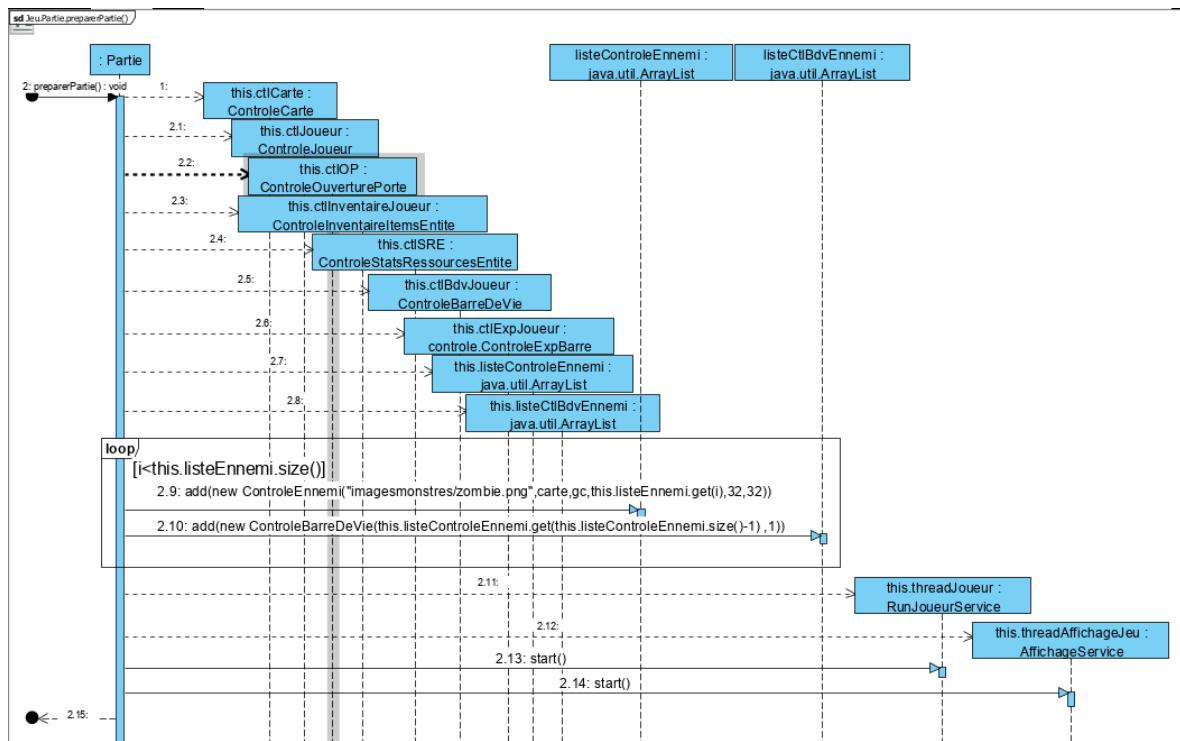
12.1 Diagrammes de séquence

1. Séquence de préparation d'une partie

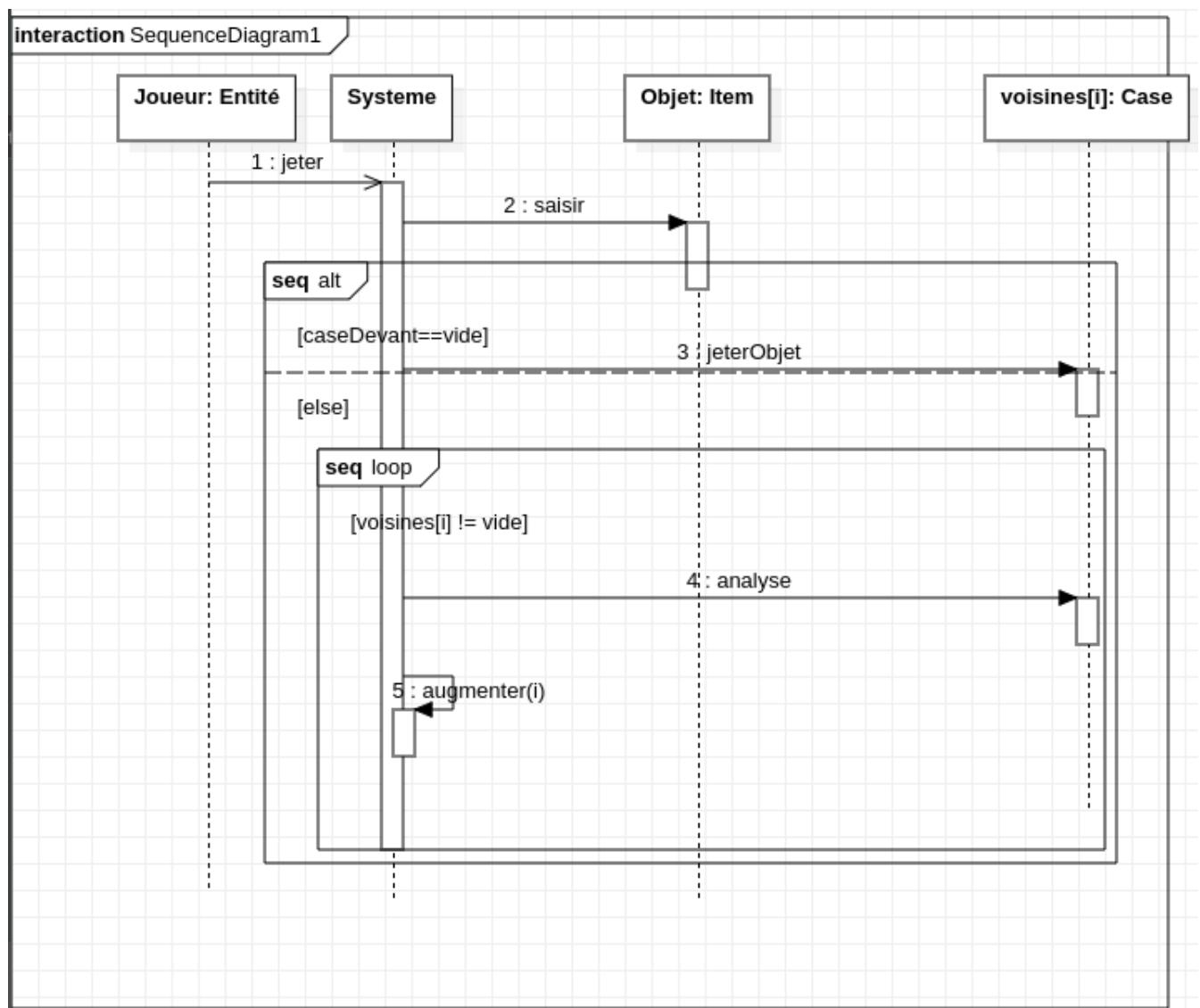
La séquence préparerPartie() de la classe Partie est appelée lorsque le joueur charge une partie ou en crée une nouvelle via le menu de l'application. Elle est chargée de lancer les deux principaux processus du jeu sous-jacents :

a. Le processus d'affichage dans le Canvas. Il s'agit de la classe AffichageService qui étend la classe AnimationTimer de JAVAFX. La librairie JAVAFX ne permettant pas qu'un autre processus que celui de l'Application modifie ses composants graphiques, elle propose la classe AnimationTimer qui se synchronise et met à jour les composants graphiques en respectant la planification de "l'horloge interne" de javafx.

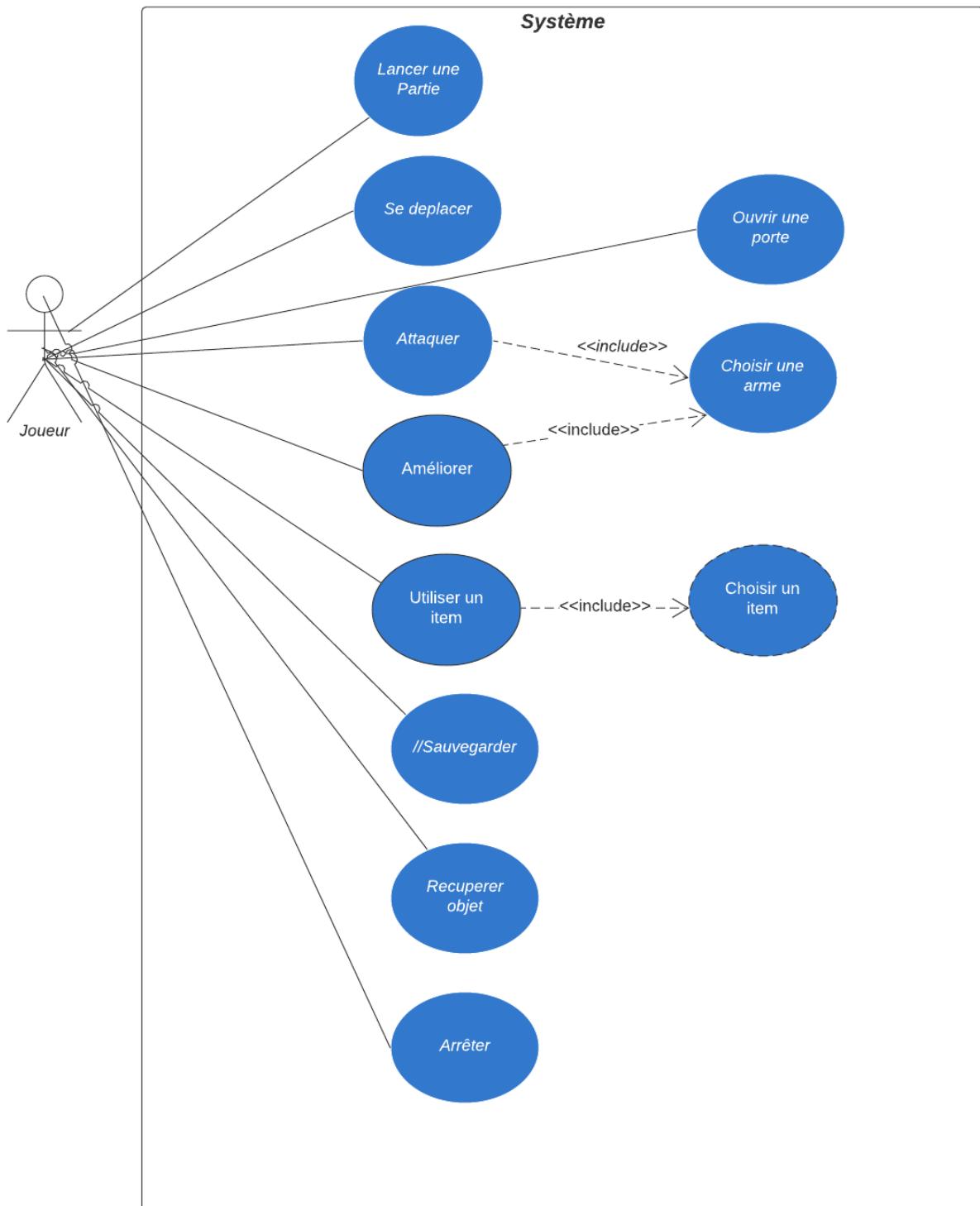
b. Le processus qui déplace le personnage en fonction des entrées clavier de l'utilisateur. Il s'agit de la classe RunJoueurService qui étend la classe Service de JAVAFX. Cette classe permet de modifier le modèle de données via un second processus. Le joueur se déplace selon 4 directions traduites dans le code par 4 booléens qui sont des attributs de la classe ControleEntite. Le RunJoueurService modifie en permanence la position du joueur en regardant ces 4 booléens. Ces booléens étant eux-mêmes modifiés par le gestionnaire d'événements de la classe ControleEntité.



2. Séquence de l'opération jeter



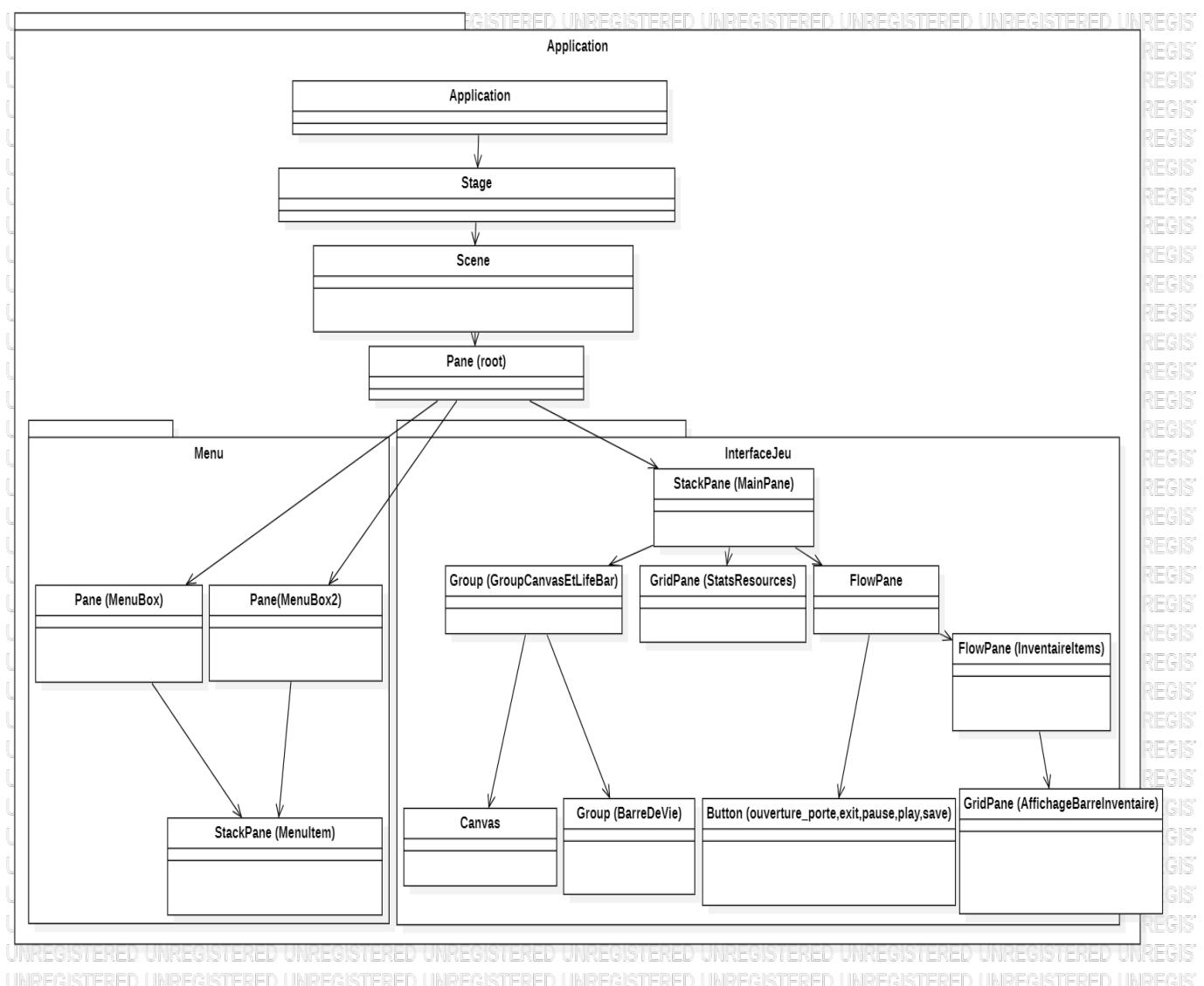
12.2 Diagramme de cas d'utilisation



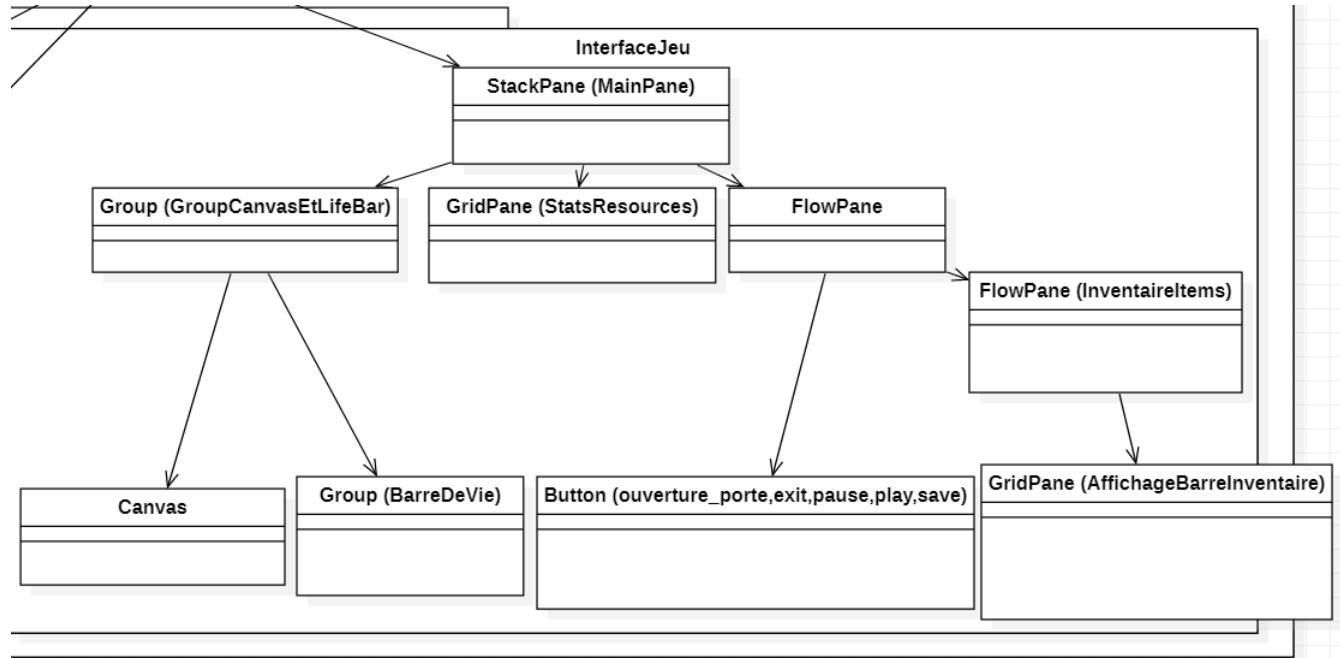
13 Structure de l'IHM

L'application étant développée avec la bibliothèque JAVAFX, elle étend la Classe Application de JAVAFX. S'ensuivent les classes Stage et Scene. Le principal découpage de l'application s'effectue dans le panneau Pane, appelé "root" dans le code, qui contient d'une part le menu de l'application et d'autre part l'interface du jeu (Ici, l'interface du jeu désigne l'écran où apparaît le personnage se promenant dans la map du jeu).

C'est lorsque l'utilisateur lance une partie que l'ensemble des observables "fils" du panneau "root" changent. On enlève les composants du Menu (ceux de gauche sur le graphique) et on ajoute en noeud fils de "root" le StackPane (intitulé MainPane dans le code) qui contient les composants de l'interface du jeu



13.1 Description du graphe de scène de l'interface du jeu



StackPane (MainPane) Le StackPane appelé MainPane est le composant-père de la fenêtre de Jeu. Le choix du StackPane permet d'ajouter tous les boutons accessibles par l'utilisateur et les stats du joueur directement sur la carte. Là où on aurait pu réduire la taille de la carte qui défile et mettre séparément les boutons, nous avons préféré utiliser cet espace pour donner au joueur plus de visibilité et ainsi laisser la carte s'étendre "sous" les boutons qui se retrouvent au premier plan, soit en haut de la pile StackPane.

GridPane (StatsRessources) Les ressources du personnage affichées n'ont pas d'interaction directe avec l'utilisateur si ce n'est au travers du personnage qui, en se déplaçant, ramasse des clés, de l'or, etc. De plus, la structure dans laquelle sont stockées ces ressources est un Map<Resource, Integer>, un GridPane convient donc à cette structure (tout composant permettant d'afficher un tableau aurait pu convenir). Ce composant est situé au second-plan dans l'ordonnancement du StackPane (MainPane).

FlowPane (éléments cliquables) Nous avons choisis de réunir tous les composants cliquables par l'utilisateur dans un même composant-père pour une raison pratique. Le FlowPane, comme son nom l'indique, reste un composant souple permettant une répartition d'éléments sur les bords du panneau. De la même manière que nous souhaitions répartir les boutons sur les bords de l'écran du jeu, nous avons opté pour un FlowPane. Ce composant est au premier plan dans le StackPane MainPane puisque c'est celui que l'utilisateur peut cliquer. Le FlowPane "InventaireItems"(repérable sur le schéma) contient lui-même un GridPane qui contient les cases de l'inventaire du personnage (Chaque case est cliquable et correspond à un item sélectionnable) cf le manuel du jeu en partie III section 1.2.1.

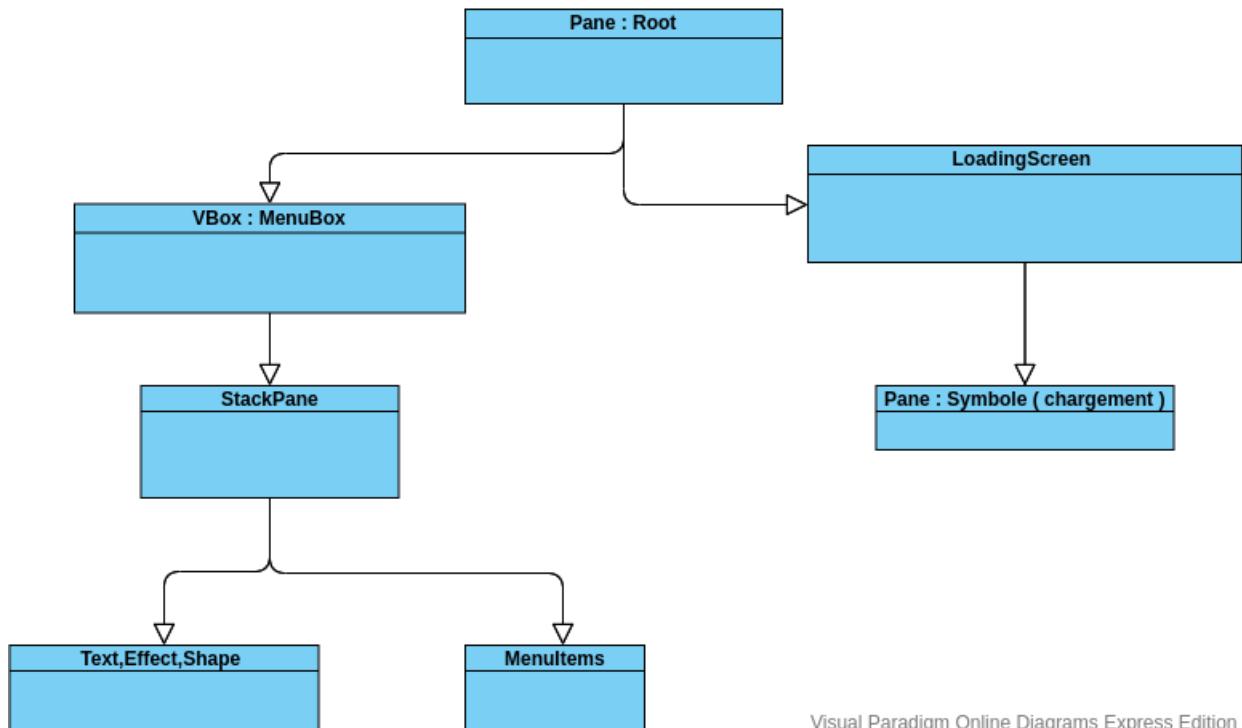
Group (GroupCanvasEtLifeBar) Ce Group, qui est situé au troisième et dernier plan du StackPane (MainPane), est composé des barres de vie de l'ensemble des personnages(héros et ennemis) ainsi que du Canvas du jeu (là où s'affichent la carte et les personnages).Il s'agit des éléments à affichage dynamique. En effet, la carte défile, les personnages se déplacent et leur barre de vie avec eux. Nous avons toutefois tenu à distinguer le Canvas des barres de vies. Les barres de vie sont en fait modulables. C'est-à-dire que l'on peut choisir de les afficher ou non indépendamment du Canvas. Elles sont fixes ou non. On peut ainsi, pour n'importe quelle entité (héros ou ennemi), choisir d'afficher sa vie au dessus de sa tête ou de manière fixe dans un coin de l'écran.Toutes les barres de vie créées sont elles-mêmes réunies dans un Group.

Le Canvas est un composant de JAVAFX qui permet un rendu graphique sur lequel on peut effectuer un grand nombre d'opérations de dessins et donnant ainsi l'illusion du mouvement à celui qui le regarde. C'est donc sur ce canvas que se redessinent indéfiniment (jusqu'à la fin du jeu) la carte et les personnages. De plus le canvas possède un GraphicsContext (une autre classe de JAVAFX) qui possède une multitude de capacités de personnalisation.

13.2 Description du graphe de scène du Menu

Visual Paradigm Online Diagrams Express Edition

MenuInterface



Visual Paradigm Online Diagrams Express Edition

Pane. Dans un premier temps lors de la réalisation de notre fenêtre Root (la racine qui est le composant père de la fenêtre), nous voulions faire un StackPane, mais au fur et à mesure de notre avancement, cela a posé des problèmes de positionnement, nous avons donc opté pour un Pane basique qui permet une liberté de positionnement de ses fils. C'est également le cas pour le loadingScreen.

VBox. Notre choix s'est porté sur Vbox au lieu de Pane pour la réalisation du MenuBox, car ce dernier est un conteneur (container), qui arrange les sous-composants sur une seule colonne, et c'est exactement ce qu'on voulait, car ce dernier contiendra des éléments qui devront être bien placés et positionnés de façon automatique.

StackPane. Nous avons hésité entre StackPane et Group pour la réalisation de MenuItems, sauf que Group positionne ses fils à (0,0) automatiquement, alors que StackPane gérera automatiquement l'emplacement de ses fils, ce qui permet d'ajouter du contenu au Menu sans avoir de problèmes. C'est également le cas pour Text (le text des éléments), Effect (l'effet flouté et changement de couleur) et Shape (Rectangle), qui forment les effets visuels lors de la sélection d'un élément du Menu, comme par exemple le changement de couleur ou bien la création d'un rectangle entourant cet élément, ce rectangle se situe ainsi au dernier plan de notre StackPane initial.

Sixième partie

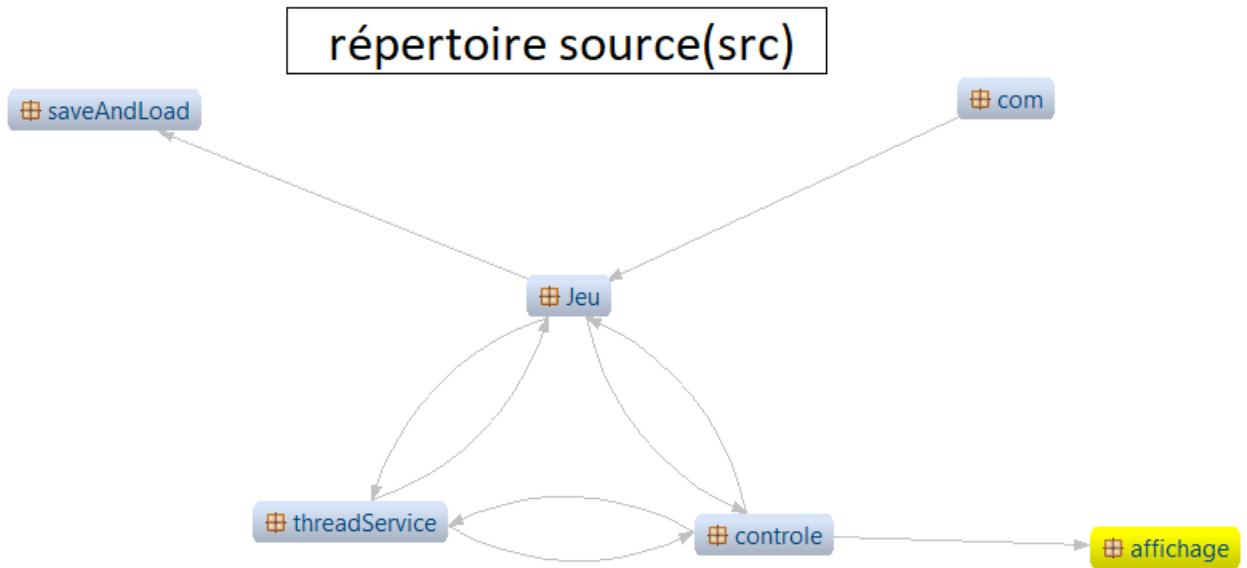
Le code : Arborescence des fichiers et choix d'implémentation

14 Arborescence des packages

Le dossier racine du projet, intitulé "JavaProject" contient la liste des répertoires suivante :

- src (là où sont contenus tous les fichiers .java)
- cartes (dossier qui contient des cartes du jeu sous la forme de fichiers .csv)
- css (dossier qui contient les styles de certains composants)
- images (ce dossier contient les images des éléments de la carte)
- imagesitems (ce dossier contient les images des items du jeu)
- imagesmenu (ce dossier contient les images du menu)
- imagesmonstres (ce dossier contient les sprites des monstres ainsi que celui du héros)
- musiques (ce dossier contient les musiques du jeu)

Tous les packages des classes de l'application sont donc regroupés dans JavaProject/src et sont organisés selon les dépendances suivantes :



15 Logique des packages

Le projet reste souple, il est facile d'ajouter de nouvelles fonctionnalités à condition de respecter la logique qui suit :

15.1 Le package Jeu

Le package Jeu renferme le noyau du modèle de données. Il contient toutes les classes qui définissent ce que les personnages et les ennemis peuvent faire ou ne pas faire. Si on est amené à modifier les règles du jeu, il faudra obligatoirement modifier les classes concernées dans ce package. En outre, si l'on souhaite ajouter un élément ou un personnage, il faudra ajouter la classe de cet élément dans le package.

15.2 Le package threadService

Le package threadService contient les processus du jeu. En effet certaines opérations nécessitent de les traiter indépendamment afin qu'elles puissent se dérouler sans bloquer l'intégralité du jeu. C'est le cas du processus qui déplace le joueur. Le processus est lancé à part pour une raison de fluidité et pour avoir un meilleur contrôle des entrées clavier. En effet les événements d'entrée clavier dans JavaFX sont gérés par la classe KeyEvent ; plutôt que de se déplacer lorsque l'événement OnKeyPressed se produit. Un programme essaie en permanence de faire avancer le joueur.

Le package contient également des Timer pour réguler le nombre de coups distribués par seconde par exemple.

15.3 Le package contrôle

Le package contrôle détient les classes de contrôle du jeu. Il assure la cohérence de la partie abstraite du jeu et fait également la liaison entre les composants d'affichage de l'IHM et la partie abstraite. Si on rajoute un élément graphique, il faudra modifier ou ajouter un contrôleur.

15.4 Le package saveAndLoad

Le package saveAndLoad n'a pas réellement vocation à être modifié, il permet la sérialisation et désérialisation des données d'une partie pour pouvoir faire des sauvegardes et les recharger lors d'un lancement ultérieur du programme.

15.5 Le package com

Le package com contient les composants du menu du jeu. Comme le package saveAndLoad, les modifications seront plus rares.

15.6 Le package affichage

Le package affichage a été créé pour ne pas surcharger le package contrôle. Il permet de personnaliser des composants graphiques complexes du jeu comme c'est le cas pour l'inventaire cliquable du joueur.

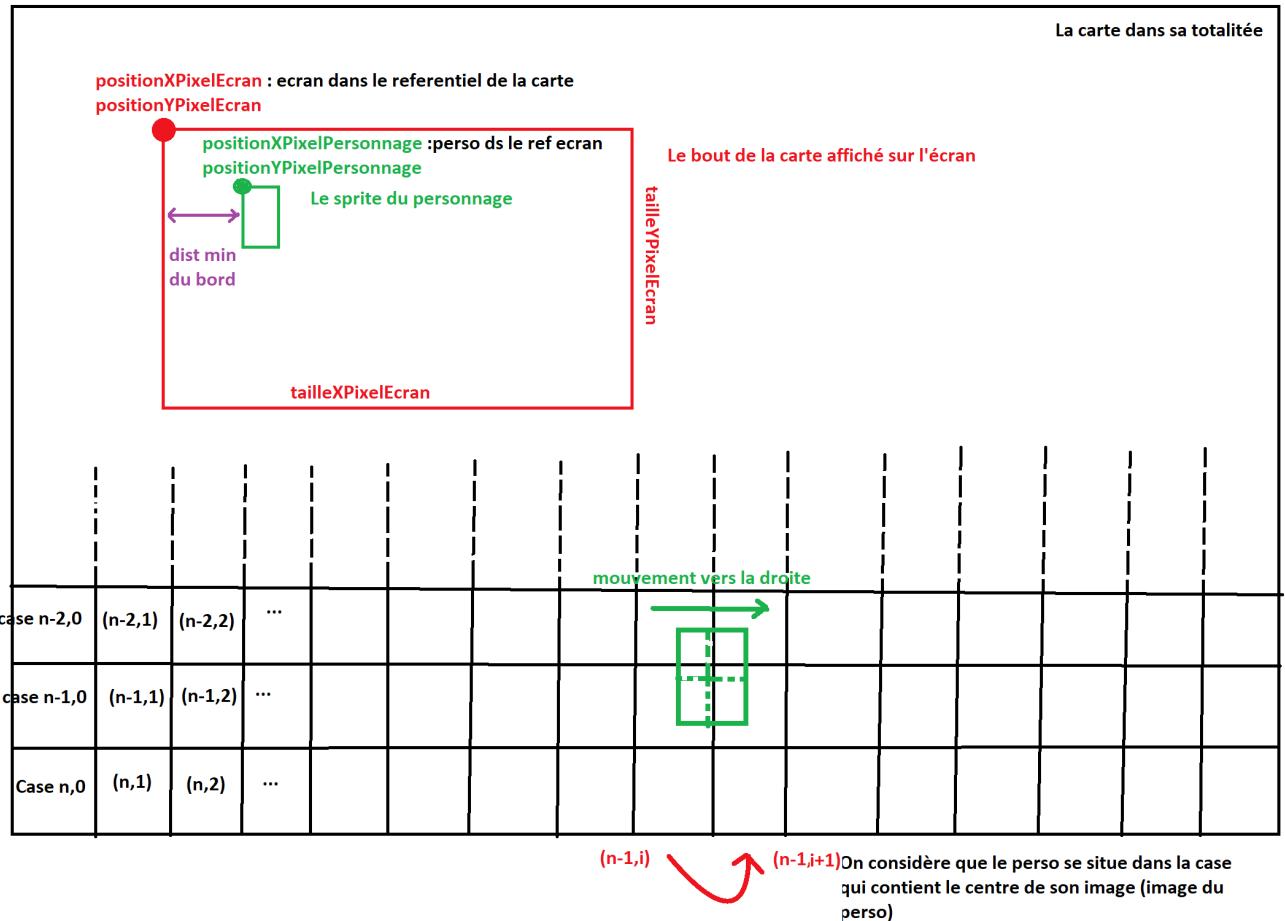
16 L'illusion du mouvement : Techniques d'affichage et Algorithmique

16.1 Mouvements du héros

Le mouvement du héros est simulé à l'aide de la succession d'images prises toutes sur la même feuille appelée "feuille de sprites". Ainsi chaque personnage dispose de sa propre "feuille de sprites". Evidemment, une fois dessinée sur le canvas, l'image du personnage ne peut être effacée pour remettre l'image qui correspond à la position suivante dans le mouvement. L'idée est donc de redessiner l'image de la carte entre les deux. On obtient ainsi l'ordre d'affichage des différents éléments du décor.

16.2 Défilement de la carte

Le défilement de la carte a été conçu à partir de la méthode décrite par le schéma ci-dessous :



L'idée est de considérer l'ensemble de la carte. Seulement, celle-ci étant bien trop grande pour être affichée par l'écran dans son intégralité, nous devons également considérer la partie de la carte que l'écran affiche. Nous avons donc donné à l'écran des coordonnées en pixel qui le place dans le référentiel de la carte (il s'agit du rectangle rouge sur le schéma). Le personnage est ensuite lui-même référencé dans le référentiel de l'écran. De cette façon, lorsque le personnage s'approche à une distance limite (en violet sur le schéma) du bord de l'écran, le personnage cesse de bouger dans le référentiel de l'écran et c'est l'écran qui bouge dans le référentiel de la carte.

Nous avons rédigé l'algorithme qui permet d'afficher la partie de la carte qui correspond à la position de l'écran.

```

procédure affichageEcran()

variables:
    ArrayList<ArrayList<Image>> imagesCases;
    int positionXPixelEcran;
    int positionYPixelEcran;
    int nbLignesCarte;
    int nbColonnesCarte;
    int varTempPourX;
    int varTempPourY;
    int largeurCase;
    int hauteurCase;
    int numColonnePremiereCaseEcran;
    int numLignePremiereCaseEcran;
    int indiceLigneTempCase;
    int indiceColonnetTempCase;

Debut:
    numLignePremiereCaseEcran =  $\left\lfloor \frac{\text{positionYPixelEcran}}{\text{hauteurCase}} \right\rfloor$ ;
    numColonnePremiereCaseEcran =  $\left\lfloor \frac{\text{positionXPixelEcran}}{\text{largeurCase}} \right\rfloor$ ;
    varTempPourX := (numColonnePremiereCaseEcran+1) * largeurCase - positionXPixelEcran ;
    varTempPourY := (numLignePremiereCaseEcran+1) * hauteurCase - positionYPixelEcran ;
    indiceLigneTempCase := numLignePremiereCaseEcran;
    indiceColonnetTempCase := numColonnePremiereCaseEcran;
    Tant que (VarTempPourY < hauteurEcran) Faire {
        Tant que (varTempPourX < largeurEcran) Faire {
            dessinImage(imagesCases.get(indiceLigneTempCase).get(indiceColonnetTempCase), varTempPourX, varTempPourY);
            varTempPourX:=varTempPourX + largeurCase;
            indiceColonnetTempCase:= indiceColonnetTempCase+1;
        }
        FinTantque
        varTempPourX := (numColonnePremiereCaseEcran+1) * largeurCase - positionXPixelEcran ;
        varTempPourY:=varTempPourY + hauteurImage;
        indiceColonnetTempCase := numColonnePremiereCaseEcran;
        indiceLigneTempCase := numLigneTempCase+1;
    }
    FinTantque
Fin

```

16.3 Attaque automatique des ennemis

La gestion des ennemis est sans aucun doute, la partie requérant la plus grande technicité algorithme, puisqu'il s'agit d'une intelligence artificielle. Dans ce jeu, les ennemis ont donc le comportement suivant : chaque ennemi regarde en permanence si un joueur se trouve à une case de distance dans les quatre directions : haut ,droite, gauche, bas. Si un joueur s'y trouve il le frappe continuellement toute les 0.5 seconde (ce paramètre pouvant être modifié) jusqu'à ce que mort s'en suive.

Septième partie L'équipe : rôles et organisation

17 Méthode de fonctionnement au sein du groupe

La planification des tâches a été hasardeuse pour les deux raisons qui suivent : Il a fallut nous former en même temps d'avancer le projet à différentes technologies.

Voici un historique de notre avancement. Il s'agit d'un journal de bord tenu en parallèle du projet.

15/05/20

- écriture des règles du jeu
- diagramme de cas d'utilisation
- diagramme de classe (partie abstraite)
- design IHM écran de jeu

16/05/20

- implémentation des classes (partie abstraite)
- mise au point sur les bibliothèques utilisées (installation de JAVAFX)
- création des classes Case+Coordonnées
- implémentation d'une fonction jeter pour les entités
- création d'une durabilité pour les armes
- conception du modèle PAC de l'application

17/05/20

- architecture des classes de contrôle basées sur le modèle PAC
- optimisation de l'utilisation des items par la classe Entité
- proposition d'interfaces graphiques de jeu (exemple de gestion des collisions)
- découverte de la classe AnimationTimer pour créer une boucle de jeu
- premier partage du code, création du dépôt sur Git
- répartition de développement des classes de contrôle
- changement de Structure pour l'inventaire des entités (HashMap -> List<Item[]>)

18/05/20

- exemples FXGL pour la gestion des collisions
- modifications dans les classes de la partie abstraction

19/05/20

- écriture au propre du cahier des charges
- changement de structure pour la classe carte
(HashMap<Coordonnées,Case> -> ArrayList<ArrayList<Case>>)

20/05/20

- réflexion sur la gestion de la carte (défilement de la carte ou carte fixe)
- la classe ControleJoueur est désormais utilisable
- recherche des premiers sprites pour le héros du jeu
- mise au point de la hitbox du héros (gestion propre des collisions)

21/05/20

- mise au point de l'attribut expérience pour le héros
- dessin de l'interface de jeu révisé
- déplacement de la méthode de traitement des collisions de la classe controle joueur vers la classe ControleEntite

22/05/20

- implémentation de la classe partie qui permet de lancer une partie
- ré-écriture de la méthode changerItem pour un joueur

23/05/20

- implémentation de la classe ControleStatsRessourcesEntite
- recherche de design pour les éléments du décor, les murs ,etc

- implémentation de l'inventaire cliquable par le joueur dans l'interface
24/05/20
 - ajout du contrôle d'ouverture des portes, boutons
 - implémentation du menu + IHM Menu + musique Menu
- 25/05/20
 - révision de la méthode jeter pour les entités et adaptation dans les différents contrôles
 - découverte de la sérialisation pour mettre en place un système de sauvegarde des parties
 - implémentation des méthodes pour quitter et mettre en pause une partie
 - implémentation de la classe saveAndLoad
 - pb avec la gestion des différents processus dans javafx, découverte de la classe Service de JavaFX
- 26/05/20
 - debug du contrôle de la barre de vie qui ne semble pas se mettre à jour correctement
 - intégration des boutons pour quitter et sauvegarder une partie
 - nouveau pb d'affichage : les ennemis morts ne disparaissent pas de la carte (sûrement un pb d'observable)
- 27/05/20
 - pb avec la sérialisation des classes (impossible de sérialiser la classe Image de JavaFX)
 - étude d'un moyen pour sauvegarder reste en suspens
 - début de l'écriture des diagrammes de séquence
 - nouvelle implémentation de la gestion d'un ennemi dans l'état étourdi.
 - début de la rédaction du rapport final
 - correction des paths pour les fichiers images du menu
- 28/05/20
 - intégration du cahier des charges au rapport
 - un ennemi étourdi reste passif sur ses cinq prochaines actions (tentatives d'attaques)
- 29/05/20
 - résolution du pb de l'ennemi mort qui ne disparaît pas
 - ajout de la barre d'expérience et adaptation des valeurs numériques du jeu
 - implémentation de la classe ControleEnclume
 - confexion des design pour les deux types d'enclume
 - ajout des diagrammes de classe pour les classes Carte, Case et FenetreEcran dans le rapport
 - rédaction de l'abstract dans le rapport
 - ajout des graphes de scènes dans le rapport
- 30/05/20
 - pb avec la mise à jour de l'inventaire des ennemis
 - début de la présentation orale du projet
 - implémentation de la classe enclume et mise en relation avec le contrôleur de l'enclume (ça marche)
- 31/05/20
 - finalisation du rapport
 - rédaction du fichier README.md

18 Les outils de communication

1. GitHub : Hébergement du code

2. Discord : Plateforme d'échange vocal et écrit
3. Microsoft Teams : Plateforme d'échange vocal et écrit
4. Messenger : Plateforme d'échange écrit
5. GoogleDrive : Partage de fichiers annexes
6. Overleaf : Rédaction du rapport+Cahier des charges
7. Lucidchart : Plateforme pour la conception architecturale
8. starUML : Plateforme pour la conception architecturale
9. Visual Paradigm : Plateforme pour la conception architecturale

19 Répartition des tâches

1. Rédaction des règles du jeu : toute l'équipe
2. Mise en place de l'architecture : toute l'équipe
3. Développement
 - (a) Classe Partie, TypeIssus : 1er développeur - Ilham Laatarsi
 - (b) Package Jeu (abstraction) : 1er développeur - Lilian Naretto
 - (c) Package com, com/res, saveAndLoad (menu+gestion sauvegarde) : 1er développeur - Hassan Yazane
 - (d) Package controle, threadService (contrôle+gestion processus) : 1er développeur - Corentin Brilliant, 2ème développeur - Sabrina Smail
4. Rédaction
 - (a) Cahier des Charges : Ilham Laatarsi, Hassan Yazane
 - (b) Rapport final : toute l'équipe
 - (c) présentation : Hassan Yazane
5. Design
 - (a) Design IHM Menu - Hassan Yazane
 - (b) Design IHM Jeu - Lilian Naretto, Corentin Brilliant
 - (c) Design carte, décors et personnages - Corentin Brilliant

Huitième partie

Configuration et Ajouts personnalisés

20 Ajouter une carte

Les cartes sont stockées dans le répertoire carte, il s'agit de fichiers csv. Le fichier csv est lu par l'application dans la méthode nouvellePartie() de la classe Partie du package Jeu. Il faut donc changer le chemin d'import du fichier csv. Ensuite, le tableau d'entiers du fichier csv sert à créer un tableau statique de String dont il faut modifier les dimensions selon celle du fichier

csv. Le tableau String[][] associe à chaque entier le nom de l'image de l'objet correspondant. Basiquement, les entiers suivants sont référencés :

- 0 :une case vide (on pourra par la suite y mettre un perso)
- 1 :un obstacle, ici un arbre
- 2 :une porte

21 Ajouter un personnage

L'ajout des joueurs et ennemis sur la carte se fait dans la même méthode un peu plus bas à la ligne 203 du fichier, chaque personnage est ajouté en indiquant sa position sur la carte (il faut veillé à mettre les personnages uniquement sur des cases vides sinon ils n'apparaîtront pas). On rentre dans l'ordre :

- le nom, uniquement s'il s'agit du joueur
- le niveau du personnage
- le numéro de la colonne du personnage
- le numéro de sa ligne
- la catégorie puis la race s'il s'agit d'un ennemi

Neuvième partie Conclusion

22 Bilan

Ce projet était particulièrement enrichissant pour nous, il nous a permis de pratiquer Java d'une façon collective entre les membres du groupes, mais également d'une façon individuelle avec les différentes erreurs et problèmes que nous avons rencontrés, nous avons beaucoup appris grâce à nos erreurs. Travailler en groupe, le partage et le respect des idées de chacun était dominant tout au long de ce projet, nous avons réussi à atteindre nos objectifs, mener à bien ce projet , et vous présenter un jeu complet qui marche. Par la suite nous allons vous parler des difficultés auxquelles nous étions exposés mais également les améliorations que nous voulions ajouter avec plus de temps.

23 Difficultés rencontrées

La principale difficulté était à notre avis, le manque de temps, qui était un facteur de stress que nous avons réussi à gérer, mais ce dernier a impacté quelques éléments que nous voulions faire en plus, comme par exemple apprendre la bibliothèque FXGL, qui allait nous permettre de faire un côté graphique plus moderne et réaliste, ainsi améliorer nos connaissances sur la partie IHM de Java.

Hormis le manque de temps, la réalisation de la partie graphique était un challenge pour nous, nous avions une idée assez flou de comment nous allions procéder (déplacements des personnages, animations, gestions des inventaires,etc) , surtout avec le manque de ressources

sur internet pour les interfaces de jeux vidéo (en comparaison avec les interfaces logiciels), mais nous avons réussi à relever ce challenge.

La gestion des imprévus était aussi une source de difficulté supplémentaire, notre jeu comporte beaucoup d'exceptions et d'interactions entre les éléments, tout marchait souvent mais pas comme nous exigeions, ainsi nous avons eu plusieurs modifications du code existant qui étaient nécessaires, mais grâce à nos phases de tests de code au fur et à mesure de notre avancement, nous avons réussi à détecter les erreurs assez rapidement.

24 Améliorations possibles

Voici une liste non-exhaustive des améliorations possibles.

- Crédation d'une IA pour déplacer les ennemis
- Terminer la sérialisation pour pouvoir effectuer des sauvegardes en profondeur
- Optimisation du code

Dixième partie

Abstract

Within the GL2 project, we decided to create an “action and adventure” RPG application named Beat The Zombies, which is one of the most popular video game genres using the object-oriented programming language Java.

The game starts when the player presses the “Nouvelle partie” button, and it ends either when he loses, wins or quits the game.

The player’s ultimate goal is to kill the big boss. But to achieve his goal, he has to fight and get rid of all the other enemies, pick up their keys (and other items such as weapons ..) and open several doors that lead to the most powerful enemy in the game.

Onzième partie

Annexes

Pour apprendre d'avantage sur Java, nous avons suivi quelques vidéos sur Youtube, pluralsight, stackoverflow était d'une grande aide également, ainsi que pour la partie IHM nous avons suivi les videos de Monsieur "Almas Baimagambetov" sur Youtube qui est notamment le créateur de FXGL la bibliothèque jeux vidéos de JavaFX.