15 Allée Claude Forbin
13600 Aix-en-Provence
Téléphone 04 86 09 10
fabienne.murgu@univ-amu.fr

Projet – Jeu JAVA

Greg, mage du temps

Luca BEVILACQUA
Hugo CHALIK
Grégory NAM
Ahmadou MBAYE

Vendredi 13 décembre 2019
Licence 3 Informatique parcours MIAGE
Projet JAVA 2019-2020
V1.7

Gestionnaire de versions

Date	Auteur(s)	Modification(s)	Version	Statut
17/03/2020	Grégory NAM	Création du document Ajout de la description globale	V1.0	Terminé
19/03/2020	Grégory NAM	Ajout de la description conceptuelle	V1.1	Terminé
22/03/2020	Grégory NAM	Ajout de la partie gestion de projet et de la conclusion	V1.2	Terminé
23/03/2020	Grégory NAM	Ajout des annexes	V1.3	Terminé
24/03/2020	Luca BEVILACQUA Grégory NAM	Ajout de la sitographie	V1.4	Terminé
28/03/2020	Hugo CHALIK	Ajout des tests unitaires	V1.5	Terminé
29/03/2020	Grégory NAM	Relecture du document et correction	V1.6	Terminé

Table des matières

Description globale

Scénario

Aucune modification du scénario n'a été effectuée depuis l'idée initiale.

Dans un monde parallèle, deux royaumes ennemis Timekeep (royaume du temps) et Soulsfort (royaume des âmes) sont en guerre. Cette dernière est sur le point de prendre fin. Soulsfort, étant au bord de la défaite, tente de retrancher ses forces dans la capitale. Un groupe de soldats est envoyé en éclaireur par Timekeep pour étudier les défenses de la capitale ennemie.

Le héros est un mage du temps : « Greg, mage du temps ». Lors de cette mission qui devait être une routine, il se fait trahir et se retrouve dans une autre dimension temporelle. Le personnage principal, coincé dans cette dimension, a une heure pour en sortir avant que ce nouveau monde ne s'effondre.

Afin de s'échapper, il doit interagir avec des horloges pour retourner à son époque et ainsi retrouver sa liberté.
Ce dernier a tout intérêt à ne pas échouer, sinon il s'effondrera avec cette dimension et disparaitra du temps et de l'histoire.

Carte

Nous avons décidé de modifier la carte que nous avions initialement modélisé. Deux nouvelles salles ont été ajoutées. Nous trouvions les déplacements trop simples avec 6 salles. Une des salles ajoutées contient l'horloge piège ainsi que deux Personnages Non-Joueurs, un donnant un item, l'autre non (perte de temps pour le joueur. L'autre salle ajoutée est juste une « salle de transition » permettant d'ajouter la salle contenant l'horloge piège (zone rayée).

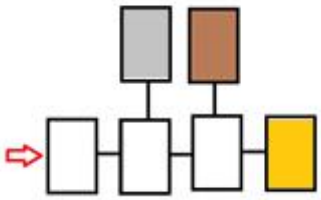

Figure 1 : Carte initiale

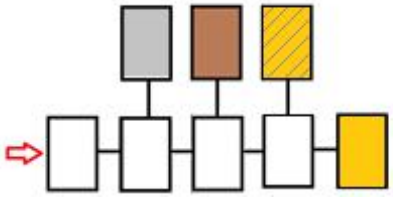

Figure 2 : Carte finale

Zones

Chaque interaction avec une horloge provoque un changement de période. La carte ne change pas, en revanche ce sont les salles du jeu qui subissent des petites modifications.

Figure 3 : Salle 2 en période 1

Figure 4 : Salle 2 en période 2

Figure 5 : Salle 2 en période 3

Personnage joueur et déplacements

Les différentes positions du personnage se trouvent ci-dessous. Les déplacements gauche et droite se font respectivement avec la flèche de gauche et la flèche de droite. Le joueur peut interagir avec plusieurs éléments et notamment avec une porte pour changer de salle en appuyant sur la flèche du haut.

Figure 6 : Déplacement droite

Figure 7 : Déplacement gauche

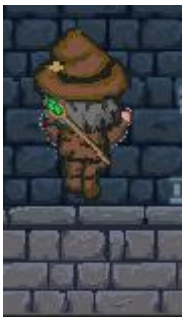

Figure 8 : Déplacement en face

Figure 9 : Arrivée dans une salle par porte murale

Éléments

Les salles sont décorées par des éléments tels que des tables, des cadres, des fenêtres. Ce sont des objets qui n'influencent pas la manière de jouer. Toutefois, les éléments ci-dessous sont des éléments nécessaires à l'avancement du jeu. Ce sont des éléments avec lesquels le joueur peut interagir.

Figure 10 : Horloge de bronze

Figure 11 : Horloge d'argent

Figure 12 : Horloge d'or

Figure 13 : Horloge piège

Figure 14 : Slyce

Figure 15 : Klace

Figure 16 : Zavie

Figure 17 : Carpentier

Figure 18 : Abitbol

Figure 19 : Aiguille de bronze

Figure 20 : Aiguille d'argent

Figure 21 : Aiguille d'or

Figure 22 : Pendule d'or

Conteneurs

Le joueur possède un inventaire dans lequel il lui sera possible de stocker les items qu'il aura ramassé (au maximum 4).

Figure 23 : Inventaire avec deux items

Description conceptuelle de la solution

Afin de réaliser le jeu **Greg, Mage du temps**, nous avons utilisé :

- l'IDE Eclipse.
- Le langage JAVA.
- La bibliothèque JAVA FX.
- La bibliothèque JSON-SIMPLE.

Vues et contrôleurs

Notre jeu est composé de différentes vues gérées par des contrôleurs, les voici :

Figure 24 : Menu du jeu

Figure 25 : Menu du jeu avec souris sur un label

Le **Menu** du jeu est la première vue (*Menu.fxml)* affichée lorsque l'on lance le jeu mais également lorsque l'on appuie sur **Echap** en pleine partie. Elle est composée de cinq **Label** visibles, dont quatre ayant une couleur de fond grise. Il y a également deux **Label** non visibles, le texte explicatif de l'histoire du jeu ainsi que le texte des commandes.
La classe **MenuControleur** qui hérite de la classe **Pane**, est le contrôleur de la vue. C'est dans cette dernière que toutes les interactions avec les éléments graphiques du menu (clique sur un label par exemple) sont gérées.

Figure 26 : Le jeu

La vue du **Jeu** (*LeJeu.fxml*) est seulement composée d'un **Label** qui permet d'afficher des messages durant la partie. L'ensemble des éléments graphiques est ajouté dans la classe **Jeu**, on y retrouve seulement des **ImageView**. On peut dire que la classe **Jeu** fait office de contrôleur mais ce n'est pas vraiment le cas. Nous reviendrons sur la classe **Jeu** plus tard.

Figure 27 : Inventaire avec deux Items

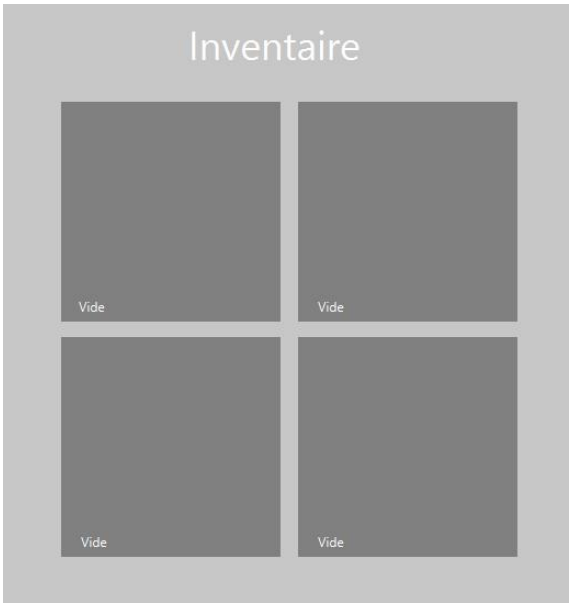

Figure 28 : Inventaire vide

La vue de l'**Inventaire** (*Inventaire.fxml*) sera affichée lorsque le joueur appuiera sur la touche I. Elle est composée de quatre **ImageView** et de cinq **Label**. La classe **InventaireControleur** qui hérite de **Pane** est le contrôleur de la vue. Le contrôleur gère les

interactions avec l'**Inventaire**. L'**Inventaire** (et non pas la vue ou le contrôleur) est géré avec une **ObservableList**. Cette liste est écoutée par le contrôleur qui va ajouter les items dans la vue selon les ajouts et suppressions d'items dans **l'Inventaire**. Il est possible de sélectionner un Item en cliquant dessus.

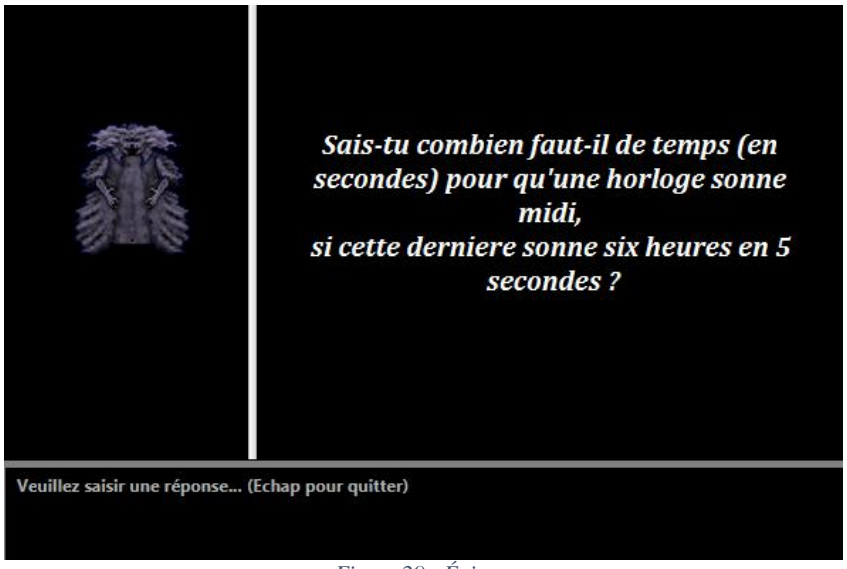

Figure 29 : Énigme

La vue d'une **Enigme** (*Enigme.fxml*) sera affichée lorsque le joueur interagira avec un PNJ. Elle est composée d'un **Label**, d'une **ImageView** ainsi que d'un **TextField.** La classe **EnigmeControleur** qui hérite de **GridPane** est le contrôleur de la vue. Ce dernier va permettre l'affichage de l'**ImageView** (un PNJ), du **Label** (un dialogue du PNJ) ainsi que la gestion du **TextField**.

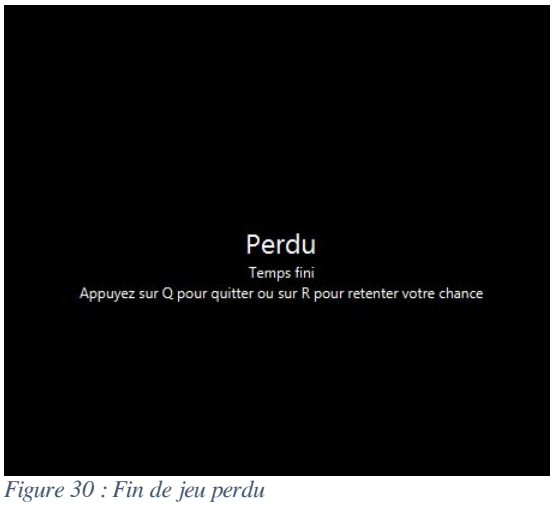

Figure 30 : Fin de jeu perdu

Figure 31 : Fin de jeu victoire

La vue de **la fin du jeu** (*EcranDeFin.fxml*) sera affichée lorsque la partie est terminée. Elle est composée de trois **Label**. La classe **FinControleur** est le contrôleur de la vue. Il permet la gestion du **Label** qui indique si le joueur a gagné ou perdu et du **Label** qui indique la raison de la fin du jeu.

Les énumérations

Énumérations	Éléments	Description
Déplacements	HAUT, DROITE, BAS, GAUCHE	Cette énumération est utilisée pour gérer les déplacements.
Matériaux	BRONZE, ARGENT, OR, PLAQUE_OR	Cette énumération est utilisée par **Horloge** et **Item**.
NomPNJ	KLACE_HEUREOUVERRE, SLYNE, CARPENTER, ABITBOL, ZAVIER_MAIS	Cette énumération est utilisée pour analyser le fichier JSON.
NomSalle	SALLE_DEPART, SALLE_1, SALLE_2, SALLE_3, SALLE_PIEGE, SALLE_OR, SALLE_BRONZE, SALLE_ARGENT	Cette énumération est utilisée pour récupérer les salles dans la **HashMap** *salles* et pour initialiser plus rapidement les salles lors d'un changement de période.
Période	PÉRIODE_1, PÉRIODE_2, PÉRIODE_3, PÉRIODE_OBJECTIF	Cette énumération est utilisée pour gérer le changement de période selon l'activation d'une **Horloge.**
TypeDialogue	QUESTION, BONNE_REPONSE, MAUVAISE_REPONSE, DEJA_REPONDU, REPONSE	Cette énumération est utilisée pour gérer les énigmes et pour analyser le fichier JSON.

Les classes

La classe **Jeu** est la classe qui va faire le lien entre l'interface graphique et les différentes interactions du joueur. C'est pour cela que nous l'avions qualifié plus haut de « contrôleur ». Elle ne gère pas à proprement parler l'interface graphique.
On va y retrouver la gestion des évènements pour le **Jeu** telle que les touches de déplacements, pour les énigmes avec les entrées utilisateurs, etc.
Elle permet également l'initialisation de tous les éléments graphiques.
Pour cette classe, nous avons mis en place le patron de conception **Singleton**, en effet, une seule instance du **Jeu** existe. Nous n'avons pas trouvé de raison à avoir plusieurs instances de cette classe dans le futur.

La classe abstraite **Interactif** est la classe mère de :

- **Personnage**
- **Horloge**
- **Item**
- **PorteExtremite**

Ce sont tous des éléments du **Jeu** avec lesquelles une interaction est possible (méthodes *interagir()*). Ces éléments sont tous représentés par des **ImageView** et comporteront donc également les positions minimum, centre et maximum en ordonnée.

La classe **Personnage** qui hérite donc de la classe **Interactif,** est la classe mère de **PersonnageJoueur** et **PersonnageNonJoueur.** Tous les personnages ont quatre **ImageView** (gauche, droite, haut, bas) et peuvent se déplacer à gauche et à droite. Nous n'avons pas fait déplacer les PNJ dans notre jeu, cependant c'est une possibilité disponible pour de futures améliorations.

Concernant la classe **PersonnageJoueur** nous avons décidé d'implémenter le patron de conception **Singleton**. En effet, notre jeu se jouant seul et avec un seul personnage, un seul **PersonnageJoueur** est donc instancié. Le **PersonnageJoueur** possède donc un **Inventaire** dans lequel se trouve les items qu'il aura récupéré durant la partie.
La classe **PersonnageNonJoueur** possède un ensemble de phrase qu'il peut dire, contenu dans une **HashMap<TypeDialogue**,**String>** et sera affichée dans la vue *Enigme.fxml*.

Une **Horloge** hérite de la classe **Interactif**, possède un tableau d'**Item** dont la taille va dépendre du nombre d'items (passé en paramètre du constructeur) nécessaires afin de l'activer. Une **Horloge** est également composée d'un **Matériaux** qui permet d'accepter ou non un **Item**.
La classe **HorlogePiege** redéfinit simplement la méthode *interagir()* en faisant terminer la partie.

Un **Item** hérite de la classe **Interactif**, possède un nom et un **Matériaux** qui permet la comparaison avec le **Matériaux** d'une horloge. Il va permettre d'activer une horloge.

Une **PorteExtremite** hérite de la classe **Interactif** possède un tableau de **Salle** qui va représenter les salles qu'elle lie. C'est le seul élément du jeu qui possède une **ImageView** égal à *null* car nous ne voulions pas afficher d'image pour ce type de porte.
Une **PorteMurale** hérite de la classe **PorteExtremite**, et possède, en revanche, deux **ImageView** : une porte fermée et une porte ouverte.

Une **Salle** possède une **ArrayList** d'**Interactif**. Des **Interactif** peuvent être ajoutés et supprimés.
Un **Inventaire** possède une **ObservableList** d'**Item** qui sera écoutée par le contrôleur (cf. vues et contrôleurs). L'**Inventaire** a une capacité de quatre **Item.**

Les classes utilitaires

La classe **CompteARebours** permet de créer le minuteur du jeu. Lorsque le minuteur arrive à 0 le jeu est terminé.

C'est dans la classe **AnalyseFichierEnigmeUtil** que nous avons utilisés la bibliothèque JSON SIMPLE.
Nous avons un fichier JSON qui répertorie chaque texte pour tous les types de dialogues pour chaque **PersonnageNonJoueur**. Cette classe qui ne peut être instanciée puisque son constructeur est privé, possède une méthode publique statique pour initialiser le dialogue pour un **PersonnageNonJoueur** et un **TypeDeDialogue**.

Les classe **FinisseurDeJeu** et **AnalyseFichierEnigmeUtil** ne peuvent être instanciées car leurs constructeurs sont privés. Ces deux classes ont, chacune, une méthode publique.

Concernant la solution avec le fichier texte contenant la suite de touche à effectuer, nous ne la trouvions pas adapté à notre jeu. Nous avons adopté une autre solution, qui consiste à faire déplacer le personnage d'un point A à un point B et de faire une interaction avec un **PersonnageJoueur**, une **Porte** ou une **Horloge**.

Les classes de test

Lors du développement des premiers tests unitaires sous JUnit 5, nous avons rencontré un problème nous empêchant d'effectuer ces derniers sur les classes des packages « elements » et « personnages ». En effet, lors du lancement d'un test JUnit, nous avions un problème d'initialisation de l'environnement graphique.
La première piste de solution trouvée fut d'utiliser la librairie TestFX permettant de faire des tests unitaires sous environnement graphique. Cette solution s'avérait trop compliquée à mettre en place car il fallait intégrer au projet le gestionnaire de dépendance Maven. Nous avons donc opté pour une solution plus simple (permettant de rester sous JUnit 5) consistant à créer une initialisation « virtuelle » de l'environnement graphique JavaFX, via la création d'une classe « **AppDeTest** ». Cette solution implique donc d'initialiser, avant tout test (via l'annotation @BeforeAll), un thread JavaFX.
De plus, nous avons eu affaire à un problème, insolvable cette fois, lors des tests des méthodes interagir() des objets interactifs. En effet, le Jeu non lancé, n'ayant pas de salle courante, certaines méthodes interagir() (ainsi que les méthodes getXMin() et getXMax() de la classe **PorteExtremite**) influent sur la suppression d'objets interactif de la salle courante ainsi que sur le changement de salle courante. Or, dans ce dernier cas, la méthode setSalleCourante() supprime le personnage joueur, qui est lui-même un objet interactif, dans une salle courante n'existant pas. La solution trouvée fut de tester certaines parties des méthodes interagir() n'impliquant pas le changement ou la récupération de salle courante.
Enfin, le dernier problème rencontré fut la comparaison d'**ImageView**, afin de tester les méthodes renvoyant des ImageView. Ce dernier fut résolu grâce à la création d'une fonction booléenne, dans la classe AppDeTest, nommée compareImages(), permettant de comparer pixel par pixel deux images passées en paramètre.

Gestion de projet

Répartition des taches

Afin de savoir ce qui était à faire dans le projet et par qui, nous avons utilisé l'outil Trello. De cette manière, nous avions une visualisation claire des tâches à effectuer, celles qui étaient en cours et celle qui étaient terminées.

À chaque réunion, nous faisions un point sur notre tableau Trello, afin d'établir les nouvelles tâches à effectuer, voir pourquoi certaines tâches étaient encore en cours.

Figure 32 : Liste des tâches à faire

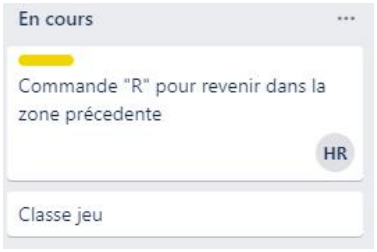

Figure 33 : Liste des tâches en cours

Figure 34 : Liste des tâches terminées

On peut voir par exemple dans les tâches en cours une barre orange : elle signifie que cette tâche n'est plus à faire. Nous avons fait le choix de pas créer cette commande car elle n'avait pas de sens dans notre jeu.

Gestionnaire de version

Afin de pouvoir travailler en équipe et de gérer la version de notre jeu nous avons utilisé GitHub. Nous avons décidé de travailler avec deux branches : la branche principale et la branche de développement. Toutes les modifications réalisées concernant le projet étaient effectuées sur la branche de développement. À chaque réunion que l'on faisait, une revue du code existant était réalisée et si toute l'équipe validait le code nous fusionnons la branche de développement sur la branche principale.

Le but de cette manipulation étant de garder une version stable du projet afin de pouvoir revenir dessus si jamais nous avions un problème.

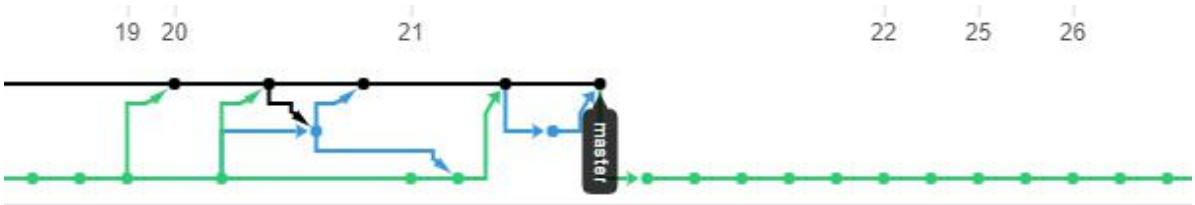

Figure 35 : État des deux branches durant le projet

Nous pouvons constater que nous avons été très assidus tout au début du projet concernant les fusions sur la branche principale. Beaucoup moins sur la suite. Si nous avions a amélioré notre gestion de version du projet, nous aurions fait des réunions bien plus fréquemment, de même pour les fusions sur la branche principale.

Conclusion

Ce projet nous a permis de travailler sur deux aspects très importants dans notre cursus : la gestion de projet et la programmation.
Nous avons pu travailler en équipe en mettant en place une méthode de gestion de projet avec un système de gestionnaire de version ainsi qu'un outil de visualisation des tâches.
Concernant la programmation, nous avons pu développer nos compétences en conception car nous avons dû réfléchir sur la manière dont notre jeu allait fonctionner et comment nous allions le mettre en place. Nous avons rencontré certains problèmes que nous avons réussi à résoudre après de nombreuses discussions en équipe.
Réaliser ce projet a également permis à l'ensemble de l'équipe d'améliorer notre capacité de recherche, notamment pour utiliser les bibliothèques JAVA FX et JSON SIMPLE.
Nous sommes heureux et fiers de notre réalisation et des connaissances acquises.

Annexes

Certaines annexes nécessiteront de zoomer afin de bien distinguer tous les éléments.

Diagramme UML global d'origine

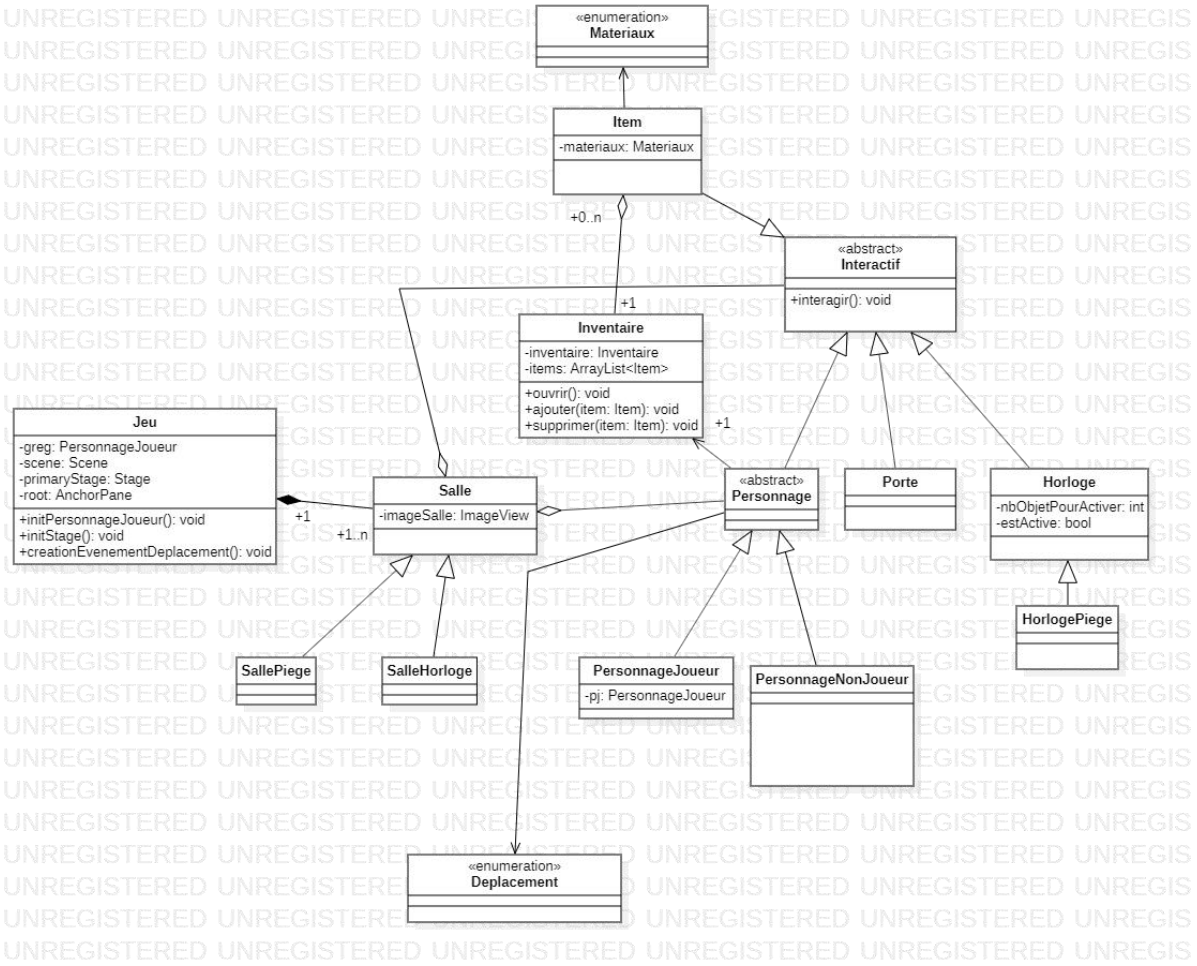

Figure 36 : Diagramme pensé avant de coder

Diagramme UML final

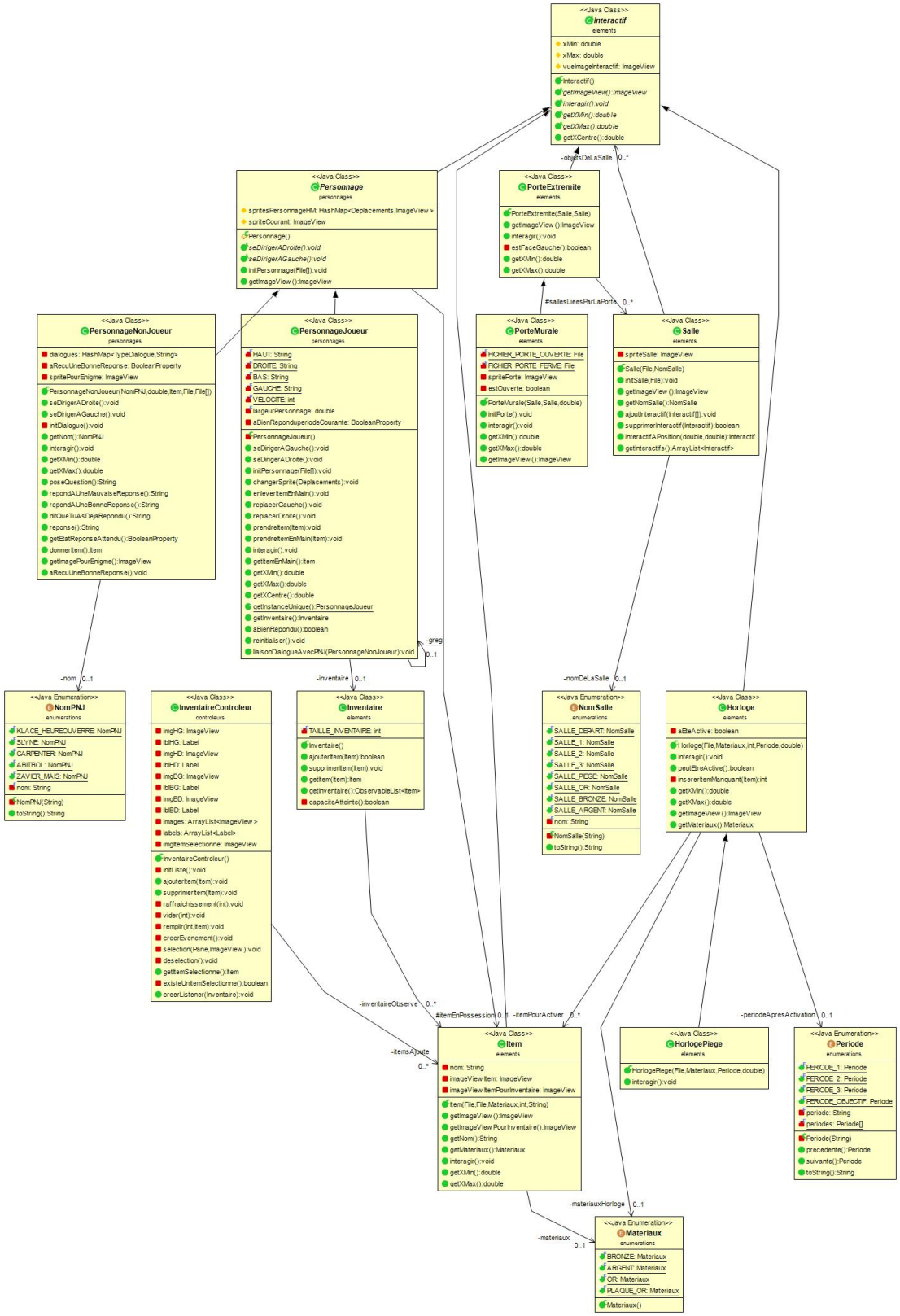

Figure 37 : Diagramme UML final

Nous retrouvons dans ce diagramme l'ensemble des classes ayant un lien entre elles. Pour des raisons de visibilité nous avons choisis de ne pas mettre certaines classes qui ne changeaient pas la compréhension de la conception. Ces classes sont tous les contrôleurs et la classe Jeu.

Sitographie

2d Sprite Ghost PNG Image. (s. d.). Consulté le 16 février 2020, à l'adresse https://www.seekpng.com/ipng/u2w7i1t4i1o0o0e6_click-to-view-full-size-2d-sprite-ghost/

3d female sprite sheet png. (s. d.). Consulté le 12 janvier 2020, à l'adresse https://ya-webdesign.com/image/3d-female-sprite-sheet-png/850310.html

Dreamstime. (s. d.). Vector pixel art weapon spear. Isolated. Consulté le 16 février 2020, à l'adresse https://www.dreamstime.com/illustration/pixel-art-icon-spear.html

Eklund, S. (2014, août 6). 2D Level Game. Consulté le 16 février 2020, à l'adresse https://blog.indiumgames.fi/2014/08/06/creating-level-2d-game/

Hylian Shield - Zelda, not a pattern yet, but easy enough to convert | Pixel art. (s. d.). Consulté le 12 janvier 2020, à l'adresse https://www.pinterest.fr/pin/71705819043802061/

M. (s. d.). Set of weapon icons in perfect pixel art style. Sword, knife, dagger,... Consulté le 16 février 2020, à l'adresse https://www.istockphoto.com/fr/vectoriel/ensemble-dic%C3%B4nes-darmes-dans-le-style-pixel-gm921312570-253041534

Pixel swords. (s. d.). Consulté le 16 fevrier 2020, à l'adresse https://www.dreamstime.com/illustration/pixel-swords.html

Sprite Sheet. (s. d.). Consulté le 16 février 2020, à l'adresse https://forums.rpgmakerweb.com/index.php?threads/need-help-identifying-sprite-sheet.93574/

Wizard Sprite by Rienquish on. (s. d.). Consulté le 12 janvier 2020, à l'adresse https://www.deviantart.com/rienquish/art/Wizard-Sprite-593011638

Google Code Archive - Long-term storage for Google Code Project Hosting. (s. d.). Consulté le 20 février 2020, à l'adresse https://code.google.com/archive/p/json-simple/

JavaFX. (s. d.). Consulté le 1 janvier 2020, à l'adresse https://openjfx.io/