



UNIVERSITÉ
DE MONTPELLIER



RAPPORT FINAL DE PROJET

Cube Escape - Jeu d'Énigme Futuriste Engagé



Réalisé par :

Eliès BILLOTTA
Adèle LAURENS
Thi-Christine NGUYEN
Léa SERRANO

Sous la direction de :

Antoine CHOLLET
Année Universitaire 2020 - 2021

Nous remercions M. Belmecheri qui nous a beaucoup orientés pour l'organisation du projet. Nous remercions aussi M. Chollet, notre tuteur de projet qui nous a guidé tout le long du projet.

Sommaire

| | |
|---|-----------|
| Introduction | 6 |
| 1. Cahier des charges (ou « Analyse ») | 7 |
| 1.1. Analyse du sujet et de son contexte | 7 |
| 1.1.1. Analyse de l'existant | 7 |
| Analyse générale | 7 |
| Analyse des cas déjà présents | 7 |
| 1.1.2. Analyse de l'environnement | 10 |
| 1.2. Analyse des besoins fonctionnels (besoins utilisateurs) | 10 |
| 1.2.1. Spécifications fonctionnelles | 10 |
| 1.3. Analyse des besoins non-fonctionnels | 13 |
| 1.3.1. Spécifications techniques | 13 |
| 2. Rapport technique | 14 |
| 2.1. Conception | 14 |
| 2.1.1. Présentation et justification des choix technologiques | 14 |
| 2.1.2. Description des algorithmes | 14 |
| 2.2. Réalisation | 19 |
| 2.2.1. Description de l'architecture du programme | 19 |
| 2.2.2. Organisation des fonctions dans le projet | 20 |
| 2.2.3. Arborescence des fichiers du programme | 20 |
| 3. Résultats | 21 |
| 3.1. La trame scénaristique | 21 |
| 3.2. Installation | 22 |
| 3.3. Manuel d'utilisation | 22 |
| 3.4. Tests/Validation | 22 |
| 3.5. Améliorations possibles | 24 |
| 4. Rapport d'activité | 25 |
| 4.1. Planification et organisation du travail : recul sur le travail effectué | 25 |
| 4.1.1. Découpe et organisation du travail | 25 |
| 4.1.2. Organisation des sessions de travail | 27 |
| Conclusion | 28 |
| Bibliographie | 29 |
| Annexes techniques | 30 |
| 1. Analyse de l'existant | 30 |
| 2. Déroulement du jeu (solutions) | 37 |
| 3. Code l'inventaire | 39 |
| 4. Compte rendus des rendez-vous avec notre tuteur | 40 |
| Résumé | 43 |

Table des figures

| | |
|--|----|
| Figure 1 - Diagramme de cas d'utilisation..... | 10 |
| Figure 2 - Diagramme de classes..... | 11 |
| Figure 3 - Code readIn() de la classe Parser..... | 14 |
| Figure 4 - code writeIn() de la classe Parser..... | 14 |
| Figure 5 - code resetFile() de la classe Parser..... | 15 |
| Figure 6 - classe Inventaire..... | 16 |
| Figure 7 - classe taquin : initialisation du taquin..... | 17 |
| Figure 8 - classe taquin : affichage du taquin..... | 17 |
| Figure 9 - controller scene panneaux : attribution de la fonction aux boutons..... | 17 |
| Figure 10 - controller scene panneaux : gestion du taquin fini | 18 |
| Figure 11 - organisation des packages sur intellij Idea..... | 19 |
| Figure 12 - Product Backlog sur Trello..... | 23 |
| Figure 13 - Description d'une Story..... | 24 |
| Figure 14 - Branches sur git..... | 24 |
| Figure 15 - Organisation de notre serveur Discord..... | 25 |

Glossaire

Les termes définis dans ce glossaire sont identifiables dans le corps du texte au moyen d'un astérisque (*).

- **ArrayList** : tableau de données dynamiques dont la taille peut être modifiée selon les besoins.
- **Branches (Git)** : Fonctionnalité permettant à chaque membre du projet de travailler de manière indépendante de la branche principale sur laquelle est située le code global du projet.
- **Creepy** : En français, effrayant. Utilisé pour décrire un jeu, événement... de glauque, qui fait peur.
- **Clone (Git)** : Permet de faire une copie locale d'un projet distant.
- **Conflit (Git)** : se produit lorsqu'un merge ne se passe pas correctement. Des modifications sont donc à apporter sur la branche qui ne peut pas être merge.
- **Controller (JavaFX)** : Permet de faire le lien entre une scène JavaFX et du code Java. Notamment, sert à définir les actions sur les boutons et/ou à modifier et/ou reconstruire la scène à laquelle il est lié.
- **Escape Game** : En français, jeu d'évasion. Type de jeu dans lequel le joueur est placé dans un lieu, généralement fermé, et doit s'y échapper en résolvant des énigmes.
- **Gameplay** : les caractéristiques d'un jeu vidéo que sont l'intrigue et la façon dont on y joue. (Wikipédia)
- **GridPane** : Objet de la bibliothèque JavaFX. Grossièrement, un tableau contenant divers objets JavaFX.
- **Merge (Git)** : Fonctionnalité permettant de fusionner les branches de chaque membre de l'équipe et de mettre en commun le code produit par chacun.
- **Point and click** : type de jeu dans lequel toutes les actions se font grâce au clic : avec la souris (sur ordinateur) ou avec le doigt (sur mobile).
- **Product Backlog** : Liste des diverses fonctionnalités à développer, chaque fonctionnalité ayant un ordre d'importance définissant sa priorité de développement.
- **Pull (Git)** : Fonctionnalité permettant de "tirer" le code global vers la branche de la machine courante sur laquelle on se situe.
- **SceneBuilder** : Logiciel permettant de construire graphiquement des scènes de la librairie JavaFX et déposant les divers éléments dans l'interface.

- **Test Unitaire** : Classe java liée à une autre, permettant de tester les diverses fonctions de la classe à laquelle elle est liée. Elle permet une meilleure stratégie de développement.
- **VBox** : Objet de la bibliothèque JavaFX. Grossièrement, un tableau dont l'affichage est vertical et qui contient divers objets JavaFX.

Introduction

De nos jours, les jeux sont de plus en plus présents dans notre quotidien. Ils ont pendant longtemps été surtout une source de distraction. En revanche, ils sont maintenant aussi utilisés pour sensibiliser, éduquer et faire passer des messages. Notre projet doit sensibiliser sur un sujet important, grâce à un escape game(*), tout en s'amusant à résoudre des énigmes.

Notre jeu aura la particularité d'avoir un contexte précis composé d'un univers futuriste et engagé, afin de sensibiliser les utilisateurs. Notre scénario et nos énigmes seront en rapport avec la place que pourraient occuper les robots dans le monde de demain. Nous allons donc utiliser des mécaniques de gameplay(*) poussées (ex: système d'inventaire) afin de créer une application mobile.

Notre projet est un jeu d'évasion, à l'image de la série *Cube Escape* (série d'escape game(*) composée de nombreuses énigmes et dotée d'histoires mystérieuses). L'objectif du joueur est de s'échapper d'une pièce à l'aide de la résolution d'énigmes. Contrairement aux Escape Game(*) classiques, notre jeu aura la particularité d'avoir une histoire avec un thème et une problématique précise. Nous avons choisi de traiter le sujet de la robotique, de la relation de confiance entre humains et machines dans un futur proche de plus en plus automatisé.

Nous allons tout d'abord vous présenter notre cahier des charges comprenant les différentes analyses. Puis nous développerons les aspects techniques de notre projet. Nous analyserons également le résultat final de notre projet (fonctionnalité, ergonomie...). Pour finir, nous reviendrons sur la façon dont notre équipe s'est organisée tout au long du projet.

1. Cahier des charges (ou « Analyse »)

1.1. Analyse du sujet et de son contexte

Les Escape Game(*) sont des jeux de plus en plus populaires aujourd'hui et sous différentes formes ; jeux de société, simulation réelle... Le concept trouve pourtant naissance dans le domaine du jeu vidéo.

Notre travail consistait à réaliser une analyse de la plupart des jeux vidéo Escape Game(*). Cette analyse offre ainsi une étude de quelques jeux ayant percés sur le marché du jeu vidéoludique, et se base sur les fonctionnalités simples qui contribuent à l'ambiance du jeu et sur les mécaniques de gameplay(*) qui assurent le confort du joueur.

Le but ici est de pouvoir proposer, dans le futur, une solution informatique efficace et optimisée, menant à terme à la création de notre propre jeu Escape Game(*).

1.1.1. Analyse de l'existant

○ Analyse générale

Pour notre analyse de jeux d'Escape Game(*), nous avons choisi d'étudier trois jeux populaires sur le marché : Cube Escape, une série de jeux mobile orientés horreur, The Stanley Parable et Professeur Layton. Bien que le dernier jeu ne soit pas réellement un escape game(*), nous avons jugé qu'il s'intégrait parfaitement dans notre corpus par ses mécaniques de gameplay(*).

En effet, tous ces jeux ont pour fonctionnalité principale de confronter le joueur à des énigmes afin de le faire progresser dans l'aventure qu'il mène.

Cependant, les jeux vont se différencier sur de nombreux points : l'ambiance, les contrôles du joueur, l'histoire, le style graphique, la difficulté des énigmes... Ces différences vont donc créer des points forts à chaque jeu, mais également des points faibles, ce qui sera le sujet de cette dernière partie d'analyse.

○ Analyse des cas déjà présents

L'analyse des différents jeux est détaillée dans les annexes (annexes 1 à 3). Nous proposons une synthèse de ces analyses dans le tableau suivant.

Tableau de comparaison de plusieurs jeux d'énigmes

| | Cube Escape | The Stanley Parable | Professeur Layton |
|---------------|--|--|--|
| Mise en scène | Petite histoire au début pour se mettre dans l'ambiance creepy(*) (+ musique) | Lors de notre arrivée dans le jeu, une cinématique nous explique l'histoire principale. On a une narration tout au long de l'histoire avec une voix-off qui commente ce qui se passe. | Histoire qui présente les personnages ainsi que l'univers du jeu. Mise en place de la trame principale et de l'objectif à atteindre. |
| Graphismes | <ul style="list-style-type: none"> - 4 murs + 1 plafond - Flèches pour aller à droite, gauche et en haut - Inventaire à droite soit déroulant (de haut en bas), soit "s'ouvrant" (de droite à gauche) - Nom de l'objet affiché quand on clique dessus - Déplacer les objets dans l'inventaire - Animation quand on clique quelque part | <ul style="list-style-type: none"> - On a plusieurs pièces qui sont de tailles différentes et en 3D | <ul style="list-style-type: none"> - Images en 2D - Couleurs chaleureuses dans la journée et froides le soir. - Beaucoup d'animations - Éléments du décor interactif - Flèches pour se déplacer |
| Son | <ul style="list-style-type: none"> - Petite musique de fond pas forte du tout - Bruits quand on touche à certains objets, quand on les ramasse | <ul style="list-style-type: none"> - On peut entendre des bruits de pas, des bruits de porte - On a la voix-off que l'on entend pendant tout le jeu. La voix-off est vraiment bien faite, le narrateur est drôle et ajoute | <ul style="list-style-type: none"> - Petite musique de fond selon les situations. - Des dialogues doublés par des acteurs - Bruitages divers |

| | | | |
|-------------------|--|--|--|
| | | de la musique parfois. | |
| Enigmes proposées | <ul style="list-style-type: none"> - Images à trouver et à reconstituer pour avoir des objets (ex: clefs) - Mots croisés à réaliser en cherchant des indices - Numéro de téléphone ou vidéo - Petites animations permettant l'accès à de nouvelles énigmes | <ul style="list-style-type: none"> - Tout le long du jeu, une seule énigme est proposée : c'est trouver une réponse à notre problème. Pour ce faire, le gameplay(*) se base uniquement sur l'exploration du bureau dans lequel on se trouve. - Le jeu comporte plusieurs fins | <ul style="list-style-type: none"> - Tout le long du jeu, une seule énigme est proposée : c'est trouver une réponse à notre problème. Pour ce faire le gameplay(*) se base uniquement sur l'exploration du bureau dans lequel on se trouve. - Le jeu comporte plusieurs fins qui sont liées aux endroits où l'on va. |
| Contrôles | <ul style="list-style-type: none"> - Pour se déplacer, on utilise des flèches qui sont implémentées dans le jeu | <ul style="list-style-type: none"> - Pour se déplacer, on utilise les touches directionnelles - La souris nous permet de pivoter - Le bouton gauche de la souris peut nous permettre d'appuyer sur des boutons - La touche Echap nous permet de quitter le jeu | |

Grâce à cette analyse, nous avons maintenant les informations nécessaires afin de créer notre propre escape game(*). Nous pouvons remarquer des éléments répétitifs entre les trois jeux que nous avons analysé tels que la présence d'une histoire, une grande variété d'énigmes et de sons. Malgré des ressemblances entre chaque jeu, il est important de noter également l'originalité de chacun que cela soit au niveau de la trame scénaristique ou des énigmes.

Nous avons pris connaissance de ce qui était attendu de nous pour notre projet S3, nous allons donc devoir prendre en considération tous les éléments observés pour pouvoir faire un escape game(*) comportant ces fonctionnalités (déplacement, système d'inventaire...) et y ajouter notre propre histoire.

1.1.2. Analyse de l'environnement

Notre jeu est jouable sur ordinateur, il est donc important de prendre en compte les caractéristiques propres à celui-ci, telles que le clavier, l'utilisation de la souris.... Le joueur n'aura pas besoin d'un ordinateur "puissant", pour profiter pleinement du jeu, il est préférable de jouer dans un environnement calme. Le jeu se joue seul mais il est libre à l'utilisateur de demander de l'aide extérieure.

Le jeu étant tout public, il se doit de respecter certaines règles comme de ne pas diffuser certaines scènes trop violentes ou interdites aux moins de 18 ans. Nous proposons un jeu pour tout âge. Il est donc important de faire attention à ces règles sur la violence dans notre jeu.

On peut dire qu'il existe une limitation interne puisque les énigmes peuvent être difficiles ou d'une logique différente à celle des enfants.

1.2. Analyse des besoins fonctionnels (besoins utilisateurs)

1.2.1. Spécifications fonctionnelles

Notre projet doit permettre d'aboutir à un escape game(*) dans le même style que le *Cube-Escape*. Nous aurons donc à créer un jeu de type *Point and click*(*).

Il va devoir, à la fin, comporter, au moins 4 pièces et avoir quelques caractéristiques:

- Le joueur doit pouvoir se déplacer entre chaque pièce et pouvoir interagir avec les objets de la pièce.
- Si l'objet sur lequel le joueur clique sur un objet qui contient une énigme, l'énigme doit s'afficher.
- Le joueur doit pouvoir avoir la possibilité de résoudre l'énigme et d'obtenir les récompenses en cas de réussite.
- Si le joueur clique sur un objet récupérable, l'objet doit venir se mettre dans son inventaire.
- Lorsque le joueur change de scène, il doit pouvoir garder la sauvegarde de ce qu'il a fait sur la scène d'avant (les énigmes résolues doivent le rester et l'inventaire doit rester intacte).

Nous avons retranscrit tous ces éléments sur un diagramme de cas d'utilisation :

Nous avons aussi retranscrit nos besoins dans un diagramme de classe :

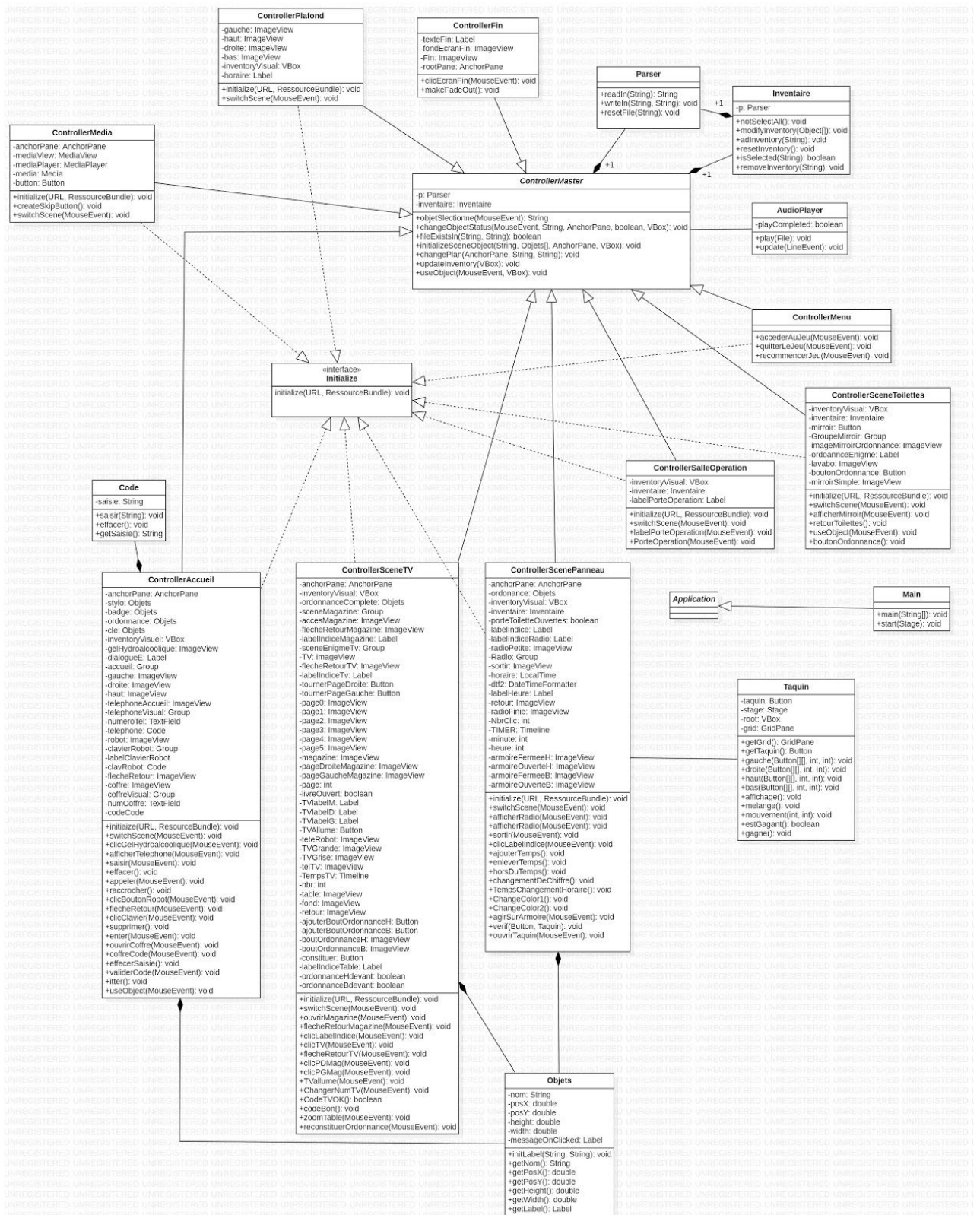


Figure 2 - Diagramme de classes

Le joueur va aussi interagir avec les objets afin d'obtenir des indices et/ou de résoudre des énigmes. Il y a aussi plusieurs énigmes (visuelles, textuelles, sonores) de types : des énigmes avec codage, avec manipulation, avec observation, avec jeux de logique.

1.3. Analyse des besoins non-fonctionnels

1.3.1. Spécifications techniques

Nous n'avons pas eu de contrainte (de langage, de logiciel) de la part de notre tuteur de projet. Cependant, il existe tout de même quelques contraintes selon nos choix, tels que le choix du langage de programmation et des contraintes visuelles.

Soit nous utilisons Unity, un logiciel qui possède une manipulation de l'interface graphique spécialisée pour les jeux vidéo et qui nous permet de transformer le jeu en application mobile très facilement. L'inconvénient était que nous ne maîtrisons pas le langage C# qui est le langage utilisé sur Unity.

Soit nous utilisons le langage Java que nous avons l'habitude d'utiliser et qui possède une bibliothèque graphique, JavaFX, mais qui n'est pas spécialisée pour la conception de gros jeux vidéo. L'avantage premier de cette solution réside dans le fait que nous avons déjà réalisé un escape game(*) avec JavaFX l'année dernière.

Après discussion avec tous les membres de l'équipe, en prenant en compte notre limite de temps, le choix s'est porté sur le langage de Java et de l'utilisation de la bibliothèque JavaFX que nous avons choisi d'utiliser sur le logiciel IntelliJ Idea.

2. Rapport technique

2.1. Conception

2.1.1. Présentation et justification des choix technologiques

Pour le développement de notre projet, de nombreuses hésitations ont vu le jour. En effet, le codage de l'Escape Game(*) convenait à deux technologies différentes ; l'utilisation de la bibliothèque JavaFX ou de moteurs de jeu classiques, notamment Unity.

Finalement, après une semaine de débat, le choix s'est porté sur JavaFX. En effet, en choisissant Unity, le code du jeu aurait été réalisé en C# mais aurait nécessité une plus grande couche de manipulation de logiciel que de codage pur. Ayant déjà des connaissances en programmation Java, il paraissait plus judicieux d'utiliser ce langage afin de bien exploiter les diverses notions travaillées lors de nos années à l'IUT, chose impossible avec l'utilisation d'Unity.

2.1.2. Description des algorithmes

Notre jeu étant assez simpliste en termes de mécanique, il n'y a pas eu à coder des algorithmes très complexes. Cependant, de nombreux problèmes ont vu le jour, notamment à cause de notre manque d'expertise du JavaFX.

- **La sauvegarde**

Un principal défi concernait la sauvegarde des objets entre les scènes. Par exemple, si le joueur se situait dans une scène A et se dirigeait vers une scène B, il perdait l'objet mis en inventaire. S'il revenait dans la scène A, les objets pris étaient à nouveau dans la pièce. Il a donc fallu réfléchir à cette problématique pour tenter de la résoudre.

Après plusieurs pistes explorées, une solution s'est finalement dégagée grâce à l'aide de M. Belmecheri que nous remercions encore. Cette solution concernait la mise en place d'un système de fichiers afin de sauvegarder tous les états d'une scène. Chaque scène se reconstruirait donc en fonction de ce que le fichier indiquerait, c'est-à-dire quel objet le joueur a pris, quelle action il a effectuée...

Pour ce faire, la classe Parser a été créée. Cette classe permet de faire la transition entre les éléments graphiques du jeu, régis notamment par la bibliothèque JavaFX, et les fichiers contenant les données de l'application. Après plusieurs tests et refactorisations du code, la classe contient trois fonctions principales ;

D'abord, la fonction *readIn(String nomFichier)*. Elle permet de lire le contenu du fichier dont le nom est passé en paramètre, sous la forme d'une chaîne de caractères. Si le nom du fichier est erroné, on renvoie une erreur indiquant que le fichier n'a pas été trouvé. En plus de lire le contenu du fichier, elle le renvoie sous la forme d'un tableau de chaînes de caractères ce qui, par la suite, va nous servir à

réécrire le contenu du fichier. En quelque sorte, cette fonction est utilisée pour sauvegarder le contenu du fichier à un instant précis dans un tableau.

```
//Renvoie un tableau de String, chaque String du tableau correspondant à une ligne (= un objet de la scène) du fichier.
//nomFichier : nom du fichier de la scène
public String[] readIn(String nomFichier){
    File file = new File( pathname: "src/Stages/Fichiers/"+nomFichier);
    StringBuilder s = new StringBuilder();
    try {
        FileReader fr = new FileReader(file);
        LineNumberReader lnr = new LineNumberReader(fr);
        String line = lnr.readLine();
        if(line == null) return null;
        else{
            while (line != null){
                s.append(line).append(" ");
                line = lnr.readLine();
            }
            return s.toString().split( regex: " ");
        }
    } catch (IOException e) {
        System.out.println("File not found");
        return null;
    }
}
```

Figure 3 - Code readIn() de la classe Parser

Ensuite, la classe Parser contient la fonction *writeIn(String nomFichier, String nomObjet)*. Cette fonction permet de modifier le contenu du fichier nomFichier en ajoutant un # devant le nomObjet dans le fichier, symbolisant le fait que le joueur a ramassé l'objet nomObjet. Ainsi, lors de la recreation de la scène, lorsque le système va lire le contenu du fichier, si un # se trouve devant un nom d'objet, il ne va pas rajouter l'objet en question à la scène.

```
//écrit dans le fichier de nom nomFichier que l'objet de nom nomObjet a été pris.
//nomFichier : nom du fichier de sauvegarde de la scène
//nomObjet : nom de l'objet qui a été pris dans la scène
public void writeIn(String nomFichier, String nomObjet){
    try{
        String[] strings = readIn(nomFichier);
        File fichier = new File( pathname: "src/Stages/Fichiers/"+nomFichier);
        FileWriter fw = new FileWriter(fichier);
        for (int i = 0 ; i<strings.length ; i++){
            if(strings[i].equals(nomObjet)){
                strings[i] = "#"+strings[i];
            }
            if (i == strings.length-1) fw.write(strings[i]);
            else fw.write( str: strings[i]+"\\n");
        }
        fw.close();
    } catch (IOException e) {
        System.out.println("Fichier non trouvé");
    }
}
```

Figure 4 - code writeIn() de la classe Parser

Enfin, une dernière fonction est *resetFile(String fileName)*. Comme son nom l'indique, elle permet de réinitialiser les fichiers dans leur état de base, ce qui signifie enlever les # devant chaque nom d'objet et donc de remettre tous les objets dans la scène lors de sa création.

```
//Réinitialise le fichier dans son état de base. Enlève tous les # devant les noms de chaque objets d'un fichier.
public void resetFile(String fileName){
    try {
        String[] fileContent = readIn(fileName);
        FileWriter fw = new FileWriter(new File( pathname: "src/Stages/Fichiers/"+fileName));
        for (int i = 0 ; i<fileContent.length ; i++) {
            if (fileContent[i].charAt(0) == '#') {
                fileContent[i] = fileContent[i].replace( target: "#", replacement: "");
            }
            if (i == fileContent.length-1) fw.write(fileContent[i]);
            else fw.write( str: fileContent[i]+"\\n");
        }
        fw.close();
    } catch (IOException e) {
        System.out.println("Fichier non trouvé");
    }
}
```

Figure 5 - code resetFile() de la classe Parser

La classe *Parser* est donc très importante car, sans elle, notre jeu n'aurait pas pu être jouable. Cette classe a par ailleurs un effet double : en modifiant les fichiers, elle permet la sauvegarde entre les scènes, comme expliqué plus haut, mais elle permet également de créer des sauvegardes du jeu, au vrai sens du terme.

Ainsi, si le joueur quitte l'application et la rouvre, il retrouvera le jeu dans l'état dans lequel il l'a laissé.

- L'inventaire

Une autre fonctionnalité phare du jeu est le système d'inventaire. Dans *GameCube - The Hospital*, le joueur a la possibilité de prendre des objets et les stocker dans un inventaire, c'est-à-dire, un tableau dans lequel il peut garder les objets et les réutiliser au cours de la partie.

Comme pour le système de sauvegarde, la conception de l'inventaire a été faite en plusieurs temps et de différentes manières : d'abord en utilisant des tableaux et ensuite en utilisant le système de fichiers mis en place. Ainsi, tout comme les scènes possèdent leur propre fichier de données, l'inventaire du jeu n'est ni un tableau, ni une arraylist(*) mais un fichier dans lequel sont stockés les noms des objets que possède le joueur. Le choix d'utiliser un fichier plutôt qu'un tableau s'est fait car l'inventaire devait se conserver entre les différentes scènes du jeu, ce qui aurait été impossible à implémenter avec un tableau, ou du moins beaucoup trop compliqué dans le temps imparti. Le fichier permet donc de stocker, de manière permanente et jusqu'à la réinitialisation du jeu, le contenu de l'inventaire du joueur.

```

public class Inventaire {

    Parser p = new Parser();

    public Inventaire() {}

    public void notSelectAll() throws IOException {...}

    public void modifyInventory(Object[] s) throws IOException {...}

    public void addInventory(String id) throws IOException {...}

    public void resetInventory() throws IOException {...}

    public boolean isSelected(String objectName) {...}

    public void removeInventory(String objectName) throws IOException {...}

}

```

Figure 6 - classe Inventaire

Il serait trop long de détailler chaque fonction de la classe *Inventaire*. Vous pourrez le retrouver en détail dans la partie Annexe Technique, annexe 6. Dans les grandes lignes, la classe est divisée en six parties, chacune d'entre elles représente une fonctionnalité.

La fonction *addInventory(String id)* permet d'ajouter un objet de nom *id* à l'inventaire. *removeInventory(String objectName)* permet de retirer un objet de l'inventaire, donc d'effacer une ligne du fichier *inventaire.txt*. *isSelected()* et *notSelectAll()* permettent respectivement de vérifier si un objet est sélectionné dans le fichier, symbolisé par un # devant le nom de l'objet, et de désélectionner tous les objets de l'inventaire. Enfin, la fonction *modifyInventory()* permet de mettre à jour le fichier *inventaire.txt* et est appelée par les fonctions d'ajout et de suppression d'objets.

- Le taquin

Nous avons décidé de réaliser un puzzle pour notre Cube Escape, nous avons donc choisi de réaliser un taquin. Ce taquin au niveau du fonctionnement ne représentait pas spécialement de défi mais nous avons eu de grandes difficultés au niveau de l'implémentation graphique. En effet, notre manque d'expérience avec l'interface graphique de javaFx nous a coûté beaucoup de temps.

Tout d'abord le problème a été l'utilisation du *scenebuilder(*)*, en effet il a été difficile de choisir quel type d'interface était intéressante (*VBox(*)*, *GridPane(*)*...). Le choix s'est finalement porté sur le *GridPane(*)*, cependant nous avons vu que l'utilisation du *scene builder(*)* même s'il permet une facilité visuelle, il est difficile de le manier pour le côté esthétique. Notre choix s'est donc porté à créer le *GridPane(*)* via le code au moment de la création du taquin permettant d'optimiser la scène. La mise en forme a été également faite dans le code pour éviter la redondance.

```

taquin[i][j].setText(Integer.toString(k));
taquin[i][j].setId("case");
grid.add(taquin[i][j], j, i);

```

Figure 7 - classe taquin : initialisation du taquin

Pour gérer l'affichage, il était plus intéressant de créer une nouvelle scène, afin de contrôler l'apparition d'images ou autre et de pouvoir fermer la scène automatiquement une fois le taquin résolu.

```

public void affichage(){
    VBox.setMargin(grid, new Insets(50, 50, 50, 50)); //Contour du grid
    root.getChildren().addAll(grid); //ajoute la grille
    Scene scene = new Scene(root, 280, 265); //235, 260
    scene.getStylesheets().add(getClass().getResource("css/style.css").toExternalForm());
    stage.setTitle("Résolvez le taquin !");
    stage.setResizable(false);
    stage.setScene(scene);
    stage.show();
}

```

Figure 8 - classe taquin : affichage du taquin

Au niveau des couleurs, nous avons créé un fichier type CSS qui regroupe les couleurs des différentes cases du taquin.

Dans un second temps le défi a été de lier le code situé dans la classe Taquin (qui contient toute les fonctions principales pour le bon déroulement du jeu), le *controller(*)* de la scène et la scène.

Le problème principal était que l'initialisation du taquin se faisait dans le *controller(*)* de la scène mais le déroulement du code et des fonctions se faisait dans la classe.

```

public void ouvrirTaquin(MouseEvent event) {

    Button[][] tab = new Button[3][3];
    Stage stage = new Stage(); //création de la scene
    VBox root = new VBox(); //fond
    root.setId("root");
    GridPane grid = new GridPane(); //grille du jeu

    Taquin jeu = new Taquin(tab, stage, root, grid);

    if (event.getSource() == armoireFermeB) {
        jeu.affichage();

        jeu.melange();

        //Mouvement des boutons
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                int x = i, y = j;
                jeu.getTaquin()[i][j].setOnAction(e -> verif(jeu.getTaquin()[x][y], jeu));
            }
        }
    }
}

```

Figure 9 - controller scene panneaux : attribution de la fonction aux boutons

Le but était de pouvoir récupérer un booléen dans le *controller(*)* de la scène afin de déterminer si le taquin est fini ou pas et de pouvoir par la suite ajouter des fonctionnalités. Pour cela, nous avons créé une fonction dans le *controller(*)* de la scène qui permet de récupérer la position du bouton cliqué dans le *gridpane(*)* afin de déterminer si le taquin est résolu ou non suite au mouvement.

```

        if (b == taquin.getGrid().getChildren().get(i + 6)) {

            if(i==1){
                y=1;
            }
            if(i==2){
                y=2;
            }
            x = 2;
        }

    }

    if(taquin.mouvement(x,y)){
        Parser p = new Parser();
        p.writeIn( nomFichier: "ScenePanneau.txt", nomObjet: "taquin");
        armoireFermeB.toBack();
        armoireOuverteB.toFront();
        AudioPlayer audioPlayer = new AudioPlayer();
        audioPlayer.play(new File( pathname: "src/Stages/Sons/porte.wav"));
        super.changePlan(anchorPane, nomObjet: "ordonnance1", plan: "front");
    }else{
        taquin.mouvement(x,y);
    }
}

```

Figure 10 - controller scene panneaux : gestion du taquin fini

2.2. Réalisation

2.2.1. Description de l'architecture du programme

Le projet est réalisé avec une architecture simple et bien organisée. Chaque sous dossier contient des fichiers du même type (dans le dossier controllers, on a tous les controllers que nous utilisons dans notre jeu). Il contient donc un dossier "out" dans lequel se trouve l'exécutable du code, il a aussi un dossier "src" dans lequel il y a le code que nous avons écrit. Enfin, il y a aussi un dossier UML qui contient tous les diagrammes qui ont été créés pour la conception du projet.

2.2.2. Organisation des fonctions dans le projet

Le projet fonctionne grâce à l'implémentation de nombreuses fonctions. Il y a des fonctions liées aux mouvements du joueur dans les salles du jeu, liées à l'affichage du jeu et des salles. Certaines fonctions gèrent le déroulement d'énigmes et leur réalisation. Il y a aussi des fonctions qui permettent de gérer l'inventaire (entrée, sortie d'un objet dans l'inventaire, son utilisation dans le jeu) par rapport à sa partie technique et visuelle.

2.2.3. Arborescence des fichier du programme

Dans le fichier *src*, se trouve un dossier Stage qui contient plusieurs dossiers. Les *controllers*(*) des différentes scènes, le css, les fichiers textes enregistrant l'inventaire et certaines actions d'une scène à une autre, les images triées par scènes, les sons, les vidéos et des classes dont le *main*, qui permet de lancer le projet.

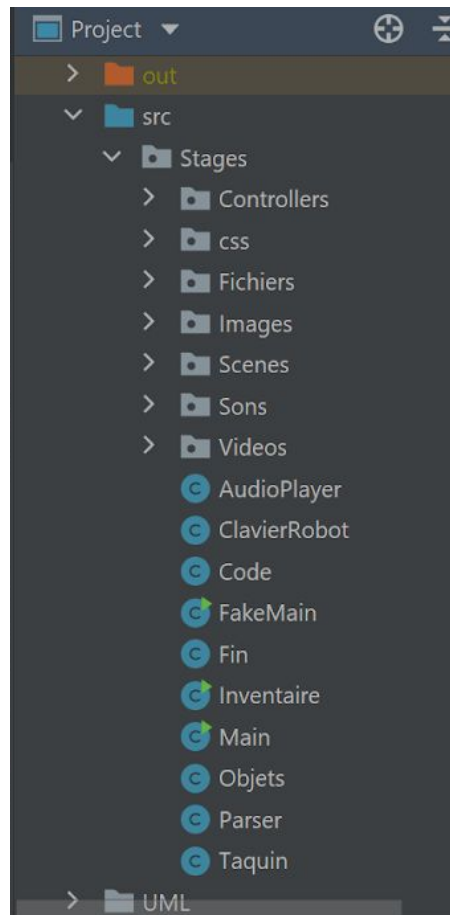


Figure 11 - organisation des packages sur intellij Idea

3. Résultats

Nous avons beaucoup changé d'avis au fil du projet dû au temps et aux conseils donnés par les professeurs. Nous voulions aussi mettre sous forme d'application notre jeu PC mais ça n'a pas été possible.

3.1. La trame scénaristique

Le joueur est représenté par une femme en 3020 dans un univers où les robots font partie de notre quotidien. Notre personnage n'a pas confiance en eux et les voit d'un mauvais œil, estimant qu'ils ne sont pas uniquement présents pour aider l'humain. En effet, cette peur est justifiée par le remplacement de l'emploi de son mari par un de ces humanoïdes.

La fille de notre personnage est victime d'un cancer du rein et a été amenée à l'hôpital pour une opération. Notre personnage se retrouve donc dans un hôpital et a pour but d'arrêter cette opération, qu'elle estime dangereuse. Pour cela, le personnage devra résoudre plusieurs énigmes afin d'obtenir un badge pour ouvrir la porte permettant d'accéder à la scène finale et de choisir si on laisse les robots finir l'opération de la petite fille ou pas. En fonction de notre choix, nous pourrions repartir de l'hôpital avec l'enfant ou non.

Pour commencer, nous avons 2 trajets possibles:

- Le premier permet de reconstituer l'ordonnance de l'enfant dont un bout se trouve dans l'accueil et le second dans une étagère que l'on peut ouvrir en résolvant un taquin.
- Le second permet d'accéder à une clé ouvrant la salle des toilettes. Pour cela, il faut trouver un code à mettre dans la TV qui nous en donne un autre à rentrer dans le téléphone. Ce dernier nous communique un mot à écrire dans un robot se trouvant à l'accueil. Une fois tout cela fait, l'automate nous délivre la clé.

Après avoir fait tout ça, le personnage pourra se rendre aux toilettes et lire la phrase de l'ordonnance en la plaçant devant le miroir. Grâce au code obtenu sur le miroir, il sera possible d'entrer un horaire dans la radio qui nous renvoie un code à mettre dans un coffre à l'accueil. Une fois ouverts, nous avons enfin accès au badge qui ouvre la porte de la salle d'opération et nous emmène à la scène finale.

Pour avoir la solution complète et le cheminement (annexes 4 et 5).

3.2. Installation

- Télécharger IntelliJ Idea
- Cloner(*) le projet
- Télécharger la librairie de JavaFX
- Ajouter le SDK si besoin (File -> Project Structure -> Project -> SDK)
- Ajouter la librairie de JavaFX sur IntelliJ Idea (File -> Project Structure -> libraries -> + -> java -> rechercher la librairie dans les documents)
- Ajouter la Path variable pour javaFX si besoin (File -> Settings -> Path Variable -> + -> rechercher la librairie dans les documents)
- Ajouter VMP option (Run -> Edit Configuration -> VMP option : --module-path \${PATH_TO_FX} --add-modules javafx.controls,javafx.fxml,javafx.media)

3.3. Manuel d'utilisation

Nous n'avons pas pu transformer notre code en un exécutable seul. Il faudrait donc, pour jouer à notre jeu, cliquer sur "Run" pour lancer le projet. Ensuite, une nouvelle fenêtre va s'ouvrir avec le jeu dessus.

3.4. Tests/Validation

Il n'y a pas de tests à proprement parler pour toutes les parties du code. Les seules parties concernées sont les classes Inventaire et Parser car elles ne font pas appel à la librairie JavaFX. Ainsi, des tests unitaires(*) classiques ont été utilisés pour leur développement et déceler d'éventuelles erreurs.

Dès qu'on code, on compile le projet afin de voir le rendu et ainsi vérifier ce qui va et ce qui ne va pas et on modifie si besoin. Pour chaque fonctionnalité, on regarde si ça marche ou pas. Afin de ne pas créer des problèmes de merge(*), chaque personne de l'équipe travaille sur sa branche.

Nous avons aussi fait tester le jeu à des personnes afin de nous aider dans l'amélioration de notre jeu.

- **Sujet 1 :**

- Fluidité ? Oui, le jeu est fluide.

- Difficulté ? Difficulté pas trop élevée, avec quelques zones cliquables difficiles à trouver (pas d'indices pour la table en bois). Le code du téléphone impossible à trouver (0 indice).

- Cohérence ? Univers avec une ambiance sympathique (il faudrait peut-être ajouter un texte pour nous présenter le contexte).

- Bug ? Les objets d'inventaire sont parfois inaccessibles, le compteur sur la radio bug, bug saisi clavier (si on clique sur le texte en rouge après on peut plus saisir de mot).

- Avis ? Très bon jeu, ambiance agréable avec des énigmes sympathiques. Peut-être qu'un onglet « aide » qui permettrait de débloquer le joueur à certain endroit pourrait être utile.

- Sujet 2 :

- Fluidité ? Il y a une bonne fluidité.

- Difficulté ? Moyen+

- Cohérence ? C'est logique.

- Bug ? L'ordonnance sort de l'inventaire après le miroir, page téléphone Samu trop court (et est un peu pixelisé), on a la possibilité de remplacer les écritures lors du clavier avant que l'on tape quelque chose.

- Avis ? Jeu qui fait réfléchir, très intéressant, j'aime beaucoup.

- Sujet 3 :

- Fluidité ? Manque un peu de feedback pour comprendre qu'on a cliqué.

- Difficulté ? Plutôt difficile

- Cohérence ? Moyenne

- Bug ? Non

- Avis ? Un peu trop compliqué

- Sujet 4 :

- Fluidité ? Nickel

- Difficulté ? Dur mais je ne suis pas une « vraie joueuse ».

- Cohérence ? Manque d'indication pour démarrer et avancer.

- Bug ? Affichages qui se superposent.

- Avis ? Manque un peu de sons, difficile mais rigolo faut réfléchir.

3.5. Améliorations possibles

Une éventuelle mise à jour du jeu pourrait être présentée avec l'implémentation de nouvelles fonctionnalités ou encore une meilleure optimisation du code.

Un des axes d'amélioration serait d'alléger le code et éviter encore plus la redondance. Puis éventuellement améliorer le côté graphique, avec une amélioration du visuel au niveau des boîtes de dialogue.

Il serait également envisageable d'exporter le jeu sur mobile afin d'offrir au joueur plus de possibilités. Si le jeu plaît aux joueurs, il est possible d'en faire une série ou d'en faire différentes versions adapter à différents types de joueurs.

4. Rapport d'activité

4.1. Planification et organisation du travail : recul sur le travail effectué.

4.1.1 Découpe et organisation du travail

Pour gérer notre projet, diverses méthodes étaient à notre disposition ; cascade, en "V" ou itératif. Notre réflexion à ce sujet fut de courte durée. En effet, nous avons très vite opté pour une méthode de développement en itératif. Ayant déjà effectué un projet web utilisant la méthode Scrum, nous connaissions ainsi déjà les outils à utiliser et la posture à adopter.

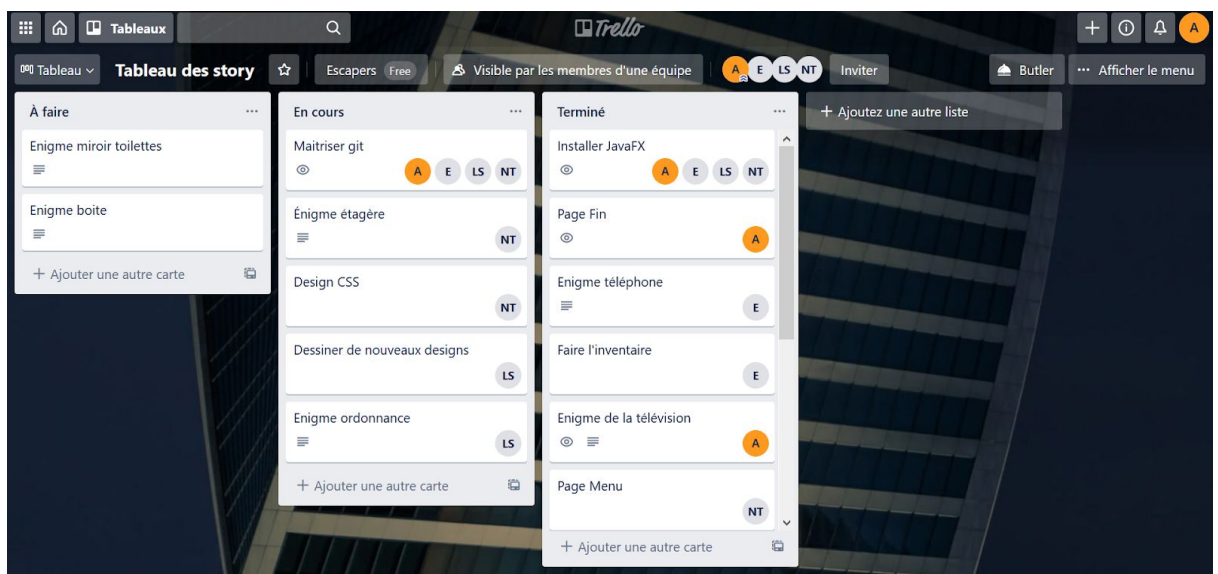


Figure 8 : Product Backlog sur Trello

Dans un premier temps, nous avons créé un product backlog(*) pour recenser les story qui définissent les "tâches" à développer et nécessaires au fonctionnement du jeu. Une fois le product backlog(*) défini, chaque membre du groupe choisissait une story et la développait.

Trello nous a donc permis d'agencer notre product backlog(*) à notre guise et d'avoir une vue globale sur le projet, afin d'éviter de nombreux problèmes de répartition des tâches. Trello permet d'organiser ses projets sous forme de tableaux, eux-mêmes composés de listes en colonnes qui répertorient des tâches sous formes de cartes.

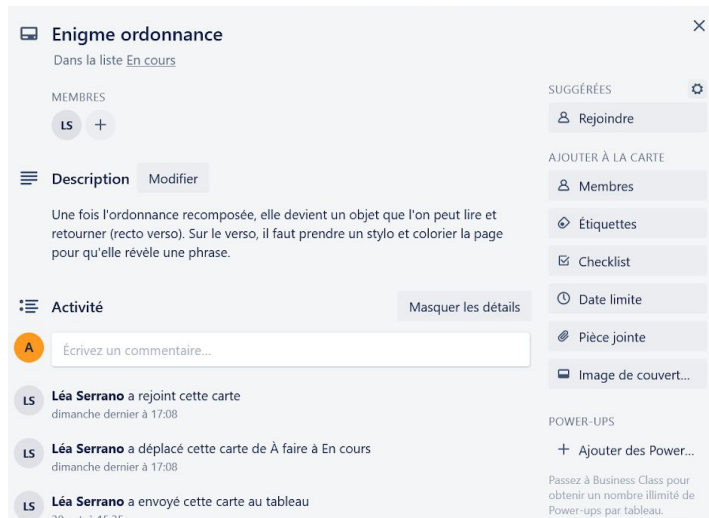


Figure 13 - Description d'une Story

Nous avons choisi Trello car nous l'avions déjà tous manipulé l'année passée pour un autre projet utilisant la méthodologie itérative.

L'une des fonctionnalités de Trello est notamment d'avoir accès au détail d'une story. Ainsi, en cliquant sur l'une d'entre elles, on avait plus d'informations sur celle-ci. Nous pouvions donc savoir exactement quel était le rôle exact de la fonctionnalité à développer dans le projet et éventuellement s'attribuer la tâche.

| All branches | | | | |
|--------------|---|---------|------------------|--|
| main | Updated 22 hours ago by elies-billotta | Default | | |
| magazine | Updated 2 days ago by adele-laurens1 | 13 1 | New pull request | |
| lea | Updated 2 days ago by lea-serrano | 12 0 | #15 Merged | |
| ade | Updated 2 days ago by adele-laurens1 | 15 8 | New pull request | |
| T | Updated 4 days ago by thi-christine-nguyen | 29 2 | New pull request | |
| Thi | Updated 10 days ago by thi-christine-nguyen | 57 4 | New pull request | |
| Adele | Updated 17 days ago by adele-laurens1 | 100 2 | New pull request | |

Figure 14 - Branches sur git

Chaque semaine, le week-end, nous faisons des rendez-vous sur discord afin d'avancer ensemble et de s'entraider si quelqu'un avait un problème pour finir sa story.

Nous utilisons aussi des branches(*) sur git afin qu'il n'y ait aucun conflit(*) de code entre chaque membre du groupe. Chaque personne devait pull(*) avant de pouvoir merge(*) sa branche(*).

Quand tout notre code était terminé, nous avons fait en sorte de l'optimiser. C'est-à-dire que nous avons essayé de redimensionner nos images, d'enlever toutes les redondances de code et de mieux organiser nos dossiers et fichiers pour qu'une personne extérieure puisse comprendre.

4.1.2 Organisation des sessions de travail

Dans un projet en équipe, la collaboration est essentielle. C'est pourquoi nous avons organisé, tout au long des phases de conceptions et d'analyse, un minimum d'une réunion par semaine pour discuter et éventuellement instaurer des sessions de programmation communes au cours desquelles nous travaillons ensemble sur le code du projet afin de s'entraider.

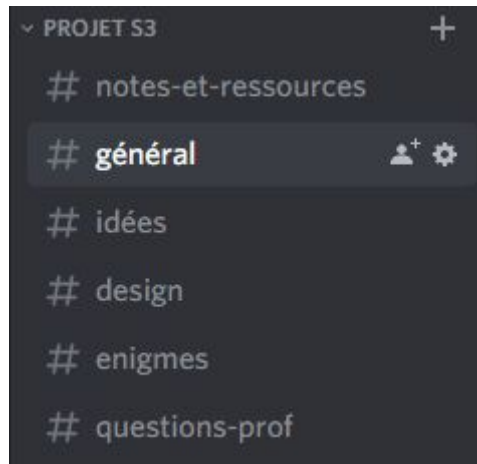


Figure 15 - Organisation de notre serveur Discord

Pour collaborer, nous utilisons Discord, un logiciel de chat vocal et textuel. Nous avons ouvert un serveur afin de nous y retrouver et regrouper, dans divers salons, nos diverses idées sur le projet, tel que le design, les énigmes... Cet outil nous a donc permis d'optimiser nos rencontres virtuelles lors de cette période de codage à distance.

C'est d'ailleurs aussi sur ce logiciel que s'organisaient nos rencontres virtuelles avec notre tuteur, M. Antoine Chollet.

Il nous a dit ce qui allait, ou ce qui n'allait pas, ce qu'il faudrait modifier ou améliorer.

Nous avons aussi montré notre code afin d'organiser tous les dossiers à M. Nassim Belmecheri qui nous a d'ailleurs beaucoup aiguillés sur la méthode à appliquer.

Conclusion

Pour conclure, nous sommes partis d'un sujet : faire un escape game(*) qui s'inspire de la licence de jeux mobiles Cube Escape avec un thème futuriste engagé que nous avons mené à terme.

Le projet a commencé par des analyses et la conception de notre jeu. Nous avons étudié des jeux d'énigmes et d'escape game(*) existant afin de nous donner des idées, de nous aider à structurer notre projet. Ensuite nous avons analysé tous les besoins, c'est-à-dire tout ce qui devait se trouver dans notre jeu.

Après tout ceci, nous avons programmé notre jeu (que nous avons réalisé en Java sur IntelliJ Idea) que nous avons ensuite testé à quelques personnes. Nous avons recueilli les avis que nous avons utilisés pour améliorer notre jeu.

La communication et l'entente dans notre équipe était très bonne. Finalement, nous sommes fiers du projet que nous avons réalisé.


Bibliographie

- [1]
« FAQ JavaFX, le club des développeurs et IT Pro ».
<https://java.developpez.com/faq/javafx> (consulté le janv. 10, 2021).
- [2]
« java - Communication between two JavaFx controllers - Stack Overflow ».
<https://stackoverflow.com/questions/22178533/communication-between-two-java-fx-controllers> (consulté le janv. 10, 2021).
- [3]
« JavaFX Communication Between Controllers with Example Project ».
<https://www.genuinecoder.com/javafx-communication-between-controllers/>
(consulté le janv. 10, 2021).
- [4]
« Learn JavaFX 8 - PDF Drive ».
<https://www.pdfdrive.com/learn-javafx-8-d41376722.html> (consulté le janv. 10, 2021).
- [5]
« S'CAPE-Vers une typologie des énigmes ».
<https://scape.enepe.fr/typologie-enigmes.html> (consulté le janv. 10, 2021).



Annexes techniques

1. Analyse de l'existant

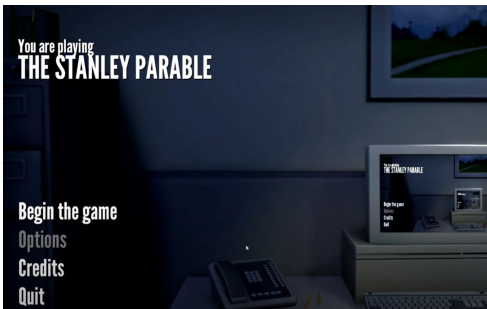
Annexe 1: Cube Escape

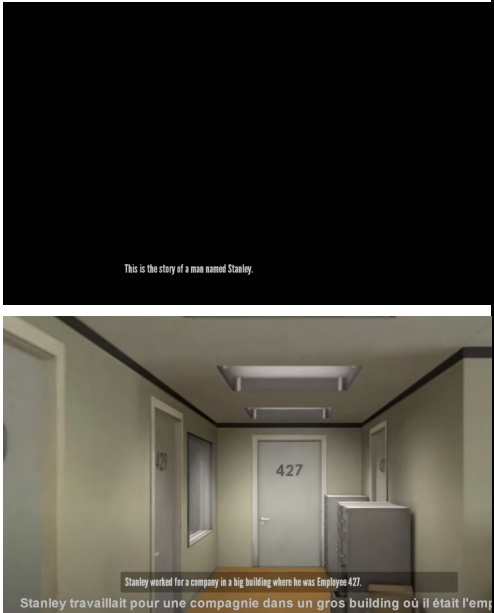
| | |
|---------------|--|
| Mise en scène | <p>Petite histoire au début pour se mettre dans l'ambiance creepy(*) (+ musique)</p>  |
| Paramètres | <ul style="list-style-type: none"> • Réglage du son • Réglage de la qualité de l'image (basse, moyenne, haute) • langue • Reset |
| Aide | <ul style="list-style-type: none"> • Comment jouer (explication du jeu) • Lien discord • Solution vidéo • Oeil qui permet de voir où cliquer |
| Lien | <ul style="list-style-type: none"> • Autres jeux de la même série à télécharger • Instagram, Twitter, Facebook • Laisser avis |
| Succès | <p>Acquisition de petits "trophées" qui résument certaines parties de l'histoire</p> |
| Premium | <ul style="list-style-type: none"> • Supprimer pubs • Soutenir le développeur |

| | |
|-------------------|---|
| Graphisme | <ul style="list-style-type: none"> • 4 murs + 1 plafond • Flèches pour aller à droite, gauche et en haut • Inventaire à droite soit déroulant (de haut en bas), soit "s'ouvrant" (de droite à gauche) • Nom de l'objet affiché quand on clique dessus • Déplacer les objets dans l'inventaire • Animation quand on clique quelque part |
| Son | <ul style="list-style-type: none"> • Petite musique de fond pas forte du tout • Bruits quand on touche à certains objets, quand on les ramasse |
| Enigmes proposées | <div data-bbox="906 891 1358 967" data-label="Image"> </div> <div data-bbox="962 981 1270 1180" data-label="Image"> </div> <div data-bbox="909 1189 1240 1523" data-label="Image"> </div> <ul style="list-style-type: none"> • Images à trouver et reconstituer pour avoir des objets (ex: clefs) • Mots croisés à réaliser en cherchant des indices <div data-bbox="906 1713 1318 1995" data-label="Image"> </div> |

| | |
|----------------------|--|
| |  <ul style="list-style-type: none"> • Numéro de téléphone ou vidéo • Petites animations permettant l'accès à de nouvelles énigmes  |
| Critiques sur le jeu | <ul style="list-style-type: none"> • Il faudrait plus de précision au niveau du tactile. • Il manque la possibilité de prendre des notes • Parfois manque d'aide pour avancer |

Annexe 2: The Stanley Parable


| | |
|---------------|---|
| Mise en Scène |  <p>Lors de l'arrivée dans le jeu, on a la page d'accueil du jeu avec le nom du jeu, la possibilité de démarrer le jeu, l'accès aux crédits et enfin, la possibilité de quitter le jeu.</p> |
|---------------|---|

| | |
|-----------|--|
| |  <p>Lors de notre arrivée dans le jeu, une cinématique nous explique l'histoire principale.</p> <p>On a une narration tout au long de l'histoire avec une voix-off qui commente ce qui se passe.</p> |
| Aide | <ul style="list-style-type: none"> • Il y a beaucoup de vidéos de ce jeu sur internet • Il y a aussi des guides de solutions disponibles sur internet |
| Graphisme | <ul style="list-style-type: none"> • On a plusieurs pièces qui sont de tailles différentes et en 3D |
| Contrôles | <ul style="list-style-type: none"> • Pour se déplacer, on utilise les touches directionnelles • La souris nous permet de pivoter • Le bouton gauche de la souris peut nous permettre d'appuyer sur des boutons • La touche Echap nous permet de quitter le jeu |
| Son | <ul style="list-style-type: none"> • On peut entendre des bruits de pas, des bruits de porte • On a la voix-off que l'on entend pendant tout le jeu. La voix-off est vraiment bien faite, le narrateur est drôle et ajoute de la musique parfois. |

| | |
|----------------------|--|
| Enigmes proposés | <ul style="list-style-type: none"> • Tout le long du jeu, une seule énigme est proposée : c'est trouver une réponse à notre problème. Pour ce faire le gameplay(*) se base uniquement sur l'exploration du bureau dans lequel on se trouve. • Le jeu comporte plusieurs fins qui sont liées aux endroits où l'on va. |
| Critiques sur le jeu | <ul style="list-style-type: none"> • Il n'y a pas vraiment d'énigmes, il faut juste explorer la zone. |

Annexe 3: Professeur Layton

| | |
|---------------|---|
| Mise en scène | <p>Histoire qui présente les personnages ainsi que l'univers du jeu.</p> <p>Mise en place de la trame principale et de l'objectif à atteindre.</p> |
| Paramètres | <ul style="list-style-type: none"> • Réglage du son • Réglage de la qualité de l'image (basse, moyenne, haute) • Langue • Reset |
| Aide | <ul style="list-style-type: none"> • Fan solution (site internet, vidéo) • Il est possible "d'acheter" de l'aide avec des pièces (SOS) |
| Lien | <ul style="list-style-type: none"> • Plusieurs jeux qui suivent l'histoire, séries animées, film... • Grande communauté |
| Succès | <ul style="list-style-type: none"> • Permet de débloquent l'histoire principale ou des histoires supplémentaires. |

| | |
|-------------------|---|
| Graphisme | <ul style="list-style-type: none"> • Générique (ville, maison, bateau...) selon l'histoire • Dessin en 2D • Couleurs chaleureuses dans la journée et froide le soir. • Beaucoup d'animation • Élément du décor interactif • Flèche pour se déplacer |
| Son | <ul style="list-style-type: none"> • Petite musique de fond selon les situations. • Des dialogues doublés • Bruitages divers |
| Enigmes proposées |  <ul style="list-style-type: none"> • Enigme généralement très interactive. • Types d'énigme très variés (mathématiques, logique, visuel...) |

089 27 picarats 0 : 23

Vous vous trouvez à un croisement et vous ignorez s'il faut emprunter la voie de gauche ou de droite. Votre intuition vous dit que le panneau qui se trouve en face de vous a pour but d'indiquer le chemin à prendre.

Trouvez parmi ces traits une flèche qui corresponde à celle qui se trouve à droite du panneau. Une fois que vous l'aurez trouvée, tracez-en les contours.

La flèche que vous cherchez n'est peut-être pas de la même taille que son modèle.



116 50 picarats 0 : 23

Neuf carrés sont creusés dans du bois.

Vous devez placer neuf nombres différents situés entre 1 et 51 de telle manière qu'en additionnant toutes les lignes horizontales, verticales ou diagonales qui passent par la case centrale, la somme des trois nombres soit exactement la même.

Bien qu'il existe de nombreuses configurations possibles, votre but est de trouver celle qui vous permettra d'obtenir la plus grande somme. Dans cette configuration précise, quel nombre occupera la case du milieu?



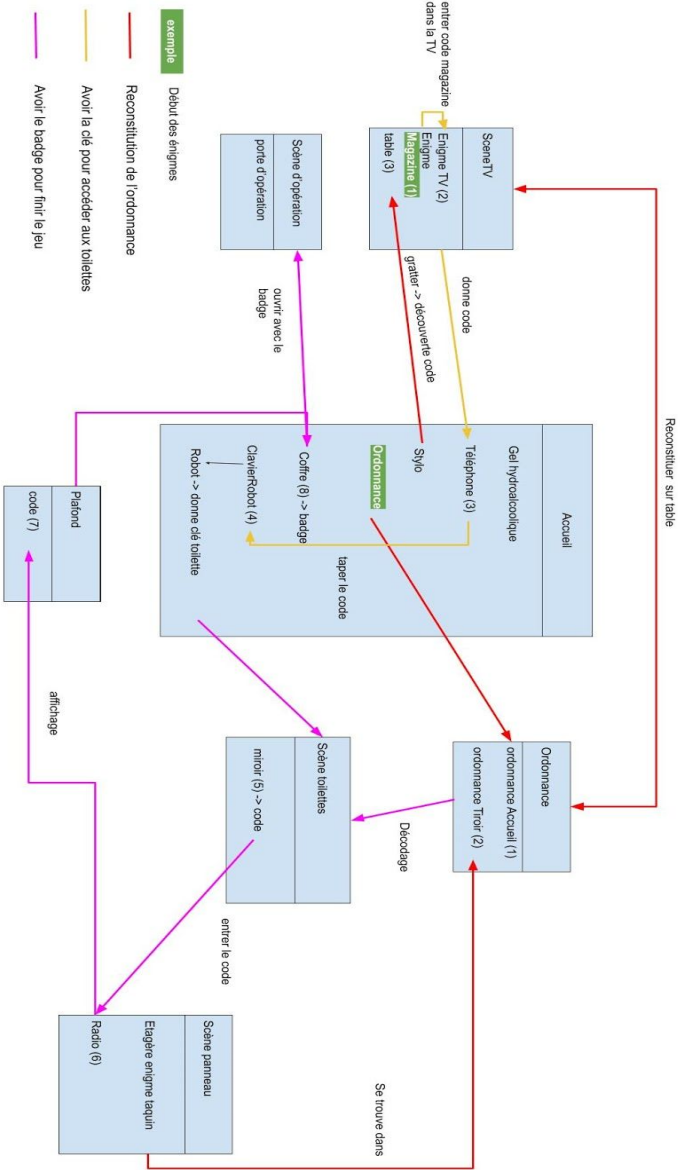
| | |
|----------------------|---|
| Critiques sur le jeu | <ul style="list-style-type: none"> • Les énigmes n'ont pas forcément de lien avec l'histoire • Certaines énigmes sont trop difficiles pour la tranche minimal du jeu (7ans) • Il faut attendre les nouvelles sorties de la série pour pouvoir expliquer certaines scènes ou la présence de certains personnages (histoire de Don Paolo...) |
|----------------------|---|

2. Déroulement du jeu (solutions)

Annexe 4: Solutions

- 1) Mettre du gel hydroalcoolique
- 2) Première partie de l'ordonnance dans l'accueil
 - a) Résoudre le taquin afin d'obtenir le deuxième morceaux de l'ordonnance
 - b) Reconstituer l'ordonnance sur la table basse devant la télévision
- 3) Résoudre l'énigme de la télévision
 - a) Chercher dans les magazine les lettres DNL (correspondant à la marque de la télévision)
 - b) Obtention du code grâce au numéro des pages (736)
 - c) Entrer le code sur la télévision
 - d) Obtention d'un code à utiliser sur le téléphone (SAMU)
- 4) Aller à l'accueil et sur le téléphone entrer le code 7268 (correspondant à la position des lettres)
- 5) Enigme sonore : solution epilepsie
- 6) Entrer le code epilepsie sur le clavier en cliquant sur le robot -> obtention de la clé des toilettes
- 7) Ouverture des toilettes avec la clé
 - a) Cliquer sur le miroir
 - b) Cliquer sur l'ordonnance puis le miroir encore une fois
 - c) Obtention de l'énigme : solution 14h30
- 8) Entrer l'heure obtenue sur la radio : obtention du code 8035
- 9) Entrer le code 8035 dans le coffre et obtention d'un badge
- 10) Utilisation du badge pour ouvrir la salle d'opération et accéder à la scène finale
- 11) Faire un choix final qui va impacter la fin.

Annexe 5: Schéma de la trame scénaristique



3. Code l'inventaire

Annexe 6: Code de l'inventaire

```
public void notSelectAll() throws IOException {
    String[] strings = p.readIn( nomFichier: "inventaire.txt");
    resetInventory();
    for (int i = 0 ; i < strings.length ; i++) {
        if(!strings[i].equals("")){
            if(strings[i].charAt(0) == '#') {
                strings[i] = strings[i].replace( target: "#", replacement: "");
            }
        }
    }
    modifyInventory(strings);
}
```

```
public void modifyInventory(Object[] s) throws IOException {
    FileWriter fileWriter = new FileWriter(new File( pathname: "src/Stages/Fichiers/inventaire.txt"));
    for (int i = 0 ; i<s.length ; i++){
        if(i!=s.length-1) fileWriter.write( str: s[i]+"\\n");
        else fileWriter.write((String) s[i]);
    }
    fileWriter.close();
}
```

```
public void addInventory(String id) throws IOException {
    boolean addToInventory = false;
    String[] strings = p.readIn( nomFichier: "inventaire.txt");
    if(strings != null){
        ArrayList<String> strings1 = new ArrayList<>();
        for (int i = 0 ; i<strings.length ; i++) {
            if(strings[i].equals("") && !addToInventory) {
                strings[i] = id;
                addToInventory = true;
            }
            strings1.add(i,strings[i]);
        }
        if(!strings1.contains(id)) strings1.add(id);
        modifyInventory(strings1.toArray());
    }
    else{
        modifyInventory(new String[]{id});
    }
}
```



```

public void resetInventory() throws IOException {
    File inventaireFile = new File( pathname: "src/Stages/Fichiers/inventaire.txt");
    FileWriter fw = new FileWriter(inventaireFile);
    fw.write( str: "");
    fw.close();
}

public boolean isSelected(String objectName) {
    String[] strings = p.readIn( nomFichier: "inventaire.txt");
    if(strings != null){
        for (String s : strings) {
            if (!s.equals("") && s.charAt(0) == '#' && s.contains(objectName)) return true;
        }
    }
    return false;
}

public void removeInventory(String objectName) throws IOException {
    String[] strings = p.readIn( nomFichier: "inventaire.txt");
    if(strings != null){
        for (int i = 0 ; i<strings.length ; i++) {
            if (strings[i].equals("") || (strings[i].charAt(0) == '#' && strings[i].contains(objectName))) {
                strings[i] = "";
            }
        }
        modifyInventory(strings);
    }
}

```

4. Compte rendus des rendez-vous avec notre tuteur

Annexe 7: Entretiens avec le tuteur

1er RDV :

Il faut coder un jeu dans le même style que le jeu mobile Cube Escape. On aura un thème défini (ex: environnement) et on devra faire passer des messages à travers les énigmes. On ne reprendra pas l'apparence creepy(*) du jeu original. On fera le code en java ou C sharp sur Unity. Il ne faudra pas utiliser les bibliothèques proposées par Unity sauf pour le graphisme si c'est vraiment compliqué.

Pour nous aider dans notre documentation sur les escape games, notre tuteur de stage nous a parlé du jeu "The Stanley Parable".

On va faire une réunion par semaine jusqu'au début des vacances de Toussaint puis nous ce sera à l'équipe de s'organiser pour prendre des rendez-vous avec les professeurs. Ce dernier aimerait en fait 1 toute les 2 semaines au plus. Pour l'instant, nous devons faire l'analyse de l'existant pour la semaine prochaine ou pour dans 2 semaines. Il faut donc l'avoir fini pour les vacances de Toussaint au plus tard.

On aura un oral ainsi qu'une fiche résumant notre travail à rendre à la fin du semestre 3 (janvier 2021).

2eme RDV:

Nous avons rendu notre analyse de l'existant au professeur. Il nous a demandé si on avait bien compris le concept du jeu, si on avait joué à Cube Escape et si on avait des idées de thèmes.

Nous lui avons dit que voulions coder sur Unity en C# pour l'instant. Il nous a dit qu'il fallait faire attention à ne pas utiliser les librairies pour coder les fonctionnalités du jeu mais qu'on pouvait s'en servir pour le graphisme.

3eme RDV:

Nous avons proposé au professeur un commencement de diagramme de classe et un use Case.

Nous hésitons entre faire une liste d'objets ou faire une casse inventaires. De plus, nous ne sommes pas non plus sûrs des liaisons entre chaque classe car nous avons montré le diagramme à notre professeur de CPOA la veille.

Pour l'instant, nous ne pouvons pas encore faire un diagramme complet car nous manquons d'éléments pour.

Eliès a présenté l'histoire générale du Cube Escape que nous allons utiliser ainsi que sa morale. Le prof l'a validé et nous a encouragés à commencer à coder pendant les vacances afin de ne prendre aucun retard sur le projet.

Il nous a bien-sûr rappelé que nous ne devons pas utiliser les bibliothèques Unity pour coder (sauf pour le graphisme car ce n'est pas ce qui est noté).

Léa a proposé de dessiner sur sa tablette graphique les décor du jeu.

4eme RDV:

Nous avons montré à notre tuteur de stage ce que nous avons fait pendant les vacances : nous lui avons montré les premiers dessins du jeu, et nos idées pour les énigmes.

Il nous a conseillé de mettre des choses dans les énigmes afin de nous rappeler notre thème sur la robotique. Il nous a dit qu'on pouvait peut-être mettre sur l'ordonnance une prescription pour un implant robotique et ainsi donner un aspect transhumanisme à notre jeu. Aussi, notre professeur nous a conseillé de mettre des informations dans le journal que nous avons prévu de mettre dans le jeu.

Nous lui avons aussi posé des questions car, au final, nous hésitons à prendre Unity ou à coder en java (sur IntelliJ Idea). Il nous a dit que cela dépendait de ce que nous voulions, il y a un côté risqué mais aussi original en prenant Unity. Mais programmer notre escape game(*) sur intellij idea est un moyen de montrer ce que nous avons acquis lors du semestre 2 et du semestre 3.

Il nous a aussi demandé de commencer à lui rendre du code la prochaine fois qu'on le verrait et de faire notre trame principale de notre histoire.

5eme RDV:

On a montré à notre tuteur de stage l'avancée de notre projet : L'ajout du magazine, de l'énigme du gel hydroalcoolique, ... Nous lui avons aussi fait part de nos soucis concernant l'inventaire et le changement de scènes.

6eme RDV:

On a montré à notre enseignant notre grande avancée : nous avons réussi à corriger notre souci sur les changements de scènes et celui sur l'inventaire. Il nous a aussi dit qu'on pourrait se voir moins souvent si on a peu de choses à lui montrer.

7eme RDV:

Nous avons montré notre téléphone, l'énigme sur le magazine et la télévision, les toilettes.

Notre enseignant nous a demandé de rajouter une trame de l'histoire, de mettre les ombres bien sur les images. Il trouve qu'on a assez d'énigmes. Il nous a aussi demandé notre avancement sur le rapport, il nous a conseillé aussi de mettre du code dans notre rapport technique. Il voulait aussi qu'on lui rende la trame des énigmes.

8eme RDV:

Nous avons surtout discuté avec notre tuteur. Nous lui avons montré notre schéma sur la trame de l'histoire et l'ordre des énigmes. Nous lui avons parlé de nos difficultés sur le taquin. Il nous a demandé d'essayer d'améliorer le graphisme et de continuer nos énigmes.

9eme RDV:

Nous avons montré nos fonctionnalités, notre tuteur nous a dit que le projet était presque fini. Il nous manquait juste quelques petits détails dont on lui avait parlé, il aurait aussi aimé que notre jeu soit sous forme de .exe.

Résumé

Ce document parle du projet que 4 étudiants de l'IUT de Montpellier du département Informatique (Eliès Billotta, Adèle Laurens, Thi-Christine Nguyen, Léa Serrano) ont réalisé lors de leur troisième semestre. Leur projet était de réaliser un jeu d'escape game(*) se basant sur la série d'escape game(*) pour téléphone mobile *Cube Escape*. Ce projet était globalement assez libre puisque les étudiants avaient le choix du thème, des énigmes, de leur nombre, etc... même s'ils devaient se baser sur un thème futuriste engagé. Les étudiants ont choisi de faire leur jeu sur un thème robotique dans le futur et ont réalisé ce projet sur IntelliJ IDEA dans le langage Java.

Dans ce document, on voit tout le déroulé du projet, de l'analyse de l'existant jusqu'aux résultats.

That document is talking about 4 student's project from the UIT of Montpellier IT department (Eliès Billotta, Adèle Laurens, Thi-Christine Nguyen, Léa Serrano) that they realized during their third semester. Their project was to realize an escape game(*) based on the *Cube Escape* series for mobile phones. That project was globally free because the students had the choice for the theme, the puzzles, and their number, etc... even if they had to base their theme on an engaged future. The students had chosen to do their game on a robotic future theme and realized that project on IntelliJ IDEA in the Java language.

In that document, we can see all the process of the project, from the analysis of what's existing to the results.