ALTERAVITA

Rapport de Soutenance 1

Réalisé par :
Hugo MEENS
Elya MAURY
Ethan PERRUZZA
Timothy DIDIERJEAN

Projet S2 - EPITA

Table des matières

Chapitre 1

Rappel du Projet

Cette section a pour but de rappeler les éléments du cahier des charges dont nous allons parler dans le rapport.

1.1 Le Groupe

1.1.1 Présentation

L'équipe en charge de la réalisation de ce projet est nommée XENOR et est constituée de :

hugo.meens : Hugo MEENS (chef de projet)

elya.maury : Elya MAURY

ethan.perruzza : Ethan PERRUZZA

timohty.didierjean : Timothy DIDIERJEAN

1.1.2 Gestion Organisationnelle

Le groupe se réunit lors de réunion hebdomadaire tous les lundi soir ou mardi midi (suivant l'emploi du temps) pour faire un point sur l'avancement et les objectifs de la semaine.

Nous utiliserons les fonctionnalités « projects » et les « issues » de GitHub pour connaître l'avancement du projet : les tâches en cours, réalisées ou à réaliser. De plus, chacun indiquera sur un serveur Discord propre à l'équipe sur quoi il travaille en ce moment.

Le serveur Discord de l'équipe permettra également de discuter des éventuels problèmes rencontrés par chacun et de garder un lien hors des réunions hebdomadaires.

1.2 Histoire

1.2.1 Synopsis

Dans un monde, où la médecine ne cesse d'évoluer, vous vous réveillez enfermé dans une cellule d'un laboratoire. Par chance, tout le monde n'a pas été complètement annihilé par le désir croissant des populations occidentales de contrôler leur santé et de repousser les limites de leur finitude. Et, alors que les scientifiques et les managers du laboratoire sont convoqués par le directeur du laboratoire Xenor pour discuter des prochains essais humains de transgénèse, c'est le moment que Clothilde, opératrice réseau choisit pour vous aider à vous échapper. Il faut dire que Clothilde n'est pas née de la dernière pluie, elle a été formée à l'EPITA, la plus grande école d'ingénieur en informatique de l'Univers.

Issue de la promo 2026, elle est la dernière épitéenne encore en vie à ce jour... En effet en 2052, les dirigeants des plus hautes sphères de la planète se firent pour mission d'éliminer tous ceux qui pourraient nuire à leur projet et la trop grande place de l'éthique chez les épitéens avait conduit à une

forte diminution du nombre d'Alumni. Clothilde était une autre personne à l'époque, elle ne souciait guère des droits humains. Mais là s'en était trop ! Sans trop de difficultés, elle parvient à passer toute la sécurité du système et à vous libérer de votre cellule.

Malheureusement, le laboratoire est hautement sécurisé et il vous faut encore sortir du bâtiment pour vous échapper. Pas de panique, Clothilde devrait pouvoir vous aider à distance, à moins que ses compétences en cybersécurité ne se soient un peu rouillées à force de nettoyer la salle de serveurs du laboratoire.

Vous et votre camarade recherchez la sortie du laboratoire afin de gagner votre liberté. Croyez-nous, ce ne sera pas une mince affaire, ce laboratoire regorge de portes et chaque porte est verrouillée par une énigme, car cela « stimule la créativité des employés » et les monstres issus d'expériences ratés que Clothilde a malencontreusement libéré en tentant de vous aider ne vont pas vous simplifier la tâche... Saurez-vous vous évadez de ce laboratoire infernal ?... Dépêchez-vous ! Le temps presse, les scientifiques peuvent revenir d'un instant à l'autre !

1.2.2 Conclusion de l'histoire

Vous parvenez enfin à vous échapper, mais lorsque vous poussez la porte de la sortie vous ne trouvez pas la lumière puissante et la douce chaleur du soleil qu'il vous a semblé apercevoir par les fenêtres du labo mais bien la lumière blafarde d'une salle éclairée aux néons. Vous êtes dans une nouvelle salle de laboratoire carrelé de carreaux blancs sans vie qui vous oppresse, seulement, vous n'est pas seul... Clothilde est là ! Vous vous sentez tout de suite rassurés. Une sorte de vague de chaleur et de bien-être vous envahit. « Ramenez-les à leurs cellules. » Quatre gardes que vous n'aviez pas vu jusqu'à présent s'avance et vous saisissent vous et votre compagnon par les bras. Vous vous laissez faire, trop choqué pour réagir...
Clothilde qui vous a pourtant aidé à vous échapper ordonne maintenant votre retour à l'incarcération. On vous retourne face à la grande porte que vous venez de prendre, vous apercevez deux écran numérique affichant respectivement « Crésus : Clone n°397 » et « Crésus : Clone n°398 » ainsi que deux notes très détaillées. Au-dessus de la porte se trouve le logo de Xenor, en dessous un slogan : « Avec Xenor, négationnez la mort ». Vous comprenez alors que Clothilde ne vous aidez pas à vous échapper mais évaluait simplement vos capacités.

En réalité, il y a quelques années de cela, le laboratoire Xenor avait réussi à comprendre où se situer la conscience de l'Homme et à la déplacer d'un corps vers un autre, proposant ainsi aux hommes fortunés de pourvoir s'offrir une seconde jeunesse. La note que vous avez vu sur l'écran ne sert en fait qu'à définir quel clone a les meilleures capacités pour définir lequel accueillera la conscience du client.

1.2.3 Intentions/Visée de réflexion du scénario

Notre but ici est de faire réagir, de choquer le joueur pour le faire réfléchir. Que ce soit par la croyance en une vie après la mort, une réincarnation ou autre, ou bien par la science, l'Homme cherche depuis toujours à trouver une solution à sa finitude. En effet, c'est bien là la vocation de la médecine : de repousser cette échéance qui caractérise l'être humain ; tout faire pour éviter notre fin ou en tout cas de la repousser au maximum.
Et on voit d'ailleurs chaque jours ses succès ! Prenons un exemple qui peut a première vue sembler anodin : si vous êtes naît par césarienne, il y a quelques dizaines d'années, vous et votre mère serait inévitablement morts. De nombreuses maladies autrefois mortelles sont désormais sans importance. On peut même guérir d'un cancer s'il est détecté assez tôt !
Et pour repousser sa fin, l'Homme est prêt à tout, car le temps est sa ressource la plus chère car fatalement limité.

Depuis quelques siècles maintenant, les philosophes tentent de caractériser notre conscience. Les scientifiques tentent-eux de comprendre ce qu'elle est et où elle est située dans le cerveau. Sommes-nous ancrés à notre corps ou notre âme est-elle volatile ? Pouvons-nous comme le laisse entendre certains sortir de notre enveloppe charnelle ?...
Imaginez maintenant que l'on réussisse à définir, situer et transplanter notre conscience dans un autre

corps. Ce serait là, la fin de la finitude. Une grande question morale apparaîtrait alors sur le droit moral de l'Homme à transplanter sa conscience dans le corps d'un autre.
De la même façon que les premières chirurgies, les premières greffes ont soulevée de nombreuses questions morales sur leur bien-fondé. La différence réside ici dans le consentement. Mais lorsque l'on parle de réaliser des greffes de porc sur l'homme, il n'y a pas de notion de consentement. Lors des essais clinique, on ne demande pas non plus leur avis aux souris.
Le but n'est pas de remettre en cause toutes ces pratiques, loin de là, mais plutôt de se questionner sur les limites. Jusqu'où pouvons-nous aller ? Est-ce que la vie éternelle est une raison suffisante pour « cultiver » des êtres humains pour y implanter sa conscience ?
Mais une chose peut être sûre, si cela s'avère possible, et même si c'est interdit, les plus fortunés pourront se l'offrir. Il existe bien un trafic d'organes, un trafic d'esclaves, alors pourquoi pas un trafic d'humain à destination de transplantation de conscience ?
Transplanté sa conscience dans un autre pourrait devenir au fil des ans aussi banal que de prendre un médicament...
Et si l'on créait un clone de nous-même pour y implanter sa conscience ?... Serait-ce plus moral ?

Nous voulons faire réfléchir sur ce qu'impliquerait la vie éternelle par transplantation de conscience et sur le bien-fondé de notre volonté de combattre notre nature en repoussant toujours plus les limites de la mort... Où doit-on s'arrêter ?

1.3 Mécaniques de GamePlay

1.3.1 Présentation Générale

Heureusement pour vous vous n'êtes pas seul, votre compagnon de cellule vous accompagne et croyez-nous, la coopération sera maître mot pour espérer sortir un jour de ce laboratoire infernal. Il vous faudra résoudre les énigmes pour accéder ou déverrouiller les portes.

Dans certaines salles, se trouveront des monstres, vous disposerez de fioles pour les combattre. Ils pourront également être combattu à la main, au corps à corps. Vous aurez également une barre de vie car les monstres ne sont pas inoffensifs. Dans le cas où l'un des deux joueurs meurt pendant un affrontement avec un monstre, où d'une quelconque autre manière, le deuxième joueur pourra réanimer le premier grâce au super kit de secours. Seulement, si les deux joueurs viennent à être éliminés par le monstre, vous avez perdu car les scientifiques vous retrouveront et n'aurons aucun intérêt à vous réanimer, la Terre grouille de cobayes...

Les scientifiques développent également des boosts de jeunesse, il y en a quelques-uns qui traînent dans le laboratoire, ils devraient vous permettre de récupérer de la vie si vous avez l'oeil.

Les joueurs seront toujours dans la même pièce. En effet, prendre une porte emmène votre camarade de cellules avec vous, vous ne voudriez pas vous retrouver seuls dans ce laboratoire...
Les énigmes sont conçues pour être résolues à deux.

Le jeu comportera un lobby principal reliant les niveaux entre eux. Ceux-ci peuvent prendre plusieurs formes : énigmes et/ou combats contre des monstres.

Certains niveaux seront générés procéduralement pour augmenter la rejouabilité.

Chaque joueur contrôlera un personnage dont les caractéristiques sont :

- une barre de vie
- un inventaire
- possibilité de déplacer
- possibilité d'attaquer
- possibilité d'interagir avec certains éléments du décors
- possibilité de collecter certains objets

- possibilité de d'utiliser certains objets collectés
- possibilité de se soigner
- possibilité de réanimer son coéquipier
- menu pause et configurations

1.3.2 Les Énigmes

Labyrinthe à Boîtes

But : Bouger des boîtes pour se frayer un chemin jusqu'à la sortie.

Les joueurs apparaissent à deux endroits différents dans la salle.
La salle est pleine de boîte de rangement (nous sommes dans un entrepôt du laboratoire).
Les joueurs peuvent interagir avec les boîtes (les tirer, les pousser, les décaler sur les côtés).
Leurs chemins jusqu'à la sortie sont bloqués par des boîtes mais ils ne pourront pas toujours se débloquer seuls, ils auront besoin de l'aide de leur coéquipier pour bouger certaines boîtes qui étaient jusqu'alors impossible à bouger pour eux.
Certaines boîtes sont trop lourdes pour être poussées.
Pour sortir de la salle, il faut que chaque joueur se retrouve sur une plaque de pression, en face d'une porte.

Il n'y a pas de condition d'échec sur cette énigme.

Labyrinthe Invisible

But : Sortir du labyrinthe invisible en coopérant avec votre coéquipier.

Un joueur aura la carte du labyrinthe et l'autre sera dans le labyrinthe, mais le labyrinthe est invisible. En effet, les obstacles de ce labyrinthe sont invisibles.
Marcher à un endroit interdit inflige des dégâts et téléporte le joueur au début du labyrinthe.

Si le joueur meurt, c'est une condition d'échec.

Enigmes des fils

But : Couper le bon fil à partir de consignes, un joueur ne peux pas couper les fils et avoir les consignes en même temps.

Ce système est inspiré du jeu « Keep Talking and Nobody Explode ».
Un joueur sera face aux fils, l'autre aura un manuel. Le manuel contiendra des phrases possédantes chacune une condition et une conséquence (ex : "si le premier fil est bleu couper le dernier fil").
Si l'un des joueurs coupe un mauvais fil, il perd de la vie en laissant une limite de vie d'un demie coeur au joueur, l'empêchant ainsi de mourir sur cette énigme.

Labyrinthe fléché

But : Sortir du labyrinthe grâce à une grande coopération

Ce labyrinthe est constitué de dalles fléchés, le joueur ne peut aller que dans la direction des flèches et ne peux pas aller en opposition à une flèche sur une autre tuile.
De plus les deux joueurs ne verront pas l'entièreté des dalles les obligeant ainsi à communiquer pour pouvoir résoudre ce labyrinthe.
Si un joueur ne respecte pas les règles précédemment citées, il perdra de la vie, et sera possiblement téléporté au début du labyrinthe.

Chapitre 2

Avancement

Durant cette période, nous avons travaillé chacun de notre coté en se répartissant les tâches.

2.1 Graphismes

Nous avons beaucoup avancé sur cette partie bien qu'il reste encore beaucoup à faire.

Nous nous contenterons ici de présenter uniquement le visuel des assets, leurs utilités/fonctionnalités étant expliquées dans la suite du rapport.

2.1.1 Les personnages

Le jeu comporte un joueur homme et un joueur femme. Afin que le design du jeu corresponde à nos attentes, nous avons fait appel a une amie d'Elya qui est en école d'art. Ainsi, grâce à ses connaissances et ses talents artistiques, nous avons d'abord réalisé les premiers aspects des personnages et enfin concrétisé les différentes planches qui ont permis de créer les sprites des joueurs.

FIGURE 2.1 – Premier asset du joueur homme

FIGURE 2.2 – Premier asset du joueur femme

Voici les différentes planches du joueur homme :

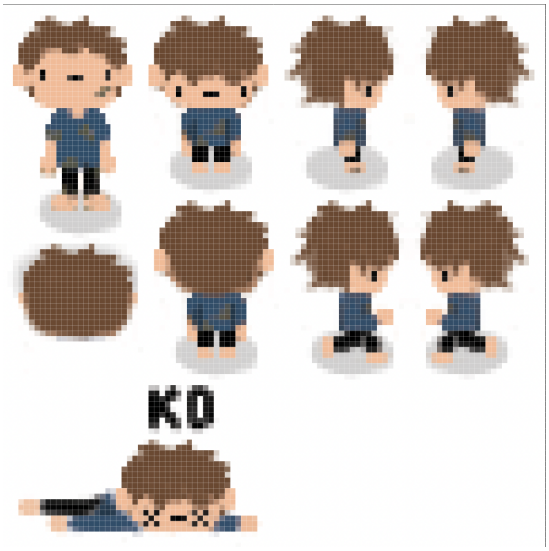

FIGURE 2.3 – Player homme

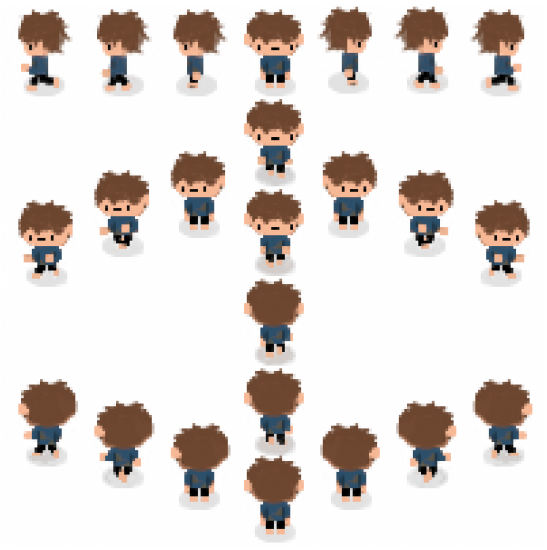

FIGURE 2.4 – Player homme

Voici les planches du joueur femme :

FIGURE 2.5 – Player femme

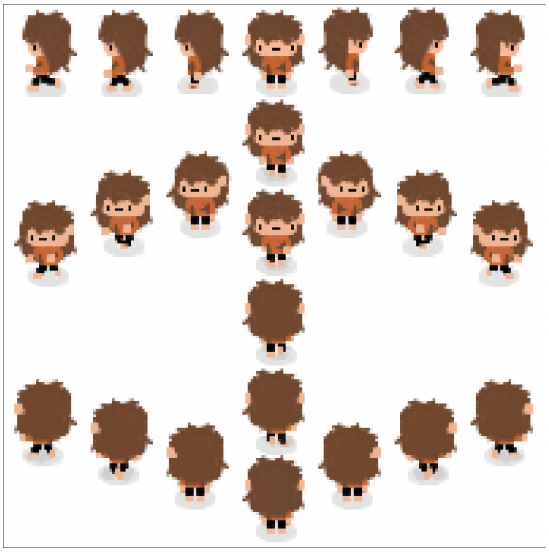

FIGURE 2.6 – Player femme

2.1.2 Les Plaque de pression

Cet asset servira pour les portes, pour changer de salle.
Il a été réalisé par Ethan.

FIGURE 2.7 – Asset « Plaque de pression »

2.1.3 Assets du Labyrinthe de Boîtes

Ces assets serviront pour l'enigme du labyrinthe de boîtes.
Ils ont été réalisés par Ethan.

Boîte que l'on peut bouger

FIGURE 2.8 – Asset « Boîte que l'on peut bouger »

Boîtes impossibles à bouger

FIGURE 2.9 – Asset « Boîte impossible à bouger »

Case spéciale dites : « Tabouret »

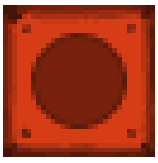

FIGURE 2.10 – Asset « Tabouret »

Les Bennes à Boîtes

FIGURE 2.11 – Asset « Benne »

2.1.4 Logo du jeu et de l'entreprise

Les logos du jeu et de l'entreprise ont été réalisés par Ethan.

Le logo de l'entreprise a pour source d'inspiration celui de SpaceX et a été réalisé sous Photoshop.
Le logo du jeu à été réalisé partir d'une image vectorielle gratuite d'internet.

FIGURE 2.12 – Logo du jeu : Mode sombre

FIGURE 2.13 – Logo du jeu : mode clair

FIGURE 2.14 – Logo de l'entreprise

2.2 Les Personnages

Cette partie a été réalisé par Elya.

Nous avons commencé par faire bouger le personnage verticalement et horizontalement. Ensuite, les direction en diagonales ont été rajoutées.
Enfin les animations ont été additionnées aux mouvements.
Pour cela, nous utilisons le code suivant :

Listing 2.1 – Variables utilisées dans la fonction précédente

```
1  public class playerwalk : MonoBehaviour
2  {
3      public string horizon;
4      public string verti;
5      public Animator animator;
6
7      // Utilisation de FixedUpdate() pour rendre le mouvement du joueur indépendant
           des FPS
8      void FixedUpdate()
9      {
10         Vector3 mouvement = new Vector3(Input.GetAxis(horizon), Input.GetAxis(verti
               ), 0.0f); //creation du mouvement avec horizon=direction
11         animator.SetFloat("Horizontal", mouvement.x); //mise en place de l'
               animation
12         animator.SetFloat("Vertical", mouvement.y);
13         animator.SetFloat("Magnitude", mouvement.magnitude);
14         transform.position = transform.position + mouvement * Time.deltaTime;
```

```
15          //deplacement du joueur (changement de coordonnées du joueur selon un temps
                proportionelle)
16      }
17  }
```

Dans ce code, nous utilisons les variables d'entrée *horizon* et *verti* qui sont les positions sur les axes x et y du joueur. Auxquelles sont associées différentes touches selon le joueur fille ou le joueur garçon. Ainsi, un vecteur sera créé et, selon l'axe, amènera le joueur aux coordonnées demandées en utilisant l'animation correspondante. Pour ce qui est de l'animation, nous avons utilisé l'animator.
Grâce à ce dernier, lorsque le joueur bouge (donc lorsque le script pour avancer se lance), on passe à l'animation mouvement. Une position est associée à chaque animation ainsi lorsque le joueur avance dans une direction précise, l'animation de la direction se met en place. Si le joueur s'arrête il reviens à son design d'origine (ici appelé *immobile*).

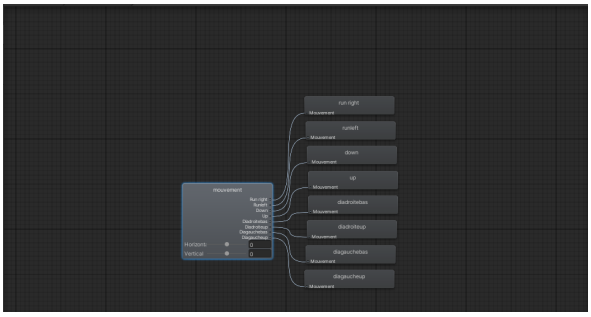

FIGURE 2.15 – Arborescence de l'animation

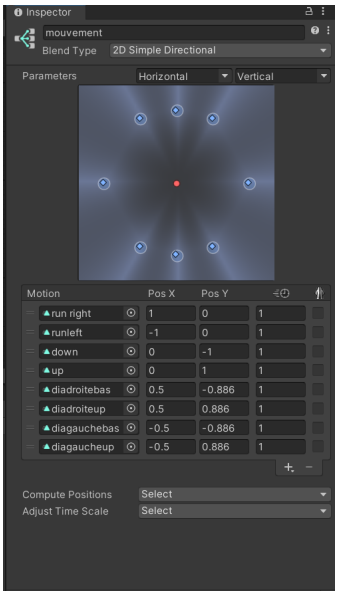

FIGURE 2.16 – Position des animations

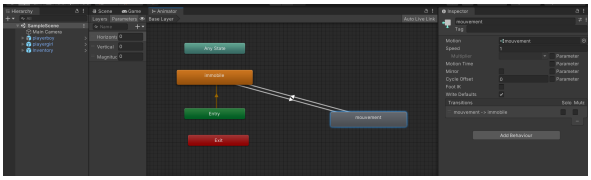

FIGURE 2.17 – Animator

Lors de cette étape nous avons rencontré plusieurs problèmes. A l'origine, la planche de départ ne contenait pas toutes les positions possibles du joueurs, empêchant ainsi la création des sprites pour certaines animations. C'est pourquoi nous possédons maintenant plusieurs planches pour chaque joueurs.

De plus, une barre de vie est associée à chaque joueur. Elle diminue différemment selon la situation. Cette dernière diminue grâce au *fill amount*.

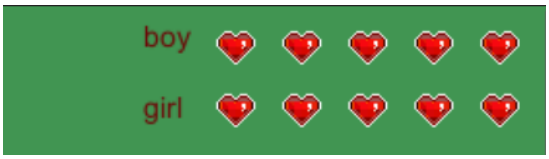

FIGURE 2.18 – Barre de vie

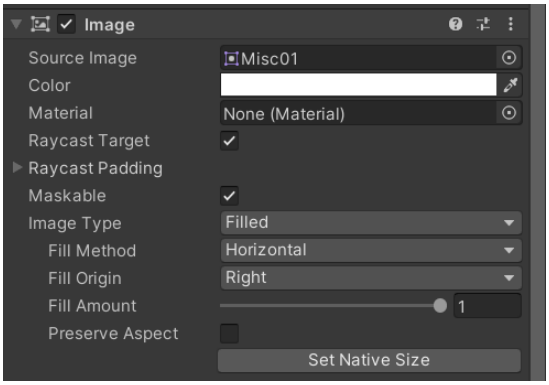

FIGURE 2.19 – Gestion de la diminution

2.3 Les Portes : Changement de Salle

Les avancements de cette section ont été réalisé dans leurs intégralités par Ethan.

2.3.1 Classe Mère : Détection de présence d'un joueur et action

Cette classe sert à détecter la présence d'un player sur l'objet sur lequel se trouve le script.
J'utilise le tag *Player* sur les joueurs pour savoir si l'objet détecté est un joueur.
Je garde en mémoire dans le script si l'objet est pressé (si un joueur est sur l'objet).
Je garde également une liste de tous les players présents sur la plaque bien que cela ne nous serve pas encore.

Listing 2.2 – Fonction qui détecte le joueur

```
/**
* <summary>Appelé lorsque qqc entre en contact avec la plaque.
*   Seulement si c'est un joueur : elle indique que l'on est en contact
*  avec la plaque + call OnPressure()</summary>
*
* <param name="other">objet avec qui est sur la plaque</param>
*
* <returns>Return nothing</returns>
*/
private void OnTriggerEnter2D(Collider2D other)
{
    //Si ce qui est entrée en collision est un joueur
    if (other.tag == "Player")
    {
        //On indique que la plaque est pressée
        pressed = true;
        // On stocke le player (au cas oú qu'on en ai   besoin plus tard
        // pour un ajout de features (mais pour l'instant  a  sert á rien)
        player.Add(other);
        //On fait l'action
        OnPressure(other);
    }
}
```

Le script gère également la sortie du player de la zone d'action de l'objet.
Lorsque la plaque est activé elle appelle une fonction qui sera redéfinie dans les classe fille (grâce à l'attribut virtual)

Listing 2.3 – Fonction a redéfinir dans les classes filles

```
//Fonction OnPressure() appellee si un player marche sur la plaque de pression
//IMPORTANT : cette fonction á pour vocation á etre modifier dans les classes
    filles (avec override)
/**
```

```
4  * <summary>Determine ce s'il faut faire qqc (et quoi) avec le player qui est sur la
       plaque</summary>
5  *
6  * <param name="other">objet avec qui on a collisione</param>
7  *
8  * <returns>Return nothing</returns>
9  */
10 protected virtual void OnPressure(Collider2D other)
11 {
12     return;
13 }
```

2.3.2 Classe Fille : Portes Simples et Doubles

Portes Simples

Cette classe (SingleDoor.cs) se base sur la classe mère PressurePlate.cs
Elle hérite donc de toutes ses propriétés et besoins.

Cette classe redéfinie la fonction *OnPressure()* pour afficher un message sur l'écran du joueur grâce à la classe décrite dans la section suivante. De plus, si le joueur sur la plaque appuie sur la touche 'E' (comme indiquer sur son écran), les joueurs sont téléportés dans la salle suivante.

Portes Double

Cette classe (DoubleDoor.cs) se base sur la classe mère PressurePlate.cs
Elle hérite donc également de toutes ses propriétés et besoins.

Cette classe stocke une référence un autre objet possédant lui aussi cette classe (les portes sont liées entre elles par deux)
Elle permet de séparer les joueurs lorsqu'ils doivent arriver à deux endroits différents dans la pièce suivante.
Cette classe redéfinie la fonction *OnPressure()* pour afficher un message sur l'écran du joueur grâce à la classe décrite dans la section suivante.
Si le joueur est le seul à être sur une plaque alors un message s'affiche pour lui indiquer qu'il faut que les deux joueurs soit sur les deux plaques distinctes pour se téléporté à la salle suivante
Sinon, si les deux joueurs sont chacun sur une plaque différentes alors s'affiche un message qui demande d'appuyer sur la touche 'E'. Si un des deux joueurs sur la plaque appuie sur la touche 'E' (comme indiqué sur l'écran), les joueurs sont téléportés dans la salle suivante.

Listing 2.4 – Fonction redéfinie de la classe mère

```
1  //Fonction OnPressure() appellee si un player marche sur la plaque de pression,
2  //  SI (y a aussi un player sur autrePressurePlate) : elle affiche le message qui
       demande l'interaction
3  //  SINON : elle dit qu'il faut que les deux joueurs soit sur une plaque différente
       pour pouvoir changer de salle
4  /**
5  * <summary>Détermine ce s'il faut faire qqc (et quoi) avec le player qui est sur la
       plaque</summary>
6  *
7  * <param name="other">objet avec qui on a collisioné</param>
8  *
9  * <returns>Return nothing</returns>
10 */
11 protected override void OnPressure(Collider2D other)
12 {
13     // affiche le bon message suivant s'il y a deja qqn sur l'autre plaque de
           pression
14     if (autrePressurePlate.pressed)
15     {
16         //On affiche le message qui indique au joueur comment interagir avec la
               porte.
```

```
17          MessageOnScreenCanvas.GetComponent<FixedTextPopUP>().PressToInteractText("
                Press E to interact with the door");
18      }
19      else
20      {
21          //SINON
22          //TODO : Changer la couleur de la lupiotte
23
24          //On affiche le message qui indique au joueur comment intéragir avec la
                porte.
25          MessageOnScreenCanvas.GetComponent<FixedTextPopUP>().PressToInteractText("
                To interact with the door you should both stand on a different pressure
                 plate");
26      }
27
28  }
```

2.3.3 Canvas : Retour écran

Cette Classe se place sur un Canvas désactivé qui possède un élément texte.
Elle possède deux fonction :

- Une fonction qui affiche n'importe quel texte à l'écran du joueur (le texte passé en argument de la fonction)
- Et une autre qui supprime le texte affiché

Listing 2.5 – Fonction qui permet d'afficher un message sur l'écran du joueur

```
1  //Fonction PressToInteractText() permet d'afficher un message
2  // utilitaire sur l'ecran du joueur
3  /**
4  * <summary>Permet d'afficher un message sur l'ecran du joueur</summary>
5  *
6  * <param name="message">message que l'on souhaite afficher</param>
7  *
8  * <returns>Return nothing</returns>
9  */
10 public void PressToInteractText(string message)
11 {
12     //Cherche l'element texte du texte dans le Canvas
13     gameObject.GetComponentInChildren<Text>().text = message;
14     gameObject.SetActive(true);
15 }
```

2.4 Système de Sauvegarde

Les avancements sur le système de sauvegarde ont été réalisés dans leur intégralité par Ethan.

J'ai réalisé une ébauche du système de sauvegarde. Il est fonctionnel bien que pour l'instant il n'y ai encore rien à sauvegarder, d'où l'appellation d'ébauche. Il enregistrera les données dans un fichier JSON, dans un dossier spécial qui ne sera jamais modifié par Unity, même lors d'une mise à jour. Pour ce faire j'utilise JsonUtility et le stocke dans le dossier « Application.persistentDataPath ».
Les sauvegardes des préférences (volumes et autres) seront enregistrés de façon séparée des autres données telles que l'inventaire et la réussite du joueur. En effet, les préférences ne seront pas modifiées à chaque scène.

Voici un exemple de la fonction qui gère la sauvegarde des préférences des utilisateurs.

Listing 2.6 – Fonction qui sauvegarde les préférences du joueur dans un JSON

```
1  //Fonction SavePlayerSettings() de sauvegarder tous les settings
2  // du player apr s une modification
3  /**
4  * <summary>Permet de sauvegarder de sauvegarder tous les settings du
```

```
5  *  player apr s une modification</summary>
6  *
7  * <returns>Return nothing</returns>
8  */
9  public void SavePlayerSettings()
10 {
11     // On utilise le module JsonUtility pour parse tout l'objet
12     string playerSettingsData = JsonUtility.ToJson(playerSettings);
13
14     // Chemin ou va  tre  enregistrer le JSON
15     // persistentDataPath est un dossier qui ne sera jamais modifier par unity
           m me mise á jour
16     string filePath = Application.persistentDataPath + "/PlayerSettingsData.json";
17
18     // On écrit le fichier
19     System.IO.File.WriteAllText(filePath, playerSettingsData);
20 }
```

Ont aussi été codé les fonctions pour recharger les données dans les classes où elles sont stockées depuis le fichier JSON. Et une fonction qui permet de réinitialiser les sauvegardes (en les supprimant).

2.5 Énigme : Le Labyrinthe de Boîte

Les avancements sur le labyrinthe de boîtes ont été réalisés dans leurs intégralités par Ethan.

Les avancements sont principalement concentrés sur la réflexion sur la logique et sur la création d'un certains nombres d'énigmes (sous format papier). Et de fait, j'ai réfléchi et créé des objets aux mécaniques spécifiques pour diversifier les énigmes et les rendre intéressantes.

Je vais donc m'attacher à décrire ces mécaniques :
Tout d'abord, il y a les boîtes que l'on peut bouger, on pourra les pousser et les tirer.

Figure 2.20 – Asset « Boîte que l'on peut bouger »

Puis, il y a les boîtes que l'on ne peut pas bouger (parce qu'elles sont trop lourdes pour être poussée). Elle se différencient par une croix sur la boîte.

Figure 2.21 – Asset « Boîte impossible à bouger »

Ensuite, il y a les « tabourets », ils sont fixés au sol avec des vis (impossible de les bouger). Une boîte ne peut pas passer dessus mais vous, vous pouvez ! Cela permettra de rajouter un intérêt supplémentaire à l'énigme en créant de nouvelles solutions.

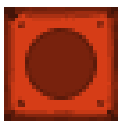

Figure 2.22 – Asset « Tabouret »

Enfin, il y a les bennes à boîtes, on peut pousser des boîtes dedans, elles sont alors détruites. Cela permettra encore une fois de créer de nouvelles mécaniques.

FIGURE 2.23 – Asset « Bennes à boîtes »

J'ai pris un peu d'avance sur cette partie pour pouvoir montrer le projet lors de la journée d'immersion du mercredi 9 mars 2022.
Ainsi, j'ai déjà commencé l'implémentation des mécaniques de base du jeu. Il est possible de pousser les boîtes (qui peuvent être poussée) (elles ne bougent que sur les axes x et y comme c'était voulu pour éviter des bugs lorsque les boîtes doivent passer entre deux murs). Ont aussi été implémenté les boîtes qui ne peuvent pas bouger. Cela m'a permis de réaliser un niveau de démonstration où l'on a seulement besoin de pouvoir pousser les boîtes. Les assets du mur et du sol ne sont pas de nous et sont seulement temporaires.

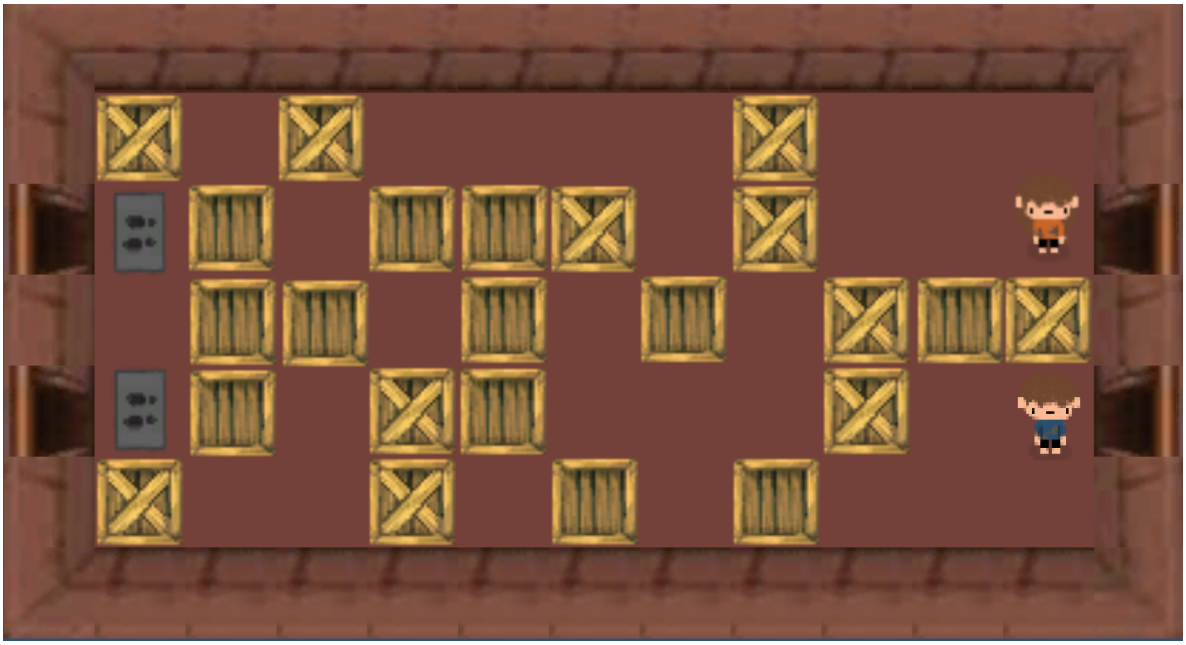

FIGURE 2.24 – Le niveau réalisé pour la démonstration

2.6 Énigme : Le Labyrinthe Invisible

Cette partie a été réalisée par Timothy.

La génération du labyrinthe a été implémenté sous forme d'objet (POO). Ce labyrinthe généré procéduralement utilise le Growing Tree algorithm [1] . Cet algorithme fonctionne de la façon suivante :

1. Marquer comme un chemin une cellule aléatoire dans le labyrinthe
2. Marquer comme un chemin une des cellules voisines pas encore visitée
3. Recommencer l'étape 2. jusqu'à arriver à une impasse
4. Lorsque l'on arrive à une impasse, faire le chemin inverse jusqu'à atteindre une cellule avec des voisins non visités

1. https://weblog.jamisbuck.org/2011/1/27/maze-generation-growing-tree-algorithm

5. Continuer à l'étape 2. jusqu'à que toutes les cellules du labyrinthe soient visitées

Ce labyrinthe est ensuite converti en matrice pour pouvoir le manipuler plus simplement. Pour convenir à nos besoins, une entrée ainsi qu'une sortie sont créées.
Ce labyrinthe est une énigme de coopération : un joueur a une carte du labyrinthe et l'autre joueur doit traverser ce labyrinthe, invisible à ses yeux, à l'aide de son partenaire. Si le joueur dans le labyrinthe marche sur un « mur », il prend des dégâts.

2.6.1 Carte du labyrinthe

Cette carte sera dans le canvas, désactivée par défaut, et activée quand le joueur la consultera. Pour plus de simplicité, chaque GameObject constituant cette carte sera instancié dans un GameObject vide et celui sera activé/désactivé selon nos besoins.
Pour créer la carte du labyrinthe, je parcours la matrice représentant celui-ci, et à chaque mur (représenté par un 1), on instancie un sprite représentant le mur aux coordonnées correspondantes.

Listing 2.7 – Fonction créant la carte du labyrinthe

```
float size = 1;                 // Taille de chaque cellule du labyrinthe dans la scene
int[] offsetMap = {0, 0};       // Offset de la position du labyrinthe (format [x,y])
                                // maze est la matrice représentant le labyrinthe

for (UInt16 i = 0; i < maze.GetLength(0); i++)
{
    for (UInt16 j = 0; j < maze.GetLength(1); j++)
    {
        if (maze[i,j] == 1)
        {
            Instantiate(wall, new Vector2(i * size + offsetMap[0], j * size +
                offsetMap[1]), Quaternion.identity, wallParent);
        }
    }
}

wallParent.SetActive(false); // Désactive la carte par défaut
```

Voici a quoi ressemble la carte du labyrinthe pour l'instant (les sprites utilisés pour les murs sont temporaires) :

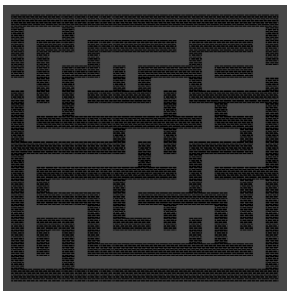

FIGURE 2.25 – Exemple de Labyrinthe 1

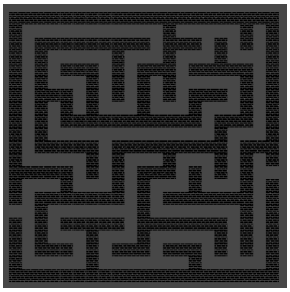

FIGURE 2.26 – Exemple de Labyrinthe 2

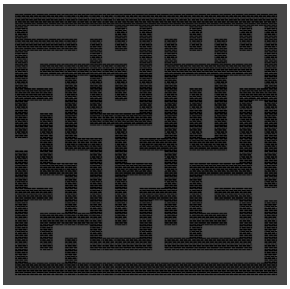

FIGURE 2.27 – Exemple de Labyrinthe 3

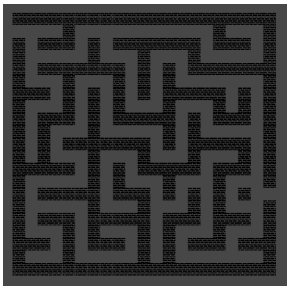

FIGURE 2.28 – Exemple de Labyrinthe 4

2.6.2 Carte du labyrinthe

La position du joueur dans le labyrinthe sera détectée grâce à sa position. Les avantages de cette méthode, est qu'elle ne nécessite qu'un script attaché à un GameObject vide pour fonctionner.

Listing 2.8 – Fonction calculant la position du joueur dans le labyrinthe à partir de ces coordonnés

```
player1Tile[0] = (player.position.x - startPos[0]) / tileSize;
player1Tile[1] = (player.position.y - startPos[1]) / tileSize;
```

Listing 2.9 – Variables utilisés dans la fonction précédente

```
int[] player1Tile;  // Coordonnees du joueur dans le labyrinthe (format [x,y])
                    // Case/Position du joueur : maze[player1Tile[1],player1Tile
                       [0]]
GameObject player;  // Joueur
int[] startPos;     // Coordonnees du labyrinthe dans la scene (offset)
int tileSize;       // Taille de chaque cellule du labyrinthe dans la scene
```

On peut ensuite vérifier si le joueur une sur une case « mur » pour lui infliger des dégâts si c'est le cas.

Listing 2.10 – Variables utilisées dans la fonction précédente

```
if (player1Tile[0] >= 0 && player1Tile[0] < maze.GetLength(1) && player1Tile[1] >=
    0 && player1Tile[1] < maze.GetLength(0))
{   // Vérifie si le joueur est dans le labyrinthe

    if (maze[player1Tile[1],player1Tile[0]] == 1)
    {
        // Damage Player1
    }
}
```

2.7 Énigme : L'énigme des Fils

Cette partie a été réalisée par Hugo.

Cette énigme est fortement inspirée du jeu de coopération "Keep talking and nobody explode". Il y a une série de fils de couleurs différentes qui sont débranchables en cliquant sur leur branchement gauche.
Un autre panneau est constitué de consignes possédant des phrases telles que :

- Si il y a X fil(s) *color* ou plus, débranchez le ième fil
- Si il y a moins de X fil(s) *color*, débranchez le ième fil
- Sinon débranchez le ième fil

Si le joueur coupe un mauvais fil, il se fait électrocuté et perd de la vie en lui laissant un minimum d'un demi-coeur. De sorte à ce qu'il ne meurt pas.
Les règles et l'armoire électrique ne peuvent pas être ouverte en même temps par le même joueur, à chaque ouverture une nouvelle génération aléatoire se fait. De sorte à forcer la coopération.

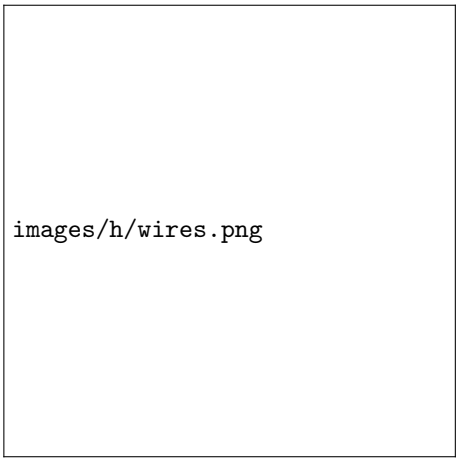

FIGURE 2.29 – Wires

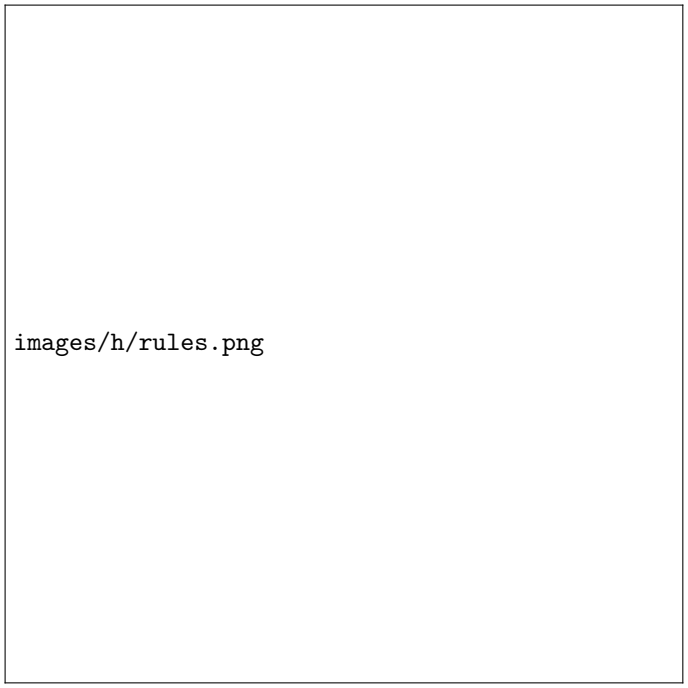

FIGURE 2.30 – Rules

2.7.1 Génération des fils

La génération des fils se fait en commençant par la génération des prises (gameObject rond) puis en ajoutant des fils (lineRenderer) avec une couleur aléatoire qui sont reliés à une prise choisie aléatoirement du côté droit.
On stocke dans un tableau la liste des couleurs des fils par ordre de branchement sur les prises de gauche.
Tous les gameObject de cette énigme sont stockés dans un objet parent appelé « WireEnigmParent » par soucis de lisibilité.

Listing 2.11 – Génération dynamique des prises

```
1     for (int i = 0; i < nbWires; i++)
2     {
3         int y = startY - i * 7;
4         GameObject plugNumber = Instantiate(plugPrefab, new Vector3(startX, y, 5),
              Quaternion.identity, parentObj.transform);
5         GameObject plugLetter = Instantiate(plugPrefab, new Vector3(startX + 30, y,
               0), Quaternion.identity, parentObj.transform);
6         plugsN[i] = plugNumber;
7         plugsL[i] = plugLetter;
8         plugNumber.GetComponent<Plug>().nb = 0;      //set to not unplug
9     }
```

Listing 2.12 – Génération des fils en les croisant

```
1 for (int i = 0; i < nbWires; i++)
2 {
3     int r = Random.Range(0, nbWires - i);
4     GameObject plug2 = plugsL[r];           //take random plug on right side
```

```
5
6       for (int j = r; j < nbWires - 1; j++){//move all element to the left
7           plugsL[j] = plugsL[j + 1];
8       }
9
10      int randomColor = Random.Range(0, 4);   //generate a random color for wire
11
12      GameObject wire = Instantiate(wirePrefab, new Vector3(0, 0, 0), Quaternion.
            identity, parentObj.transform);      //create a new wire
13      plugsN[i].GetComponent<Plug>().wire = wire;
                                                                    //adding the wire
            to the plug script
14      wire.GetComponent<Wire>().Positions = new Transform[2] { plugsN[i].transform,
            plug2.transform };     //adding two plugs to the wire script
15      wire.GetComponent<Wire>().color = randomColor;      //setting the wire color
16      wiresColors[i] = randomColor;       //saving the wire color
17  }
```

2.7.2 Génération des règles

La génération des règles se fait dynamiquement et de manière aléatoire à l'aide de phrases type (données plus haut). Dans cet exemple elles sont affichées dans la console mais dans la pratique elles sont mises dans un canvas.

Listing 2.13 – Variables utilisées dans la fonction précédente

```
1
2   int nbFils, color, unplug, rn, fil;
3   bool finished = false;                  //a rule above is already valid
4   for (int i = 0; i < nbRules; i++)
5   {
6       rn = Random.Range(0, 3);            //choose a random type of rule
7       color = Random.Range(0, nbColors);  //determine the color of the rule
8       unplug = Random.Range(0, nbWires);  //the wire to unplug if rule validated
9       if (rn == 0)
10      {
11          nbFils = Random.Range(2, nbWires / 2);
12          if (findNbColors(color) >= nbFils && !finished) //if conditions valid and
                no condition above is valid set wire to be cut
13          {
14              plugsN[unplug].GetComponent<Plug>().nb = 1;
15              finished = true;
16          }
17          Debug.Log($"Si il y a {nbFils} fil(s) {randomColor(color)} ou plus, dé
                branchez le {nbToWord(unplug + 1)}");
18      }
19      else if (rn == 1)
20      {
21          fil = Random.Range(0, nbWires);
22          if (wiresColors[fil] == color && !finished)
23          {
24              plugsN[unplug].GetComponent<Plug>().nb = 1;
25              finished = true;
26          }
27          Debug.Log($"Si le {nbToWord(fil + 1)} est {randomColor(color)}, débranchez
                le {nbToWord(unplug + 1)}");
28      }
29      else
30      {
31          nbFils = Random.Range(1, nbWires / 2);
32          if (findNbColors(color) < nbFils && !finished)
33          {
34              plugsN[unplug].GetComponent<Plug>().nb = 1;
35              finished = true;
36          }
37          Debug.Log($"Si il y a moins de {nbFils} fil(s) {randomColor(color)}, dé
                branchez le {nbToWord(unplug + 1)}");
38      }
```

```
39 }
40
41 unplug = Random.Range(0, nbWires);
42 Debug.Log($"Sinon débranchez le {nbToWord(unplug + 1)}");
43
44 if (!finished)
45 {
46     plugsN[unplug].GetComponent<Plug>().nb = 1;
47 }
```

Pour la deuxième soutenance l'implémentation de cette énigme sera faite.

2.7.3 Labyrinthe fléché

Cette partie a été réalisée par Hugo.

Ce labyrinthe est constitué de dalles fléchés, le joueur ne peut aller que dans la direction des flèches et ne peux pas aller en opposition à une flèche sur une autre tuile.
De plus les deux joueurs ne verront pas toutes des dalles les obligeant ainsi à communiquer pour pouvoir résoudre ce labyrinthe. (chacun voit la moitié des dalles)
Si un joueur ne respecte pas les règles précédemment citées, il perdra de la vie, et sera possiblement téléporté au début du labyrinthe.

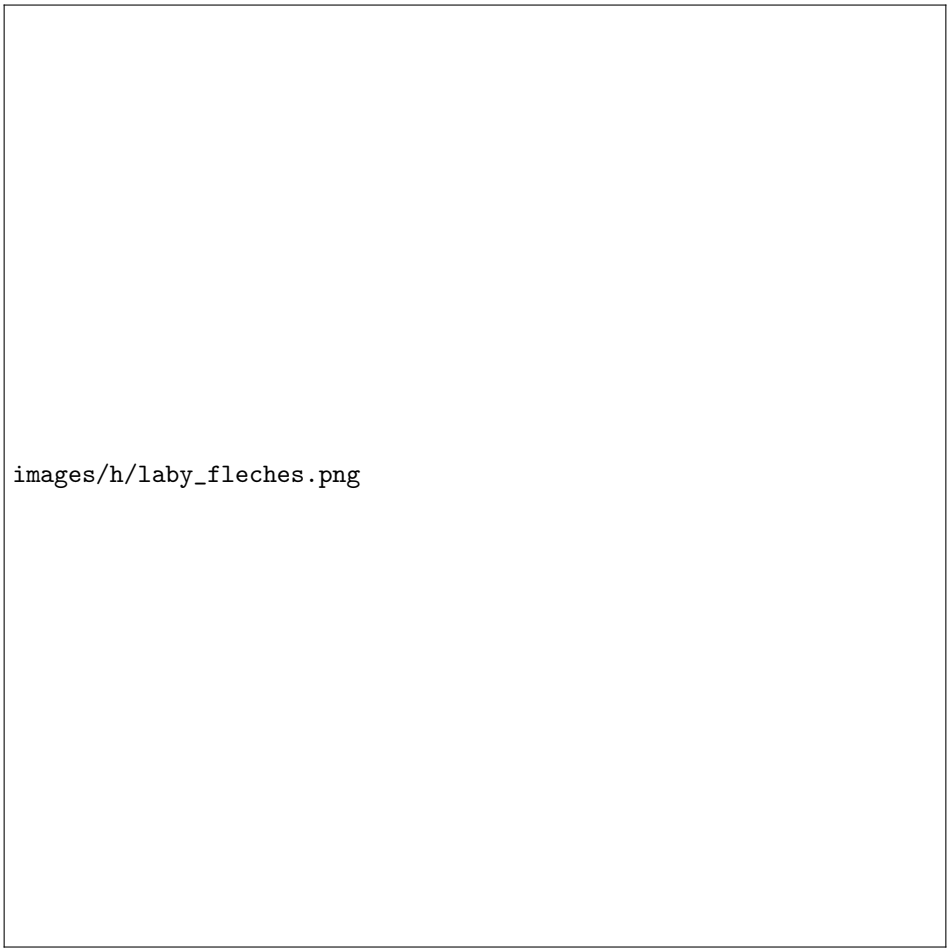

FIGURE 2.31 – Labyrinthe fléché

Cette énigme n'est pas finie, les erreurs des joueurs lors de leurs déplacements ne sont pas gérés et

les labyrinthes générés n'ont pour l'instant pas toujours une solution.
Ceci est prévu pour la deuxième soutenance.

2.8 Génération Procédurale du Donjon

Cette partie a été réalisée par Timothy.

Le donjon généré procéduralement est fait pour accueillir des monstres. Son algorithme est en réalité très simple :
Le script possède 4 tableaux contenant des préfabs de salles à instancier. Chacun de ces tableaux corresponds à une direction (Nord, Est, Sur, Ouest) et chacun des ces tableaux contient des salles ayant au moins une ouverture vers cette direction.
Exemple : Le tableau Nord ne contient que des salles avec une ouverture vers le Nord. Une salle ayant une ouverture vers le Nord et vers l'Est convient. Cette salle sera alors dans le tableau Nord est Est.
Chaque salle a plusieurs GameObject empty : 1 empty pour chaque direction, il indique la position des prochaines salles.

1. On instancie un salle avec une ouverture dans les 4 directions
2. Pour chaque GameObject empty, on instancie une nouvelle salle à la position de la direction correspondante. La salle à instancier provient du tableau de la direction opposée à l'ouverture de la salle déjà présente.
   Exemple : si une salle a une ouverture à l'Est, une nouvelle salle sera créée à sa droite. Cette salle doit donc avoir une ouverture à gauche pour pouvoir relier ces salles par un chemin.
3. Si deux GameObject empty collisionnent entre eux cela signifie que le premier des deux objets vides à être instancié à déjà créé une salle à sa position. L'autre est alors détruit pour éviter que deux salles ne se superposent. Cependant, si ces deux objets ont été créés en même temps, une salle avec aucune ouverture est alors placée à leur position.
4. On répète les étapes 2. et 3. jusqu'à ce qu'il n'y ait plus d'ouverture.

Chacune des salles créées est stockée dans une liste. On peut donc facilement trouver la dernière salle créée par ce programme, et y mettre la fin du niveau.

Voici les salles utilisées par le labyrinthe (les carrés verts représentent les GameObject empty) :

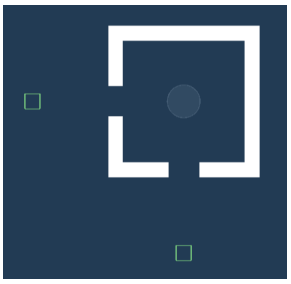

FIGURE 2.32 – Exemple de salle avec ouverture Ouest, Sud

FIGURE 2.33 – Exemple salle avec ouverture Est

Voici des exemples de donjons générés par cette méthode (les sprites utilisés sont temporaires et la tête de mort représente la dernière salle à avoir été créée) :

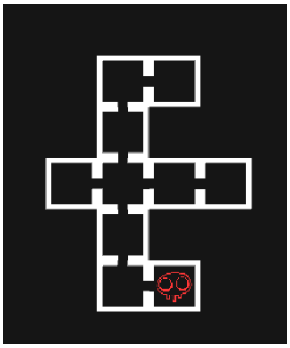

FIGURE 2.34 – Exemple de Donjon 1

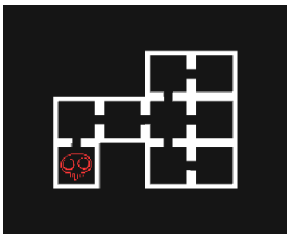

FIGURE 2.35 – Exemple de Donjon 2

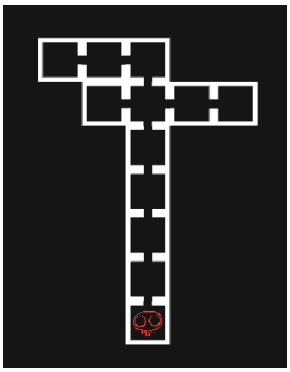

FIGURE 2.36 – Exemple de Donjon 3

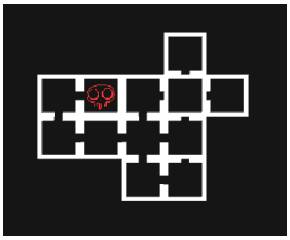

FIGURE 2.37 – Exemple de Donjon 4

2.9 Conclusion

Pour la première soutenance nous devions avoir réalisé :

- Création d'une première énigme : l'énigme des fils.
- Début de l'énigme : le labyrinthe fléché.
- Création et et début d'implémentation d'une deuxième énigme : le labyrinthe invisible.
- Création et implémentation de la génération de donjon procédurale.
- Création et implémentation des animations et des mouvements des joueurs mâle et femelle.
- Création de la barre de vie des joueurs.
- Création et implémentation des portes pour changer de salle.
- Création de la logique et exemple de niveau : Labyrinthe à boites.

Toutes ces tâches ont été réalisée avec succès.

Chapitre 3

Prévision

Pour la soutenance 2, nous devrons avoir réalisé ces différents points :

- Création du Lobby.
- Intégration des premières énigmes dans le jeu.
- Création des premiers monstres.
- Création des premières potions.
- Création du système de Narration.
- Implémentation de l'énigme du labyrinthe à boîtes.
- Début de création du texte de la narration.
- Début de création du site (FrontEnd) : description du projet.
- Création des assets principaux (sols, murs, portes)
- Début création du launcher
- Début création système multijoueur
- Début recherche musique

Chapitre 4

Ressources

Outils et Plateformes utilisés :

Unity • Unity est un moteur de jeu multiplateforme. Il est très populaire dans l'industrie du jeu vidéo, aussi bien pour les grands studios que pour les indépendants.
Nous utiliserons sa version gratuite.

Github • GitHub est un service d'hébergement de code et de gestion de développement de logiciels. Cela nous permettra de pouvoir accèder au fichier du projet facilement, et permettra de coder en simultané sans risque de perte. Cela simplifiera aussi la communication grâce aux issues, pull request, et project view disponible avec GitHub.

Plateforme du CRI • Le projet sera aussi synchronisé sur la plateforme du CRI si cela se révèle être nécessaire grâce à l'outil Gickup [1].

Photoshop • Photoshop est un logiciel de retouche photo, de traitement d'image et de dessin assisté par ordinateur.
Ce logiciel nous sert à créer le logo du jeu ainsi que certains assets.

Bosca Ceoil • Bosca Ceoil est un logiciel gratuit et simple à prendre en main de création de musique. Il nous servira pour réaliser les bruitage de notre jeu.

Overleaf • Overleaf est un éditeur LaTeX en ligne, collaboratif en temps réel.
Il nous servira pour rédiger les diverses comptes-rendus demandés.

Discord • Discord est un logiciel gratuit de messagerie instantanée.
Il nous sert à communiquer entre nous, de nous tenir au courant de l'avancée du projet, de poser des questions si un membre s'en pose.

1. `https://github.com/cooperspencer/gickup`

Chapitre 5

Conclusion

Ainsi nous sommes très satisfait de notre avancement sur le projet. En effet, nous avons pu réalisé tout ce que nous avions prévu de faire et nous avons même eu le temps de prendre un peu d'avance sur notre planning. Nos énigmes ont bien avancé chacune de leur côté, l'objectif, pour la prochaine soutenance, est donc maintenant de commencer à relier les énigmes entre elles et de débuter l'implémentation des monstres et des fioles. Nous terminerons en citant Henry Ford : « Qu'est-ce qui conditionne la réussite ? La capacité à soutenir un effort continu. » Alors à nous de continuer les efforts.