ALTERAVITA

Rapport de Soutenance Finale

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 Présentation succincte du projet

1.1.1 Type du projet

Notre projet est un jeu vidéo 2D créé sous Unity en C

1.1.2 Accroche

Vous vous réveillez dans une cellule, en face de vous, derrière un mur de verre se trouve un autre prisonnier. Clothilde, une ancienne étudiante de l'EPITA, vous aide à vous évader. Malheureusement, elle ne parvient pas à déverrouiller la porte principale du labo. Il vous faudra donc en trouver les clefs. Mais pour cela il faudra affronter les monstres créés par les scientifiques lors de leurs manipulations géniques et résoudre les énigmes qui permettent de passer d'une salle du labo à l'autre... Y parviendrez-vous ?

1.2 Le Groupe

1.2.1 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

timothy.didierjean : Timothy DIDIERJEAN

1.2.2 Gestion Organisationnelle

Le groupe se réunit lors de réunions hebdomadaires tous les lundis soir ou mardis 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 indique 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.

1.3 Histoire

1.3.1 Synopsis

Dans un monde, où la médecine ne cesse d'évoluer, vous vous réveillez enfermé(e) 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 !

1.3.2 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 percevoir 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'êtes pas seul... Clothilde est là ! Vous vous sentez tout de suite rassuré(e)s. 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'avancent et vous saisissent vous et votre compagnon par les bras. Vous vous laissez faire, trop choqué(e)s 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 écrans 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 aidait 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 situe 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.

1.3.3 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 né(e) par césarienne, il y a quelques dizaines d'années, vous et votre mère seraient inévitablement morts. De nombreuses maladies autrefois mortelles sont désormais sans importance. On peut même guérir d'un cancer [1] s'il est détecté assez tôt !
Et pour repousser sa fin, l'Homme est prêt à tout, le temps étant sa ressource la plus chère car fatalement limitée.

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 cliniques, 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 ?

1.4 Mécaniques de GamePlay

1.4.1 Présentation Générale

Heureusement pour vous vous n'êtes pas seul(e), 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.

1. https ://www.cancer.be/le-cancer/de-la-r-mission-la-gu-rison/gu-rison-du-cancer

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

1.4.2 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 murs 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.

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 téléporté au début du labyrinthe.

Connecte les produits chimiques

But : Relier les tuyaux composants chimiques circuits de création de fioles du laboratoire (en les faisant tourner) pour déverrouiller la porte.

Ce jeu s'inspire des jeux « Connect the Water ».
En effet la salle a été verrouillée car les produits chimiques se sont mélangés ! Dépêcher vous car les produits créent une réaction toxique qui vous tuera au bout d'un certain temps.
Les joueurs sont séparés et des tuyaux passent à travers le mur, ils doivent donc tous deux résoudre leur partie de l'énigme pour réussir le niveau.
Il y a une limite de temps, après quoi les joueurs perdront de la vie jusqu'à ce que mort s'en suive.
Pour sortir de la salle, il faut que chaque joueur se retrouve sur une plaque de pression, en face d'une porte, après avoir réussi l'énigme.
Si les joueurs meurent, c'est une condition d'échec.

1.4.3 Les Monstres

Les monstres seront des ennemis se situant sur notre chemin un peu partout dans les niveaux. Ils auront tous un lien avec leur environnement : le laboratoire. La taille, force, vie des monstres ainsi que leurs attaques varient d'un monstre à l'autre et d'un endroit à un autre.

Les différents monstres sont :

Slime : Un des vestiges des expériences précédentes des scientifiques.
Ils peuvent être de différentes tailles, cela influe sur ses statistiques.
Ce monstre combat au corps-à-corps.

Robots : Ils sont chargés de protéger le laboratoire et d'éliminer les intrus tels que vous.
Il y a plusieurs types de robots, de différentes formes.
Leurs attaques varient, certains attaquent au corps-à-corps, d'autres à distance.

1.4.4 Les Fioles

Les fioles sont le fruit de plusieurs années d'expériences menées par les scientifiques de ce laboratoire. Elles donnent plusieurs effets : des bonus comme un regain de vie par exemple, mais certains ratés peuvent également donner un malus au joueur comme la perte de PV.

Les effets recensés par les scientifiques sont les suivants :

- Effet de soin
- Augmentation de la force
- Effet de dégâts

A cause de ce large nombre d'effets négatifs, la production de ces fioles ont cessé. De plus, certains effets n'ont pas encore été découvert par les scientifiques.

1.4.5 Les Portes (système de changement de salle)

Il y a deux possibilités lors d'un changement de salle :

- Soit les deux joueurs doivent emprunter la même porte :
  Dans ce cas, lorsqu'un joueur marche sur la plaque de pression, un message s'affiche lui proposant d'appuyer sur une touche pour interagir avec la porte. S'il appuie, les deux joueurs sont alors téléportés vers la salle située derrière la porte.
- Soit les deux joueurs doivent emprunter chacun une porte :
  Dans ce cas, il faut que les deux joueurs soit chacun sur les deux plaques de pression menant à la salle suivante. Un message s'affiche alors, proposant d'appuyer sur une touche pour interagir avec la porte. Si un des deux joueurs appuie, les deux joueurs sont alors téléportés vers la salle située derrière la porte.

1.4.6 Narration : La Voix

La narration se fera par l'intermédiaire d'une voix arrivant par les hauts parleurs présents dans toutes les pièces.

La voix permettra en outre de l'aspect de narration du jeu d'expliquer le fonctionnement de certains objets et de certaines énigmes.

1.4.7 Mise en Place du Mode Multijoueur

Le multijoueur en ligne utilisera Photon PUN [2] dans sa version gratuite, un outil populaire et grandement utilisé dans la création de jeux en ligne sur Unity. Ce service est utilisé par de grands éditeurs de jeux tels que Ubisoft, SEGA, Square Enix, Bandai Namco.

L'implémentation de Photon Pun dans notre projet se fera à l'aide de leur asset gratuit disponible sur le Unity Asset Store.

1.4.8 Conditions de Victoire

Il faut avoir récupéré un certain nombre de clefs pour pouvoir ouvrir la porte de sortie du laboratoire. Précisément, une clef par niveau.

1.4.9 Conditions de Défaite

Si les deux joueurs meurent lors d'une énigme ou d'un combat contre un monstre, ils sont simplement téléportés au lobby.

Cependant, plusieurs niveaux de difficultés sont disponibles et une option de « mort permanente » pourra être sélectionné. Dans ce cas, si les deux joueurs meurent lors d'une énigme ou d'un combat contre un monstre, le jeu est fini. C'est une défaite, il faudra recommencer depuis le début.

1.4.10 Traitement du Score

Le score est un nombre censé refléter les capacités des binômes s'étant échappé du laboratoire. Pour cela ils devront développer leurs capacités de coordination, de réflexion, leur force, leur endurance et leur réflexe. Ce score est utilisé par les scientifiques afin de classer les clones en fonction leur compétences et de leurs « compatibilités » avec les personnes voulant les acquérir.

Ce score sera calculé à la fin du jeu l'aide d'une formule prenant en compte plusieurs paramètres tels que :

2. https ://www.photonengine.com/en/PUN

- Le niveau de difficulté (hardcore ou non)
- Le nombre de mort
- Le temps mis à s'échapper du laboratoire
- Le nombre de dégâts subis
- Le nombre de monstres abattus

Cette formule reste encore à déterminer, et les paramètres sur lesquels elle se base peuvent également différer.

Chapitre 2

Rappel Des Objectifs pour la Soutenance Finale

Pour la soutenance finale, nous devions avoir réalisé ces différents points :

TABLE 2.1 – Répartition des Responsabilités pour la Soutenance Finale

Tâches	Elya	Ethan	Hugo	Timothy
Ajout des musiques				R
Création et Implémentation de l'énigme « Connecte les produits chimiques »		R		
Intégration des dernières énigmes dans le jeu			R	
Mise en place du système de combat joueur contre monstres	R			
Asset Spécifiques (détails, objet)	R			
Finalisation du FronEnd du site		R		
Finalisation du backend du site			R	
Fin d'implémentation système multijoueur				R
Finalisation du launcher			R	
Finalisation et implémentation du script de narration		R		

Légende :

Symbole	Signification
Responsable	R
Suppléant	S

Chapitre 3

Avancement

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

Le code montré dans ce rapport est modifié par rapport au fichier sources afin de le rendre plus compréhensible.

3.1 Player

Cette partie a été réalisé par Timothy et Elya

3.1.1 Mode en Local : Player

Partie réalisé par Elya.
Lors de la dernière soutenance, le Player possédait un début d'inventaire. J'ai créé une classe inventaire et une classe image associé à l'inventaire et les panels qui le composent pour le finalier.

Mode Local : L'Inventaire

Listing 3.1 – Classe de l'inventaire

```
1
2  public class inventory : MonoBehaviour
3  {
4      bool activation = false;
5      public GameObject panel;
6      public int[] slot;
7
8      PhotonView view;
9
10     // Start is called before the first frame update
11     void Start()
12     {
13         GetComponent<Canvas>().enabled = false;
14         slot = new int[panel.transform.childCount];
15         view = GetComponent<PhotonView>();
16     }
17
18     // Update is called once per frame
19     void Update()
20     {
21         if (view.IsMine && Input.GetKeyDown(KeyCode.I))
22         {
23             activation = !activation;
24             GetComponent<Canvas>().enabled = activation;
25         }
26     }
27 }
```

La fonction `UpdateNumber` augmente ou diminue l'affichage du nombre de potions.

Listing 3.2 – Classe Image

```
public class image : MonoBehaviour
{
    inventory inventaire;
    public bool PotionStrength = false;
    public bool PotionDamage = false;
    public bool PotionHealth = false;
    // Start is called before the first frame update
    void Start()
    {
        inventaire = GameObject.Find("Inventory").GetComponent<inventory>();
    }

    public void ShowImage()
    {
        int slot_number = transform.parent.GetSiblingIndex(); //number of slots

        if (inventaire.slot[slot_number]>0)
        {
            inventaire.slot[slot_number] -= 1;
            inventaire.UpdateNumber(slot_number, inventaire.slot[slot_number].
                ToString());
            switch (slot_number)
            {
                case 0://strength
                    PotionStrength = true;
                    break;

                case 1 ://damage
                    PotionDamage = true;
                   break;

                case 2://health
                    PotionHealth=true;
                    break;
            }
        }
    }
}
```

Cette classe est associée aux images des potions qui composent l'inventaire. La fonction `Start` récupère l'inventaire, et la fonction `ShowImage` est la fonction qui permet que l'utilisation des potions lorsque le joueur clique sur l'image.

Mode Local : Les Potions

Lors de la dernière soutenance, les potions disposaient seulement d'une animations et d'une classe associée. Mais pour les utiliser ils faut tout d'abord les récupérer c'est pour cela que la classe Player a été modifiée.

Listing 3.3 – Detection de potion

```
    void OnCollisionEnter2D(Collision2D colision)
    {
        switch (colision.gameObject.tag)//take the tag of the object
        {
            case "slot"://strength potion
                inventaire.slot[0] += 1;
                inventaire.UpdateNumber(0, inventaire.slot[0].ToString());//add to
                    the inventory
                Destroy(colision.gameObject);// destroy the GameObject of the scene
                break;

            case "slot (1)"://damage potion
```

```
12                  inventaire.slot[1] += 1;
13                  inventaire.UpdateNumber(1, inventaire.slot[1].ToString());
14                  Destroy(colision.gameObject);
15                  break;
16
17              case "slot (2)"://health potion
18                  inventaire.slot[2] += 1;
19                  inventaire.UpdateNumber(2, inventaire.slot[2].ToString());
20                  Destroy(colision.gameObject);
21                  break;
22          }
23      }
```

Cette fonction permet que, lorsque le joueur entre en contact avec une des potions, il la récupère, supprime l'objet de la scène, et incrémente le nombres de potions dans l'inventaire.

Listing 3.4 – Utilisation des potions

```
1  void Update()
2      {
3          if (GameObject.Find("Image").GetComponent<image>().PotionStrength)//
               Strength potion
4          {
5              Strength = Strength * 1.2;//increase the Strength
6              GameObject.Find("Image").GetComponent<image>().PotionStrength = false;
                   //remove the object of the inventory
7          }
8          if (GameObject.Find("Image(1)").GetComponent<image>().PotionDamage )//
               damage potion
9          {
10
11                 Vector2 mouvement = new Vector2(Input.GetAxis(GetComponent<
                       playerwalk>().horizon), Input.GetAxis(GetComponent<playerwalk
                       >().verti));//create a mouvement for the potion
12                 mouvement.Normalize();
13                 GameObject potion = Instantiate(potionPrefab, transform.position,
                       Quaternion.identity);
14                 potion.GetComponent<Rigidbody2D>().velocity = mouvement * 3.0f;//
                       throw the potion
15                 Destroy(potion, 2.0f);//destroy the object on the map
16
17                 GameObject.Find("Image(1)").GetComponent<image>().PotionDamage =
                       false;//remove the object of the inventory
18         }
19         if (GameObject.Find("Image(2)").GetComponent<image>().PotionHealth)//
               health potion
20         {
21             Potion potion1 = new Potion(Potion.Type.Heal, 5);
22             potion1.Effect(this, vie); //heal the palyer
23             GameObject.Find("Image(2)").GetComponent<image>().PotionHealth = false;
24         }
25         if (!vie.die)
26         {
27             animator.SetFloat("Die", 0);//Remove the die animation
28         }
29         if(vie.die)
30         {
31             animator.SetFloat("Die", 1);//make the die animation
32         }
33         StartCoroutine(waiter());//make wait 30 before put back Strength to 15
34     }
```

Lors de la fonction `Update`, la potion est utilisée. Selon la potion, une action différente est effectuée :
La potion de vie (le kit de secours) appelle la fonction `Effect` de la classe potion qui remet la barre de vie au maximum ou le réssucite s'il est mort.
La potion d'attaque est lancer sur le monstre.
La potion de force qui augmente la force du joueur.
De plus, dans cette même fonction, l'animation de mort est mise en place si la barre de vie du joueur

est minimum.
Mais la potion de force ne fait effet que durant 15 secondes qui est appelée avec la fonction `StartCoroutine(waiter())` dans `Update`.

Listing 3.5 – chronomètre de la potion de force

```
IEnumerator waiter()
    {
        if (Strength > 15)
        {
            yield return new WaitForSeconds(15);
            Strength = 15;
        }
    }
```

Ici, la fonction fait attendre 15 secondes avant de remettre la force du joueur par défaut à 15.

3.1.2 Mode en Ligne : Player

Partie réalisé par Timothy.
Lors de la dernière soutenance, le Player de base fonctionnait déjà en ligne, mais celui-ci n'avait pas encore de vie ou d'inventaire, il pouvait seulement se déplacer. Cela a été fait à l'aide de plusieurs scripts fournis par Photon PUN [1] :

Photon View : Permet au Player d'être visible sur les différentes instances du jeu.

Photon Transform View : Permet de synchroniser le composant Transform du Player.

Photon Animator View : Permet de synchroniser les animations du Player.

De plus, le script `playerWalk.cs`, s'occupant de faire bouger le Player, a du être modifié :

— Avant la partie du script s'occupant déplacement du joueur a été rajoutée une condition, afin de savoir si le Player *appartient* au joueur. Cela a été fait avec l'attribut `isMine` [2] du script Photon View.

Listing 3.6 – Déplacement du joueur en ligne

```
if (photonView.isMine)
{ // C'est notre joueur
    Move(); //Déplacement
}
```

— Les caméras des autres joueurs sont désactivées afin d'éviter de voir une caméra autre que la sienne.

Listing 3.7 – Désactivation des autres caméras

```
foreach (GameObject player in GameObject.FindGameObjectsWithTag("Player))
{
    if (!player.GetComponent<PhotonView>().isMine)
    { // Si ce n'est pas mon joueur
        player.GetComponentsInChildren<Camera>().enabled = false; // Desactive la
            camera du joueur
    }
}
```

Mode en Ligne : La Vie

La vie des joueurs a été synchroniser entre les instances du jeu avec des fontions RPC [3], sinon les points de vie des mêmes joueurs peuvent différer entre les instances du jeu.

1. Photon PUN est un Asset permettant d'implémenter un jeu en ligne dans Unity.
2. cf. documentation Photon View : True if the PhotonView is *mine* and can be controlled by this client.
3. Une méthode RPC est une fonction pouvant être executée sur les différentes instances du jeu en ligne d'une même session.

Mode en Ligne : L'Inventaire

Cette partie a été réalisée par Timothy.
Par simplicité, il a été choisit de ne pas synchroniser les inventaires des 2 joueurs. La seule fonction qui a changé pour le multijoueur et celle qui affiche l'inventaire lorsque l'on appuie sur la touche `I`, une condition vérifiant que c'est bien *notre* Player a été rajouté :

Listing 3.8 – Affichage de l'inventaire

```
if (photonView.IsMine && Input.GetKeyDown(KeyCode.I))
{ // La touche I a été appuyé et c'est l'inventaire de notre joueur
    ShowInventory(); // Montre l'inventaire du joueur
}
```

L'inventaire est synchronisé entre les scènes, les joueurs ne perdront donc pas leurs potions. Cela a été fait avec le `GameManager`[4] qui stocke le nombre de potions actuels entre les scènes.

Listing 3.9 – Synchronisation de l'inventaire

```
void Start()
{
    gameManager = GameObject.FindGameObjectWithTag("GameManager").GetComponent<
        GameManager>();

    if (PhotonNetwork.IsMasterClient) // Sers a differencier les joueurs
    {
        inventaire.slot[index] = gameManager.potions[0, index]; // Cherche les
            nombres de potions chez le gameManager pour la case numera index de l'
            inventaire
    }
    else
    {
        inventaire.slot[index] = gameManager.potions[1, index];
    }

    inventaire.UpdateNumber(index, inventaire.slot[index].ToString()); // Met a
        jour le texte de l'inventaire
}

private void Update()
{
    if (PhotonNetwork.IsMasterClient)
    {
        gameManager.potions[0, index] = inventaire.slot[index];
    }
    else
    {
        gameManager.potions[1, index] = inventaire.slot[index];
    }

    inventaire.UpdateNumber(index, inventaire.slot[index].ToString()); // Met a
        jour le texte de l'inventaire
}
```

Mode en Ligne : Potions

Cette partie a été réalisée par Timothy.
Afin de rendre compatible les potions avec le mode multijoueur en ligne, il suffisait de leur attacher le scripts `Photon View` et `Photon Animator View` afin de synchroniser leurs animations.
Les potions sont aussi synchronisées entre les différentes scènes du jeu grâce au `GameManager`, de la m^m*emanirequelespotionsdel'inventaire*.

4. Le `GameManager` est un script qui gère l'ensemble du jeu, comme par exemple la gestion des scènes à charger à quel moment.

3.1.3 Mode en Local : Monstres

Partie réalisé par Elya.

Animations des monstres

Pour faire avancer le monstre, il nous fallait de nouveaux assets [5] pour se déplacer :

FIGURE 3.1 – Robot poing de face 1

FIGURE 3.2 – Robot poing de face 2

FIGURE 3.3 – Robot poing droite 1

FIGURE 3.4 – Robot poing droite 2

FIGURE 3.5 – Robot poing droite 3

FIGURE 3.6 – Robot poing gauche 1

FIGURE 3.7 – Robot poing gauche 2

FIGURE 3.8 – Robot poing gauche 3

FIGURE 3.9 – Robot poing de dos 1

FIGURE 3.10 – Robot poing de dos 2

FIGURE 3.11 – Robot poing de dos 3

FIGURE 3.12 – Robot fusil de face 1

FIGURE 3.13 – Robot fusil de face 2

FIGURE 3.14 – Robot fusil de face 3

FIGURE 3.15 – Robot fusil droite 1

FIGURE 3.16 – Robot fusil droite 2

FIGURE 3.17 – Robot fusil droite 3

5. Un asset est la représentation de tout élément pouvant être utilisé dans votre jeu ou projet. Une ressource peut provenir d'un fichier créé en dehors de Unity, tel qu'un modèle 3D, un fichier audio, une image ou tout autre type de fichier pris en charge par Unity.

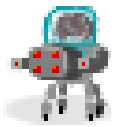


FIGURE 3.18 – Robot fusil gauche 1

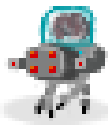


FIGURE 3.19 – Robot fusil gauche 2

FIGURE 3.20 – Robot rouge fusil de face

FIGURE 3.21 – Robot fusil rouge droite

FIGURE 3.22 – Robot fusil rouge gauche

FIGURE 3.23 – Robot rouge fusil de dos

FIGURE 3.24 – Robot de face

FIGURE 3.25 – Robot droite

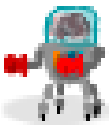


FIGURE 3.26 – Robot gauche

FIGURE 3.27 – Robot de dos

Pour créé les animations de chacun des monstres, j'ai utilisée l'animator [6].

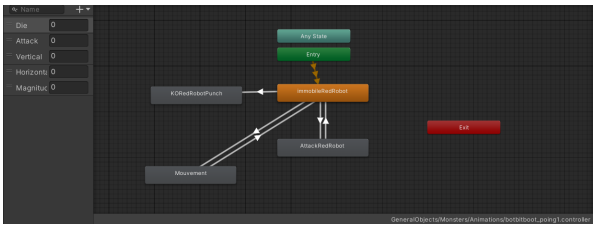


FIGURE 3.28 – Animator d'un robot

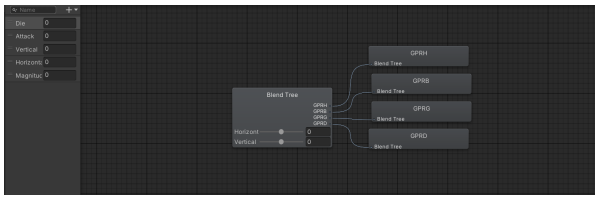


FIGURE 3.29 – Ensemble des animations de déplacement du robot

Déplacement des monstres

Pour faire en sorte que les monstres suivent le joueur et le déplacent vers lui la classe monstre a été modifié.

Listing 3.10 – fonction de detection du player

```
1  void DetectPlayer()
2     {
3         if(Time.time-PlayerDetectTime>PlayerDetectRate)//can the player be detect
4         {
5             PlayerDetectTime = Time.time;//detect the player on the moment
6             foreach(playerwalk player in FindObjectsOfType<playerwalk>())//research
                   all the player
7             {
8                 if(player!=null)
9                 {
```

6. L'animator est un outil de unity qui sert à donner des animations à un GameObject.

```
                    float dist = Vector2.Distance(transform.position, player.
                        transform.position);//go to the player

                    if(player == targetPlayer)
                    {
                        if(dist>chaseRange)
                        {
                            targetPlayer = null;//no player on the path
                        }
                    }

                    else if (dist < chaseRange)
                    {
                        if (targetPlayer == null)
                            targetPlayer = player;//change the player
                    }
                }
            }
        }
    }
```

Cette fonction permet de détecter s'il y a des joueurs dans la zone de détection du monstre en parcourant la listes des joueurs présents dans la scène.

Listing 3.11 – Fonction de déplacement du monstre

```
private void FixedUpdate()
{
    if (targetPlayer != null && !dead )
    {
        float dist = Vector2.Distance(transform.position, targetPlayer.transform.
            position);// distance
        if (dist > attackRange)//check if the distance
        {
            if (path == null)
                return;
            if (currentPath >= path.vectorPath.Count)//check the good path
            {
                reachEndPath = true;
                return;
            }
            else
            {
                reachEndPath = false;
            }
            Vector2 direction = ((Vector2)path.vectorPath[currentPath] - rb.
                position).normalized;//create the direction
            Vector2 force = direction * speed * Time.fixedDeltaTime; //create a
                mouvement
            walk(direction);

            rb.velocity = force;//move
            float distance = Vector2.Distance(rb.position, path.vectorPath[
                currentPath]);
            if (distance < NextWayPointDistance)//check the distance between the
                next position and the current one
            {
                currentPath++;
            }
        }
        else
            rb.velocity = Vector2.zero;//stop the monster
    }
    DetectPlayer();
}
```

Cette fonction permet de détecter le joueur et, si un joueur est dans la zone de détection du monstre, alors ce dernier se dirige vers lui. Enfin, les animations de déplacement sont mis en place par la fonction `Walk`.

Listing 3.12 – Fonction Walk

```
public void Walk(Vector2 mouvement)
{
    animator.SetFloat("Horizontal", mouvement.x);//mise en place de l'animation
    animator.SetFloat("Vertical", mouvement.y);
    animator.SetFloat("Magnitude", mouvement.magnitude);
}
```

L'IA des monstres

Les monstres sont capables de choisir quel joueur attaquer.
Au début, ils s'en prendront au joueur le plus proche. Ensuite, ils attaquent les dernier joueur les ayant attaqués.

Listing 3.13 – Changement de cible par le monstre lors d'un collision

```
void OnCollisionEnter2D(Collision2D colision)
{
    if (colision.gameObject.tag == "Player") // Si le joueur touche le monstre
        if (!dead && !onePlayer) // Si le monsre n'est pas mort et il reste plus d'
            un joueur en vie
            targetPlayer = colision.gameObject.GetComponent<playerwalkOnline>();
            // La cible est ce joueur
}
```

Si il reste uniquement un joueur en vie, les monstres l'attaqueront.

Listing 3.14 – Détection du nombre de joueurs en vie

```
GameObject[] players = GameObject.FindGameObjectsWithTag("Player");
int i = 0; // Nombre de joueurs en vie
foreach (var item in players)
{
    if (!item.GetComponent<playerOnline>().vie.die) // Si le joueur n'est pas en
        vie
        i++;
}
if (i == 1) // Il y'a uniquement un joueur en vie
{
    onePlayer = true;
    foreach (var item in players)
    {
        if (!item.GetComponent<playerOnline>().vie.die)
            targetPlayer = item.GetComponent<playerwalkOnline>(); // Cibler le
                joueur en vie
    }
}
else
    onePlayer = false;
```

3.1.4 Mode en Ligne : Monstres

Cette partie a été réalisée par Timothy.
De base, les monstres semblaient fonctionner en ligne, en effet, les joueurs étant synchroniser sur les 2 instances du jeux, les monstres semblaient attaquer les mêmes joueurs, recevoir et infliger les mêmes dégâts. Le problèmes était que quelques fois, les point de vie des monstres différaient d'un ou plusieurs points de vie à travers les instances du jeu. Pour palier cela, des fonctions RPC [7] été utilisés.

7. Une méthode RPC est une fonction pouvant être executée sur les différentes instances du jeu en ligne d'une même session.

3.1.5 Mode en Local : Système de combat

Partie réalisée par Elya

Le système de combat se compose en plusieurs parties : Le joueur peut attaquer grâce a des potions.
Le joueur peut diminuer de peu la vie du monstre en rentrant en collision avec ce dernier.
Le monstre attaque lorsque qu'il rentre en contact avec un joueur.
Donc j'ai donc modifié la classe player.

Listing 3.15 – Fonction de collision du player

```
void OnCollisionEnter2D(Collision2D colision)
{
    switch (colision.gameObject.tag)//take the tag of the object
    {
        case "slot"://strength potion
            inventaire.slot[0] += 1;
            inventaire.UpdateNumber(0, inventaire.slot[0].ToString());//add to the
                inventory
            Destroy(colision.gameObject);// destroy the GameObject of the scene
            break;

        case "slot (1)"://damage potion
            inventaire.slot[1] += 1;
            inventaire.UpdateNumber(1, inventaire.slot[1].ToString());
            Destroy(colision.gameObject);
            break;

        case "slot (2)"://health potion
            inventaire.slot[2] += 1;
            inventaire.UpdateNumber(2, inventaire.slot[2].ToString());
            Destroy(colision.gameObject);
            break;

        case "Monster":
            colision.gameObject.GetComponent<Monstre>().GetDamage((float)(Strength
                /2));//make damage on monster
            break;
    }
}
```

En rajoutant les trois dernières lignes, le joueur retire la moitié de sa force à la vie du monstre dès qu'ils entrent en collision.

Listing 3.16 – Lancé de potion dans la fonction update du player

```
void Update()
{

    if (GameObject.Find("Image(1)").GetComponent<image>().PotionDamage )//damage
        potion
    {

            Vector2 mouvement = new Vector2(Input.GetAxis(GetComponent<playerwalk
                >().horizon), Input.GetAxis(GetComponent<playerwalk>().verti));//
                create a mouvement for the potion
            mouvement.Normalize();
            GameObject potion = Instantiate(potionPrefab, transform.position,
                Quaternion.identity);
            potion.GetComponent<Rigidbody2D>().velocity = mouvement * 3.0f;//throw
                the potion
            Destroy(potion, 2.0f);//destroy the object on the map

            GameObject.Find("Image(1)").GetComponent<image>().PotionDamage = false;
                //remove the object of the inventory
    }

}
```

En rajoutant ces quelques lignes dans dans la fonction `Update` alors des que le joueur utilise les potions d'attaques de son inventaire alors un mouvement sera créé dans la direction du joueur et sera lancer dans cette même direction. Si elle rencontre un monstre cette dernière retire a la vie du monstre une valeur égale a la force du joueur. Dans tout les cas cette dernière sera détruite.

Listing 3.17 – Fonction de collision du monstre

```
1   void OnCollisionEnter2D(Collision2D colision)
2     {
3         switch (colision.gameObject.tag)//take the tag of the object
4         {
5             case "Potion"://damage potion
6                 playerwalk current_player = ChangeTarget(GameObject.
                      FindObjectsOfType<playerwalk>()[0]);
7                 GetDamage((float)current_player.GetComponent<player>().Strength);//
                      have damage
8                 Destroy(colision.gameObject);
9                 break;
10            case "Player":
11                Attack();//attack animation
12                colision.gameObject.GetComponent<player>().GetDamage();//reduce
                      player life
13                break;
14        }
15
16    }
```

Au moment où le joueur et le monstre entrent en contact, l'animation d'attaque du monstre se met en place puis réduit de un demi-coeur la vie du joueur.

3.2 Les Portes : Changement de Salle

3.2.1 Soutenance 1

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

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 3.18 – Fonction qui détecte le joueur

```
1  /**
2  * <summary>Appelé lorsque qqc entre en contact avec la plaque.
3  *   Seulement si c'est un joueur : elle indique que l'on est en contact
4  *  avec la plaque + call OnPressure()</summary>
5  *
6  * <param name="other">objet avec qui est sur la plaque</param>
7  *
8  * <returns>Return nothing</returns>
9  */
10 private void OnTriggerEnter2D(Collider2D other)
11 {
12     //Si ce qui est entrée en collision est un joueur
13     if (other.tag == "Player")
14     {
15         //On indique que la plaque est pressée
16         pressed = true;
17         // On stocke le player (au cas oú qu'on en ai   besoin plus tard
18         // pour un ajout de features (mais pour l'instant  a  sert á rien)
19         player.Add(other);
20         //On fait l'action
```

```
21          OnPressure(other);
22      }
23  }
```

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 3.19 – Fonction a redéfinir dans les classes filles

```
1  //Fonction OnPressure() appellee si un player marche sur la plaque de pression
2  //IMPORTANT : cette fonction á pour vocation á etre modifier dans les classes
       filles (avec override)
3  /**
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 }
```

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 3.20 – 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 }
```

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 3.21 – 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 }
```

Soutenance finale

Cette partie a été réalisé par Hugo.

J'ai modifié les portes afin qu'elles puissent fonctionner dans un environnement multi joueur. La logique a aussi légèrement changé avec cette mise à jour.
En effet, maintenant les plaques, quand elles sont pressé et que l'utilisateur a appuyé sur la touche E, appellent une fonction dans le game manager, qui prendra des actions en conséquences.
Les portes du côté client, doivent appeler une fonction intermédiaire dans un script de l'objet manager de la scène actuel afin de pouvoir avoir accès au game manager du maître.
Tous ce qui est vient d'être expliqué et illustré dans le code ci-dessous.

Listing 3.22 – Code d'une plaque de pression seule

```
void Update()
{
    if (isActive && Input.GetKeyDown(KeyCode.E))
    {
        //call gamemanager function going through manager if not master
        if (PhotonNetwork.IsMasterClient) gameManager.DoorUpdate(1, false);
        else GameObject.FindGameObjectWithTag("Manager").GetComponent<GlobalScript
            >().DoorUpdate(1, false);
        canvas.GetComponent<FixedTextPopUP>().PressToInteractText("Level is not
            finished"); //add text if error
        isActive = false;
    }
}

public void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.tag == "Player" && collision.GetComponent<PhotonView>().IsMine)
    {
        isActive = true;
        canvas.GetComponent<FixedTextPopUP>().PressToInteractText("Press E to
            interact with the door");
    }
    //setup nextscene door for client and overwrite it if is one other door in
        update
    if(collision.tag == "Player") if (NextScene != null) gameManager.NextSceneDoor
        = NextScene;
}

public void OnTriggerExit2D(Collider2D collision)
{
    if (collision.tag == "Player" && collision.GetComponent<PhotonView>().IsMine)
    {
        isActive = false;
        //inform the game manager that a user have left a plate
        if (PhotonNetwork.IsMasterClient) gameManager.DoorUpdate(indent, false);
        else GameObject.FindGameObjectWithTag("Manager").GetComponent<GlobalScript
            >().DoorUpdate(indent, false);

        canvas.GetComponent<FixedTextPopUP>().SupprPressToInteractText();   //
            remove text
    }
}
```

3.3 Énigmes

3.3.1 Connecte les Produits Chimiques

L'Enigme et sa génération à été réalisée par Ethan.
Le multijoueur sur cette énigme a été réalisé par Hugo.

Soutenance finale

Présentation Le principe de l'énigme et que l'on doit faire pivoter des tuyaux pour que les produits chimiques passant à l'intérieur suivent le bon chemin j'usqu'à la sortie. Il y a trois couleurs de produits chimiques différents.

Principe de la génération Le « Labyrinthe » de tuyaux est généré de façon procédurale. Ainsi, une varaible permet de choisir la taille de labyrinthe souhaitée ce qui permet de rendre leur résolution plus ou moins longue.

On définit aléatoirement trois entrées et les tuyaux suivent un chemin aléatoire jusqu'à arriver à l'autre bout de la salle. Les tuyaux peuvent aller tout-droit, faire un angle ou se croiser (mais s'il se croisent il doivent nécessairement se croiser en allant tout-droit. Pour gérer les cas où les tuyaux se bloquent tous seul lors de la génération j'utilise une méthode de backlog qui permet de revenir à un

moment où il n'y avait pas de problème et de recommencer en empruntant un autre chemin. Lors de la génération du chemin pour éviter que la résolution ne devienne trop complexe j'ai interdit aux tuyaux de revenir sur leurs pas. Il peuvent ainsi : monter, aller à droite ou aller à gauche. Ainsi, l'énigme est toujours solvable car je commence par générer 3 bons chemins pour les tuyaux.

Ensuite, j'ajoute d'autres tuyaux aléatoirement sur le terrain pour créer des chemins alternatifs qui ne mèneront potentiellement à rien et complexifier la résolution de l'énigme.

Cette construction se fait en mode PPO (Programmation Orientée Objet) avec une class contenant l'intégralité du labyrinthe composée de PCTile une autre class qui définit chaque tile du labyrinythe.

Affichage Je commence par transformer toute les tiles du labyrinthe avec class Tuyaux (qui définit est gère l'affichage et les intéraction entre le layrinthe et le joueur. Par défaut, tous les tuyaux vides.

Ensuite je crée une rangée de vitre autour pour aérer le labyrinthe et enfin j'instancie le sol autour et les murs ainsi que les portes et les plaques de pression (tout est obligatoirement instancier par le script car la taille du labyrinthe est définie par un variable).

Traitement des couleurs de tuyaux Ce traitement ce fait à chaque rotation d'un tuyau.

A chaque rotation de tuyau, on commence par réinitialiser les couleurs pour tous les tuyaux. Ensuite, on ajoute la couleur sur les sources. Puis, pour chaque tuyaux suivant, on regarde la direction du liquide, et on appelle la fonction qui gère le changement de la couleur du tuyaux (si le liquide arrive dans la bonne direction et surtout si il y a un tuyaux sur la tile). Le traitement s'arrête dès que l'on envoie du liquide sur un case vide ou que le tuyaux ne l'accepte pas.

Listing 3.23 – Fonction appelant le prochain tuyau

```
1  private void NextTuyauxColor(PCTile.PCFluidDirection pCFluidDirection, PCTile.
       PCFluidColor color)
2      {
3          //A chaque fois on va inverser la direction du tuyaux parce que on a en
               param tre la direction de sortie de CE tuyaux et qu'on veut la
               direction d'entre du tuyau PROCHAIN
4          switch (pCFluidDirection)
5          {
6              case PCTile.PCFluidDirection.Down:
7                  if (CoordY + 1 < Map.TuyauxMaze[0].Length && Map.TuyauxMaze[CoordX
                       ][CoordY + 1] != null)
8                  {
9                      //On renvoie une direction différente parce que l'ont veux
                           savoir d'oú arrive le liquide sur la case
10                     Map.TuyauxMaze[CoordX][CoordY + 1].ColorUpdate(PCTile.
                           PCFluidDirection.Up, color);
11                 }
12                 break;
13             case PCTile.PCFluidDirection.Left:
14                 if (CoordX - 1 >= 0 && Map.TuyauxMaze[CoordX - 1][CoordY] != null)
15                 {
16                     Map.TuyauxMaze[CoordX - 1][CoordY].ColorUpdate(PCTile.
                           PCFluidDirection.Right, color);
17                 }
18                 break;
19             case PCTile.PCFluidDirection.Up:
20                 if (CoordY - 1 >= 0 && Map.TuyauxMaze[CoordX][CoordY - 1] != null)
21                 {
22                     Map.TuyauxMaze[CoordX][CoordY - 1].ColorUpdate(PCTile.
                           PCFluidDirection.Down, color);
23                 }
24                 break;
25             case PCTile.PCFluidDirection.Right:
26                 if (CoordX + 1 < Map.TuyauxMaze.Length && Map.TuyauxMaze[CoordX +
                       1][CoordY] != null)
27                 {
28                     Map.TuyauxMaze[CoordX + 1][CoordY].ColorUpdate(PCTile.
                           PCFluidDirection.Left, color);
29                 }
30                 break;
31             case PCTile.PCFluidDirection.End:
32                 //c bon pour un tuyaux (le liquide arrive á la fin
```

```
33                  Map.nbPipeOk++;
34                  //si c bon pour les trois tuayux
35                  if (Map.nbPipeOk == 3)
36                  {
37                      //le jeux est finit !
38                      Map.EndOfGame();
39                  }
40                  break;
41              default:
42                  break;
43          }
44      }
```

Ci-dessus, la fonction qui appelle le rafraîchissement de la couleur du tuyau suivant en fonction de la direction du fluide. Et également une référence à la couleur en cours de traitement.

Le jeu ne bug pas non plus si l'on fait un boucle avec 2 couleurs, une des deux couleurs sera simplement dominante sur l'autre.

Ci-dessous, un extrait du traitement de la couleur pour un simple tuyau droit.

Listing 3.24 – Extrait de la fonction gérant l'affichage de la couleur sur le tuyau

```
1  /**
2       * <summary>Ajoute le liquide dans le tuayux actuel (si il viens de la bonne
            direction) (et si le liquide passe, appel la prochaine case pour une é
            ventuelle actualisation)</summary>
3       *
4       * <param name="commingFrom">Sens d'oú provient le liquide</param>
5       * <param name="color">Couleur en cours de traitement</param>
6       */
7      public void ColorUpdate(PCTile.PCFluidDirection commingFrom, PCTile.
           PCFluidColor color)
8      {
9          PCTile.PCFluidDirection pCFluidDirection;
10         //S'il faut mettre de la couleur
11         switch (TileData.TileType)
12         {
13             case PCTile.PCTileType.Strait:
14                 //Si on est sur un tiuyaux droit
15                 //Si le fluid viens de la bonne direction (peut importe le sens)
16                 if (commingFrom == fluidCommingDirection || commingFrom ==
                       fluidDirection)
17                 {
18                     switch (color)
19                     {
20                         case PCTile.PCFluidColor.blue:
21                             this.GetComponent<SpriteRenderer>().sprite =
                                   Strait_Blue;
22                             break;
23                         case PCTile.PCFluidColor.pink:
24                             this.GetComponent<SpriteRenderer>().sprite =
                                   Strait_Rose;
25                             break;
26                         case PCTile.PCFluidColor.green:
27                             this.GetComponent<SpriteRenderer>().sprite =
                                   Strait_Green;
28                             break;
29                         default:
30                             break;
31                     }
32                     //Suivant le sens de circulation du fluide, le liquide sort d'
                           un coté ou de l'autre
33                     if (commingFrom == fluidCommingDirection)
34                     {
35                         pCFluidDirection = fluidDirection;
36                     }
37                     else
38                     {
39                         pCFluidDirection = fluidCommingDirection;
40                     }
41                     //Appel du prochain tuyau
```

```
                    NextTuyauxColor(pCFluidDirection, color);
                }
                break;
```

Le tuyaux accepte le liquide par ses deux potentielle entrée et en fonction de la couleur du liquide qui y passe change son affichage pour la bonne image le représentant.

Gestion de la condition de réussite La condition de réussite est recalculée à chaque fois que l'on met à jour l'affichage des couleurs des tuyaux. En effet, à chaque fois que le tuyaux suivant correspond à la fin, une variable (réinitialisée avant la mise à jour) est incrémentée, si elle atteint le chiffre fatidique qu'est trois. Alors cela veut dire que tous les liquides de tous les tuyaux arrivent à la fin de leur chemin. Ainsi, le jeu est finit et la porte de sortie est déverrouille.

Comme ce calcul est réalisée à chaque update, la porte se reverrouille si un tuyau est touché.

Multijoueur L'implémentation du multijoueur pour cette énigme a été particulièrement compliqué en effet la complexité du niveau a poussé Ethan a avoir une réflexion et un code orienté objet. Hors, photon ne supporte pas le passage d'objet à travers les fonctions PUN (fonction qui se fait appeler à travers le réseau) et ne permet pas d'éditer les propriétés des objets sur le réseau non plus.
J'ai donc du décortiquer le code et créé de nouvelles fonctions regroupant toutes les propriétés des objets, en l'occurrence les tuyaux pour les créé du côté du client de toute pièce, fonction en annexe ci-dessous.

De plus au lieu d'appeler en local l'évènement de rotation sur son objet, je le remonte au script parent qui l'appelle ensuite sur tous les clients avec les coordonnées.
Chaque client arrive donc lui même à orienter les tuyaux de la même manière, faire les calculs pour les couleurs et à déduire si le niveau est finis ou non.

Listing 3.25 – Fonction PUN générant les tuyaux sur le réseau

```
[PunRPC]
public void GlobalInstantiatePipe(int coX, int coY, int secondY, int source, int
    direction, int from, int val)
{
    PCTile tile;
    if (from == 0) tile = new PCTile((PCTile.PCTileType)source, (PCTile.
        PCFluidDirection)direction, (PCTile.PCFluidDirection)val);  //multi
        direction
    else tile = new PCTile((PCTile.PCTileType)source, (PCTile.PCFluidDirection)
        direction);

    Tuyau pipe = Instantiate(tuyau, new Vector3((float)0.32 * coX - (float)0.16, (
        float)0.32 * coY + (float)0.16, 0), Quaternion.identity).GetComponent<Tuyau
        >();
    pipe.TileData = tile;
    pipe.MessageOnScreenCanvas = canvaTextPopUP;
    pipe.AffichageUpdate();
    pipe.Map = this;
    pipe.InitaliseRotation(coX, secondY);

    if (from == 2) pipe.ColorUpdate(PCTile.PCFluidDirection.None, (PCTile.
        PCFluidColor)val); //source
    else Tuyaux.Add(pipe);

    if(from == 3) tuyauxMaze[coX][secondY] = pipe; //exit
    else tuyauxMaze[coX][secondY] = pipe;
}
```

3.3.2 Énigme du Labyrinthe de Boîte

Soutenance 1

Les avancements sur le labyrinthe de boîtes lors de la première soutenance avait été réalisés dans leurs intégralités par Ethan.

Les avancements étaient 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 me suis 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 3.30 – 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 3.31 – 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.

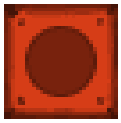


FIGURE 3.32 – Asset « Tabouret »

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 3.33 – Asset « Bennes à boîtes »

J'avais 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'avais déjà commencé l'implémentation des mécaniques de base du jeu. Il était déjà 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). Avaient 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.

Soutenance 2

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

Création des Objets pour l'Enigme Tous les objets nécessaires à cette énigme ont été créé.

Boite Immobile La boîte immobile, à l'aider d'un simple BoxCollider2D [8], elle empêche le joueur de passer sur la boîte (l'objet bloque son avancement, forçant ainsi le joueur à le contourner).

Boite qui bouge La boîte qui bouge a deux fonctionnalités, elle doit pouvoir être poussée si le joueur marche dans sa direction et tirée si le joueur appuie sur une touche pour la tirer.

Pour pousser la boîte, j'ai utilisé un simple BoxCollider2D un Rigidbody2D [9]

Pour tirer la boîte, j'ai eu besoin d'un autre BoxCollider2D (légèrement plus grand) qui, lorsque le joueur entre en collision, active un script qui permet de lier la boîte au player s'il appuie la touche demandée. La liaison est réalisée avec un FixedJoint2D [10] et d'un script qui lie/délie la boîte

Listing 3.26 – Script de la boîte qui bouge

```
//Appelée a chaque frame
private void Update()
{
    //check si on appuie sur E ssi en contact avec la boîte est préssée
    if (Input.GetKeyDown(KeyCode.E))
    {
        //si qqn entre en contact avec la boîte ET que la boîte n'est pas déjá liée á qqn
        if (pressed && !linkedToPlayer)
        {
            // on la lie au perso
            Link();
        }
        //sinon, si la boîte est déjá liée á ce qqn, on lui fais l cher la boîte
        else if (linkedToPlayer)
        {
            UnLink();
        }
    }
}

/**
* <summary>Affiche le message pour proposer á l'utilisateur d'intéragir</summary>
*
* <param name="other">objet avec qui on a collisioné (ici le joueur)</param>
*
* <returns>Return nothing</returns>
*/
protected override void OnPressure(Collider2D other)
{
    //On affiche le message qui indique au joueur comment intéragir avec la porte.
    //Grace á fonctionnalité lock le message ne s'affichera que si c pas lock
    MessageOnScreenCanvas.GetComponent<FixedTextPopUP>().PressToInteractText("Press E to start pulling the box");
}

/**
* <summary>Lier l'objet avec le premier player rentré en collision avec l'objet</summary>
*
* <returns>Return nothing</returns>
*/
private void Link()
{
    //le premier joueur qui est entré en collision avec la boîte stocké dans PressurePlate.cs
    playerLinked = player[0].gameObject;

```

8. Composant de Unity qui permet de gérer les collision avec des Objets
9. Autre composant de Unity qui permet de gérer la position d'un objet en fonction des forces qui s'appliquent sur lui
10. Egalement un élément de Unity qui permet de lier entre eux, deux objets et donc de lier leur déplacement, boîtes de collisions, etc.

```
45      //on récupere le component qui permet de faire le lien
46      FixedJoint2D lienActuel = playerLinked.GetComponent<FixedJoint2D>();
47      //On stocke la position de la boîte et du joueur
48      //Cela permet d'éviter que le point d'ancrage fasse bouger la boîte est le
            joueur au moment de l'attache
49      Vector2 posBox = this.transform.position;
50      Vector2 posPlayer = playerLinked.transform.position;
51      //On active le component
52      lienActuel.enabled = true;
53      lienActuel.anchor = posBox;
54      lienActuel.connectedAnchor = posPlayer;
55      //On lie au component le rigidbody de la boîte
56      lienActuel.connectedBody = this.GetComponent<Rigidbody2D>();
57
58      //on indique que le lien a été établi avec succ s
59      linkedToPlayer = true;
60  }
61
62  /**
63  * <summary>Délier l'objet du player avec lequel il était lié</summary>
64  *
65  * <returns>Return nothing</returns>
66  */
67  private void UnLink()
68  {
69      //on récupere le component qui permet de faire le lien
70      FixedJoint2D lienActuel = playerLinked.GetComponent<FixedJoint2D>();
71      //On le desactive
72      lienActuel.enabled = false;
73      //On réinitialise le lien
74      lienActuel.connectedBody = null;
75
76      //On réinitialise les varaibles
77      linkedToPlayer = false;
78      playerLinked = null;
79  }
```

Dans ce script, est géré par la fonction `OnPressure()` l'affichage d'un message sur le Canvas dans le cas où l'on entre en contact avec une boîte. Par la fonction `Update()` est gérée la détection des touches pressées et si le script doit lier ou délier la boîte du joueur (en fonction de s'il était déjà liée ou non à la boîte). Ainsi la fonction `Link()` lie le joueur à la boîte et la fonction `Unlink()` le délie.

Un autre script ajoute également au chargement de la scène un élément FixedJoint2D aux joueurs pour pouvoir les lier à la boîte.

Listing 3.27 – Script qui ajoute les liens aux joueurs

```
1  // Start is called before the first frame update
2  void Start()
3  {
4      // Initialisation de moyen pour lier les boîtes et le joueur lorsque l'on veut
           tirer une boîte
5      foreach (GameObject player in GameObject.FindGameObjectsWithTag("Player"))//
           pour tout les players
6      {
7          // On ajouter un <FixedJoint2D> et on met autoConfigureConnectedAnchor á
               false (sinon  a  lie pas les objets)
8          // POur lier les obj sur chaque boîte y aura un script qui permettra de
               lier la boîte au FixedJoint2D ou de les délier (avec touche E)
9          // Attention : faudra penser á verifier que y a pas déjá une boîte accroché
               e...
10         player.AddComponent<FixedJoint2D>();
11         player.GetComponent<FixedJoint2D>().enabled = false;
12         player.GetComponent<FixedJoint2D>().autoConfigureConnectedAnchor = false;
13     }
14 }
```

Le Tabouret Le tabouret à été réalisé l'aide d'un BoxCollider2D et d'un script lui permet de bloquer la progression d'une boîte mais pas celle du joueur.

Listing 3.28 – Script du tabouret

```
//Fonction OnCollisionEnter2D() appelé á chaque collision
/**
* <summary>Appelé a chaque collision : permet que le joueur ignore les collisons avec cet objet, mais pas les autres objets</summary>
*
* <param name="collision">objet qui est entré en collision avec l'objet qui porte le script</param>
*
* <returns>Return nothing</returns>
*/
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.tag == "Player")
    {
        Physics2D.IgnoreCollision(collision.gameObject.GetComponent<Collider2D>(), this.GetComponent<Collider2D>());
    }
}
```

Ici le script va regarder à chaque collision avec le BoxCollider2D si l'objet qui est rentré en collision porte le *Tag*[11] : « Player », si c'est le cas il le laisse entre dans sa boîte de collision, autrement non.

La Benne à boîte La benne à boîte a subit un redesign car l'ancien n'était pas assez explicite. Au fond de la poubelle, il y a maintenant de la lave qui permet de comprendre plus facilement qu'une boîte pousser sur la benne sera détruite.

FIGURE 3.34 – La nouvelle benne

Elle a également été réalisé à l'aide d'un BoxCollider2D et d'un script.

Listing 3.29 – Script de la benne à boîte

```
// Si qqc rentre en contact avec le dumpster
/**
* <summary>Détruit l'objet qui entre en collision si c'est une boĭte (appellé auto par unity)</summary>
*
* <param name="other">objet entrer en collision avec</param>
*
* <returns>Return nothing</returns>
*/
private void OnTriggerEnter2D(Collider2D other)
{
    //si le qqc est une boĭte
    if (other.tag == "Box")
    {
        //On détruit l'objet boĭte.
        Destroy(other.gameObject);
    }
}
```

Lorsque l'objet entre en collision avec la benne, si c'est une boîte, il est détruit.

11. Étiquette en français

Création de la Salle La scène utilisée est la même pour tous les niveaux du labyrinthe, les boîtes, tabourets et bennes sont posées par un script et le jeu possède une banque d'énigmes stockée en fichier JSON [12].

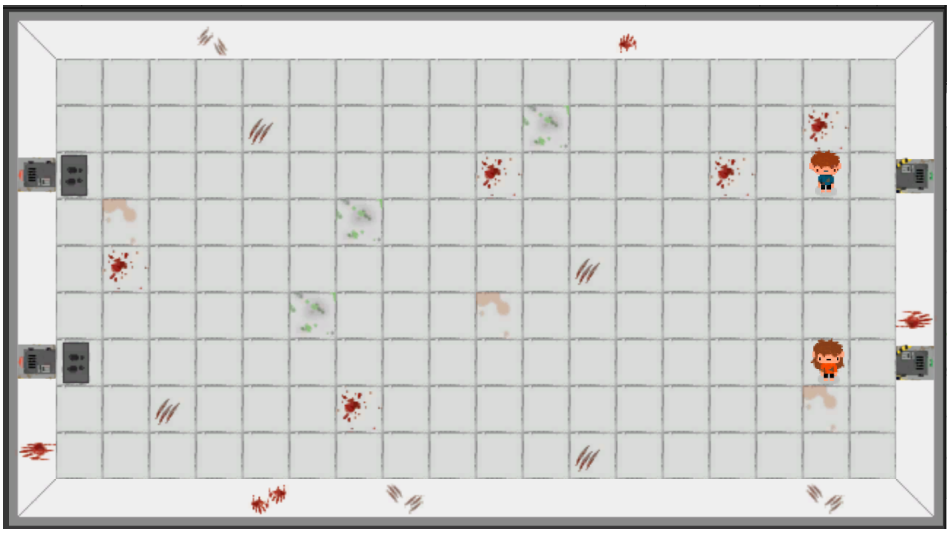


FIGURE 3.35 – La salle du labyrinthe de boîtes vide

Design de la salle vide La scène vide possède simplement le design (porte et sols), les murs qui empêche le joueur de sortir de la zone et les plaques de pression (codées lors de la première soutenance) qui permettent de se téléporter dans la salle suivante.

Banque d'énigme La première liste de coordonnées correspond aux boîtes qui bougent, la deuxième à celles qui ne bougent pas, la troisième aux tabourets et la dernière aux bennes.

Listing 3.30 – Script de la benne à boîte

```
1 private List<Vector3[]> getData(string filename)
2 {
3     //Lecture du fichier JSON
4     var fileContent = Resources.Load<TextAsset>("CrateRooms/" + filename);
5     //Analyse du fichier
6     // Cree une liste de liste de vector3
7     List<Vector3[]> roomData = JsonConvert.DeserializeObject<List<Vector3[]>>(
          fileContent.text);
8     return roomData;
9 }
```

Le script charge les données sous forme de tableau de tableau de Vecteur3 [13]. Les fichiers de données sont stocké dans un dossier ressource qui est compilé avec le jeu lors du build.

12. JSON est un format de données textuelles permettant de représenter de l'information structurée
13. Permet de définir un Vecteur dans un espace tridimensionnel dans Unity

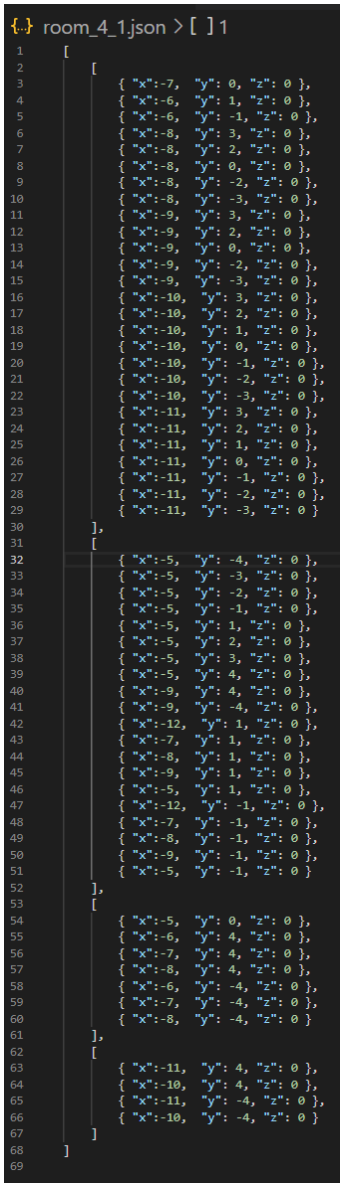


FIGURE 3.36 – Représentation d'une salle dans la base de données

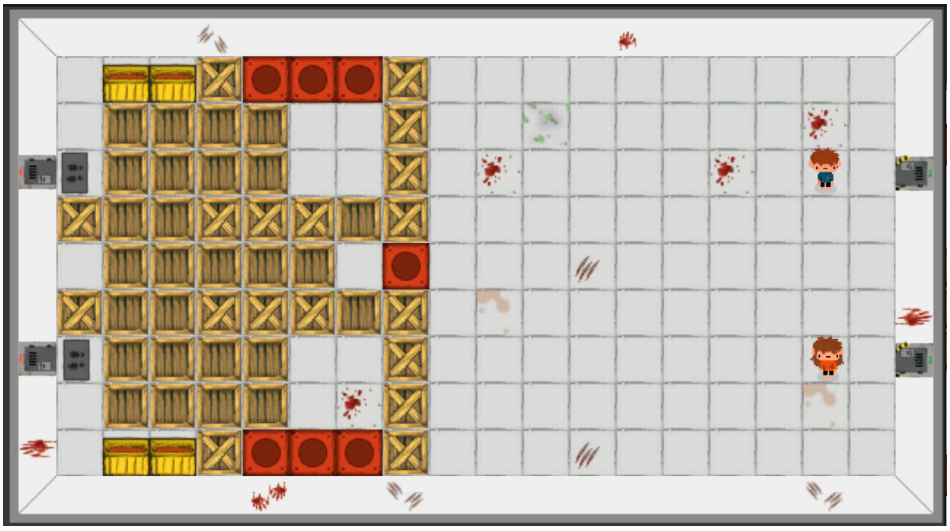


FIGURE 3.37 – La salle du labyrinthe de boîtes générée à partir de données de la figure ci-dessus

Listing 3.31 – Script de la benne à boîte

```
1  public void loadScene(string roomToLoad)
2  {
3      //Récupérer dans une variable le canvas d'ineraction
4      GameObject canvaTextPopUP = GameObject.Find("TextPopUpCanvas");
5
6      //Chargement des coordonnées des boîtes de la pi ce
7      List<Vector3[]> roomData = getData(roomToLoad);
8
9      //Instanciation des Boites qui bouge
10     foreach (Vector3 coord in roomData[0])
11     {
12         Instantiate(movableCratePrefab, new Vector3((float)0.32 * coord.x - (float)
               0.16, (float)0.32 * coord.y + (float)0.16, 0), Quaternion.identity);
13     }
14
15     //Instanciation des Boites qui bouge pas
16     foreach (Vector3 coord in roomData[1])
17     {
18         Instantiate(unmovableCratePrefab, new Vector3((float)0.32 * coord.x - (
               float)0.16, (float)0.32 * coord.y + (float)0.16, 0), Quaternion.
               identity);
19     }
20
21     //Instanciation des tabourets
22     foreach (Vector3 coord in roomData[2])
23     {
24         Instantiate(stoolPrefab, new Vector3((float)0.32 * coord.x - (float)0.16, (
               float)0.32 * coord.y + (float)0.16, 0), Quaternion.identity);
25     }
26
27     //Instanciation des poubelles
28     foreach (Vector3 coord in roomData[3])
29     {
30         Instantiate(dumpsterPrefab, new Vector3((float)0.32 * coord.x - (float)
               0.16, (float)0.32 * coord.y + (float)0.16, 0), Quaternion.identity);
31     }
32
33     //Lier aux boîtes qui bougent le canvas d'intéraction
34     foreach (GameObject movableCrate in GameObject.FindGameObjectsWithTag("Box"))
35     {
36         movableCrate.GetComponent<MovableCrate>().MessageOnScreenCanvas =
               messageOnScreenCanvas;
37     }
38 }
```

Ce script va d'abord récupérer le lien vers le Canvas [14] tant qu'il n'y a pas beaucoup d'objets à analyser, puis récupérer les coordonnées et placer les objets à ces coordonnées. Enfin, il lie les boîte qui peuvent bouger au Canvas pour afficher un message au joueur lorsqu'il est près d'une boîte et souhaite la tirer.

Soutenance Finale

Le mode multijoueur de cette énigme a été réalisé par Timothy.

L'énigme des boîtes a été adapté pour le mode en ligne de notre jeu. Lors de cette conversion, il y a eu 2 changements notables :

— Les boîtes sont créées pas le maître de la session une fois que les 2 joueurs sont dans la scène, sinon un des joueurs n'aura pas les boîtes dans sa scène.

Listing 3.32 – Attente des 2 joueurs

```
private void Update()
{
    if (load && GameObject.FindGameObjectsWithTag("Player").Length == 2) // Wait for the 2 players and then the master spawns it
    {
        if (master)
        {
            loadScene(); // Spawn crates
        }

        load = false;
    }
}
```

— Une fonction RPC [15] est utilisé afin de lié le joueur et une boîte lorsqu'il souhaîte la faire bouger. Le joueur lié à la boîte est récupéré dans la scène à l'aide de son ViewID [16].

Listing 3.33 – Création du lien entre le joueur et la boîte

```
private void Update()
{
    //check si on appuie sur E ssi en contact avec la boite est pressee
    if (Input.GetKeyDown(KeyCode.E))
    {
        //si qqn entre en contact avec la boite ET que la boite n'est pas deja liee a qqn
        if (collision && !linkedToPlayer) // collision est un bool indiquand si la boite est en contact avec un joueur
        {
            // on la lie au perso, UpdateLink est execute par les 2 joueurs
            photonView.RPC("UpdateLink", RpcTarget.All, player[0].gameObject.GetPhotonView().ViewID);
        }
        //sinon, si la boite est deja liee a ce qqn, on lui fais lacher la boite
        else if (linkedToPlayer && playerLinked.GetPhotonView().IsMine)
        {
            // on la delie au perso, UpdateLink est execute par les 2 joueurs
            photonView.RPC("UpdateLink", RpcTarget.All, 0);
        }
    }
}

[PunRPC]
```

14. Element permettant d'afficher un objet à un endroit précis de l'écran du joueur.
15. Une méthode RPC est une fonction pouvant être executée sur les différentes instances du jeu en ligne d'une même session.
16. ViewID est un ID fournit par son script `PhotonView` permettant d'identifier les objets à travers les différentes instances du jeu dans une même session pour pouvoir l'attacher à la boîte.

```
22 void UpdateLink(int id)
23 {
24     if (linkedToPlayer)
25     { // Delie la boite
26         UnLink();
27         linkedToPlayer = false;
28     }
29     else
30     { // Lie la boite
31         Link(id);
32         playerLinked = PhotonView.Find(id).gameObject;
33         linkedToPlayer = true
34     }
35 }
```

3.3.3 Labyrinthe Fléché

Cette énigme a été réalisé par Hugo.

Soutenance 1

Génération du labyrinthe fléché de manière dynamique. La taille du labyrinthe peux être réglée avec des variables permettant ainsi de faire un niveau plus facile en guise du tutoriel.

Pour cette génération on considère que chaque tuile est un nombre composé de 4 chiffres représentant respectivement la possibilité d'aller en haut, à droite, en bas ou à gauche.
Par exemple le nombre 2001 indique que la tuile n'a pas de flèche vers le haut (2 en première position) qu'elle pourrait aller à droite et en bas et est obligé d'avoir une flèche vers la gauche.

Le script ci-dessous créé donc un chemin indiquant à chaque tuile la flèche nécessaire pour continuer et les flèches interdites (permettant d'arriver sur la tuile) grâce à la fonction `addPrevious` ;

Listing 3.34 – Génération dynamique du labyrinthe fléché

```
1  //create logic laby
2  while (Xt < x)  //while current height is inferior to max height
3  {
4      int r = Random.Range(0, 3);
5      if (r == 0) // go down
6      {
7          laby[Xt, Yt] = (int)Direction.down + addPrevious(previous); //add minimal
               tile
8          Xt += 1;
9          previous = (int)Direction.down; //store previous direction
10     }
11     else if (r == 1 && Yt + 1 < y && previous != (int)Direction.left) //go right
12     {
13         laby[Xt, Yt] = (int)Direction.right + addPrevious(previous);
14         Yt += 1;
15         previous = (int)Direction.right;
16     }
17     else if (r == 2 && Yt - 1 >= 0 && previous != (int)Direction.right)//go left
18     {
19         laby[Xt, Yt] = (int)Direction.left + addPrevious(previous);
20         Yt -= 1;
21         previous = (int)Direction.left;
22     }
23 }
```

Soutenance 2

Pour cette soutenance, la salle a totalement été dessinée dynamiquement (sols, portes et murs autour du labyrinthe). Ce qui permet de n'avoir que un script pour tout gérer.

Les erreurs des joueur sont aussi géré grâce à un petit script qui à partir de leur position actuel et leur position précédente, vérifie les deux règles qui correspondent à leur mouvement.

Ainsi pour un mouvement vers la gauche on vérifiera que la tuile actuel ne possède pas de flèche allant vers la droite en appelant la fonction ci-dessous et on vérifiera que la tuile précédente possède une flèche allant vers la droite.

Listing 3.35 – Vérification d'une règle à partir de la tuile et la direction

```
private bool RespectRules(int tile, Direction dir)
{
    int[] tileBitArray = DeciToArray(tile,4);
    int[] dirBitArray = DeciToArray((int) dir, 4);
    for (int i = 0; i < tileBitArray.Length; i++)
    {
        if (tileBitArray[i] < dirBitArray[i]) return false;
    }
    return true;
}
```

De plus, le multijoueur a aussi été implémenté. Les tuiles que le joueur maître doit voir sont instanciées localement et les tuiles que le joueur client doit voir sont générée sur le réseau grâce à la ligne `PhotonNetwork.Instantiate` puis désactivées sur le joueur maître.

Le tableau comportant les conditions minimum de chaque tuile est parcouru et avec un ratio de 1/2 la tuile est attribué soit au joueur local ou au joueur distant. Une tuile est ensuite attribué aléatoirement à partir des règles minium grâce à la fonction `prefabRules`. On peut voir tout cela dans le script ci-dessous.

Listing 3.36 – Affichage du labyrinthe

```
for (int x = 0; x < laby.GetLength(0); x++)
{
    for (int y = 0; y < laby.GetLength(1); y++)
    {
        prefabNb = prefabRules(laby[x, y]);
        laby[x, y] = toBin(prefabNb);
        if(Random.Range(0, 8)%2 == 0) //setup local
        {
            Instantiate(prefabsL[prefabNb], new Vector3(startX + y * 0.32f+0.1f,
                startY - x * 0.32f - 0.32f), Quaternion.identity, parentObj.
                transform);
            t = PhotonNetwork.Instantiate(prefabsL[0].name, new Vector3(startX + y
                * 0.32f + 0.1f, startY - x * 0.32f - 0.32f), Quaternion.identity);
            t.SetActive(false);
        }
        else //setup distant
        {
            Instantiate(prefabsL[0], new Vector3(startX + y * 0.32f + 0.1f, startY
                - x * 0.32f - 0.32f), Quaternion.identity, parentObj.transform);
            t= PhotonNetwork.Instantiate(prefabsL[prefabNb].name, new Vector3(
                startX + y * 0.32f + 0.1f, startY - x * 0.32f - 0.32f), Quaternion.
                identity);
            t.SetActive(false);
        }
    }
}
```

Le maître s'occupe toujours de la gestion d'erreur, et envoie juste une commande au joueur pour le re-téléporter au début.

Soutenance finale

Mise à jour des assets pour être d'avantage en cohésion avec les autres salles.
Les conséquences lors du non respect des règles ont été mise en place, les joueurs perdent dorénavant un quart de coeur à chaque erreur qu'ils commettent.

Conclusion

Cette énigme est donc totalement fini. Elle sera utilisé trois fois. Permettant ainsi aux joueurs d'avoir un tutoriel avec une toute petite énigme de (5 par 5). Mais aussi deux fois la même énigme d'une taille supérieur leurs permettant d'échanger les rôles.
L'apercu final est visible ci-dessous.

FIGURE 3.38 – Arrows

3.3.4 Enigme des Fils

Cette énigme a été réalisé par Hugo.

Soutenance 1

La génération des fils se fait en commençant par la génération des prises (gameObject rond) puis en ajoutant des fils (lineRenderer [17]) avec une couleur aléatoire qui sont reliés à une prise choisie aléatoirement du côté droit (visible sur la figure ci-dessous).
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 3.37 – Génération dynamique des prises

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

Listing 3.38 – Génération des fils en les croisant

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

    for (int j = r; j < nbWires - 1; j++){//move all element to the left
         plugsL[j] = plugsL[j + 1];
    }

    int randomColor = Random.Range(0, 4);   //generate a random color for wire

    GameObject wire = Instantiate(wirePrefab, new Vector3(0, 0, 0), Quaternion.
        identity, parentObj.transform);      //create a new wire
    plugsN[i].GetComponent<Plug>().wire = wire;
                                                                    //adding the wire
        to the plug script
    wire.GetComponent<Wire>().Positions = new Transform[2] { plugsN[i].transform,
        plug2.transform };     //adding two plugs to the wire script
    wire.GetComponent<Wire>().color = randomColor;      //setting the wire color
    wiresColors[i] = randomColor;       //saving the wire color
}
```

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 (visible sur la figure ci-dessous).

Listing 3.39 – Variables utilisées dans la fonction précédente

```

int nbFils, color, unplug, rn, fil;
bool finished = false;                  //a rule above is already valid
for (int i = 0; i < nbRules; i++)
```

17. Line Renderer est un composant d'Unity permettant de tracer une ligne entre 2 points.

```
{
    rn = Random.Range(0, 3);            //choose a random type of rule
    color = Random.Range(0, nbColors);  //determine the color of the rule
    unplug = Random.Range(0, nbWires);  //the wire to unplug if rule validated
    if (rn == 0)
    {
        nbFils = Random.Range(2, nbWires / 2);
        if (findNbColors(color) >= nbFils && !finished) //if conditions valid and
            no condition above is valid set wire to be cut
        {
            plugsN[unplug].GetComponent<Plug>().nb = 1;
            finished = true;
        }
        Debug.Log($"Si il y a {nbFils} fil(s) {randomColor(color)} ou plus, dé
            branchez le {nbToWord(unplug + 1)}");
    }
    else if (rn == 1)
    {
        fil = Random.Range(0, nbWires);
        if (wiresColors[fil] == color && !finished)
        {
            plugsN[unplug].GetComponent<Plug>().nb = 1;
            finished = true;
        }
        Debug.Log($"Si le {nbToWord(fil + 1)} est {randomColor(color)}, débranchez
            le {nbToWord(unplug + 1)}");
    }
    else
    {
        nbFils = Random.Range(1, nbWires / 2);
        if (findNbColors(color) < nbFils && !finished)
        {
            plugsN[unplug].GetComponent<Plug>().nb = 1;
            finished = true;
        }
        Debug.Log($"Si il y a moins de {nbFils} fil(s) {randomColor(color)}, dé
            branchez le {nbToWord(unplug + 1)}");
    }
}

unplug = Random.Range(0, nbWires);
Debug.Log($"Sinon débranchez le {nbToWord(unplug + 1)}");

if (!finished)
{
    plugsN[unplug].GetComponent<Plug>().nb = 1;
}
```

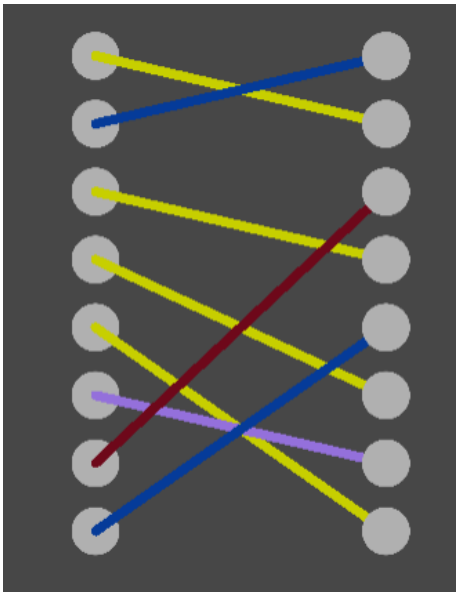


FIGURE 3.39 – Wires

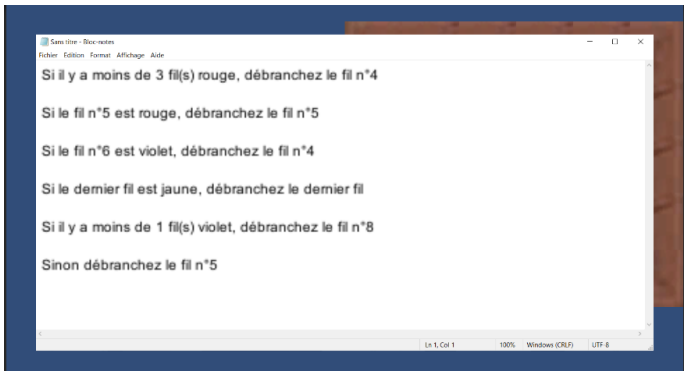


FIGURE 3.40 – Rules

Soutenance finale

L'affichage a été modifié et se fait actuellement directement dans la salle. Au même endroit que les fils. Le déclenchement de la génération des règles et des fils se fait à l'aide de plaques de pression disposé dans la salle, comme vous pouvez le voir dans le code si dessous.

Listing 3.40 – Gestion des plaques

```
if (isGenerationDone) //already generated
{
    if (isOnPressureWire) //someone one pressure plate
    {
        if (isOnPressureRule) //some one on pressure rules
        {
            if (isMasterOnWire) //if master show local wire
            {
                ShowDistantWires(true);
            }
            else //show client wire
            {
                PhotonView photonView = PhotonView.Get(this);
                photonView.RPC("ShowDistantWires", RpcTarget.All, true);
            }
        }
        else// could be same player who activated both pressure, punish them
        {
            PhotonView photonView = PhotonView.Get(this);
            photonView.RPC("DestroyAll", RpcTarget.All);
        }
    }
}
else
{
    if (isOnPressureRule)
    {
        GenerateAll();
    }
}
```

Pour la partie multijoueur, je génère toujours tout du côté maître, et en fonction de qui se tient sur la plaque qui a généré les règles, le maître appelle une fonction PunRPC chez le client pour instancier

les prefabs de règles ou de fils. Quand le client coupe les fils il appelle une fonction chez le maître lui faisant part du résultat. Si le fils coupé et le bon, ça ouvre les portes et les joueurs peuvent passer à la salle suivante, sinon ça détruit les fils qu'il faut ensuite regénérer et fait perdre de la vie au joueur qui a coupé le fils (électrocution).

Conclusion

Cette énigme est complètement finis et intégré aux jeux.
L'apercu final est visible ci-dessous.

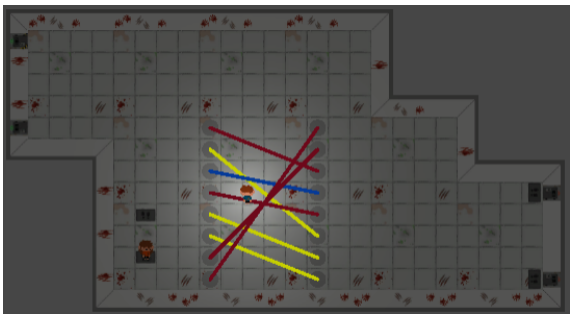


FIGURE 3.41 – Wires

FIGURE 3.42 – Rules

3.3.5 Labyrinthe Invisible

Cette partie a été réalisé par Timothy.

Soutenance 1

Lors de la première soutenance, l'algorithme de génération du labyrinthe avait été fait. Il est détaillé dans le Rapport de la première soutenance.

Soutenance 2

Lors de la deuxième soutenance, le labyrinthe avait été adapté pour le mode multijoueur, et la carte du labyrinthe avait été créée. Un prototype du niveau qu'il constitue avait également était implémenté dans une scène du jeu.

Soutenance finale

Le prototype montré lors de la dernière soutenance a été améliorée et est maintenant complétement intégré dans un niveau complétement fonctionnel. Un joueur à l'aide de la carte du labyrinthe doit aider son compagnon à le franchir, un faux pas dans le labyrinthe et il prend des dégâts. Puis, une fois de l'autre coté, le joueur libère l'accès aux portes de sorties à son camarade en poussant les boîtes qui le gène dans la benne à boite.

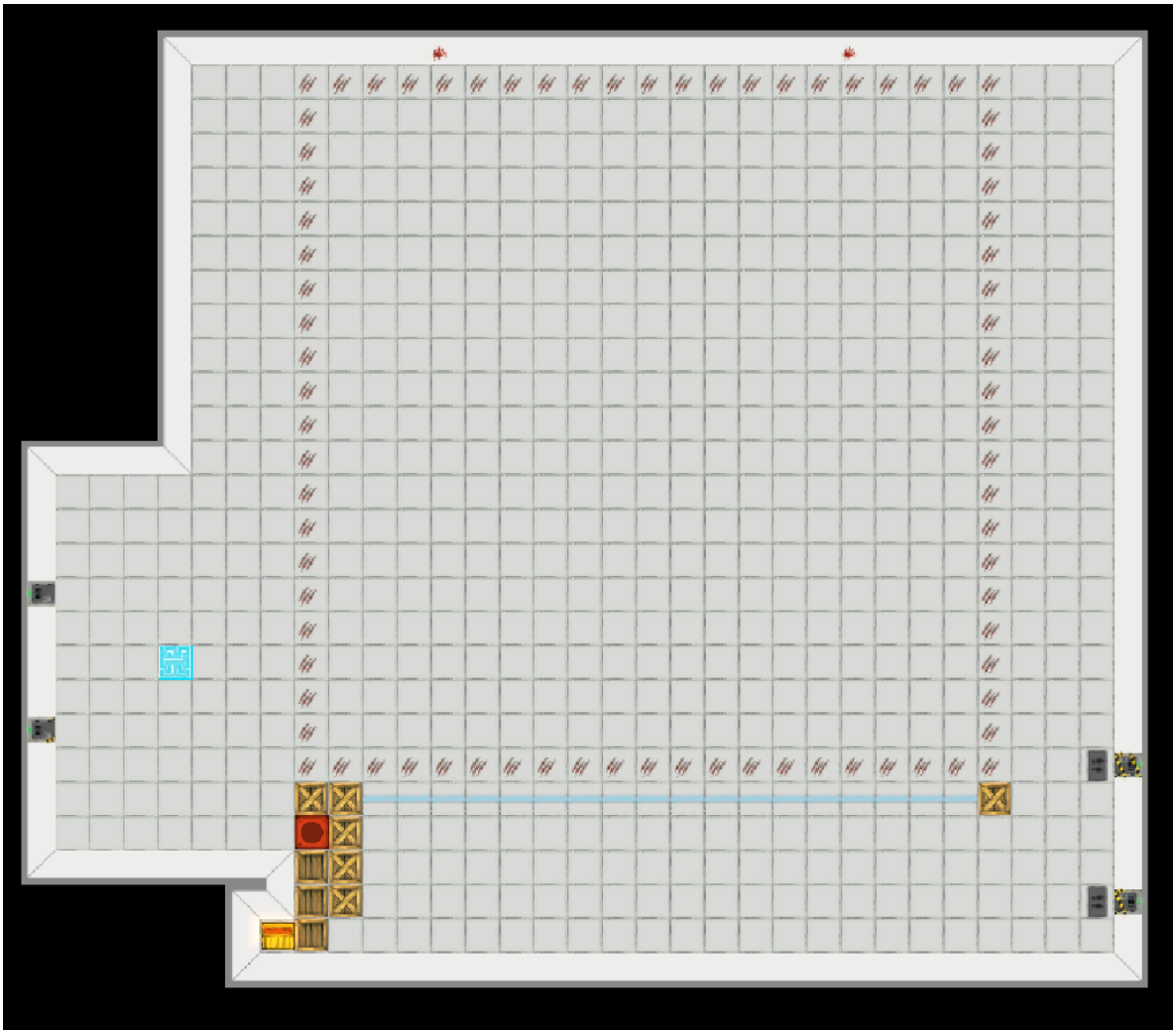


FIGURE 3.43 – Menu de pause

3.3.6 Donjon Aléatoire

Soutenance 1

L'algorithme de création du donjon a été implémenté avec des images temporaires en guise de salle, et une tête de mort en guise sortie. La méthode utilisée pour la création du donjon est détaillée dans le Rapport de Soutenance 1

Soutenance Finale

Cette énigme a été intégrée dans un niveau complétement fonctionnel. Les salles finales du donjons ont été créées et intégrés dans le donjon, et le donjon a été adapté pour le multijoueur en ligne avec seulement quelque modifications apportées au code préexistant.
Dès que 2 joueurs sont dans la scène, les salles sont instantiées sur le serveur uniquement par la personne ayant créé la session en ligne.

Listing 3.41 – Création d'une salle du donjon

```
private void Update()
{
    if (GameObject.FindGameObjectsWithTag("Player").Length == 2)
    {   // Quand les 2 joueurs sont lá
        if (!spawned && PhotonNetwork.IsMasterClient) // Si Spawn() n'a pas deja
            ete appele
            Spawn()
    }
}

void Spawn() // Spawn rooms
{
  PhotonNetwork.Instantiate(rooms[Random.Range(0, roons.Length)].name, transform.
      position, Quaternion.identity); // Choose random room and spawn it on the
      network
    spawned = true;
}
```

Il en va de même pour créer la salle finale comportant la sortie du niveau.

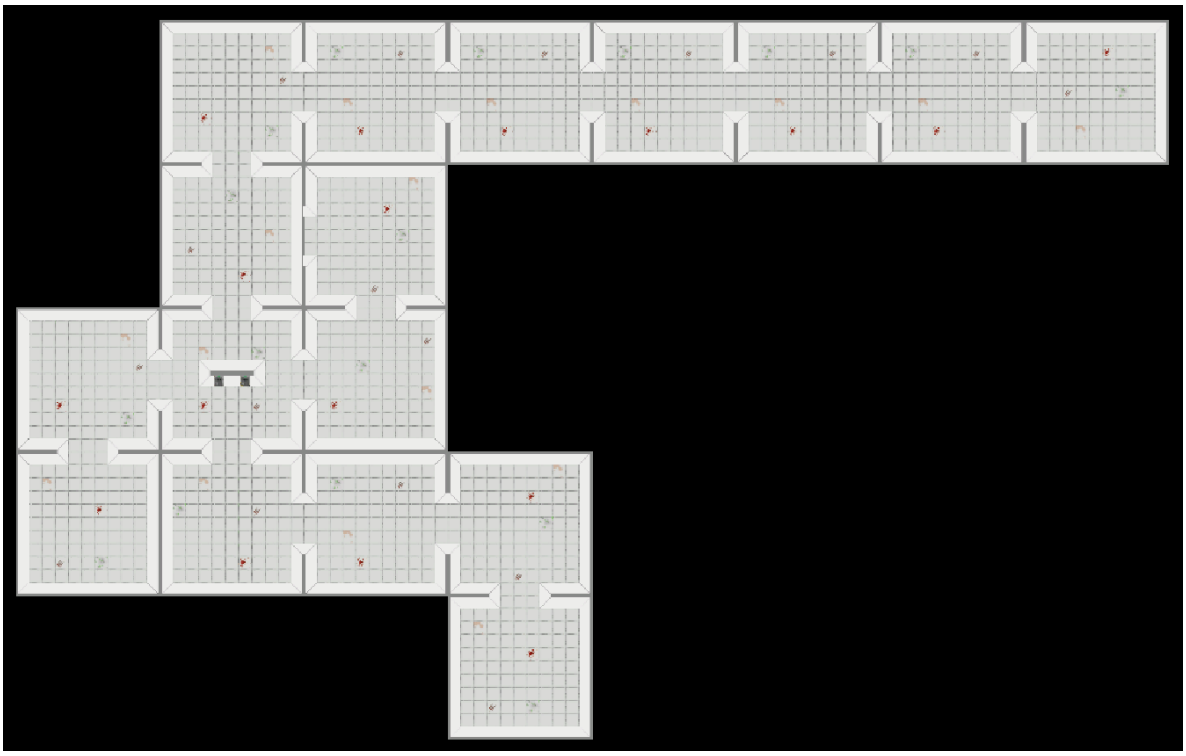


FIGURE 3.44 – Exemple de donjon aléatoire

3.4 Narration

3.4.1 Soutenance 2

Rédaction du scénario

La rédaction du scénario à été réalisée par Ethan.

Une majorité des scripts ont déjà été écrit, notamment celui de l'introduction et de la conclusion, tous ceux de l'énigme du labyrinthe de boîtes et ceux d'avant et après la première rencontre avec un monstre.

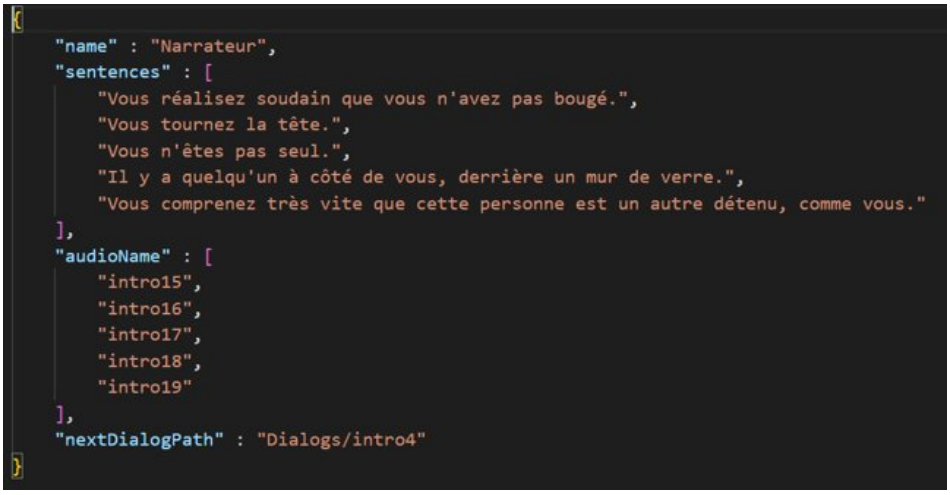


FIGURE 3.45 – Script de l'introduction

Système de Dialogue

Le système de dialogue est utile à la narration du jeu. C'est grâce à cela que les joueurs et les PNJ [18] du jeu c'est intéragissent ensemble. Il a été créé par Timothy et modifié par Ethan pour convenir aux besoins du jeu

Création Cette partie a été réalisée par Timothy.

Les dialogues utilisent un trigger placé dans divers niveaux pour se déclencher ainsi qu'une classe Dialogue afin d'enregistrer le nom de la personne qui parle ainsi que ce qu'elle dit. Le trigger a également une variable de type Dialogue afin de pouvoir déclencher un dialogue différent pour chaque trigger.

Listing 3.42 – Classe Dialogue

```
1 public class Dialogue
2 {
3     // Name of the speaker
4   public string name;
5     // Sentences that the character say
6   public string[] sentences;
7 }
```

Chaque phrase présente dans le dialogue est enregistrée dans une File [19], permettant ainsi de facilement accéder à la phrase à afficher dans le canvas, avec une animation où chaque lettre apparaît un court instant après la précédente. Cet instant de « pause » est géré par une Coroutine [20] en utilisant du mot clé yield [21].

18. Personnage Non Joueur
19. Une File type abstrait basée sur le principe « premier entré, premier sorti »
20. Une Coroutine est une méthode permettant de pauser l'execution d'un code et de le reprendre plus tard
21. Yield permet de retourner plusieurs éléments en un appel

Listing 3.43 – Affichage d'une phrase

```
// Beginning of the Coroutine & call of function TypeSentence()
StartCoroutine(TypeSentence(sentence));

IEnumerator TypeSentence (string sentence)
{
    // Add text to canvas (dialogueText is the text field of the dialogue in the
        canvas)
    dialogueText.text = "";

    // sentences est la File contenant les phrases á afficher
    foreach (char letter in sentences.Dequeue())
    {
        // Append letter to canvas
        dialogueText.text += letter;
        // Restart Coroutine to make a break
        yield return null;
    }
}
```

Importation du texte depuis des fichiers JSON Cette partie a été réalisée par Ethan.

J'ai modifié le système de dialogue de Timothy pour que le système dialogue aille chercher le phrases du script dans un fichier JSON et qu'il supporte le fait qu'il y ait potentiellement plusieurs personnes participant à la discussion.

Ci-dessous un exemple de fichier JSON stockant les informations nécessaires au système de dialogue, à savoir : le nom du locuteur, son discours séparé en un tableau de phrases et le lien (éventuel) vers le discours suivant (d'une autre personne).

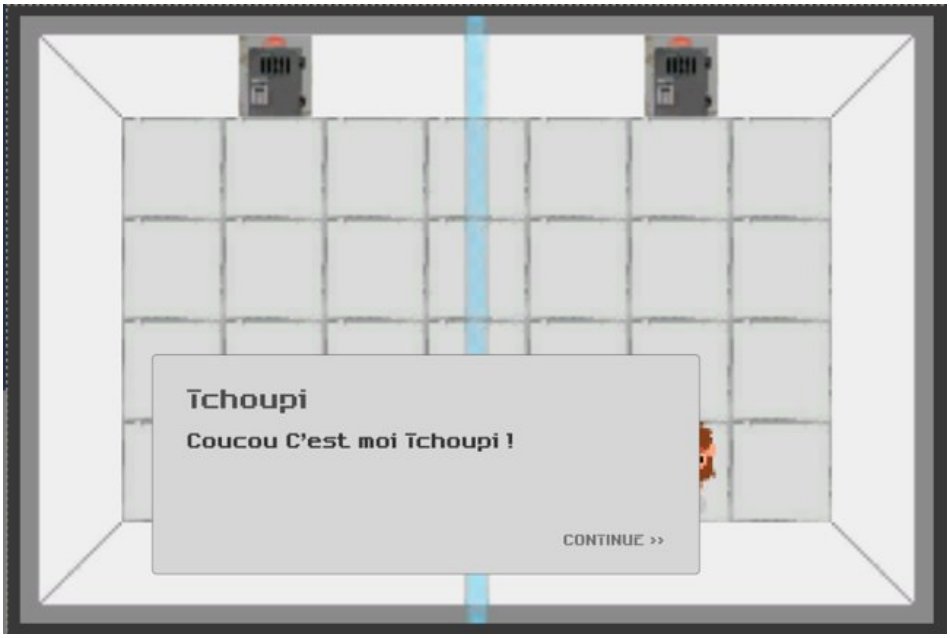


FIGURE 3.46 – Exemple d'un dialogue de Tchoupi

Listing 3.44 – Création du labyrinthe

```
public void TriggerDialogue()
{
  if (filePath != "") {
    //Lecture du fichier json
    var dialoguesData = Resources.Load<TextAsset>(filePath);

    //création de l'objet qui stocke les phrases du dialogue en fonction du json
```

```
    Dialogue dialogue = JsonUtility.FromJson<Dialogue>(dialoguesData.text);

    //changement du chemin du nom du prochain fichier
    filePath = dialogue.nextDialogPath;

    //Début du dialogue
    FindObjectOfType<DialogueManager>().StartDialogue(dialogue, this);
  }
    else if (objectsToActivate != null && objectsToActivate.Count != 0)
    {
        foreach (GameObject obj in objectsToActivate)
        {
      obj.SetActive(true);
        }
    }
}
```

Le script permet de charger le fichier et de réarmer le script avec l'éventuel suivant dans le cas ou la string du suivant est vide, alors le dialogue est finit et on active les objet qui permettent de passer à la salle suivante.

3.4.2 Soutenance Finale

Rédaction du scénario

La rédaction du scénario à été réalisée par Ethan.

Tous les scénarios non-rédigés à la soutenance 2 ont été rédigés. Et les fichiers JSON de tous les dialogues ont été généré

Système de Dialogue

Originellement crée par Timothy, modifié par Ethan, le système de dialogue fut adapté par Hugo pour convenir au besoin du multijoueur.

Ainsi, les dialogues s'affichent chez le client et le host mais chacun peut les lire à son rythme. Ils sont lancé via la game manager décrit plus en détails ci-dessous.

Audio : Tournage

Les audios des dialogues ont été tournée par Ethan.

La voix de Clothilde à été interprétée par Léana Perruzza, la soeur de Ethan et la voix du narrateur fut réalisée par Ethan.

Audio : Ajout dans le jeu

Cette partie a été réalisée par Ethan.

Dans les fichiers JSON, un tableau a été rajouté contenant les noms des fichiers audio à jouer pour chaque phrase.
Les fichiers audio sont stocké dans un dossier du dossier Ressource et sont ainsi appelés avec un Ressource.Load()
Ainsi le fonctionnement du traitement n'a pas été changé, il est calqué sur le changement de phrase, en effet, à chaque changement de phrase, on appelle simplement l'audio suivant.
Pour jouer l'audio j'ai crée un audio manager qui va charger le fichier audio à partir des ressources, crée un AudioSource qui contient le fichier son et le jouer.
Avant de jouer un audio, le manager arrête le précédent pour éviter une cacophonie si le joueur spamme le bouton suivant.

Game Manager

Cette partie a été réalisé par Hugo.

Le game manager s'occupe de gérer les scènes. Il va ainsi permettre d'avoir une cohérence dans tous le jeux.
Il va lancer plusieurs fois les salles d'énigmes, en prenant soin de lancer les dialogues correspondant pour les tutoriels. Il va aussi adapter les paramètres permettant d'avoir des énigmes plus ou moins compliqué.
De plus, il s'occupe aussi du score et permet d'accéder à la scène final uniquement quand tous les autres niveaux sont finis.
Pour les besoins de cette soutenance il s'occupe aussi des raccourcis claviers permettant de passer plus vite d'un niveau à l'autre.

Pour cela il a une fonction qui attend l'évènement de pression sur une plaque de porte. Il va ensuite vérifier si la salle est bien finis en appelant une autre fonctions dans le script du manager de la salle. Il appellera ensuite la nouvelle scène correspondante (qui sera transmise sur la salle) en se basant sur la plaque activé et sur la salle actuel.

3.5 Site Web

3.5.1 Site Web : Font-End

Cette partie a été réalisé dans son intégralité par Ethan.
J'ai d'abord réalisé tout un plan du site que l'on a validé ensemble avec toute l'équipe.
Le site est réalisé en pur HTML [22]/CSS [23] et JavaScript [24].
Le site s'adapte à tout taille d'écran d'ordinateur.

Soutenance 2

Cette partie a été réalisé dans son intégralité par Ethan.
J'ai d'abord réalisé tout un plan du site que l'on a validé ensemble avec toute l'équipe.
Les efforts ont principalement été centré sur le design.
Ont pour l'instant été réalisé :

- La NavBar (pour naviguer d'une page à l'autre).
- Une Pop-Up intempestive qui propose de télécharger notre magnifique jeu .
- La page de présentation des membres de l'équipe.
- La page de présentation du scénario du jeu.

Il ne reste plus que la page d'accueil pour laquelle il était nécessaire d'avoir des images du jeu fini et pour les autres pages, ce n'est que du contenu à créer, le style étant déjà fait.
Le site est réalisé en pur HTML [25]/CSS [26] et JavaScript [27].
Ci-dessous, quelques images du site.

22. HTML est un langage de balisage conçu pour représenter les pages web
23. CSS est un langage informatique qui décrit la présentation des documents HTML et XML
24. JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives
25. HTML est un langage de balisage conçu pour représenter les pages web
26. CSS est un langage informatique qui décrit la présentation des documents HTML et XML
27. JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives

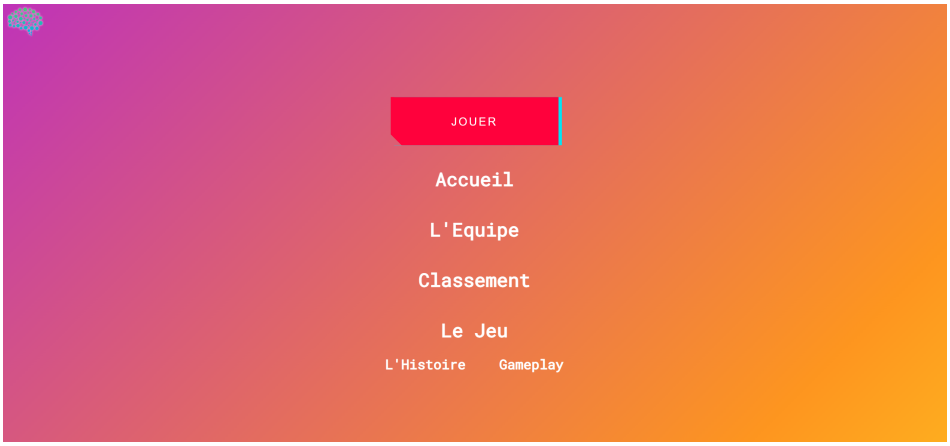


FIGURE 3.47 – Navigation du site

FIGURE 3.48 – Extrait de la page de présentation du scénario

FIGURE 3.49 – Titre de la page de présentation du groupe

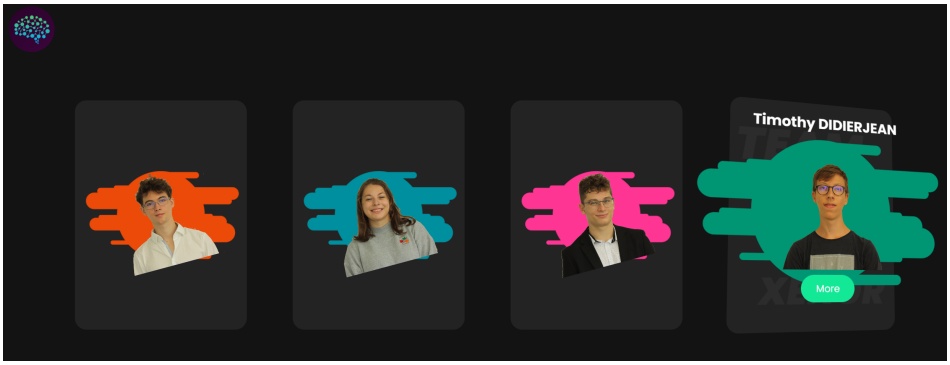


FIGURE 3.50 – Carte des membres qui bougent avec la souris

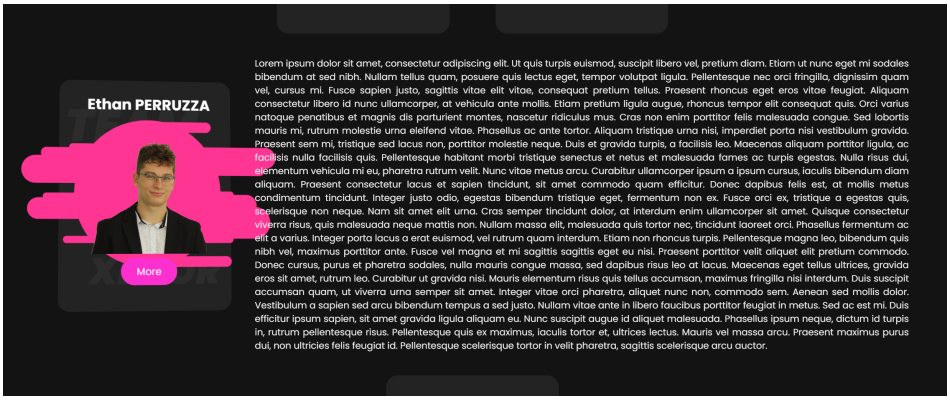


FIGURE 3.51 – Texte déployable avec un bouton sur la carte

Soutenance Finale

Tout le reste du site web à été réalisé. A savoir :

Page d'accueil Cette page fut la plus complexe à réaliser. Ayant une idée très précise de à quoi elle devait ressembler, j'ai créé les images svg du fond de la page à la main (en codant les courbes et leurs couleurs). J'ai également apris à réaliser un effet de paralaxe pour créer la surprise et l'émerveillement dans les yeux du potentiel joueur et l'influencer pour qu'il télécharge notre jeu.

FIGURE 3.52 – Page d'accueil du site (légèrement déroulée)

Page de présentation du Gameplay Cette page reprend les éléments de gameplay principaux afin de les présenter au potentiel jouer et de lui donner envie de télécharger notre jeu.

Page de téléchargement

Cette page est la page qui apparait derière le bouton « Jouer » et derrière celui de la pop-up « »> Play ». Elle propose évidement de télécharger le jeu (en version Windows, MacOs ou Unix). Elle permet également de télécharger l'installeur (uniquement disponible en version Windows). En outre, on peut y trouver nos 3 rapports (première soutenance, soutenance intermédiaire et soutenance finale). Ces dernier permettent de suivre l'avancée du projet.

FIGURE 3.53 – Page de téléchargement

Redesign de la page de Classement Le design de cette page (originellement crée par Hugo) n'avait pas encore été réalisé. Cette partie était ma responsabilité mais elle était prévue pour la soutenance finale. C'est chose faites.

FIGURE 3.54 – Nouveau design du tableau

Finitions Cette partie comprend toute les petites modifications effectuées au site qui ne nécessitent pas une section à par entière.

Page de présentation de l'équipe Le Lorem-Ipsum jusqu'à présent afficher en tant que texte de présentation de l'équipe et de ses membres à été remplacé par une véritable description. Les descriptions personnelles ont été rédigée individuellement par les membres de l'équipe.

FIGURE 3.55 – Présentation générale de l'équipe

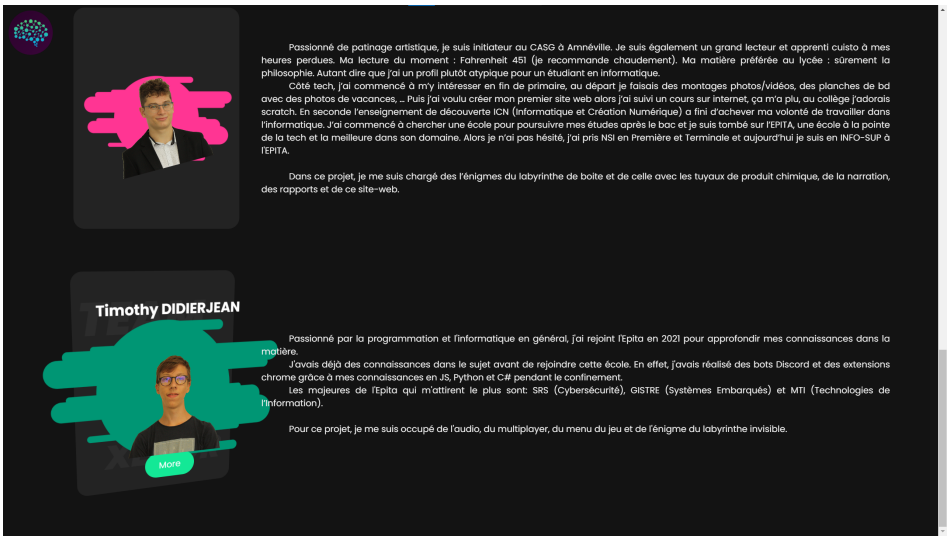


FIGURE 3.56 – Présentations Individuelles

Légères modifications du design La couleur du texte sur certaines page a notamment été modifiée pour offrir une meilleure lisibilité.

Mise à jour de la NavBar Beaucoup de page n'ayant pas été crée au précédent n'était pas référencée dans les liens de la NavBar, ainsi, cette dernière à vu ses liens être mis à jour.

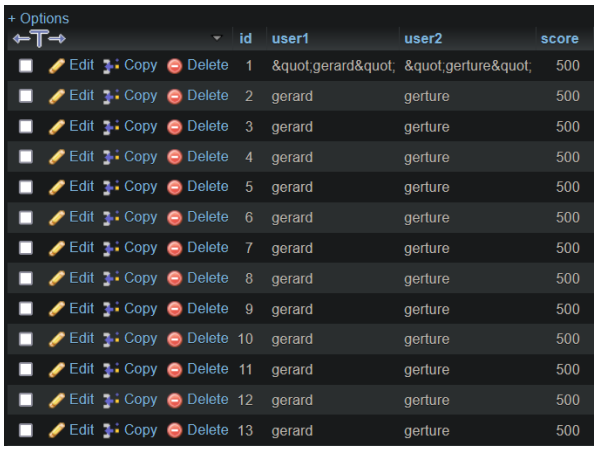
3.5.2 Site Web : Back-End

Cette partie a été réalisé par Hugo.

Soutenance 2

Le backend du site a pour mission de récupérer les scores des joueurs à partir d'Unity et de les envoyer au frontend du site lorsqu'il est chargé.

Pour stocker les scores nous utilisons une base de donnée MySQL [28].



+ Options

	id	user1	user2	score
Edit Copy Delete	1	"gerard"	"gerture"	500
Edit Copy Delete	2	gerard	gerture	500
Edit Copy Delete	3	gerard	gerture	500
Edit Copy Delete	4	gerard	gerture	500
Edit Copy Delete	5	gerard	gerture	500
Edit Copy Delete	6	gerard	gerture	500
Edit Copy Delete	7	gerard	gerture	500
Edit Copy Delete	8	gerard	gerture	500
Edit Copy Delete	9	gerard	gerture	500
Edit Copy Delete	10	gerard	gerture	500
Edit Copy Delete	11	gerard	gerture	500
Edit Copy Delete	12	gerard	gerture	500
Edit Copy Delete	13	gerard	gerture	500

FIGURE 3.57 – Vu de la base de donnée depuis PHPMyAdmin [29] rempli avec des données temporaires

Le premier script PHP [30] permet de charger les données à partir d'une requête GET qui pourra ou non contenir un offset afin d'obtenir les donnée à partir d'un certain rang.

Listing 3.45 – Script load.php

```php
<?php
include('./config.php');

//allow CORS
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: GET, POST');
header("Access-Control-Allow-Headers: X-Requested-With");


$offset = 0;

//get offset is exist and not empty
if(isset($_GET["offset"]) && !empty($_GET["offset"])){
    $offset = htmlspecialchars($_GET["offset"]);
}


//requests to bdd to fetch 10 element based starting at the offset
// $req = $bdd->prepare("SELECT * FROM scores ORDER BY score ASC LIMIT 10 OFFSET ".$offset."");
$req = $bdd->prepare("SELECT * FROM scores LIMIT 10 OFFSET ".$offset."");
$req->execute(array());

$info = $req->fetchAll();

```

28. MySQL est un système de gestion de bases de données relationnelles
30. PHP est un langage de programmation libre, principalement utilisé pour produire des pages Web dynamiques via un serveur HTTP

```php
header('Content-Type: application/json');
echo json_encode($info);
?>
```

Le second script PHP permet d'ajouter des scores à la base de donnée depuis une requête GET contenant les pseudos des deux joueurs et leur score.

Listing 3.46 – Script add.php

```php
<?php
include('./config.php');

//check if values are present and not empty
if(isset($_GET["u1"]) && isset($_GET["u2"]) && isset($_GET["score"]) && !empty($_GET["u1"]) && !empty($_GET["u2"]) && !empty($_GET["score"])){
    $u1 = htmlspecialchars($_GET["u1"]);
    $u2 = htmlspecialchars($_GET["u2"]);
    $score = htmlspecialchars($_GET["score"]);

    //insert in bdd
    $addScore = $bdd->prepare("INSERT INTO scores (user1, user2, score) VALUES (?, ?, ?)");
    $addScore->execute(array($u1, $u2, $score));
    header("HTTP/1.1 200 OK");
}else{
    //respond with error
    header("HTTP/1.1 400 Bad Request");
}

?>
```

Soutenance finale

J'ai pour cette soutenance installé php ainsi que mysql et nginx sur un serveur personnel. Le backend est donc maintenant totalement opérationnelle pour être appelé depuis le jeu et le site.

3.6 Launcher

Cette partie a été réalisé par Hugo.

3.6.1 Soutenance 2

Le launcher a trois mission :

— Installer le jeux

— Faire une mise à jour du jeux

— Désinstaller le jeux

Le launcher a été commencé en C#, il s'exécute sous la forme d'une console (visible ci dessous). Pour l'instant, il fait l'installation du jeux depuis les releases Github dans un dossier fournis (ou non) par l'utilisateur puis il ajoute un raccourci du jeu sur le bureau.

Il manque encore l'étape de l'ajout du jeux dans le Registre de Windows [31] (afin de le retrouver dans le paneau de contrôle) et la mémorisation du chemin d'installation (pour pouvoir le désinstaller). Une fois que cela sera fait, la mise à jour et la désinstallation se feront sans aucun soucis pour la dernière soutenance.

31. Le Registre Windows (ou Regedit) est une base de données définie par le système dans laquelle les applications et les composants système stockent et récupèrent les données de configuration

```
What do you want to do ?
1) Install Altera Vita
2) Update Altera Vita
3) Uninstall Altera Vita
4) Cancel
Enter the number corresponding to desired action.
```

FIGURE 3.58 – Console du launcher

Le système de release Github permet d'héberger les binaires du jeux et grâce à l'API [32] de la plateforme et de la librairie C# Octokit [33] il est simple de télécharger la dernière version du jeu comme montré ci-dessous.

Listing 3.47 – Création du labyrinthe

```
1 GitHubClient client = new GitHubClient(new ProductHeaderValue("SomeName"));
2 IReadOnlyList<Release> releases = await client.Repository.Release.GetAll("s2xenor",
      "xenor");
3 var latest = releases[0];
4
5 WebClient webClient = new WebClient();
6 webClient.Headers.Add("user-agent", "Anything");
7 await webClient.DownloadFileTaskAsync(new Uri(latest.Assets[0].BrowserDownloadUrl),
      destination+"/tmp.zip");
8 ZipFile.ExtractToDirectory(destination+"/tmp.zip", destination + "/AlteraVita");
```

3.6.2 Soutenance finale

Pour la soutenance final le launcher a été finalisé et il peut maintenant remplir toutes ses missions. Il installe correctement le jeux, avec un raccourcis sur le bureau, et il ajoute aussi deux entré dans le registre de windows pour connaître la version installé et le path d'installation.
Grâce à ces clés dans le path, le launcher peux sans difficultés désinstaller le jeux et ses raccourcis ainsi que le mettre à jour.

En revanche il se trouve que la version compilée de ce launcher a un certains nombre de fichiers, ce qui ne simplifie pas énormement la tâche de l'utilisateur. Il a toute fois le mérite de permettre à l'utilisateur de mettre à jour simplement le jeux grâce au Github.
Le jeux n'ayant pas de mise à jour prévue, je suis passé sur une autre solution clé en main utilisant Inno, qui est un logiciel permettant de créer un installateur d'application grâce à une interface graphique.

3.7 Audio

Cette partie a été réalisée par Timothy.

3.7.1 Effets Sonores

Pour augmenter l'immersion dans le jeu, des bruitages ont été rajoutés dans le jeu.
La majorité proviennent soit de Envato Elements, soit de YouTube.
Ils ont été implémentés grâces aux composants `AudioClip` [34] et `AudioSource` [35].

Listing 3.48 – Lancement du son clip

```
1 public AudioClip clip;
```

32. API (ou interface de programmation d'applications) est un ensemble normalisé de classes, de méthodes, de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels
33. Octokit est le client officiel de l'API Github
34. Un `AudioClip` représente un fichier audio dans Unity
35. Un `AudioSource` est un composant Unity permettant de joueur un son dans le jeu

```
2 AudioSource source;
3
4 public void Die()
5 {
6     source.clip = clip;
7     source.Play();
8 }
```

3.7.2 Musiques

La musique provient du site Chosic, offrant de la musique de fond gratuite.
Le jeu joue une musique choisit aléatoirement entre 5 musiques, avec 20 secondes d'écart entre chaque musique.

Listing 3.49 – Script gérant la musique

```
1 public AudioClip[] musics;
2 AudioSource _audioSource;
3
4 private void Update()
5 {
6     if (!_audioSource.isPlaying)
7         Invoke("PlayMusic", 20); // Appelle la fonction PlayMusic() dans 20
              secondes
8 }
9
10 public void PlayMusic() // Play random music and wait until the end
11 {
12     if (_audioSource.isPlaying) return;
13
14     int i = Random.Range(0, musics.Length); // Index aléatoire de la musique
15
16     // Joue la musique
17     _audioSource.clip = musics[i];
18     _audioSource.Play();
19 }
```

3.8 Détails

Cette partie a été réalisé par Timothy.

3.8.1 Lumières

Les lumières ont été rajouté dans le jeux en guise de "touche finale". Celles-ci non pas pour vocation de modifier le gameplay du jeu mais sont là pour améliorer l'immersion en modifiant l'ambiance générale du jeu. En conséquence, même si le jeu s'est globalement assombri, il a gagné en réalisme.

3.8.2 Ecran de Chargement

Un écran de chargement a été rajouté au jeu suite à la transformation de certaines énigmes du mode local au mode multijoueur. En effet, pour permettre à certains niveaux d'être jouable en ligne, ceux-ci attendent que les 2 joueurs soient présent dans la salle avant de s'initialiser, autrement le niveau ne sera jouable que chez un des deux joueurs. De ce fait, entre le moment où une joueur rejoint la partie et celui où le deuxième arrive, le niveau n'est pas prêt. Et au lieu que le premier joueur arrive dans une salle vide en attendant le deuxième, un écran de chargement s'affiche, signifiant intuitivement au joueur déjà présent que le deuxième joueur n'est pas encore arrivé dans la scène, ou que le niveau n'a pas encore finit de charger dans le cas où les 2 joueurs sont présents.
L'écran de chargement est en fait un Prefab [36] de canvas modifié. Pour l'utiliser, il suffit de le glisser

36. Un Prefab est un object créé dans Unity et sauvegardé à part des autres objets, modifier le Prefab modifiera toutes ses instances dans toutes les scènes du jeu

dans une scène, il désactivera automatiquement le déplacement des joueurs et modifiera leur caméras pour que les personnes jouant au jeu voyent le chargement. Il désactivera aussi les monstres our des raisons évidentes.

Listing 3.50 – Set-up de l'écran de chargement dans son script

```
GameObject[] monstres;

private void Start()
{
    foreach (GameObject obj in GameObject.FindGameObjectsWithTag("Player"))
    {
        if (obj.GetComponent<PhotonView>().IsMine) // Verifie que c'est notre joueur
        {
            gameObject.GetComponent<Canvas>().worldCamera = obj.transform.GetComponentInChildren<Camera>(); // Assigne la camera du joueur au canvas
            obj.GetComponent<playerwalkOnline>().enabled = false; // Desactive le mouvement des joueurs
        }
    }

    monstres = GameObject.FindGameObjectsWithTag("Monster");

    foreach (GameObject obj in monstres)
    {
        obj.SetActive(false);
    }
}
```

Afin supprimer l'écran de chargement, il suffit d'appeler sa fonction `Del` pour le supprimer. Cela rétablira les joueurs ainsi que les monstres.

Listing 3.51 – Fonction Del() de l'écran de chargement

```
public void Del()
{
    foreach (GameObject obj in GameObject.FindGameObjectsWithTag("Player"))
    {
        if (obj.GetComponent<PhotonView>().IsMine) // Verifie que c'est notre joueur
        {
            obj.GetComponent<playerwalkOnline>().enabled = true; // Reactivation du mouvement des joueur
        }
    }

    foreach (GameObject obj in monstres)
    {
        obj.SetActive(true);
    }
}
```

Listing 3.52 – Appel de la fonction Del()

```
void RemoveLoadingScreen() // Fonction qui supprime l'ecran de chargement
{
    GameObject.FindGameObjectWithTag("Loading").GetComponent<FetchCam>().Del();
}
```

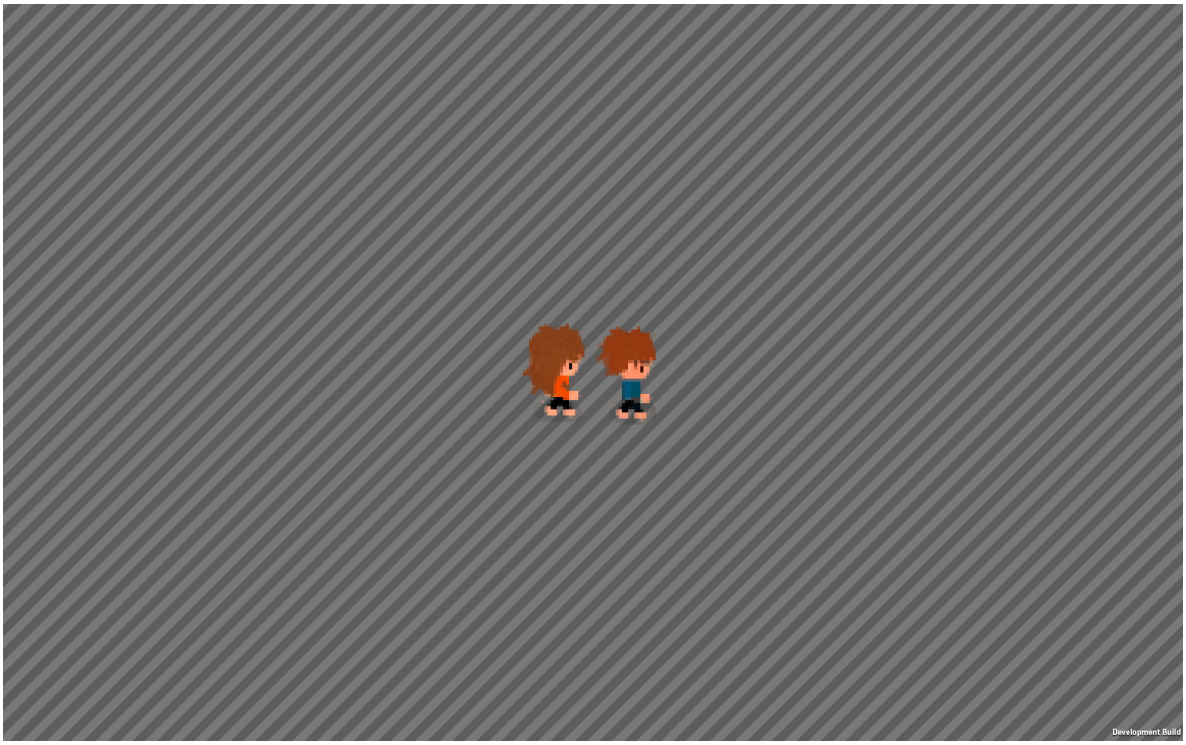


FIGURE 3.59 – Ecran de chargement

3.8.3 Menu de pause

Un menu de pause a été rajouté au jeu, celui-ci permet de modifier les paramètres du jeu en pleine partie. Cependant, lorsque le menu pause est activé, le jeu continue.
Il a été implémenté dans le `GameManager`.

Listing 3.53 – Affichage du menu de pause

```
1  private void Update()
2  {
3      if (Input.GetKeyDown(KeyCode.Escape)) // Si la touche echap est appuyee
4      {
5          GameObject pauseMenu = GameObject.FindGameObjectWithTag("Pause");
6
7          if (pauseMenu != null)
8          { // Menu de pause deja affiche, on le supprime
9              Destroy(pauseMenu);
10         }
11         else
12         { // On affiche le menu de pause
13             Instantiate(menu, Vector2.zero, Quaternion.identity);
14         }
15     }
16 }
```

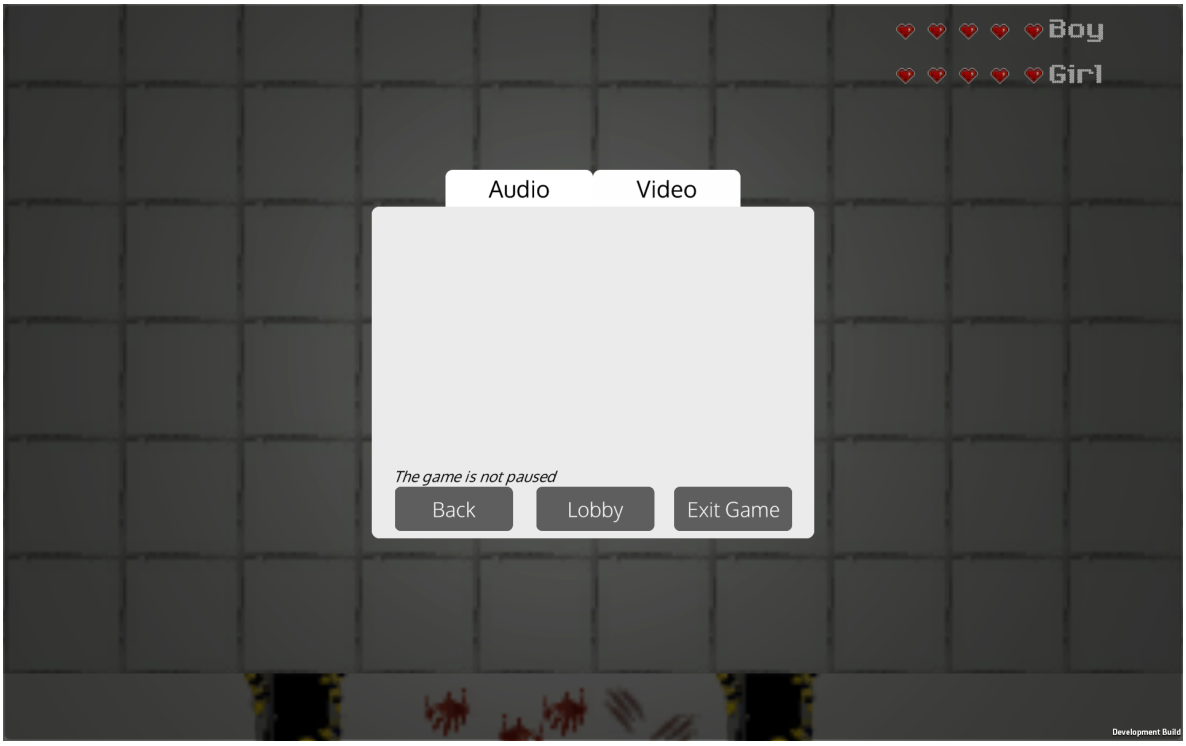


FIGURE 3.60 – Menu de pause

3.8.4 Jaquette de rendu

Pour rendre la clé avec la version finale du jeu, nous avons choisis d'imiter une boîte de jeu vidéo. C'est pourquoi nous avons designer une jaquette de couverture pour la boîte qui contiendra la clé.

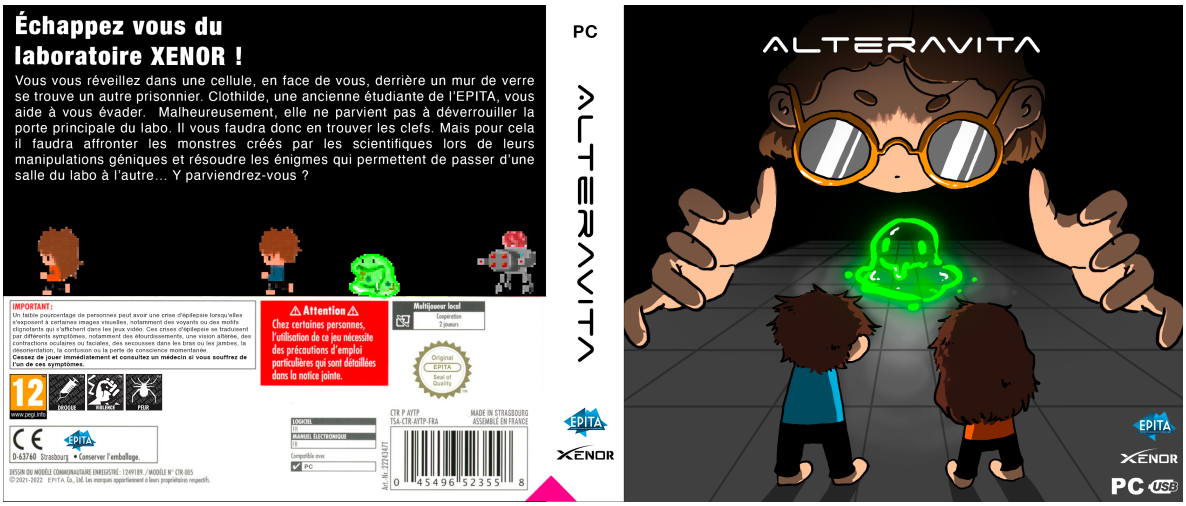


FIGURE 3.61 – Couverture de la boîte de jeu

3.9 Résumé de l'Avancement

Table 3.1 – Résumé de l'Avancement

Tâches	État
Création Menu Principal	F
Création du Lobby	F
Création des Cellules	F
Ajout du multijoueur en ligne	F
Intégration de l'énigme : « Enigme des fils »	F
Intégration de l'énigme : « Labyrinthe de Boites »	F
Intégration de l'énigme : « Labyrinthe Fléché »	F
Intégration de l'énigme : « Labyrinthe Invisible »	F
Intégration de l'énigme : « Connecte les Produits Chimiques »	F
Création du menu principal	F
Intégration du système de dialogue	S : Audio des dialogues
Intégration des bruitages	F
Création du Front-End du site	F
Création du Back-End du site	F
Création du launcher	F
Création des monstres	F
Création des potions	F
Système de combat	F
Implémentation des animations et des mouvements des joueurs mâle et femelle	F
Création de la barre de vie des joueurs	F

Légende :

Symbole	Signification
Fini	F
En Cours	C
Pas Encore Réalisé	N
Objectif Surpassé	S

3.10 Conclusion

Table 3.2 – Répartition des Responsabilités pour la Soutenance Finale

Tâches	Elya	Ethan	Hugo	Timothy
Ajout des musiques				A
Création et Implémentation de l'énigme « Connecte les produits chimiques »		A		
Intégration des dernières énigmes dans le jeu			A	
Mise en place du système de combat joueur contre monstres	A			
Asset Spécifiques (détails, objet)	A			
Finalisation du FronEnd du site		A		
Finalisation du backend du site			A	
Fin d'implémentation système multijoueur				A
Finalisation du launcher			A	
Finalisation et implémentation du script de narration		A		
Ajout d'un audio sur les dialogues		B		
Jeu de lumière autour des personnages				B

Légende :

Symbole	Signification
Atteint	A
En Cours	EC
Échoué	E
Bonus	B

Toutes ces tâches ont été réalisé avec succès par rapport aux attentes.

Chapitre 4

Ressenti

Nous sommes très content de ce que nous avons produit, ce fût très instructif de coder et travailler de manière général en équipe. Nous avons rempli nos objectifs et même dépassé sur certains points ce qui nous remplis de fierté.
On a aussi appris beaucoup sur la programmation orienté objet et sur le fonctionnement de Unity. Cependant de manière général ce que nous avons préféré est bien évidemment le code et la logique qui va avec, Unity pouvant être frustrant sur sa manière de fonctionner.
Nous avons tous déterminer au cours de se projet que nous ne travaillerons pas dans ce domaine.
Mais au final, nous repartons avec de l'expérience et un jeu dont nous ne sommes pas peu fière.

Chapitre 5

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.

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.

Envato Elements • Envato Elements est un service distibuant un stock illimité de vidéos, musiques, photos. Ici, il nous serviera pour les bruitages du jeu.

Chosic • Chosic est un service distibuant de la musique de fond gratuitement. Les musiques pouvant être entendues dans jeu proviennent de ce site

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.

Chapitre 6

Conclusion

Nous pouvons ainsi être fier du rendu final. Notre jeu fonctionne et correspond a nos attentes. En effet, nous avons réussi à réaliser l'intégralité de nos objectifs. Nous pouvons maintenant vivre la formidable histoire tout droit sortie de notre imagination et la partagée avec les autres passionnés de jeux d'aventure et d'énigme. Cette expérience nous a permis de découvrir le travail en équipe et les secrets qui se cachent derrière nos jeux vidéos préférés. Pour conclure, nous citerons en toute modestie John Fitzgerald Kennedy : « L'art de la réussite consiste à savoir s'entourer des meilleurs. ». Nous sommes les meilleurs, c'est pourquoi nous avons réussi avec brio.