RAPPORT DE SOUTENANCE n°1

EPITA - Projet S2

Ctrl + Alt + Elite

KOPFF Louis, MITTELBRONN Pierre,
PONSOT Thibault, RAUGER Axel, TOUFIR Bashir

Janvier 2025

Table des matières

1 Introduction

Ce rapport de soutenance présente l'avancée du projet de jeu vidéo en 2D intitulé *Syndrome Fear*, un jeu axé sur l'exploration et la résolution d'énigmes. À ce stade, le développement du projet a progressé sur plusieurs aspects clés. Le rapport détaille les étapes déjà accomplies, les fonctionnalités développées et les choix techniques et créatifs réalisés jusqu'à présent.

À ce stade, plusieurs aspects clés du projet ont été réalisés. Les déplacements du joueur ont été implémentés, offrant ainsi une base solide pour l'exploration du monde du jeu. Les premiers designs des personnages et des environnements ont été créés, contribuant à l'ambiance visuelle du jeu. De plus, les menus ont été conçus. Le développement du mode multijoueur a également débuté. Ces avancées posent les fondations pour l'expérience de jeu complète et immersive que nous envisageons.

Le document offre également une vue d'ensemble des défis rencontrés et des solutions apportées, tout en détaillant les étapes à venir pour mener le projet à son terme. Les objectifs pour la prochaine phase de développement sont également précisés afin de garantir la continuité et la qualité du projet.

2 Etat de l'art

Syndrome Fear fait partie de la catégorie de jeux "jeux d'évasion". L'histoire du genre se fait en parallèle avec la création des escape games.
Le premier jeu qui définit ce type est le jeu *Behind Closed Doors* sorti en 1988 sur l'ordinateur Sinclair ZX Spectrum. Dans ce dernier, le joueur commence enfermé dans une pièce et doit s'échapper. Ce n'est qu'en 2001 qu'un autre jeu du type est paru. Il s'agit de *The Mystery of Time and Space* (MOTAS), un jeu Flash. On peut considérer le japonais Toshimitsu Takagi comme étant le père du genre. En effet, dès 2005, il l'a popularisé avec ses jeux Flash *Crimson Room* et *QP-Shot*.

2.1 Inspirations

Notre jeu s'inspire librement d'autres mécaniques présentes dans des jeux des trente dernières années.

Professeur Layton

Les mécaniques des énigmes sont similaires à celles de la série des *Professeur Layton* développée par Level-5. *Professeur Layton* est le jeu d'énigmes par excellence : dix-huit millions de jeux vendus au total entre le premier opus en 2007 et 2023. La licence se découpe en huit jeux : une trilogie initiale, une trilogie "prequel", un titre qui fait office de suite ainsi qu'un nouvel opus prévu pour 2025, qui devrait être la suite de la première trilogie. Dans cette série, le joueur progresse dans son enquête à l'aide d'énigmes de différents types et de différentes difficultés. Le joueur est récompensé par un score (exprimé en picarats) qui décroît avec la difficulté de l'énigme ainsi qu'avec le nombre d'essais infructueux.
Les deux principales qualités de cette licence sont son intemporalité et son adaptation aux consoles sur lesquelles ils ont été publiés. En effet, les *Professeur Layton* ont tous été conçus pour Nintendo, et ils exploitent tous à merveille les fonctionnalités incontournables de la DS (puis de la 3DS) que sont le double écran et le tactile. Pour le nouvel opus qui sortira sur Nintendo Switch, les graphismes se referont une beauté avec des menus plus modernes ainsi que des effets 3D à la hauteur des attentes actuelles.

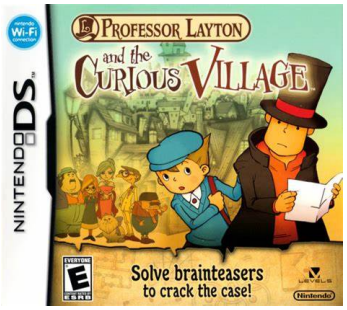

Figure 1: Box art de "Professeur Layton et l'étrange village", premier opus de la série, en version US.

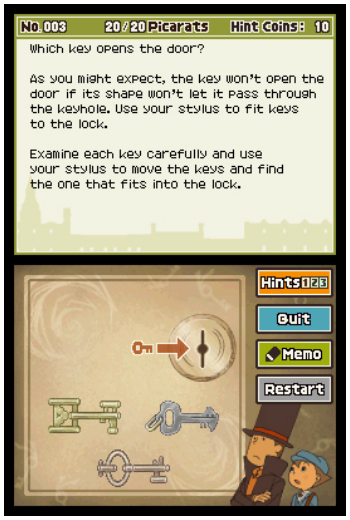

Figure 2: Aperçu d'énigme (Énigme 3 - Professeur Layton et la boîte de Pandore) Ici, la solution est la clé dorée qu'il faut insérer à l'envers.

Papers Please

La mécanique de manipulation de documents est analogue à celle du jeu Papers, Please de Lucas Pope sorti en 2013. Dans ce jeu de réflexion, le joueur incarne un garde frontière chargé de laisser passer, ou non, des hommes souhaitant rentrer en Arstozka, en étudiant les documents qu'ils ont à lui présenter (d'où le nom signifiant "Vos papiers, s'il vous plaît"). Ce jeu est un jeu dont l'ambiance est très immersive, ce qui en fait un jeu encore très joué actuellement.

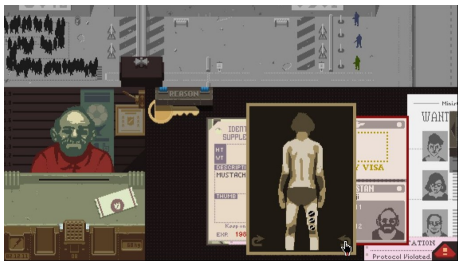

Figure 3: Capture d'écran du jeu Papers, Please. Le joueur doit juger la validité des preuves présentées pour laisser passer, ou non, le visiteur.

2.2 Jeu similaire

Notre jeu est similaire sur certains points au jeu Among Us publié par Innersloth en 2018. Dans ce jeu d'ambiance multijoueur, les joueurs évoluent dans un vaisseau spatial. Certains d'entre eux sont des imposteurs, et leur but est de tuer leurs coéquipiers d'une façon ou d'une autre. Les autres doivent faire leurs tâches, c'est-à-dire résoudre des énigmes qui font avancer un curseur de victoire.
Inconsciemment, notre projet de jeu se rapproche un peu d'Among Us : les graphismes simples en 2d, l'ambiance sonore relativement silencieuse et pesante, et la présence d'énigmes sont des éléments communs.

Figure 4: Capture d'écran du jeu Among Us

Notre jeu présente des similitudes avec The Escapists, un jeu développé par Mouldy Toof Studios et publié par Team17 en 2015. Ce jeu est caractérisé par son style graphique distinctif en pixel art 2D dans lequel les environnements et les personnages adoptent une esthétique rétro et minimaliste.

De manière inconsciente, notre projet de jeu s'inspire de cet aspect visuel de The Escapists. Les graphismes minimalistes en 2D, l'utilisation d'un style pixel art clair et épuré, ainsi que la simplicité des décors et des personnages sont des points communs qui rapprochent nos deux jeux.

Figure 5: Capture d'écran du jeu The Escapists.

3 Le choix de Unity

Pour produire notre jeu vidéo, nous avions le choix entre Godot et Unity. Nous avons finalement choisi Unity pour plusieurs raisons. La première est que Unity a l'avantage de posséder une diversité de scripts et de plug-ins intégrés. Il possède également une grande communauté, ce qui facilite l'apprentissage, la recherche de documentation, et la correction de nos erreurs.

Godot a, lui aussi, ses avantages, comme son optimisation pour la 2D et sa légèreté, ainsi que par son côté Open-Source. Cependant, Unity étant plus utilisé dans l'industrie, et étant donné que nous utilisons C# dans le cadre de notre cursus, nous avons trouvé cela beaucoup plus intéressant d'utiliser Unity dans le choix de notre projet.

4 Avancée du planning

Lors de la création de ce projet, nous avons défini un planning à travers le diagramme de Gantt, que nous avons donc suivi jusqu'ici.

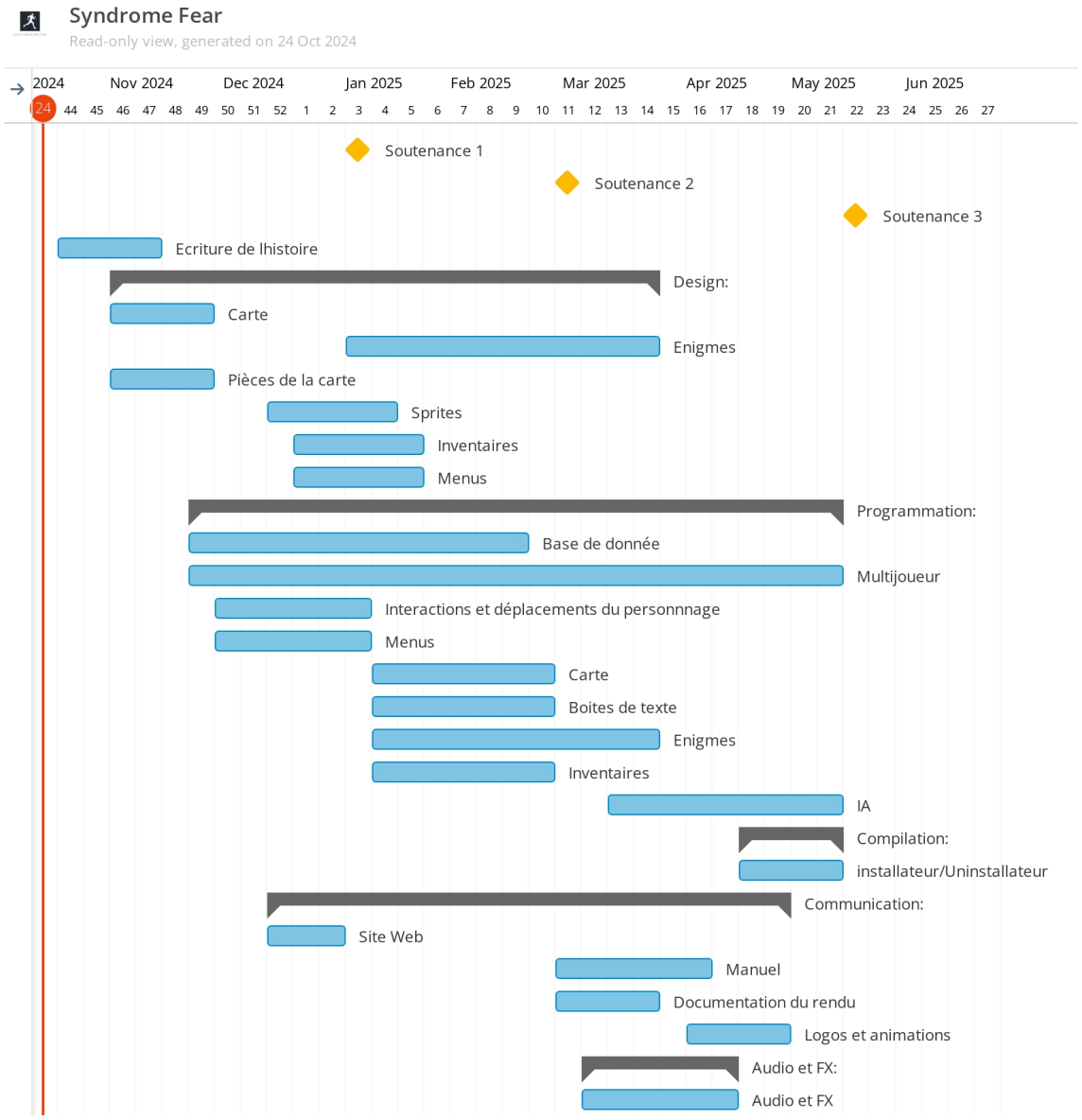

Figure 6: Diagramme de Gantt

Nous avons débuté par l'écriture de l'histoire du jeu, une étape fondamentale. En effet, un jeu vidéo sans trame narrative serait comme un film sans scénario : dépourvu de sens et de direction.

Par la suite, nous avons entamé la conception des designs du jeu. Nous avons choisi de commencer par créer les éléments graphiques, afin de faciliter l'étape de la programmation et d'obtenir une vision claire du rendu visuel de notre jeu. Ce processus

permet également de simplifier le développement des interactions des personnages avec des objets ou des menus, les visuels correspondants étant déjà disponibles ou en cours de création. Nous avons ainsi conçu la carte du jeu, dans laquelle les joueurs pourront se déplacer, ainsi que les différentes pièces qui la composent. En parallèle, nous avons réalisé un premier personnage et produit une première version des designs de l'inventaire et des menus, ces derniers étant encore en cours d'élaboration.

En parallèle, nous avons débuté l'implémentation du mode multijoueur, un élément central de notre jeu vidéo. Ce travail se poursuivra tout au long de la phase de programmation. Nous avons également commencé à développer les fonctionnalités relatives aux interactions et aux déplacements des personnages, ainsi que les menus.

Enfin, nous avons conçu notre site Internet, qui sert désormais de vitrine à notre projet.

4.1 Designs du jeu

Les designs du jeu sont essentiels dans l'expérience du joueur. Il est donc important de les soigner.

4.1.1 Le menu

Afin de réaliser le menu, nous avons récupéré le logo d'origine et nous l'avons étendu pour avoir un effet de profondeur, qui rappelle l'intrigue du jeu avec une porte verrouillée. Nous l'avons légèrement modifié afin que le personnage principal n'apparaisse pas ici. Nous avons fait ce choix afin de ne pas détourner le regard du joueur derrière le logo, ce qui donne un meilleur contraste à l'écriture "syndrome fear", en conséquence de quoi nous avons pris ce vert pour sortir de la pénombre qu'est l'arrière-plan. Le vert rappelle la gentillesse dans l'univers d'Undertale et la possibilité de se faire aider dans ce jeu tout en avançant au cours de l'histoire. Les boutons sont bleus et violets pour respecter la palette indiquée du cahier des charges.

Figure 7: Le menu du jeu.

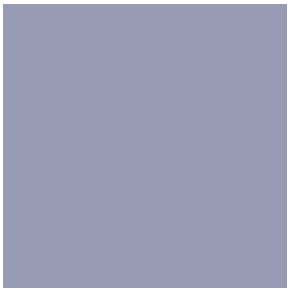

Figure 8: Le violet sélectionné, de code Hexadécimal #868ca7.

4.1.2 La carte et les pièces de la carte

Nous avons réfléchi à l'agencement de la carte, afin de permettre une aventure agréable et logique tout au long de l'aventure du joueur. Pour cela, nous avons élaboré le plan le plus logique et le plus cohérent.

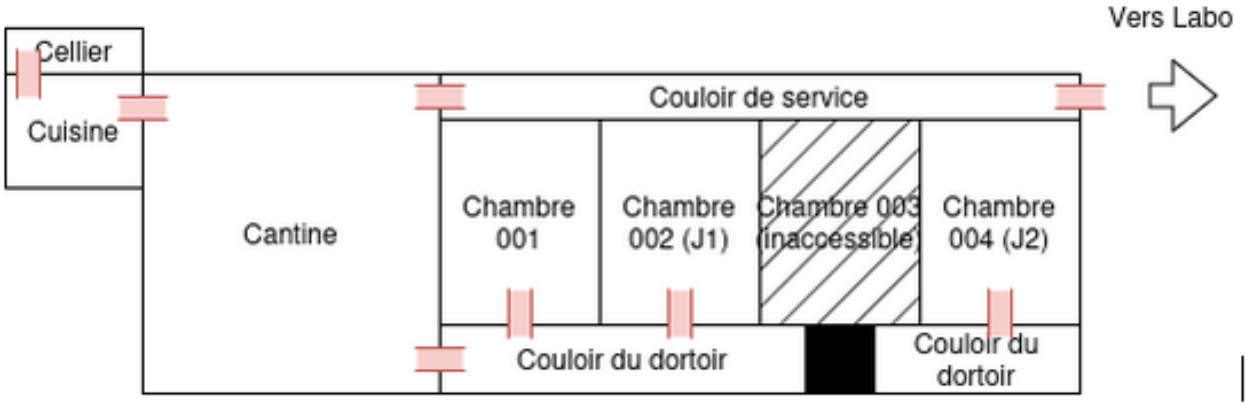

Figure 9: Premier plan réalisé.

Nous avons commencé par travailler sur la cantine, étant donné qu'il s'agit de la pièce la plus grande. Pour le design de sa forme, nous nous sommes inspirés du style de "The Escapists - Food Hall". Ensuite, nous avons poursuivi la conception de l'ensemble du plan en adoptant un sol et des murs évoquant un lieu abandonné, plus précisément un asile. Pour renforcer cette ambiance, nous avons utilisé des éléments comme des fauteuils roulants, une cantine avec des plats vides, etc. Toutes les pièces ont été découpées afin de pouvoir être importées dans Unity. Cette approche permet de "bloquer" le joueur dans une salle dans laquelle il doit résoudre une énigme pour avancer dans le jeu. Cela simplifie également la gestion des collisions sur la carte.

Figure 10: Capture d'écran de la carte.

Figure 11: Capture d'écran du jeu vidéo.

4.1.3 Les outils utilisés

Afin de réaliser les designs de ce jeu, nous avons utilisé plusieurs logiciels : Photoshop principalement, ainsi que RPG Maker pour les assets de la carte uniquement. L'ensemble des sprites est donc "custom", c'est-à-dire réalisé par nos soins. Nous avons également utilisé Unity afin d'animer le personnage notamment.

4.2 Programmation du mode multijoueur

Dans le cadre de notre projet Syndrome Fear, nous avons étudié et implémenté un mode LAN (Local Area Network) pour laisser aux joueurs la chance d'une expérience immersive en multijoueur. Dans cette section, nous allons explorer les fondements du mode LAN, expliquer son fonctionnement, puis présenter une approche technique dans le but de réaliser cette fonctionnalité dans notre jeu.

Définition du mode LAN

Par définition, le mode LAN ou réseau local en français, est un réseau informatique dans lequel les terminaux participants s'envoient des trames au niveau de la couche de liaison sans utiliser l'accès à internet. Plus concrètement, ce mode permet à plusieurs appareils situés dans un même réseau local de communiquer directement sans passer par un serveur externe. Il s'appuie sur les protocoles réseaux locaux pour établir des connexions rapides et stables. Ce mode est très utile pour des jeux multijoueurs, car il offre une latence faible et une configuration facile.

Ce mode a trois caractéristiques particulières : la première est qu'il n'y a pas besoin d'Internet pour utiliser ce mode, puisque les appareils communiquent via le réseau local; il y a donc une connexion directe entre les appareils. La seconde est que les données transitent rapidement entre les appareils connectés, et la dernière est que toutes les perturbations dues à des problèmes de connectivité Internet sont éliminées.

Fonctionnement

Détection des appareils : chaque appareil qui participe (client ou hôte) doit être identifié via une adresse IP unique sur le réseau local.
Communication via protocole : les protocoles couramment utilisés incluent TCP/IP (Transmission Control Protocol) pour la fiabilité des connexions ou UDP (User Datagram Protocol) pour des transmissions rapides.
Gestion des sessions : une session multijoueur est créée par un hôte, et les clients s'y connectent en utilisant l'adresse IP de l'hôte.
Synchronisation des données : les événements du jeu (mouvements, actions, résolution d'énigmes) sont synchronisés entre les joueurs via des échanges de paquets.

Implémentation

Dans le cadre de notre projet Syndrome Fear, nous avons intégré un mode multijoueur LAN en utilisant Unity et le langage C#. Pour cela, nous avons choisi d'utiliser Netcode for GameObjects, qui est la solution réseau officielle d'Unity pour le développement multijoueur. Après avoir évalué les différentes options, telles que Mirror ou d'autres bibliothèques réseau, notre choix s'est porté sur Netcode for GameObjects en raison de sa compatibilité native avec Unity et de la richesse de sa documentation.

La première étape a consisté à installer les outils nécessaires, notamment les packages Netcode for GameObjects, Multiplayer Tools et ParrelSync. Ce dernier nous a permis de

lancer plusieurs instances du projet Unity simultanément, sans devoir exporter le jeu, ce qui simplifie grandement les phases de tests. Grâce à ParrelSync, nous avons pu simuler un environnement multijoueur directement dans l'éditeur Unity en ouvrant une seconde instance du projet.

Nous avons ensuite créé un Network Manager, qui constitue l'élément central de la gestion multijoueur. Ce composant utilise UnityTransport comme protocole de communication réseau. Pour chaque joueur qui se connecte, le Network Manager attribue un prefab configuré comme NetworkObject. Cela permet de définir cet objet comme réseau, de le répliquer sur les autres instances et de le créer automatiquement pour chaque nouveau joueur rejoignant la session. Afin de faciliter l'interaction, nous avons ajouté des boutons in-game pour créer ou rejoindre une session multijoueur. Le bouton "Start Host" permet au joueur local de créer une session en hôte sur son IP et son port, tandis que le bouton "Start Client" connecte un client à une session hôte existante. Le script associé à ces boutons gère les actions nécessaires pour démarrer l'hôte ou le client en utilisant les fonctions StartHost() et StartClient() du Network Manager.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Script : MonoBehaviour
{

    public void StartClient()
    {
        NetworkManager.Singleton.StartClient();
    }

    public void StartHost()
    {
        NetworkManager.Singleton.StartHost();
    }
}
```

Lorsqu'on clique sur Start Host, Unity crée une session hôte sur l'IP locale et un port défini. Ensuite, lorsqu'une seconde instance clique sur Start Client, celle-ci détecte automatiquement la session hôte et y connecte un autre joueur.

Une fois les connexions établies, un problème majeur est apparu : les mouvements des joueurs n'étaient pas correctement synchronisés sur le réseau. Par défaut, tous les joueurs semblaient se déplacer simultanément, ce qui n'était pas fonctionnel. Pour résoudre cela, nous avons créé un script qui désactive certains composants du prefab en fonction de l'état réseau. Ainsi, seul le propriétaire du prefab peut en contrôler les déplacements. Grâce à l'utilisation de la propriété IsOwner, nous avons pu activer ou désactiver le composant PlayerController en fonction du joueur qui possède le prefab.

Cela garantit que chaque joueur ne peut contrôler que son propre personnage.

```
using System.Collections;
using System.Collections.Generic;
using Unity.Netcode;
using UnityEngine;

public class NetworkPlayer : NetworkBehaviour
{


    public PlayerController playerController

    public override void OnNetworkSpawn()
    {
        base.OnNetworkSpawn();

        playerController.enabled = !IsOwner;
    }
}
```

Pour la synchronisation des déplacements, nous avons utilisé le composant NetworkTransform fourni par Unity. Ce composant permet de répliquer l'état du Transform d'un GameObject entre les différentes instances du jeu. Par défaut, Unity autorise uniquement la synchronisation dans le sens hôte-vers-clients, afin d'éviter des comportements indésirables, comme la triche. Cependant, dans notre jeu coopératif, nous avons choisi de permettre au client d'envoyer des données de mouvement au serveur. Pour cela, nous avons implémenté un script dans lequel nous avons redéfini le comportement d'autorité réseau, en autorisant les clients à transmettre leurs informations au serveur. Cela permet une synchronisation fluide et cohérente des mouvements tout en évitant les abus.

```
using System.Collections;
using System.Collections.Generic;
using Unity.Netcode.Components;
using UnityEngine;

public class NetworkPlayer : NetworkBehaviour
{


    protected override bool OnIsServerAuthoritative()
    {
        return false;
    }
}
```

Avant d'implémenter ces solutions, nous avons effectué un travail approfondi pour comprendre les principes fondamentaux du mode LAN et des réseaux en général. Nous avons étudié les protocoles TCP/IP et UDP afin de comparer leurs avantages et inconvénients dans le contexte du jeu multijoueur. Nous avons également consulté de nombreuses ressources en ligne, notamment la documentation officielle d'Unity sur Netcode for GameObjects, des tutoriels vidéo tels que Introduction to Multiplayer Networking with Unity et LAN Multiplayer Implementation (Unity Mirror Tutorial), ainsi que des documents techniques sur l'hébergement de serveurs de jeu. Ces recherches nous ont fourni une base solide pour mettre en place les connexions et synchroniser les données en réseau local.

Le choix du mode LAN s'explique par ses nombreux avantages. Ce mode est accessible hors connexion, ce qui est essentiel pour un jeu coopératif local. Il offre également une latence réduite par rapport aux jeux en ligne classiques et s'avère relativement simple à implémenter grâce aux outils fournis par Unity. Ce travail nous a permis de concevoir un mode multijoueur stable et adapté aux besoins de notre jeu Syndrome Fear.

4.3 Programmation des interactions des personnages

La gestion des déplacements du personnage dans *Syndrome Fear* repose sur une série d'étapes essentielles, de l'importation des ressources graphiques à la création des animations et de la gestion des collisions.

4.3.1 Étapes de développement

1. **Importation du sprite :** Le sprite du personnage principal a été importé et positionné sur le fond de base de la scène.

2. **Création du script de mouvement :** Un script a été développé pour gérer les déplacements. La logique principale repose sur l'utilisation d'un vecteur directionnel basé sur les entrées utilisateur (*input*). La vitesse de déplacement (*speed*) est initialisée, et le vecteur est mis à jour dynamiquement en fonction des touches pressées par le joueur. Unity prend en charge le déplacement du personnage en ajustant automatiquement sa position selon les calculs du script.

3. **Animation des déplacements :** Cinq animations ont été créées pour représenter les différents états du personnage : déplacement vers le haut, vers le bas, vers la gauche, vers la droite et une position statique. Les postures ont été définies directement dans Unity à l'aide du *Sprite Editor* et des outils d'animation. Ces animations sont ensuite associées aux différentes entrées utilisateur via l'outil *Animator*, permettant une transition fluide entre les états du personnage.

4. **Gestion des collisions et fond de scène :** Le fond de scène a été conçu et intégré avec des *collider boxes* pour définir les zones accessibles et interdites du jeu. Cette étape a permis de garantir que le personnage interagit correctement avec son environnement.

5. **Création de la navigation entre les scènes :** Une nouvelle scène contenant le menu principal a été développée. Un *Canvas* a été créé pour accueillir les éléments de l'interface utilisateur, comme les boutons. Un script a été associé à ce *Canvas* pour permettre la transition entre le menu principal et la scène du premier niveau.

4.3.2 Difficultés rencontrées

Lors du développement, certaines difficultés ont été rencontrées :

- **Gestion des hitbox :** Le traitement des collisions avec l'arrière-plan s'est révélé complexe. Initialement, nous avions envisagé un système de *parsing*, mais il s'est avéré mal adapté aux besoins et a été remplacé par une solution plus simple et robuste basée sur des *collider boxes*.

Voici un extrait du code des déplacements :

```
//Type :
//  - v = Vector
//
//Que fait il :
//  - Gere les deplacements du personnage jouable.

//Dependence :
using UnityEngine;

public class Player : MonoBehaviour
{

    //Initialisation :
    public Rigidbody2D rb;
    public float fspeed = 5f;
    Vector2 vmoov;
    public Animator animator;

    //Repetition :
    void Update()
    {
        //Gere les deplacements :
        vmoov.x = Input.GetAxisRaw("Horizontal");
        vmoov.y = Input.GetAxisRaw("Vertical");
        if (vmoov.x != 0 && vmoov.y != 0)
        {
            vmoov.y = 0;
        }
        rb.MovePosition(rb.position + vmoov * fspeed * Time
           .deltaTime);
        //  Gere l'animation :
        animator.SetFloat("Horizontal", vmoov.x);
        animator.SetFloat("Vertical", vmoov.y);
        animator.SetFloat("fspeed", vmoov.magnitude);

    }
}
```

4.4 Programmation et design des menus

4.4.1 Conception graphique des éléments

Le titre du jeu *Syndrome Fear* et les boutons (*Jouer*, *Options*, *Quitter*) ont été réalisés dans un style pixel-art, correspondant à l'esthétique générale du jeu. Les graphismes ont été créés en utilisant une palette de couleurs limitée pour conserver un aspect rétro. Les boutons possèdent un effet de relief pixelisé et des contours nets pour donner un rendu interactif.

4.4.2 Mise en place du fond et du Canvas

L'arrière-plan du menu est constitué d'une image de couloir sombre, basée sur le logo du jeu. Cette image a été intégrée comme sprite dans Unity et placée en arrière-plan au sein d'un *Canvas* Unity.

4.4.3 Configuration des boutons dans Unity

Les boutons ont été créés à l'aide du système d'interface utilisateur (*UI*) d'Unity. Voici les étapes principales :

- Chaque bouton est un *GameObject* avec un composant `Button`.
- Les sprites graphiques des boutons sont définis dans les propriétés du composant `Button`.
- Les textes des boutons (*Jouer*, *Options*, *Quitter*) sont ajoutés via des objets enfants `TextMeshPro`.

Chaque bouton est ensuite relié à un script C# pour gérer les actions associées :

- **Jouer** : Charge la scène principale du jeu.
- **Options** : Charge la scène des paramètres.
- **Quitter** : Quitte l'application.

4.4.4 Navigation entre les scènes

La navigation entre les scènes est réalisée grâce au *SceneManager* d'Unity. Voici un extrait du script associé :

```
//Ce script gere les boutons du main menu.

//Dependances :
using UnityEngine;
using UnityEngine.SceneManagement;

public class ChangeScene : MonoBehaviour
{
    //Permet de changer de scene.
    public void Scene(string sceneName)
    {
        SceneManager.LoadScene(sceneName);
    }

    public void Quit()
    {
        Application.Quit();
    }
}
```

4.4.5 Difficultés rencontrées

- **Alignement des éléments :** S'assurer que les éléments du menu (titre, boutons, arrière-plan) restent correctement centrés et proportionnés sur différents types d'écrans (résolutions et ratios variés).

4.5 Le serveur GIT

Au début de notre projet, nous nous sommes posés la question de la gestion du code source, afin que chaque membre de l'équipe puisse le modifier simultanément sans conflit.

Git s'est rapidement imposé comme l'outil idéal : ce système, basé sur l'utilisation de branches, permet un travail en parallèle tout en conservant l'historique des modifications. Il offre également la possibilité de maintenir plusieurs versions du code et de revenir à un état antérieur en cas de problème.

Dans un premier temps, nous avions envisagé d'utiliser GitLab, un outil open source très complet. Il propose une interface graphique conviviale, l'intégration des clés SSH pour une meilleure sécurité, la gestion de plusieurs projets et groupes, ainsi que de nombreuses fonctionnalités avancées.

Nous avions décidé de l'installer sur un Raspberry Pi 3 B+ avec Raspberry OS, dans une optique d'indépendance totale. Cependant, nous avons rencontré plusieurs difficultés dès le début de cette mise en place.

1. Incompatibilité d'architecture

Le système d'exploitation installé sur le Raspberry Pi était en 64 bits, alors que GitLab n'est compatible qu'avec une version 32 bits pour cette architecture.

2. Problèmes de performance

Une fois l'installation terminée, nous avons constaté un manque de puissance flagrant de la part du Raspberry Pi. Celui-ci peinait à exécuter GitLab et ses outils associés. Par ailleurs, la carte réseau était trop limitée pour gérer efficacement les échanges, héberger le site web et assurer les fonctions du serveur Git.

Face à ces contraintes techniques, nous avons décidé de changer de stratégie. Nous avons opté pour une solution plus légère, à savoir l'installation d'un simple serveur Git sur le Raspberry Pi.

Infrastructure actuelle

Nous utilisons désormais un Raspberry Pi 3 B+ équipé de Raspberry OS en version 32 bits, avec un serveur Git fonctionnel. Cette configuration nous offre une solution stable et adaptée à nos besoins, tout en restant indépendante.

4.6 Le site internet

Avoir un site Internet est important : il s'agit de la vitrine de notre projet et de notre entreprise. Notre site web est accessible à tout moment à l'URL `http://syndromefear.chefmine.me:16385/`. Ce site web est hébergé par nos soins sur la Raspberry Pi susmentionnée.

Les différentes sections du site web sont accessibles depuis une barre de navigation située en haut de chaque page. Le plan du site est le suivant :

- **Accueil** : L'accueil du site est composé d'une présentation rapide ainsi que du synopsis du jeu.
- **Ctrl+Alt+Elite** : Cette page présente notre équipe et regroupe nos biographies individuelles.
- **Déroulement du projet** : Cette section décrit l'origine de l'idée *Syndrome Fear* et inclut une timeline présentant l'avancée du projet, actualisée régulièrement. Elle comporte également un résumé des problèmes rencontrés et de leur résolution.
- **Téléchargements** : Les différents cahiers des charges et rapports déjà produits y sont disponibles. Certains téléchargements, présentés sous une autre couleur, ne sont pas encore disponibles, mais indiquent aux visiteurs qu'ils le seront prochainement.
- **Liens utiles** : Cette page regroupe les différents outils utilisés, sous forme de liens cliquables prêts à l'emploi. Cette liste est actualisée régulièrement.

Graphiquement parlant, le site utilise des couleurs définies dans une palette réalisée au préalable. Cela garantit une certaine harmonie avec le logo de *Syndrome Fear*, autour duquel est centré le site.

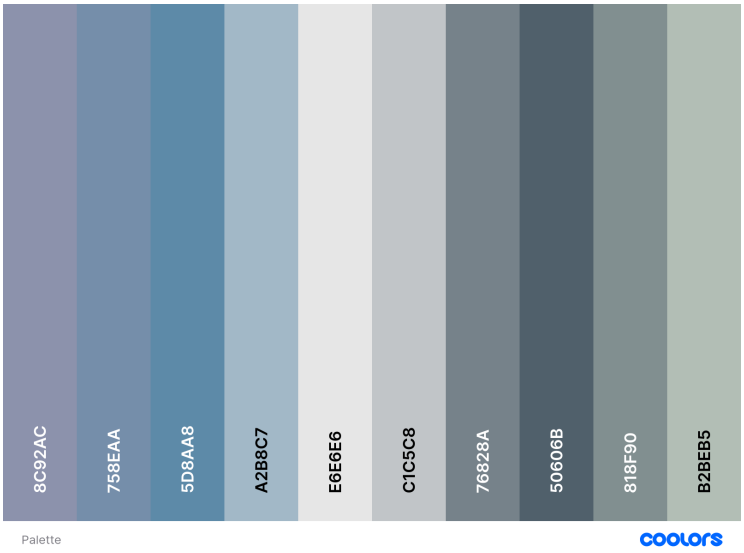

Figure 12: Palette de couleurs. Les éléments d'interface utilisateur du jeu respecteront cette palette.

4.6.1 Création du site

Lors de la création du site Internet, nous avons dû réfléchir aux outils que nous allions utiliser. La première option qui s'offrait à nous était d'utiliser le trio HTML, CSS et JavaScript, trio qui constitue la base de tout site web.

Nous avions aussi la possibilité d'utiliser des frameworks modernes, comme React, Vue, Next.js, qui proposent une approche plus structurée, réduisent le temps de développement, et sont plus faciles à utiliser lorsque les projets se complexifient.

Notre attention s'est portée sur un framework que nous ne connaissions pas auparavant : Astro. Nous avons finalement choisi de l'utiliser pour plusieurs raisons.

Tout d'abord, Astro permet d'intégrer d'autres frameworks comme React, Vue ou Svelte si nécessaire, tout en offrant la possibilité de travailler directement avec du HTML, CSS et JavaScript, ce qui nous donne une très grande liberté.

Étant donné que notre site ne nécessite pas une interaction serveur-client complexe, Astro s'est révélé être un choix idéal pour générer des pages statiques très rapidement, notamment, parce qu'il permet de créer facilement des composants qui seront réutilisables.

4.6.2 Déployer le site

Pour déployer notre site web sur notre Raspberry Pi, nous avons choisi d'utiliser Lighttpd, un serveur web léger et performant. Ce choix a été guidé par les ressources limitées du Raspberry Pi. Pourquoi Lighttpd ?

Lighttpd est particulièrement bien adapté aux environnements avec peu de ressources grâce à sa légèreté et à sa faible consommation de mémoire. De plus, il est simple à configurer, ce qui nous a permis de mettre rapidement en ligne notre site. Cependant, Lighttpd intègre très peu d'outils pour rendre un site dynamique, ce qui ne pose pas de problème immédiat dans notre projet actuel, mais pourrait en poser à l'avenir si nous avons besoin de fonctionnalités plus avancées. Dans ce cas, une solution comme Apache pourrait être envisagée.

5 Planning prévisionnel

Nous venons de dépasser le premier jalon de la création de notre jeu vidéo, et nous nous dirigeons vers le second jalon, prévu pour le mois de mars.

Dans cette seconde étape, nous allons finaliser le design de notre jeu vidéo, avec notamment la création des énigmes que nos joueurs auront à résoudre.

Nous allons poursuivre la programmation du jeu, avec l'implémentation de la carte de jeu, boîtes de texte et de l'inventaire, ainsi que la progression du mode multijoueur et le commencement de l'implémentation des énigmes.

6 Conclusion

Le jeu Syndrome Fear n'est qu'à sa première étape de construction. Cependant, nous sommes fiers du travail déjà accompli, et nous sommes confiants sur la réussite de la suite du projet. Jusqu'ici, grâce à notre organisation, notre répartition des tâches et la cohésion de l'équipe, notre planning n'a pris aucun retard, et le jeu tel que nous l'avons imaginé prend forme.