

Cahier Des Charges

LockedInGames

Escape



Crée par:

Guilhem Petit, Rafael Magalhaes, Baptiste Batrancourt

Contents

1	Introduction	2
1.1	L'histoire du jeu	2
1.2	Répartition	4
2	Cahier des charges fonctionnel	5
2.1	Origine et nature	5
2.2	Objet de l'étude	5
2.3	Etat de l'art	5
2.4	Votre entreprise	6
2.5	Votre équipe	7
3	Avancement	8
3.1	Multijoueur	8
3.2	Site Internet	10
3.3	Décors	10
3.4	Logiciel et langage utilisé	12
3.5	Apparence du personnage et du monstre	14
3.6	Interface Menu	16
3.7	Enigmes	18
3.8	Énigme 1 : La salle de démarrage	18
3.9	Énigme 2 : Le système d'extraction	19
3.10	Énigme 3 : Le marché et la plante	20
3.11	Énigme 4 : Le grand saut final	22
3.12	Gestion du temps et difficulté	25
3.13	Musique	25
3.14	Inventaire	26
3.15	Barre de vie	28
3.16	PNJ Trader	29
3.17	Épreuve du morpion	31
3.18	Ennemis et pièges	34
3.19	Implementation du dash : entre gameplay, animation et synchronisation	36
3.20	La limite des solutions brutales	36
3.21	Une nouvelle approche : la coroutine et le tempo	36
3.22	Les défis de l'animation dynamique	37
3.23	Et le multijoueur dans tout ça ?	38
3.24	Séparer les responsabilités : un modèle plus propre	38
3.25	une petite fonctionnalité, un grand impact	39
3.26	La pince interactive : conception et réalisation	39
3.27	Plateforme mouvante synchronisée en multijoueur	40
3.28	Pathfinding et IA du monstre	40
3.29	Scène de crédits	42
4	Conclusion	43
4.1	Conclusion	43
4.2	Les joies	44
4.3	Les peines	44
4.4	Les leçons tirées	44
5	Répartition	46

1 Introduction

Ce rapport présente le fruit de plusieurs mois de travail sur le projet Escape, un jeu vidéo en 2D à l'ambiance immersive et angoissante. À travers ce document, nous retracons l'évolution du projet, depuis la formulation des besoins jusqu'à la réalisation concrète du jeu, en passant par les phases de conception, de développement, de tests, et d'itération.

L'objectif initial était de concevoir une expérience interactive où les joueurs, plongés dans un univers sombre, seraient amenés à résoudre des énigmes pour échapper à un danger omniprésent. Ce projet, au croisement de nos passions communes pour la programmation, les jeux vidéo et les escape games, nous a permis de confronter notre créativité à des contraintes techniques réelles, de renforcer notre travail en équipe et de mieux appréhender la gestion de projet logiciel.

Au fil des pages, nous reviendrons sur les décisions prises, les difficultés rencontrées, les enseignements tirés, et les perspectives d'amélioration envisagées. Ce rapport vise à offrir une vision complète, technique et humaine, de l'élaboration d'un jeu indépendant, de sa conception jusqu'à sa première version jouable.

1.1 L'histoire du jeu

Si le nom de LockedIn Energy ne vous dis rien. Alors voici une présentation de ce que représente ce nom. Une société qui produit de l'énergie, mais pas n'importe laquelle. En effet, LockedIn Energy représente l'une des plus grandes sociétés énergétiques du monde. Elle met en avant son côté scientifique pour produire de l'énergie, ce qui, en prenant en considération les résultats, a l'air d'être très efficace, bien que les réel méthodes utilisées ne sont pas vraiment connues. Cette description peut s'apparenter à de la publicité pour LockedIn Energy. Une personne en particulier s'est fait avoir, un nouvel arrivant qui vient d'être embauché à LockedIn energy, très fier d'avoir travaillé dur pour intégré la société de ses rêves, la meilleure de sa catégorie. Mais en arrivant dans le laboratoire qui d'où vient toute cette énergie, il y fait une découverte terrifiante a propos de cette société. Une découverte qui restera marquée à jamais dans sa tête

Voici la réalité :

Une espèce "animale" plutôt considéré comme des monstres à fait surface, on ne sait pas réellement pourquoi ils sont sortis de sous terre ni vraiment quand. Cependant, naturellement l'humanité, plus précisément les scientifiques du monde entier se sont intéressés à ces monstres, les ont étudiés et analysés en détail. A première vue ils ne sont pas dangereux et ont étaient acceptés à la surface sans vraiment savoir pourquoi ils ont décidés de sortir. Mais LockedIn Energy ont remarqué quelque chose que personne n'a vue. Une capacité surprenante de ces monstres. Ils produisent de l'énergie, mais pas sous n'importe quelles conditions. La voila, la part d'ombre de cette société, l'exploitation d'une espèce inconnue pour le bien de leurs porte-feuilles.

Mais est-ce vraiment une bonne idée de ce servir de quelque chose dont on ignore encore tout ?



Figure 1: ESCAPE

1.2 Répartition

Comme deux membres de l'équipe ne sont désormais plus présents, la répartition a changé.

Répartition	guilhem.petit	rafael.magalhaes-boulan	baptiste.batrancourt
IA	Adjoint	Aide	Responsable
Multijoueur	Aide	Responsable	Adjoint
Déplacement du joueur	Aide	Adjoint	Responsable
Site Internet		Responsable	Adjoint
Sauvegarde			
Enigmes	Responsable		
Décors salles	Adjoint		Responsable
monstres/personnages	Responsable		Adjoint
Interface	Responsable	Adjoint	
Musique	Responsable	Adjoint	
FX	Responsable		Adjoint
Histoire	Responsable		
Cinématique	Aide	Responsable	Adjoint
Design Site WEB		Responsable	Adjoint
Publicité		Responsable	Adjoint

2 Cahier des charges fonctionnel

2.1 Origine et nature

Le projet "Escape" a été imaginé par l'équipe de LokedInGames, un groupe d'étudiants d'EPITA passionnés par l'univers des escape games. Leur objectif est de créer un jeu d'escape game en 2D innovant, où le joueur incarne un scientifique qui doit s'échapper d'un monstre de laboratoire dangereux qui a réussi à s'échapper. Inspirés par leur propre expérience de jeu, les membres de l'équipe ont souhaité accompagner l'expérience du joueur dans une atmosphère sombre et immersive. Dans "Escape", les joueurs doivent naviguer à travers des environnements périlleux, résoudre des énigmes et faire preuve de rapidité et de stratégie pour éviter d'être capturés. En combinant des graphismes 2D dynamiques et une narration engageante, "Escape" cherche à redéfinir l'expérience des escape games.

2.2 Objet de l'étude

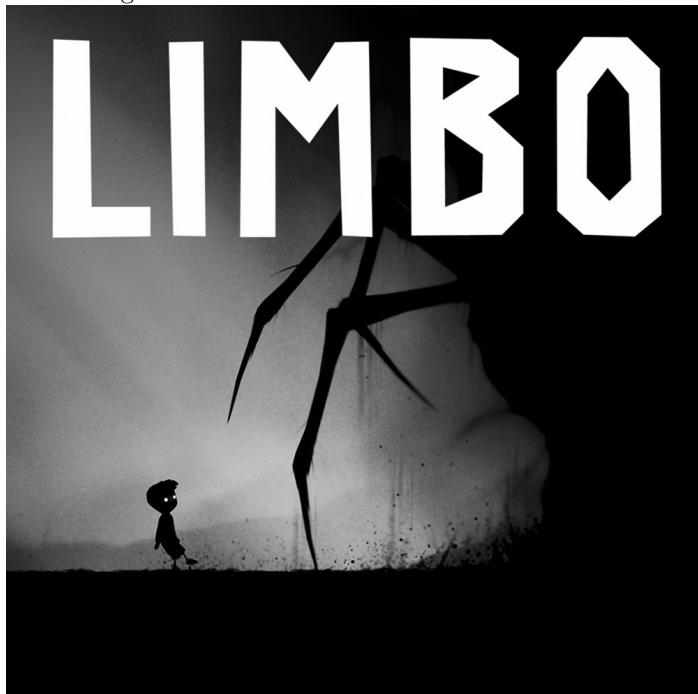
Le but premier de Escape est de plonger les joueurs dans un univers riche en mystères et en défis. Dans un monde où les jeux vidéo atteignent des milliards de joueurs, Escape cherche à allier divertissement, immersion et stimulation intellectuelle, permettant aux joueurs de développer des compétences en résolution de problèmes et en observation. Ce projet offre aussi une opportunité d'apprentissage pour l'équipe, favorisant la collaboration, la communication et le partage des tâches tout au long du processus de développement. Enfin, chaque membre de l'équipe aura l'occasion de développer ses compétences dans divers domaines, notamment la création artistique de niveaux, ainsi que la programmation.

2.3 Etat de l'art

L'histoire des jeux d'aventure en 2D sombres trouve ses racines dans des classiques comme Limbo (2010) et Inside (2016) de Playdead, qui ont su captiver les joueurs avec leur esthétique minimaliste et leur ambiance oppressante. Ces jeux sont des exemples parfaits d'immersion grâce à des environnements riches en détails et des mécaniques de jeu basées sur l'exploration et la résolution d'énigmes.

Hollow Knight (2017), développé par Team Cherry, est une autre référence majeure qui a révolutionné le genre avec son vaste monde interconnecté et sa direction artistique inspirée des contes sombres. L'aspect de découverte et l'importance de l'exploration font également écho à des jeux comme Celeste (2018), qui, bien que plus lumineux, partage l'idée d'un voyage introspectif à travers des défis significatifs. Escape s'inspire de ces titres pour offrir une expérience de jeu immersive et engageante, combinant atmosphère sombre et mécaniques d'énigmes.

Figure 2: Illustration de Limbo



2.4 Votre entreprise

Fondée en 2024 par cinq étudiants d'EPITA : Guilhem, Georgina, Baptiste, Ioan et Rafael, LokedInGames est née de la passion partagée des escape games et de la volonté de réinventer cette expérience. Ces étudiants se retrouvaient régulièrement pour résoudre des énigmes, se plonger dans des univers mystérieux et vivre l'adrénaline unique des escape games. Mais après des années à enchaîner les mêmes schémas, un sentiment d'ennui les a envahis. La frustration grandissait. Les jeux se ressemblaient, la tension des débuts s'estompait, et ils n'étaient plus aussi captivés. C'est cette lassitude qui a déclenché l'idée fondatrice de LokedInGames : ils allaient eux-mêmes créer des jeux qui leur redonneraient ce frisson perdu et offrirait une expérience unique aux autres amateurs d'énigmes. LokedInGames n'est pas une entreprise d'escape games comme les autres. Ici, chaque aventure est pensée pour être une expérience immersive où les joueurs sont constamment mis au défi, non seulement par des énigmes originales, mais aussi par une ambiance sombre, mystérieuse et anxiogène. L'objectif est clair : sortir de la routine, surprendre à chaque instant, et transporter les joueurs dans des scénarios inédits, où l'émotion et la coopération deviennent des éléments centraux. En repoussant les limites du jeu d'énigmes, ils souhaitent créer des univers où chaque détail, chaque interaction compte, et où les joueurs sentent la pression monter à chaque étape. Chaque membre de l'équipe apporte son expertise pour que chaque jeu soit une fusion parfaite entre le design narratif, la technique et l'innovation. Escape est le premier jeu inventé par LokedInGames. Malgré des ressources limitées, ce jeu est la concrétisation de cette vision. Inspiré par leur propre expérience des escape games, Escape plonge les joueurs dans un environnement oppressant et mystérieux, où chaque élément du décor peut être une clé pour progresser. Ce jeu ne se contente pas de proposer des énigmes à résoudre, il pousse les joueurs à réfléchir vite, à réagir sous pression et à observer attentivement leur environnement. La tension monte au fur et à mesure que l'on progresse, et l'immersion est totale. Escape s'impose déjà comme une réinvention du genre. Rassembler les joueurs, développer des histoires toujours plus intrigantes et résoudre des mystères sont la clé pour renouveler l'expérience escape game.



Figure 3: LockedInGames

2.5 Votre équipe

Batrancourt Baptiste

Etudiant à l'école d'ingénieur informatique EPITA et passionné de programmation, jeux ou entre autres, escape games. Ce projet avec mes camarades m'interessent énormement. En effet, il réunit les deux choses que j'aime le plus, à savoir, les escape games et la programmation, en plus de l'ambiance qu'il aura, une ambiance inquiétante et mystérieuse, l'horreur me plaît, alors c'est encore mieux. De ce fait, je pense être adapté pour m'occuper de la partie programmation du jeu. Je pense que c'est une expérience très enrichissante principalement d'un point de vue de l'expérience en codage, mais également en travail d'équipe. J'ai vraiment hâte de pouvoir concrétiser ce jeu, et de, moi même y jouer, car étant l'un des meilleurs je pense avoir fais le tour de l'escape game, ainsi pouvoir tester quelque chose de ce genre. Même si j'aurais créé une partie du jeu, et que dans ce cas là, cela perd de son intérêt.

Petit Guilhem

Actuellement étudiant à l'école d'ingénieurs en informatique EPITA, je suis passionné par l'univers numérique et les jeux vidéo. Il y a six ans, j'ai découvert avec ma famille le concept des escape games, où une équipe doit résoudre des énigmes pour progresser dans une histoire captivante. Cette expérience a éveillé ma curiosité, et j'ai ensuite décidé de m'associer à un groupe de quatre camarades pour explorer les différentes facettes de ce genre de jeu. C'est ainsi que j'ai rejoint notre entreprise, LockedInGames, avec l'ambition d'apporter ma créativité et mon expérience en matière d'escape games. Je souhaite contribuer à la conception d'énigmes intrigantes et immersives, tout en utilisant mes compétences informatiques pour développer un jeu vidéo qui saura captiver les joueurs. Ensemble, nous aspirons à créer une expérience unique qui allie challenge et plaisir.

Rafael Magalhaes

Depuis l'enfance, les jeux vidéo et les escape games m'ont toujours fasciné, capturant mon imagination et mon intérêt. Aujourd'hui, en tant qu'étudiant à l'EPITA, cette passion s'est transformée en véritable vocation professionnelle. Mon rôle de chef de projet chez LockedInGames me permet d'allier cette passion à une démarche concrète et structurée, et la réussite de ce projet est devenue une mission essentielle pour moi.

Mon objectif est de créer une expérience immersive où l'ambiance captivante et la dynamique d'équipe s'harmonisent de manière fluide. Avec un esprit d'équipe soudé et un leadership affirmé, je suis convaincu que nous réussirons ensemble à atteindre nos objectifs communs.

Mon expérience en tant que joueur m'offre une vision claire des attentes du public. Je souhaite transformer cette expertise en un atout précieux pour l'équipe, en combinant mes compétences en programmation avec une bonne compréhension des mécaniques de jeu, ce qui rendra l'expérience vraiment inoubliable.

Le projet "Escape" vise à créer un jeu vidéo 2D immersif inspiré des escape games, où les joueurs devront résoudre des énigmes et surmonter des obstacles dans un environnement sombre et mystérieux. L'objectif est d'offrir une expérience captivante et stimulante, alliant stratégie, réflexion rapide. Ce projet permettra également à l'équipe de développer des compétences en programmation, design artistique et gestion de projet, tout en proposant une réinvention du genre escape game.

3 Avancement

3.1 Multijoueur

Dans le cadre de notre projet Unity, nous avons décidé d'intégrer une fonctionnalité multijoueur en utilisant le framework **Mirror**. Nous avions déjà une base de jeu fonctionnelle en solo, mais le passage au multijoueur s'est révélé plus complexe que prévu.

Nous avons commencé par importer le package Mirror et à configurer une scène de base avec un **NetworkManager** pour gérer la connexion des joueurs. La mise en place de la connexion a été relativement simple grâce à l'utilisation du **NetworkManagerHUD**, ce qui nous a permis de tester rapidement la création et la connexion des joueurs. Cela nous a permis de vérifier si le système de connexion fonctionnait correctement et si les joueurs pouvaient rejoindre une partie sans problème majeur.



Figure 4: Mirror

Une des premières difficultés a été la création du prefab de joueur. Nous avons dû ajouter un **NetworkIdentity** au prefab pour qu'il soit correctement pris en charge par le réseau et configurer le **NetworkManager** pour qu'il instancie automatiquement le joueur lors de la connexion. Cela impliquait de s'assurer que chaque joueur était bien reconnu comme une entité distincte par le serveur. Nous avons aussi dû mettre en place des mécanismes de synchronisation des déplacements et des actions des joueurs. Pour cela, nous avons utilisé des [**Command**] (pour envoyer des ordres du client au serveur) et des [**SyncVar**] pour gérer la synchronisation des états entre les clients. Ce processus a demandé un certain temps de réflexion et d'ajustements pour que tout fonctionne correctement.

Un autre problème est apparu avec la gestion de la caméra. En solo, la caméra était déjà configurée dans la scène et suivait le joueur sans problème. Cependant, lorsque nous avons intégré le multijoueur, nous avons rencontré une difficulté liée à la différence entre un élément de la scène et un prefab. Pour résoudre ce problème, nous avons utilisé **isLocalPlayer**, une propriété de Mirror. Cette propriété permet de déterminer si un objet est celui contrôlé par le joueur local. En d'autres termes, **isLocalPlayer** retourne **true** uniquement pour l'instance du joueur local, ce qui nous a permis de nous assurer que la caméra suivait uniquement le joueur que le client local contrôlait. Cela a permis de corriger le com-

portement de la caméra et de faire en sorte qu'elle suive correctement chaque joueur local sans interférer avec les autres joueurs.

Nous avons également rencontré des difficultés lors de l'adaptation des boutons et des interactions. En solo, les boutons interagissaient correctement avec les éléments de la scène, comme l'ouverture des portes. Cependant, en multijoueur, le problème venait du fait que les boutons n'étaient pas correctement reliés au joueur local, ce qui empêchait l'action d'être déclenchée de manière appropriée. Cette difficulté était en grande partie liée au fait que `isLocalPlayer` ne fonctionnait pas correctement dans ce contexte. La solution a été d'intégrer le joueur au script des boutons, ce qui nous a permis de vérifier que l'action était bien exécutée uniquement pour le joueur qui l'avait déclenchée. Une fois cette liaison correctement effectuée, les boutons ont commencé à fonctionner comme prévu, et l'interaction entre les joueurs et les objets de la scène a été correctement synchronisée.

Un autre défi a été la gestion des animations. Au départ, les animations fonctionnaient bien pour chaque joueur, mais le problème résidait dans le fait que les animations étaient déclenchées pour les deux joueurs en même temps, même si un seul d'entre eux était censé les initier. Ce problème était lié à la manière dont les événements de déclenchement des animations étaient synchronisés entre les clients. En utilisant les outils d'Unity, nous avons pu résoudre ce problème. En ajustant la gestion des commandes et la synchronisation via `SyncVar` et en nous assurant que seules les actions du joueur local déclenchaient les animations, nous avons évité que les animations ne se déclenchent simultanément pour tous les joueurs. Cela a permis de garantir que seul le joueur local active les animations au moment approprié. La partie multijoueur a représenté le principal défi technique du projet. Bien que nous ayons pu synchroniser plusieurs éléments avec succès (comme la pince ou les plateformes), d'autres aspects restent imparfaits.

Les problèmes principaux :

- **Changement de caméra** lors de la mort d'un joueur : parfois, les clients ne voyent plus rien, car la caméra n'était pas activée correctement ou les méthodes utilisées n'ont pas les autorisations nécessaires.
- **Comportement du monstre** : en multijoueur, il devenait difficile d'assigner correctement la cible et de synchroniser ses déplacements.
- **Silent failures** : certains bugs n'avaient pas de messages d'erreur clairs, et pouvaient venir d'une mauvaise authority, d'une variable non synchronisée, d'un client RPC mal appelé, ou d'une déconnexion temporaire. Identifier la cause précise était très complexe.

Un projet jouable, mais perfectible

Malgré les difficultés, nous avons réussi à mettre en place une expérience jouable, cohérente, avec plusieurs mécaniques originales. Le jeu est pleinement fonctionnel en local, et jouable en multijoueur, même si certains bugs peuvent altérer l'expérience et nuire à l'immersion de certains joueurs.

Nous préférions être transparents : le multijoueur fonctionne, mais n'est pas exempt de soucis techniques, notamment au niveau du pathfinding et de la gestion de caméra. Des ajustements seraient nécessaires pour garantir une stabilité totale.

Pour autant, l'essentiel est là : des mécaniques synchronisées, une pince dynamique, un monstre actif, des énigmes interactives. Ce projet nous a permis de découvrir en profondeur la complexité du multijoueur, la gestion de l'authority, et les subtilités du netcode avec Mirror.

3.2 Site Internet

Créer le site n'a pas été une tâche facile, et nous avons rencontré plusieurs difficultés au cours du processus. Par exemple, nous avons eu du mal à gérer les images. Insérer une image en arrière-plan nous a particulièrement posé problème, car nous ne savions pas comment la configurer pour qu'elle s'affiche correctement sans déformer les proportions ou dépasser les marges. Ajouter des images sur le site en général n'a pas été évident non plus, car certaines ne s'intégraient pas comme nous le voulions, que ce soit au niveau des tailles ou des emplacements. Trouver comment insérer un lien vers la page du jeu a également nécessité pas mal d'essais, car nous n'étions pas certains de la syntaxe à utiliser ni des paramètres pour qu'il fonctionne comme prévu.

En plus de ces aspects techniques, nous avons également eu des difficultés à choisir une palette de couleurs cohérente et agréable à l'œil. Nous avons essayé plusieurs combinaisons, comme le noir et le rouge, mais les couleurs se heurtaient souvent ou rendaient la lecture inconfortable. Cela nous a obligés à expérimenter davantage et à faire plusieurs ajustements avant d'arriver à un résultat satisfaisant. Finalement, nous avons réussi à intégrer une image de laboratoire en arrière-plan, ce qui donne un style distinct au site. Nous avons également ajouté des photos des membres du groupe, qui apportent une touche personnelle et rendent le tout plus vivant. Enfin, nous avons inséré un lien vers la page du jeu, permettant ainsi à ceux qui visitent le site de découvrir notre projet en un clic.

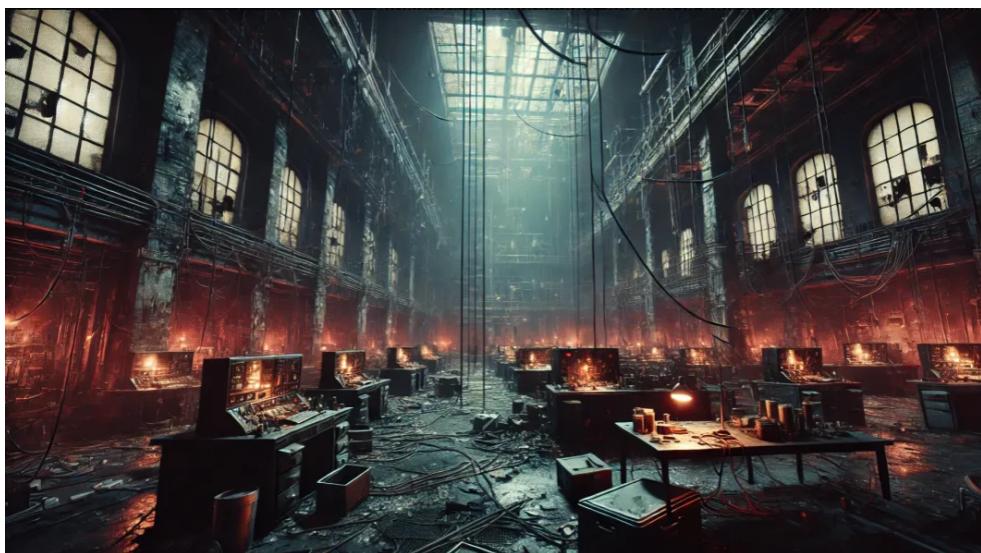


Figure 5: Background Site Web

3.3 Décors

Pour les décors, comme le jeu est en 2D. Nous avons utilisé les tilemaps de Unity. C'est outil nous permet de "dessiner" notre décor. En effet, cela va créer un quadrillage dans laquelle nous pouvons y placer des cases d'une "tilesheet" créer précédemment. Une tilesheet est une image qui à était quadriller par un outil de unity en différent carrés de pixels. De cette manière on peut prendre les différentes zones de la tilesheet pour dessiner sur notre tilemap ce que l'on veut a partir de l'image d'origine. En utilisant ce système, nous avons créer deux tilemaps. Une pour les décors en fond qui ne possède pas de boîte de collision, purement présente pour le décor. Et une tilemap pour les objets qui auront une collision avec le joueur, on y met notamment le sol, les murs par exemple. Enfin, les décors sont constitués d'assets que l'on a principalement récupérés en ligne et quelques-uns que l'on a fait nous mêmes. Le jeu se déroulant dans un laboratoire, ce sont des décors scientifiques, futuristique, où on

peut voir par exemple des machines entre autres.



Figure 6: Tilemaps hiérarchie

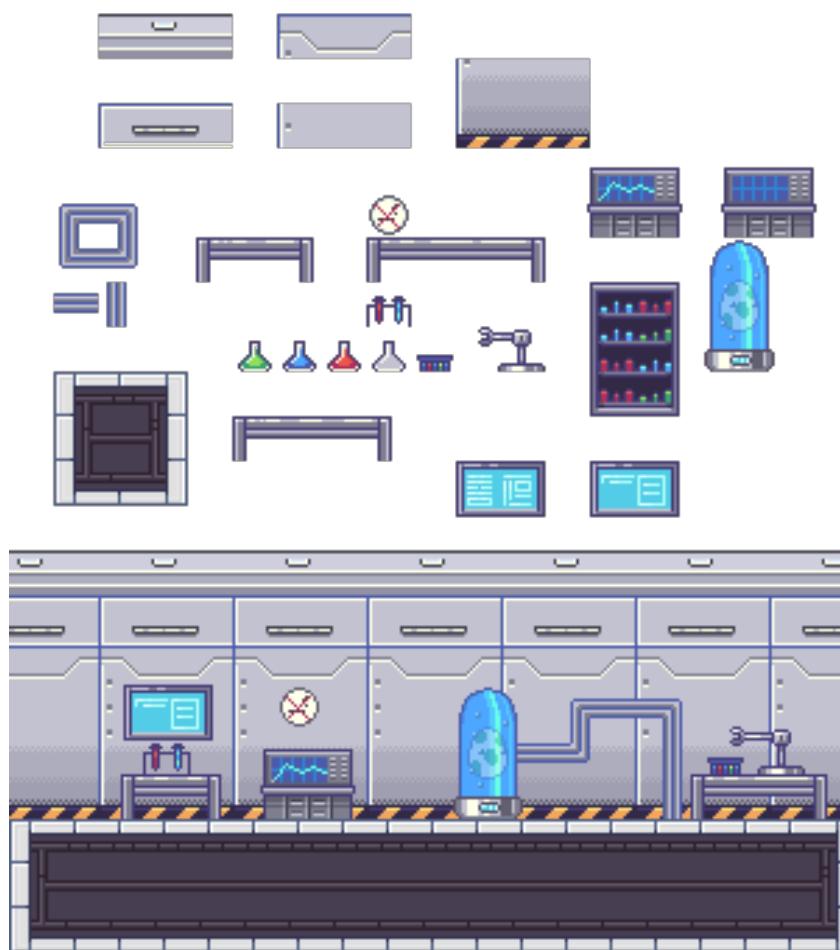


Figure 7: Exemple d'asset utilisé

3.4 Logiciel et langage utilisé

Nous avons utilisé le logiciel Unity pour développer le jeu.

Pour les fonctionnalités actuellement présentes, comme le bouton permettant d'ouvrir une porte et la plateforme mouvante, nous avons rencontré des difficultés, mais aussi trouvé des solutions pour résoudre ces problèmes.

Notamment, pour le bouton qui ouvre la porte, les deux principales difficultés étaient que l'objet qui effectue l'animation et l'objet qui la déclenche sont deux objets différents. Découvrant Unity, nous avons eu du mal à les relier, mais en comprenant un peu mieux comment fonctionne Unity avec la programmation orientée objet (POO), nous avons pu résoudre ce problème. De plus, un autre problème majeur était que l'animation de la porte s'effectuait, mais que la boîte de collision ne bougeait pas. Ainsi, même si la porte semblait ouverte, il était impossible de passer.

Pour régler ce problème, nous avons utilisé un vecteur2 pour déplacer la boîte de collision de la porte vers le haut (dans le plafond) lorsqu'on appuie sur le bouton.

```
public void Move_Hitbox()
{
    Offset = new Vector2(0,5);
    hitbox = GetComponent<Collider2D>();

    hitbox.offset = Offset;
}
```

Figure 8: Bouger la hitbox

Nous avons également ajouté une lumière sur le bouton, indiquant qu'il s'agit d'un élément important et utile. À l'appui du bouton, cette lumière disparaît. Le seul problème rencontré pour cela était d'utiliser un certain renderer pour qu'Unity reconnaisse le namespace Light2D, afin d'ajouter la lumière 2D au code du bouton.

Enfin, nous avons travaillé sur les mouvements et les animations du personnage, en mettant en place une animation de marche et une animation idle pour les moments où le personnage ne bouge pas. Les déplacements du personnage sont gérés via un script qui permet de se déplacer à gauche et à droite, tout en affichant l'animation adaptée. Nous avons également ajouté une condition pour détecter si le personnage est au sol, évitant ainsi qu'il puisse sauter lorsqu'il est en l'air ou en contact avec un mur sur le côté. Enfin, nous avons réglé le changement de direction du sprite pour qu'il s'oriente en fonction des mouvements du joueur.

Pendant ce processus, nous avons rencontré plusieurs défis. L'animation idle ne se déclenchaient pas correctement lorsque le personnage cessait de bouger, et il était crucial d'empêcher l'animation de marche de se poursuivre dans des situations inappropriées. Nous avons donc ajusté les transitions dans l'Animator et ajouté des conditions spécifiques dans le script pour résoudre ces problèmes. Un autre défi consistait à empêcher le personnage de sauter lorsqu'il touchait un mur latéral. En améliorant la gestion des collisions et en apportant des modifications ciblées au code, nous avons réussi à corriger ces dysfonctionnements.



Figure 9: Sprite Mouvement

La conception des plateformes et des plateformes mobiles dans notre projet a été une étape cruciale pour ajouter de la variété et du dynamisme à l'environnement de jeu. Dès le départ, nous avons commencé par créer des plateformes statiques, qui constituent les bases du niveau. Ces plateformes servent de points d'appui et sont essentielles pour guider le joueur dans sa progression. Pour les réaliser, nous avons utilisé des sprites simples que nous avons configurés avec des BoxCollider2D pour définir les limites physiques. Ces colliders assurent que le personnage peut marcher dessus sans passer à travers. Nous avons également ajusté leur position et leur taille pour créer une disposition intéressante et progressive, encourageant le joueur à explorer l'espace.

Cependant, les plateformes statiques seules risquaient de rendre le niveau monotone. C'est pourquoi nous avons décidé d'intégrer des plateformes mobiles. Ces plateformes ajoutent une nouvelle couche de complexité au gameplay en introduisant des éléments en mouvement qui nécessitent une synchronisation et une anticipation de la part du joueur. Pour les concevoir, nous avons utilisé des scripts pour contrôler leur déplacement. Par exemple, en utilisant des fonctions comme Vector3.Lerp ou Vector3.MoveTowards, nous avons pu définir des trajectoires précises, qu'il s'agisse de mouvements linéaires, de va-et-vient ou même de déplacements circulaires. Ces trajectoires ont été configurées de manière à offrir des défis variés, comme sauter au bon moment ou suivre une plateforme jusqu'à un nouvel endroit.

La gestion des plateformes mobiles a présenté plusieurs défis techniques. Un des principaux problèmes était de s'assurer que les objets en mouvement interagissent correctement avec les autres éléments du jeu, notamment le personnage. Par exemple, si une plateforme se déplaçait rapidement, il arrivait que le personnage glisse ou soit mal positionné lors du contact. Pour résoudre ce problème, nous avons utilisé des configurations spécifiques dans les Rigidbody2D des plateformes mobiles, en ajustant leur mode de collision pour qu'elles fonctionnent en tant qu'objets cinématiques. Cela garantit que la plateforme déplace également tout ce qui est en contact avec elle, sans provoquer de bugs ou de comportements étranges.



Figure 10: Plateforme Mouvante

3.5 Apparence du personnage et du monstre

Pour les graphismes du jeu, nous avons fait le choix de nous orienter vers un style Pixel Art, qui apporte une touche rétro et facilitant la personnalisation des assets. Afin de donner vie à notre univers, nous avons combiné des assets déjà existants et créé de nouveaux éléments personnalisés. Par exemple, le personnage principal, un scientifique ainsi que le fil électrique cassé ont été personnalisés par notre équipe pour correspondre à l'esthétique et à l'ambiance de l'histoire.

En revanche, le monstre qui poursuit le scientifique est basé sur un asset préexistant que nous avons intégré tel quel dans le jeu. Son apparence a été choisie pour inspirer une peur immédiate chez le joueur et renforcer l'atmosphère angoissante du jeu.

De plus, l'atmosphère chaotique du laboratoire est renforcée par des éléments visuels tels qu'un fil électrique cassé, un détail qui accentue la tension du jeu. Ce genre d'asset, à la fois fonctionnel et narratif, permet de plonger immédiatement le joueur dans un environnement de danger imminent. Chaque choix graphique a été pensé pour non seulement être esthétique, mais aussi pour renforcer l'immersion et la tension de l'expérience de jeu.



Figure 11: Fil électrique cassé

Ce fil électrique cassé est un symbole de l'atmosphère chaotique qui règne dans le laboratoire. Il donne au joueur une indication visuelle immédiate de la situation dangereuse dans laquelle se trouve le personnage principal.



Figure 12: Personnage Principal

Le personnage principal, un scientifique piégé dans le laboratoire.



Figure 13: Monstre

Le monstre, principal antagoniste du jeu, est un asset existant que nous avons intégré directement dans notre jeu. Son apparence a été soigneusement sélectionnée pour correspondre à l'univers visuel tout en instaurant une peur immédiate chez le joueur.

Il y aura également des petits monstres que l'on pourra retrouver le long du jeu, tel que celui-ci.



Figure 14: Exemple de petit monstre

3.6 Interface Menu

Pour mettre en place le menu principal du jeu, nous avons créé une nouvelle scène Unity nommée **MainMenu**. Cette scène contient un *Canvas* qui sert de base à l'interface utilisateur. Le *Canvas* comprend :

- Une image de fond, générée par une intelligence artificielle puis personnalisée pour correspondre à l'ambiance du jeu.
- Un logo du jeu, affichant le titre **ESCAPE** sous forme d'image, placé au centre de l'écran.
- Quatres boutons interactifs :
 - **Play** : Permet de lancer la partie.
 - **Settings** : Ouvre le menu des paramètres.
 - **Story** : Affiche le contexte narratif du jeu.
 - **Quit** : Ferme l'application.

Le script **MainMenu.cs** permet de gérer les interactions des boutons :

- **Lancer le jeu** : Le bouton *Play* charge une nouvelle scène via `SceneManager.LoadScene(leveltoload)`, où `leveltoload` est une variable définissant le niveau à charger.
- **Ouvrir et fermer le menu des paramètres** :
 - `SettingsButton()` active un objet `settingsWindow` qui contient les paramètres du jeu.
 - `CloseSettingsWindow()` désactive cette fenêtre lorsque l'utilisateur souhaite revenir au menu principal.
- **Quitter le jeu** : `QuitGame()` utilise `Application.Quit()` pour fermer l'application (ne fonctionne que dans un build, pas dans l'éditeur Unity).

Le menu des paramètres a été conçu pour permettre au joueur d'ajuster plusieurs aspects de son expérience de jeu. Il est géré via le script **SettingsMenu.cs** et comprend trois options principales :

1. Gestion du volume

- L'utilisateur peut modifier le volume général du jeu grâce à un *Slider*.
- `SetVolume(float volume)` met à jour le paramètre `volume` de l'`AudioMixer` pour refléter le réglage choisi.

2. Mode plein écran

- Un *Toggle* permet d'activer ou désactiver le mode plein écran.
- `SetFullScreen(bool isFullScreen)` applique la modification via `Screen.fullScreen`.

3. Sélection de la résolution

- Un menu déroulant (*Dropdown*) permet au joueur de choisir une résolution parmi celles disponibles sur son écran.
- `Start()` récupère toutes les résolutions compatibles et les affiche dans le *Dropdown*.
- L'option correspondant à la résolution actuelle de l'écran est sélectionnée par défaut.
- `SetResolution(int resolutionIndex)` applique la résolution choisie



Figure 15: Menu D'accueil

3.7 Enigmes

Le jeu est un escape game, il y aura alors des énigmes à résoudre tout au long du jeu.

3.8 Énigme 1 : La salle de démarrage

Le joueur débute dans une salle close, dépourvue d'éléments interactifs à l'exception d'un bouton unique fixé au mur. En l'activant, la porte métallique s'ouvre, dévoilant un passage vers la suite de l'aventure. Cette première interaction simple a pour but de familiariser le joueur avec les mécanismes d'activation, qui seront récurrents dans les énigmes suivantes.

Pour réaliser cette énigme, le bouton utilise le script de la porte et lance son animation lorsqu'il est pressé, cependant la boîte de collision de la porte, elle ne bouge pas malgré l'animation qui se joue. Pour régler ce problème nous avons relié sa boîte de collision au bouton et fais se déplacer la boîte de collision vers le haut en utilisant un vecteur.

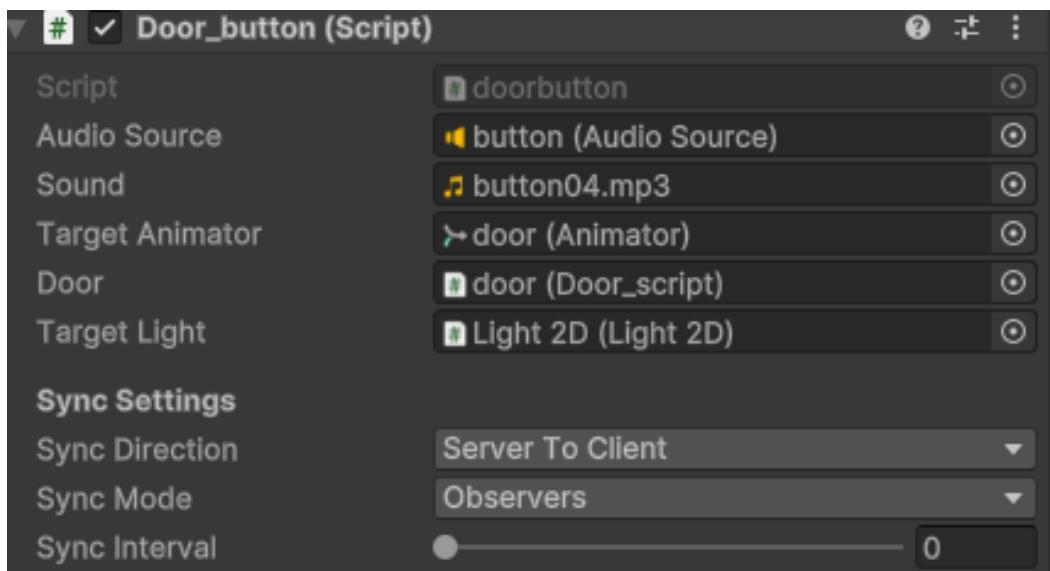


Figure 16: Fields bouton



Figure 17: Porte avec bouton

3.9 Énigme 2 : Le système d'extraction

Le joueur accède ensuite à la salle d'extraction d'énergie, un espace vaste et industriel, dominé par une pince mécanique effectuant des mouvements répétitifs. Celle-ci saisit un matériau et le déverse dans un puits de lave. À proximité, un pupitre de commande muni de trois leviers permet de contrôler la direction de la pince (haut, milieu, bas).

Pour modifier le comportement de la pince et l'amener vers une sortie sécurisée, le joueur doit d'abord collecter trois indices disséminés dans la pièce : deux sont révélés en appuyant sur des boutons cachés, le troisième se trouve dans un coffre. Chaque indice donne un numéro et une direction correspondant à l'un des leviers. Une fois la bonne combinaison trouvée, la pince transporte le joueur vers une nouvelle salle.

La pince constitue l'un des éléments emblématiques de notre gameplay. Elle a pour but d'attraper un joueur et de le déplacer selon une trajectoire précise, jusqu'à une destination dépendant de son statut dans le jeu.

Pour gérer son comportement, nous avons défini plusieurs points de passage :

- **A** : Point de départ de la pince (en haut).
- **A2** : Point correspondant à la position basse de la pince.
- **B / B2** : Zone des flammes qui infligent 50 points de dégâts en continu.
- **C / C2** : suite de l'aventure Si le joueur a réussi l'épreuve précédente

Lorsque le joueur a réussi l'épreuve précédente (identifié par l'activation du "booléen"), la pince l'emmène jusqu'à la zone de sortie et le dépose sans danger. Si ce n'est pas le cas, il est délibérément dirigé vers les flammes ou dans le vide. Cette variation ajoute une tension dramatique au gameplay.

Nous avons également mis en place un bouton permettant, une fois l'épreuve réussie, de supprimer les flammes et d'activer une plateforme permanente, afin que les autres joueurs n'aient plus besoin d'utiliser la pince.

3.10 Énigme 3 : Le marché et la plante

La salle suivante contient un personnage non-joueur (PNJ) qui propose une potion de saut amélioré (JumpBoost) en échange de cinq minutes du chronomètre global. Cette salle présente également une plante, que le joueur devra faire pousser pour progresser. Pour cela, il lui faut un seau d'eau, uniquement accessible via un parcours de saut dans une salle située au tout début du jeu.

En revenant sur ses pas grâce à la potion, le joueur découvre une porte verrouillée par un code inscrit sur un mur. Après avoir déchiffré ce code et réussi un nouveau parcours de saut, il accède à une plateforme où une partie de morpion l'oppose à une intelligence artificielle. En cas de victoire, il obtient le seau d'eau ; en cas de défaite, la plateforme disparaît et le joueur doit recommencer tout le parcours.

La plante possède une boîte de collision en mode IsTrigger, ce qui signifie que le joueur peut aller dedans, et le script de la plante détecte si le joueur est dedans, qu'il appuie sur "E" avec un seau d'eau dans la main, si cela est le cas l'animation de pousse de l'arbre se lance et s'arrête sur la dernière image de l'arbre, entièrement poussé. De plus le seau d'eau se vide pour alors devenir un seau d'eau vide. Egalement il y a une autre boîte de collision en mode IsTrigger en dessous de la plante depuis le début, cette dernière va monté au niveau de l'arbre lorsque le joueur l'aura fait poussé, cette boîte de collision va agir comme une échelle pour permettre au joueur de monter et d'accéder à la suite. Enfin, si jamais le joueur a perdu son seau d'eau avant de faire pousser l'arbre, il pourra simplement regagner une partie de morpion en refaisant le parcours.

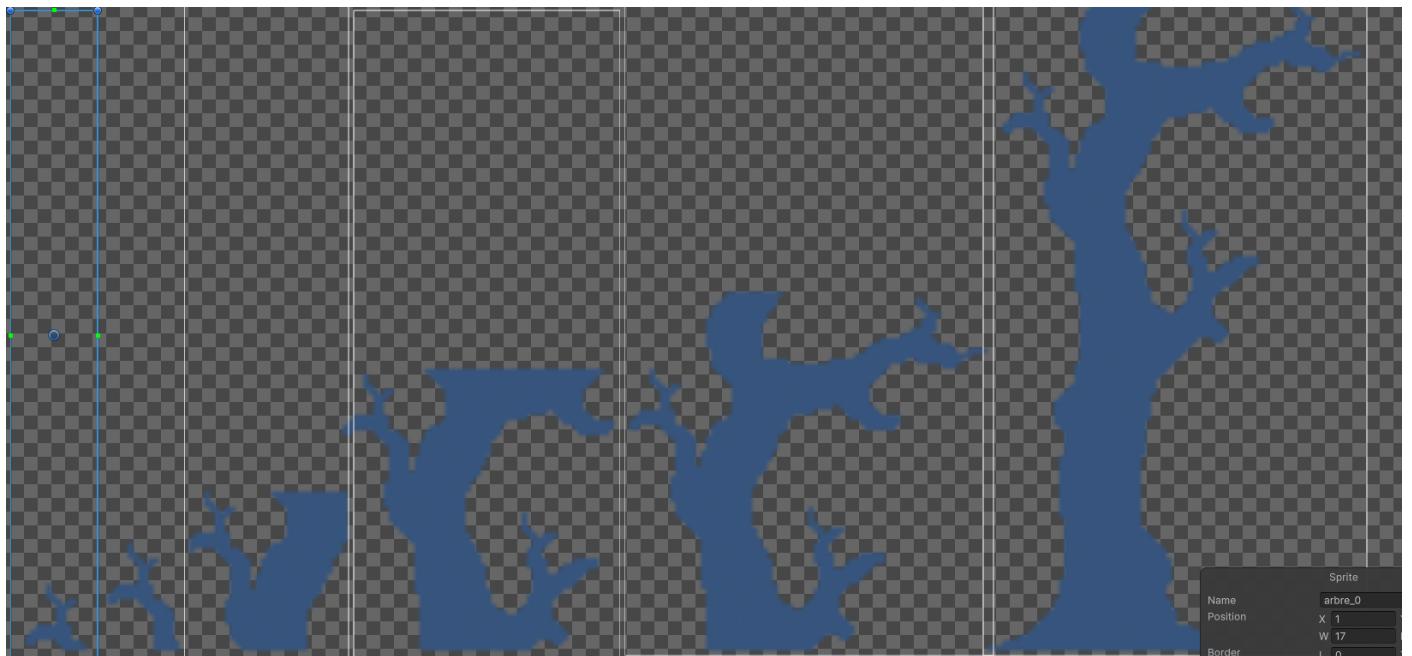


Figure 18: Animation de l'arbre

3.11 Énigme 4 : Le grand saut final

Avec le seau d'eau en main, le joueur peut désormais faire pousser la plante. Celle-ci donne accès à une nouvelle zone où se trouve une fontaine et un large trou bloquant la progression. L'objectif est de reboucher partiellement ce trou pour pouvoir y verser de l'eau et nager jusqu'à l'autre rive. Pour cela, le joueur doit d'abord couper un morceau de bois de l'arbre qu'il a fait pousser, afin d'obstruer le fond du trou. Une fois rempli d'eau, le passage devient praticable, et le joueur peut s'échapper du laboratoire.



Figure 19: planche dans l'inventaire

Le joueur, pour faire apparaître la planche au fond du trou, devra appuyer sur "E" dans la zone de la planche tout en ayant la planche précédemment récupérée sur l'arbre dans sa main. Cela va avoir pour effet de faire avancer la planche sur l'axe de la profondeur. En effet, la planche était au préalable déjà présente mais derrière tout le décor, à l'appui du bouton elle avance de 2 crans vers l'avant et apparaît pour le joueur. De plus, un booléen "ActivePlank" passe en true.

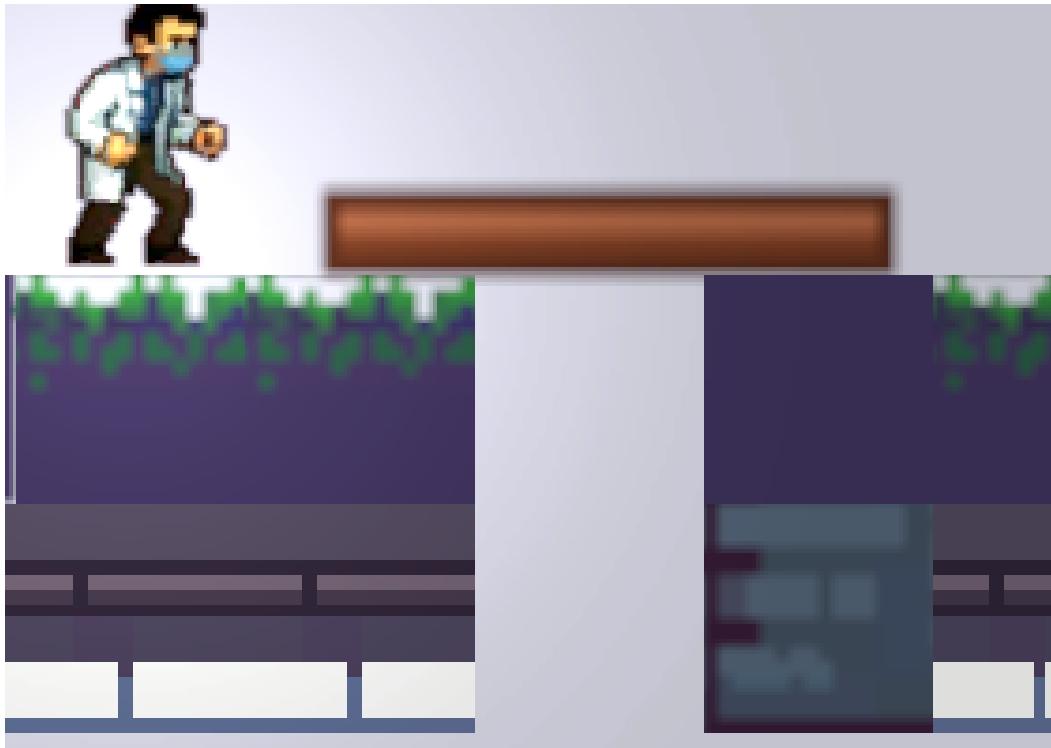


Figure 20: Planche posée

Le joueur pourra donc désormais verser de l'eau dans le trou. Pour ça, il doit aller en haut de l'échelle et appuyer sur "E" avec un seau d'eau rempli dans la main. Cela aura pour effet de faire légèrement monter le niveau de l'eau. Il devra le faire 3 fois pour remplir entièrement le trou. Pour cela, il pourra aller chercher de l'eau à la fontaine et y remplir son seau plus loin derrière. Cette énigme nous a posé plusieurs problèmes, notamment pour faire monter le niveau de l'eau. Nous avons utilisé la méthode "transform.scale" sur l'eau pour changer sa taille à chaque fois que le joueur rajoute de l'eau. Cependant, cela faisait grandir l'eau vers le haut mais aussi vers le bas. Pour corriger cela, sur le sprite de l'eau, nous avons dû changer son pivot, afin que la méthode "transform.scale" ne fasse monter l'eau uniquement vers le haut. De cette façon, cette fonction était terminée. Cependant, il restait à modifier la taille de la boîte de collision faisant flotter et permettant au joueur de nager. N'étant pas un sprite et n'agissant pas de la même façon, nous ne pouvons pas simplement modifier le pivot de la boîte de collision. Nous avons donc décidé de mettre 3 zones différentes que l'on active ou désactive en fonction du nombre de fois que le joueur a ajouté de l'eau pour que la zone d'effet de l'eau coïncide avec ce que le joueur voit.

```

if (isPlayerNearby && Input.GetKeyDown(KeyCode.E) && inventaire.content[inventaire.contentcurrentIndex].itemName == "WaterBucket" && nb_fill <3 && planche.ActivePlank)
{
    Debug.Log("Hello");

    if (nb_fill==0)
    {
        ZoneEau1.enabled = true;
        Debug.Log("Hello1");
        Vector3 currentScale = Water.transform.localScale;
        currentScale.y += 1;
        Water.transform.localScale = currentScale;

    }
    else if (nb_fill==1)

    {
        ZoneEau1.enabled = false;
        ZoneEau2.enabled = true;
        Debug.Log("Hello2");
        Vector3 currentScale = Water.transform.localScale;
        currentScale.y += 2;
        Water.transform.localScale = currentScale;

    }
    else if (nb_fill==2)

    {
        ZoneEau2.enabled = false;
        ZoneEau3.enabled = true;
        Debug.Log("Hello3");
        Vector3 currentScale = Water.transform.localScale;
        currentScale.y += 3;
        Water.transform.localScale = currentScale;

    }

}

nb_fill+=1;
inventaire.content[inventaire.contentcurrentIndex] = EmptyBucket;
inventaire.GetNextItem();
}

```

Figure 21: Script remplir le trou

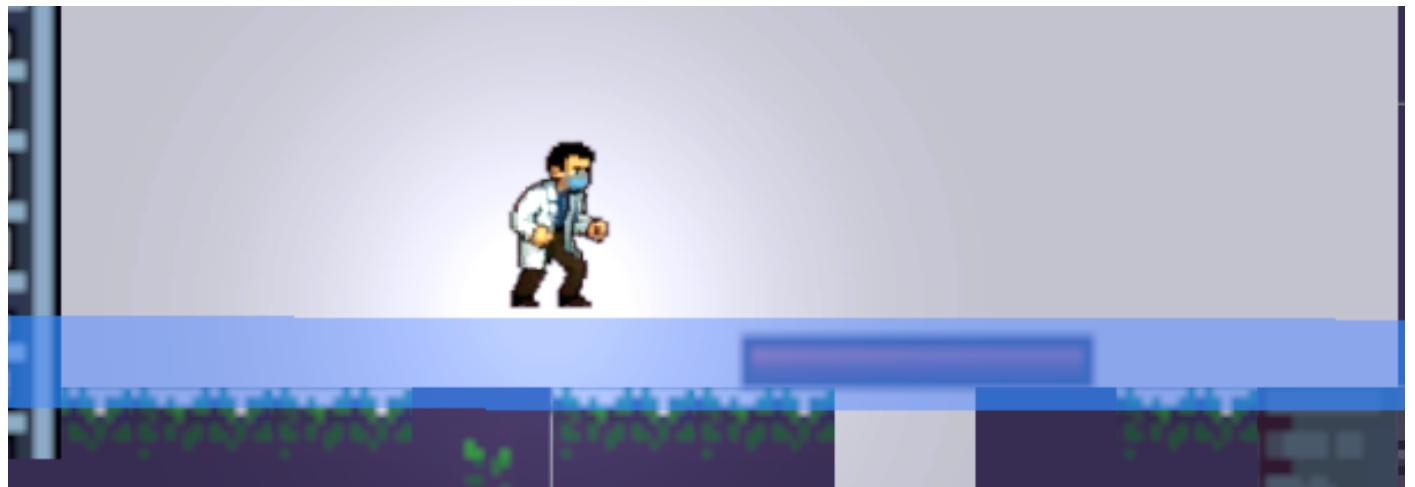


Figure 22: Eau niveau 1

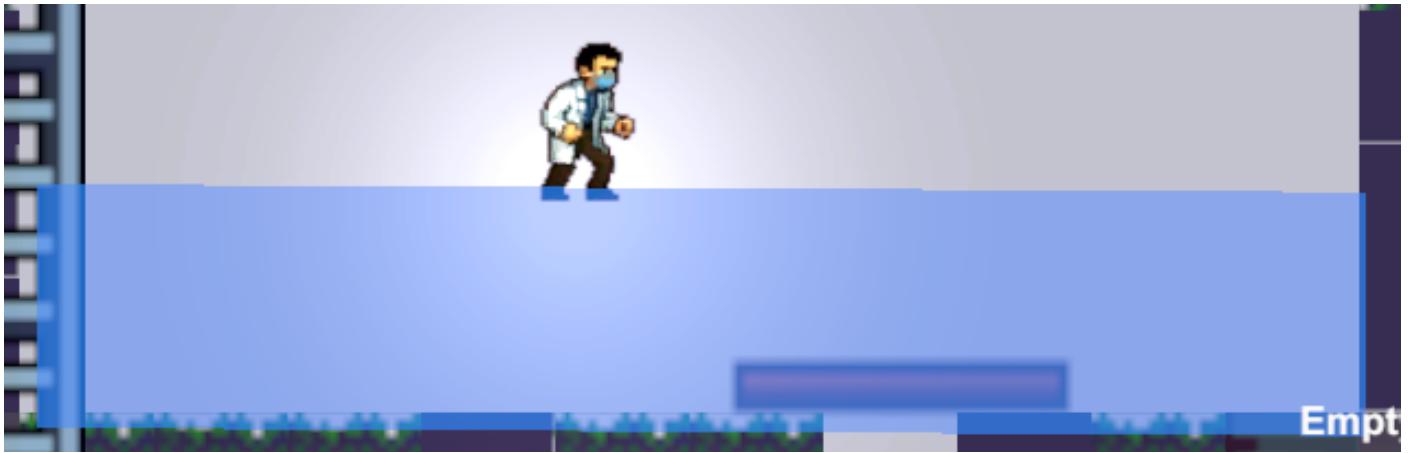


Figure 23: Eau niveau 2

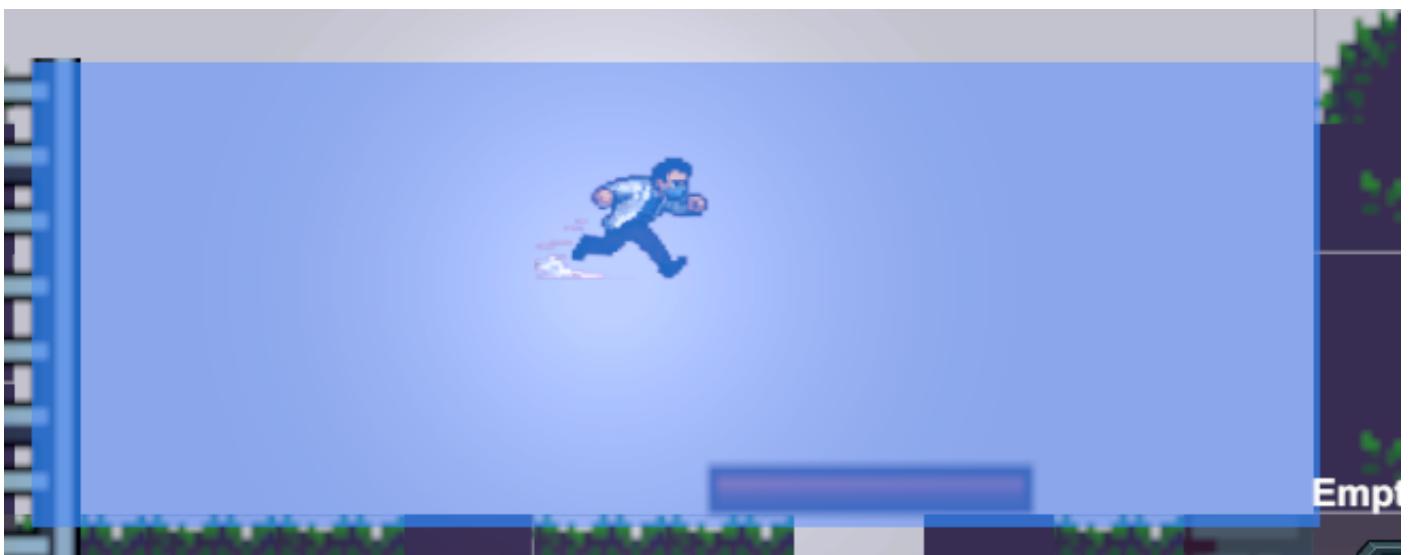


Figure 24: Eau niveau 3

3.12 Gestion du temps et difficulté

Le jeu est soumis à une contrainte temporelle forte : un chronomètre d'une heure est lancé dès le début de la partie. Chaque mort du personnage entraîne une pénalité de dix minutes. Si le temps s'écoule complètement, le monstre enfermé dans le laboratoire est libéré, mettant fin au jeu de manière brutale et définitive. Cette pression constante impose une gestion stratégique du temps, rendant chaque décision cruciale.

3.13 Musique

Concernant la musique, nous avons sélectionné et généré à l'aide de l'IA trois compositions distinctes, que nous avons ensuite personnalisées afin de mieux correspondre à l'ambiance du jeu. Les deux premières musiques ont été pensées pour instaurer une ambiance immersive et captivante, alliant à la fois calme et sérénité, mais avec une touche de tension subtile, afin de renforcer l'atmosphère stressante du jeu. Ces musiques d'ambiance sont conçues pour accompagner le joueur dans ses explorations, tout en maintenant un équilibre entre relaxation et pression.

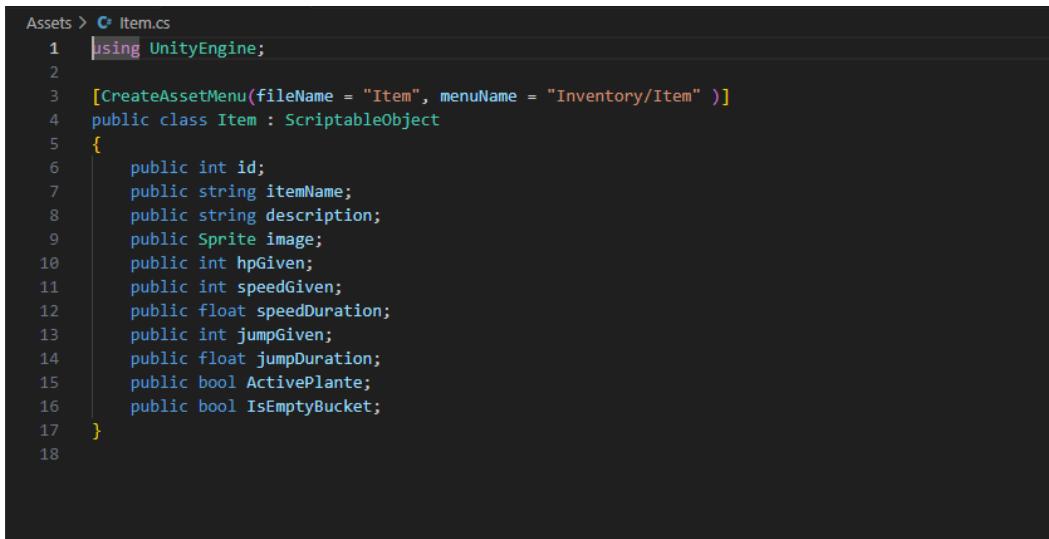
La troisième musique, quant à elle, est bien plus intense et stressante, et elle intervient lorsque le joueur se retrouve face au monstre. Cette transition musicale vise à amplifier l'angoisse et l'urgence du

moment, accentuant l'effet de panique et d'imminence du danger. Chaque morceau a été soigneusement ajusté par notre équipe pour maximiser l'impact émotionnel et renforcer l'expérience immersive du joueur.

3.14 Inventaire

L'inventaire du joueur repose sur une structure simple mais efficace, fondée sur l'utilisation de ScriptableObject dans Unity. Cette approche permet une gestion modulaire et facilement extensible des objets du jeu.

Structure des objets Chaque objet du jeu est défini par un script nommé `Item`, implémenté sous forme de ScriptableObject. Ce script contient toutes les informations nécessaires à la description d'un item : nom, description, image, et effets éventuels (comme la régénération de vie ou l'augmentation de la vitesse). Il comporte également des booléens spécifiques, comme `EmptyBucket`, permettant d'identifier si l'objet correspond à un seau vide, par exemple.



```
Assets > C# Item.cs
 1  using UnityEngine;
 2
 3  [CreateAssetMenu(fileName = "Item", menuName = "Inventory/Item" )]
 4  public class Item : ScriptableObject
 5  {
 6      public int id;
 7      public string itemName;
 8      public string description;
 9      public Sprite image;
10      public int hpGiven;
11      public int speedGiven;
12      public float speedDuration;
13      public int jumpGiven;
14      public float jumpDuration;
15      public bool ActivePlante;
16      public bool IsEmptyBucket;
17  }
18
```

Figure 25: ScriptableObject

Grâce à cette structure, la création d'un nouvel objet est facilitée : il suffit de se rendre dans le menu `Assets > Inventory > Item` pour générer un nouvel item.

Objets disponibles Actuellement, six objets ont été créés :

- Trois potions : **Vitesse**, **Saut**, et **Soin**.
- Trois objets utiles pour la progression dans le jeu : **seau d'eau**, **seau vide** et **planche de bois**.

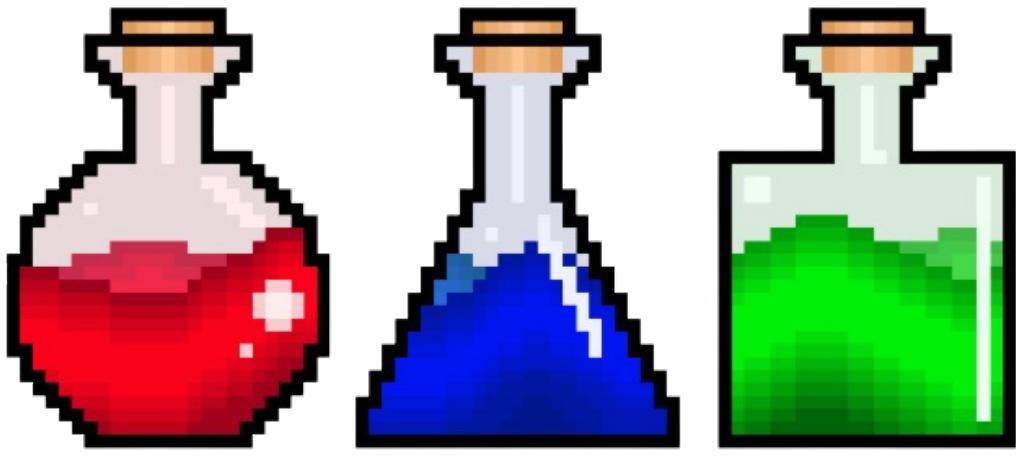


Figure 26: 3 Potions

Système d'inventaire Le script `Inventory` gère l'ensemble des objets possédés par le joueur à l'aide d'une liste d'objets `Item`. Ce script comprend notamment une méthode `ConsumeItem`, qui applique les effets associés à l'objet utilisé (comme les bonus de vitesse ou de saut).

L'inventaire est intégré dans l'interface utilisateur via un `Canvas` comprenant trois boutons :

- Un bouton pour passer à l'objet précédent.
- Un bouton pour passer à l'objet suivant.
- Un bouton pour consommer l'objet actuellement sélectionné.

Lorsque le bouton de consommation est utilisé, la fonction `ConsumeItem` est appelée. Elle affiche l'image et le nom de l'objet utilisé, applique les effets au joueur, supprime l'objet de l'inventaire et met à jour l'interface.

Navigation et sécurité L'inventaire utilise une variable `contentcurrentIndex`, initialisée à 0, pour suivre l'élément actuellement sélectionné. Les fonctions `PreviousItem` et `NextItem` permettent de naviguer dans l'inventaire. Elles s'assurent que l'indice reste dans les bornes de la liste, et implémentent un système de boucle (retour au début ou à la fin selon la direction) afin d'éviter les erreurs de dépassement.

Chaque fonction vérifie que l'inventaire contient au moins un objet avant d'exécuter une action, évitant ainsi les erreurs à l'exécution. Enfin, si l'inventaire est vide, l'image affichée est remplacée par une image transparente pour signaler l'absence d'objet disponible.



Figure 27: Navigation de l'inventaire

3.15 Barre de vie

La barre de vie du joueur est gérée via l'interface utilisateur à l'aide d'un **Canvas** attaché à son objet dans Unity. Elle permet de visualiser en temps réel l'état de santé du joueur.

Structure de la barre de vie Le **Canvas** contient deux images :

- Une image de fond, représentant le contour vide de la barre.
- Une image enfant, appelée **fill**, représentant le remplissage de la vie.

Un composant **Slider** est utilisé pour animer dynamiquement le niveau de remplissage de cette barre. Il est configuré pour aller de 0 à 100, correspondant aux points de vie du joueur.

Script HealthBarre Un script spécifique est attaché à ce composant pour gérer les mises à jour :

- **SetMaxHealth()** initialise le **Slider** avec une valeur maximale de 100.
- **SetHealth(int health)** met à jour le niveau de vie visible après avoir reçu des dégâts ou avoir été soigné.

Script PlayerHealth Le joueur possède également un script dédié à la gestion de la vie :

- **maxHealth** est initialisée à 100.
- **currentHealth** suit la vie actuelle du joueur.
- **TakeDamage(int damage)** applique les dégâts reçus. Cette fonction est appelée lors d'un contact avec un monstre, un piège, etc.

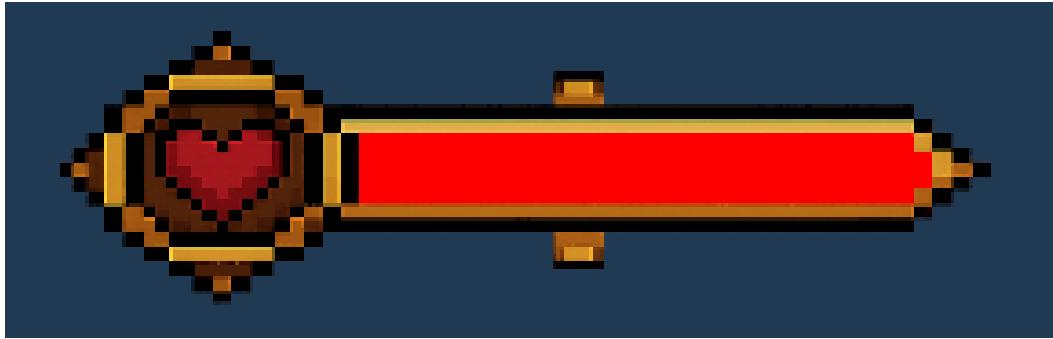


Figure 28: Barre de vie

Gestion de la mort Lorsque la vie du joueur tombe à zéro ou moins, une routine de réapparition est déclenchée :

- Le joueur est téléporté à son point de départ.
- 10 minutes sont retranchées au chronomètre principal.

3.16 PNJ Trader

Nous avons choisi un sprite représentant un scientifique comme personnage non-joueur (PNJ), auquel nous avons associé un **Animator** afin de lui donner des animations de mouvement et renforcer son aspect vivant.



Figure 29: Sprite PNJ

Interface de dialogue Pour gérer les dialogues avec le PNJ, nous avons conçu un panneau nommé **DialogueUI** contenant :

- Un champ de texte pour le nom du PNJ.
- Un champ de texte pour les répliques du dialogue.
- Un bouton transparent couvrant l'ensemble de la bulle de dialogue, servant à passer à la réponse suivante.

Structure des dialogues Un script nommé **Dialogue** permet de stocker les informations liées à chaque conversation. Il contient :

- Le nom du personnage.
- Un tableau de chaînes de caractères nommé **sentences** contenant les réponses successives.

Déclenchement des dialogues Chaque PNJ possède un composant `DialogueTrigger`, qui contient le nom du PNJ ainsi que les zones de texte à remplir. Pour détecter la proximité du joueur, nous utilisons un `BoxCollider2D` configuré en mode `IsTrigger`. Deux méthodes sont utilisées :

- `OnTriggerEnter()` : permet de détecter l'entrée du joueur dans la zone d'interaction.
- `OnTriggerExit()` : permet de détecter sa sortie.

Dans la méthode `Update()`, nous vérifions si le joueur est à portée et appuie sur la touche E, afin de déclencher l'affichage du dialogue.

Affichage des dialogues Le script `DialogueManager` fait le lien entre les données du dialogue et l'interface utilisateur `DialogueUI`. Il utilise une `Queue` pour gérer les phrases et les faire défiler via une méthode `StartDialogue()`, suivie de la méthode `DisplayNextSentence()`, reliée au bouton de progression.

Pour un effet esthétique, les phrases sont affichées caractère par caractère grâce à une `Coroutine`, créant un effet de machine à écrire.



Figure 30: Dialogue PNJ

Animation de l'interface Le panneau `DialogueUI` est initialement positionné en dehors de l'écran du joueur. Lorsqu'un dialogue est déclenché, une animation le fait apparaître progressivement à l'écran. Cette animation est activée par un booléen déclenché lors de l'interaction.

Échange avec le PNJ Un deuxième panneau, dédié au commerce avec le PNJ, est prévu pour apparaître à la fin du dialogue. Il contient :

- L'image d'une potion de saut.
- Le prix en temps affiché sous forme de texte : 5 minutes

- Deux boutons : Oui et Non.

Si le joueur clique sur Oui, 10 minutes sont soustraites de son chronomètre, et un objet `JumpPotion` est ajouté à son inventaire. Le panneau se ferme automatiquement après l'interaction, quel que soit le choix du joueur.



Figure 31: Trade PNJ

3.17 Épreuve du morpion

Interface du jeu Nous avons créé un `Canvas` nommé `CanvasMorpion` contenant un `Panel` disposant d'un `Grid Layout Group`, afin de positionner automatiquement les éléments en grille. Nous avons utilisé des boutons UI pour créer une grille 3x3 représentant le plateau de morpion. Chaque bouton est nommé de 0 à 8, ce qui permet de les identifier facilement via un tableau d'indices. Leur propriété `interactable` est utilisée pour empêcher l'utilisateur de cliquer plusieurs fois sur le même bouton.

Fin de partie Un second panneau nommé `GameOver` est superposé au plateau de jeu pour signaler la fin d'une partie. Ce panel contient :

- Un texte UI indiquant le résultat de la partie (victoire, défaite ou match nul).
- Un bouton `Restart` permettant de réinitialiser le jeu pour recommencer une nouvelle partie.

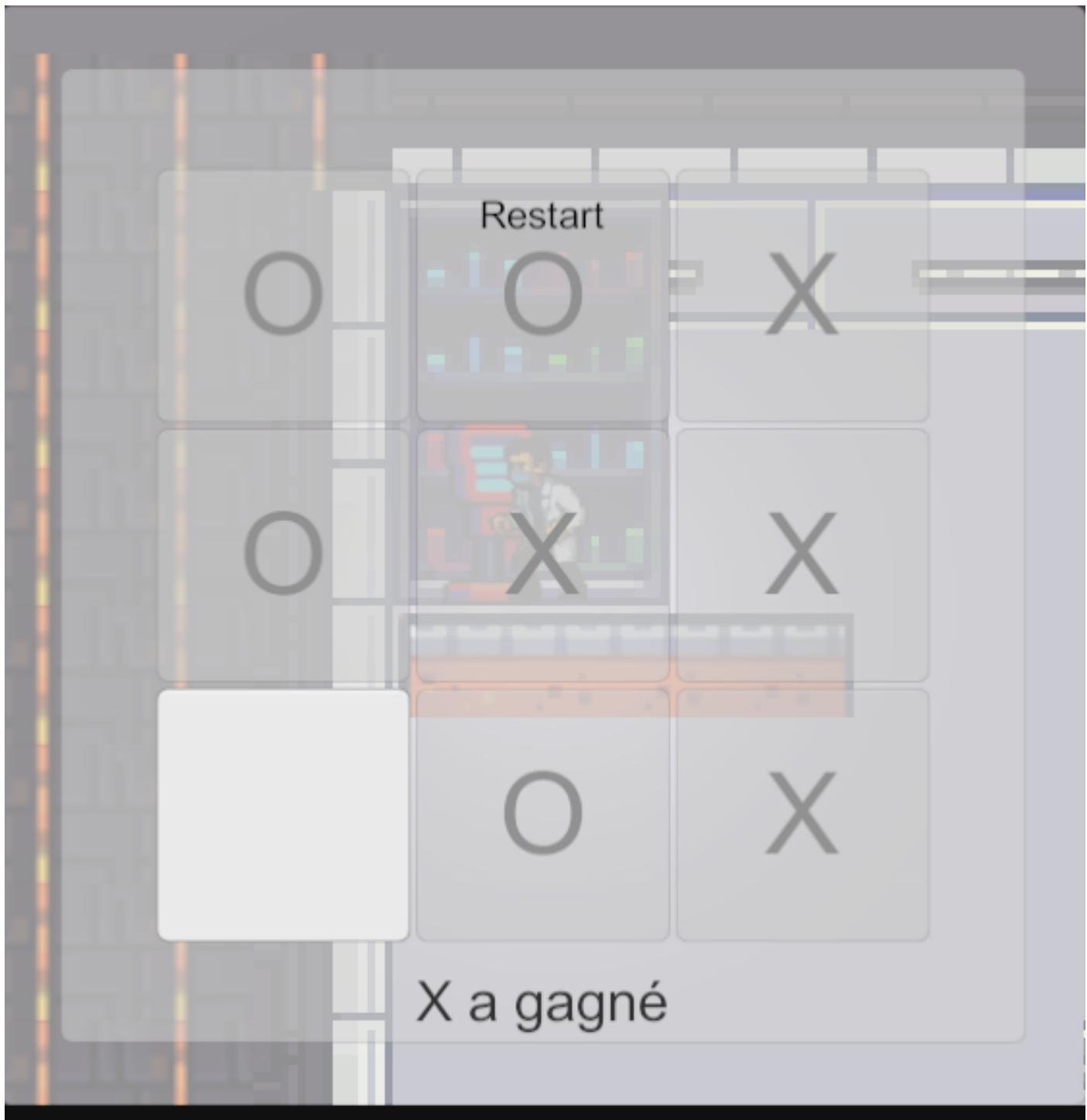


Figure 32: Fin de partie

Gestion du jeu Un script `GameManager` est assigné au `CanvasMorpion`. Il déclare un tableau de chaînes de caractères nommé `matrix` pour suivre l'état des cases.

Une fonction permet de déterminer aléatoirement si le joueur ou l'ordinateur commence la partie.

Interaction des boutons Chaque bouton est associé à un script `ButtonScriptMorpion` contenant la fonction `Select`. Celle-ci inscrit un 'X' sur le bouton sélectionné, le rend inactif (`interactable = false`), et met à jour la case correspondante dans le tableau `matrix` du `MorpionManager`, à l'aide d'un `int.Parse` du nom du bouton.

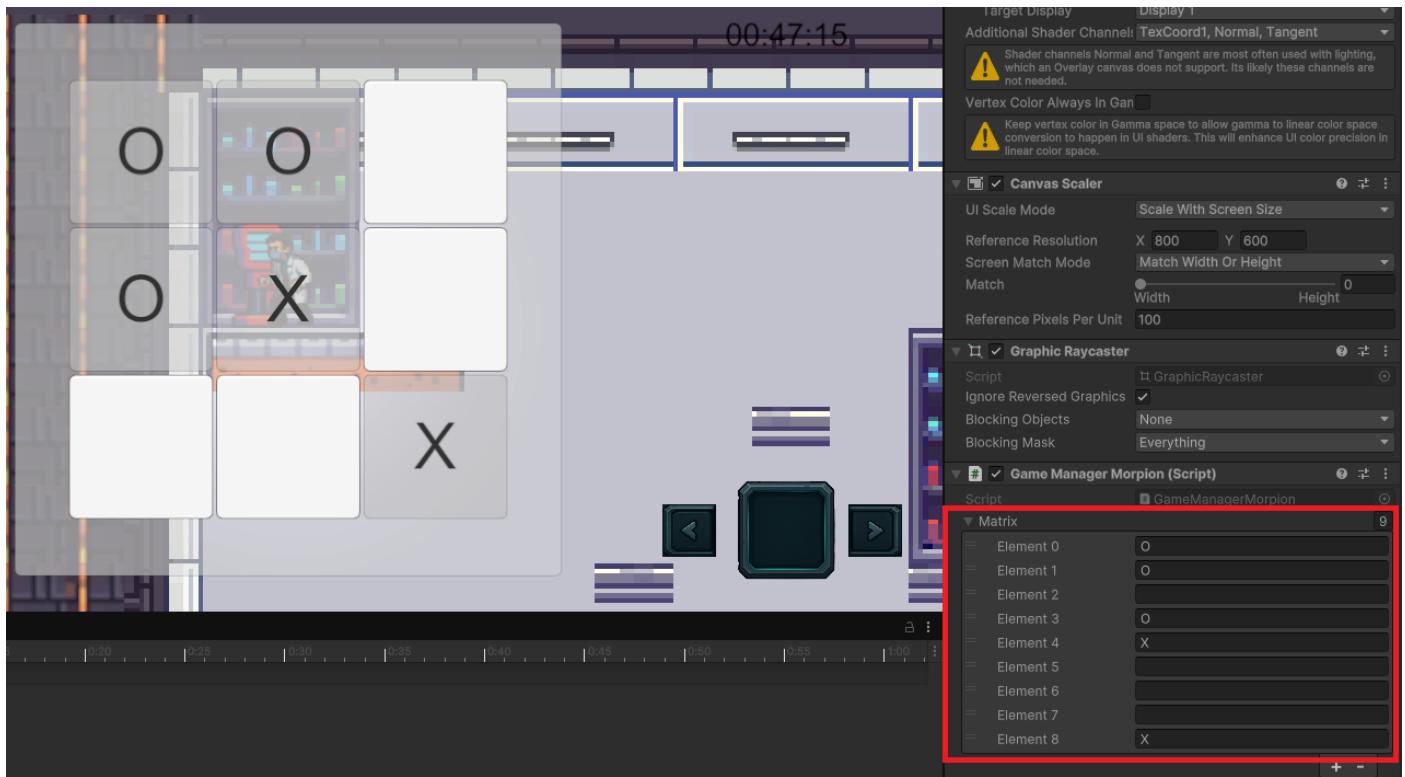


Figure 33: Tableau Matrix

Tour de l'ordinateur Le script `MorpionManager` contient la fonction `ComputerPlay`, qui repose sur une autre fonction `MajChoice`. Cette dernière génère une liste d'indices correspondant aux cases encore vides du plateau, parcourant le tableau `matrix`. Si cette liste est vide, cela signifie qu'aucun mouvement n'est possible, donc la partie est déclarée nulle.

Sinon, une case est choisie aléatoirement parmi celles disponibles. Le bouton correspondant est récupéré, son texte est défini à 'O' et il est rendu inactif.

Conditions de victoire Pour vérifier s'il y a un gagnant, le script analyse toutes les combinaisons gagnantes :

- 3 combinaisons horizontales
- 3 combinaisons verticales
- 2 combinaisons diagonales

Ces vérifications sont faites à chaque tour, juste après chaque coup.

Réinitialisation et récompense Le script `GameOverMorpion`, attaché au panneau `GameOver`, vérifie si le joueur a gagné ou non. Dans tous les cas, le bouton `Restart` remet le plateau à zéro.

Si le joueur gagne, il reçoit dans son inventaire un objet spécial : un seau d'eau, indispensable pour progresser dans le jeu.

Dans le cas d'une défaite, le redémarrage de la partie entraîne la disparition de la plateforme sous les pieds du joueur, l'obligeant à recommencer le parcours de saut menant à l'épreuve du morpion.

3.18 Ennemis et pièges

Comportement du monstre Les monstres sont équipés d'un `BoxCollider2D` en mode `isTrigger`, leur permettant de détecter les collisions sans provoquer de blocage physique. Un script leur permet de se déplacer et d'infliger des dégâts au joueur.

Le déplacement du monstre se fait à l'aide de waypoints. Un tableau de `Transform (waypoints)` définit les différentes positions que le monstre doit atteindre successivement. Un entier `destPoint` représente l'indice du prochain waypoint cible.

À chaque mise à jour, on calcule la direction à suivre en soustrayant la position de l'ennemi à celle de la cible. Si la distance entre le monstre et le waypoint est inférieure à un seuil de sécurité (`0.3f`), on passe au waypoint suivant dans la liste.

Un système de *flip* est également mis en place pour que le sprite du monstre regarde toujours dans la direction de son déplacement.

Élimination de l'ennemi Chaque monstre possède une zone de faiblesse appelée `Weak Spot`, représentée par un `BoxCollider2D` positionné au niveau de sa tête, également en mode `isTrigger`. Si le joueur entre en collision avec cette zone (par exemple en lui sautant dessus), le monstre est détruit.



Figure 34: Monstre waypoints

Pièges du laboratoire Le laboratoire contient divers pièges, comme :

- des flammes dans les couloirs,
- une hache rotative sur le parcours menant au jeu du morpion.

Ces pièges sont animés à l'aide d'un composant `Animator`. À un moment précis de l'animation (par exemple, lorsque la flamme est allumée), un `Animation Event` est déclenché. Cet événement appelle une fonction qui vérifie si le joueur entre en collision avec le piège durant cette phase critique, et lui inflige alors des dégâts.



Figure 35: Flamme Trap

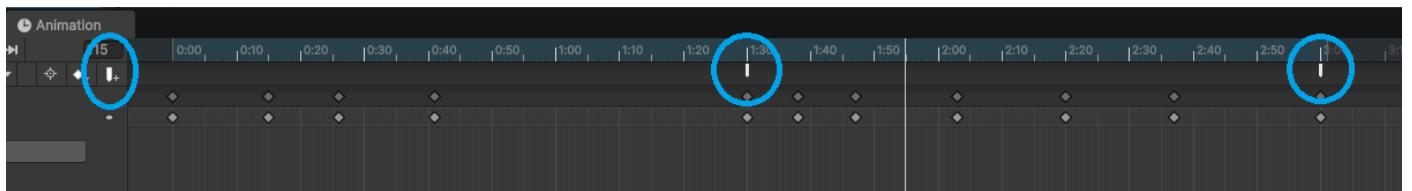


Figure 36: Add Event

Ce système permet de synchroniser précisément les dégâts avec l'état visuel du piège, renforçant ainsi l'immersion et la difficulté du jeu.

3.19 Implementation du dash : entre gameplay, animation et synchronisation

Au commencement, l'idée était simple : offrir au joueur une manière de s'échapper rapidement d'une situation dangereuse. Une impulsion rapide, un mouvement de fuite, une respiration dans la tension. Le dash, en tant que mécanique de gameplay, s'est donc imposé naturellement. Nous pensions qu'il suffirait de modifier brièvement la vitesse du joueur ou de lui appliquer un vecteur directionnel boosté. Quelques lignes de code dans notre `PlayerMovement`, et l'affaire serait réglée. Mais ce que nous avons vite compris, c'est que le ressenti joueur ne se code pas si facilement.

3.20 La limite des solutions brutales

Notre première version consistait à simplement appliquer un vecteur de déplacement augmenté durant quelques frames. Sur le papier, cela fonctionnait : le joueur était propulsé vers l'avant, et reprenait ensuite sa vitesse normale. Mais très vite, les tests ont révélé un problème plus profond : le mouvement était saccadé, sans fluidité. Le dash paraissait forcé, rigide, détaché du reste des animations. Le joueur avait l'impression de "glisser" brutalement, sans que son personnage ne réagisse visuellement à ce changement de dynamique. Ce manque de cohérence entre l'intention du joueur, le comportement du personnage et l'affichage à l'écran nuisait gravement à l'immersion.

3.21 Une nouvelle approche : la coroutine et le tempo

Nous avons donc décidé de tout reprendre. Cette fois, l'idée était de créer une coroutine, à l'aide d'un `IEnumerator`, qui permettrait de répartir le dash dans le temps, de façon lisse. Le mouvement n'était plus un simple saut de position, mais une progression fluide, calculée à chaque frame pendant une durée définie. Ce changement a tout modifié. Nous pouvions maintenant contrôler l'accélération, ralentir l'arrivée, et intégrer des effets visuels ou sonores de manière synchronisée. Cela a aussi ouvert la porte à une animation spécifique pour le dash, ce qui nous paraissait indispensable pour enrichir l'expérience utilisateur.



Figure 37: Enter Caption

```

177     IEnumerator Dash()
178     {
179         animator.SetBool("IsDashing", true);
180         animator.SetBool("IsWalking", false);
181
182         float originalGravity = rb.gravityScale;
183         rb.gravityScale = 0f;
184
185         float elapsed = 0f;
186         float dashDirection = isFacingRight ? 1f : -1f;
187
188         Vector2 startPosition = rb.position;
189         RaycastHit2D hit = Physics2D.Raycast(startPosition, Vector2.right * dashDirection, 3f, groundLayer);
190
191         float actualDashDistance = hit.collider != null ? hit.distance - 0.1f : 3f;
192         Vector2 targetPosition = startPosition + new Vector2(dashDirection * actualDashDistance, 0);
193
194         while (elapsed < dashDuration)
195         {
196             elapsed += Time.deltaTime;
197             float t = elapsed / dashDuration;
198             Vector2 newPosition = Vector2.Lerp(startPosition, targetPosition, t);
199             rb.MovePosition(newPosition);
200             yield return null;
201         }
202
203         rb.gravityScale = originalGravity;
204         animator.SetBool("IsDashing", false);
205         lastDashTime = Time.time;
206     }

```

① Do you want to install the recommended 'Unity' extension?

Figure 38: Ienumerator Dash

3.22 Les défis de l'animation dynamique

C'est ici que nous avons rencontré un autre mur : celui de la gestion des animations dans Unity. Jusqu'ici, nous pensions avoir une bonne maîtrise du système Animator. Mais l'ajout d'une animation de dash a mis en lumière plusieurs difficultés :

- Gérer l'enchainement entre l'animation de course et celle de dash
- Comprendre et maîtriser les Exit Time pour éviter les transitions non naturelles
- Intégrer des conditions dynamiques pour la reprise de contrôle
- Synchroniser l'animation avec le mouvement réel (et non pas décalé)

Cette phase a été extrêmement instructive. Elle nous a obligés à revoir toute notre logique d'animation et à adopter une approche plus rigoureuse, avec des tests poussés et un travail de fine réglage sur l'Animator Controller.

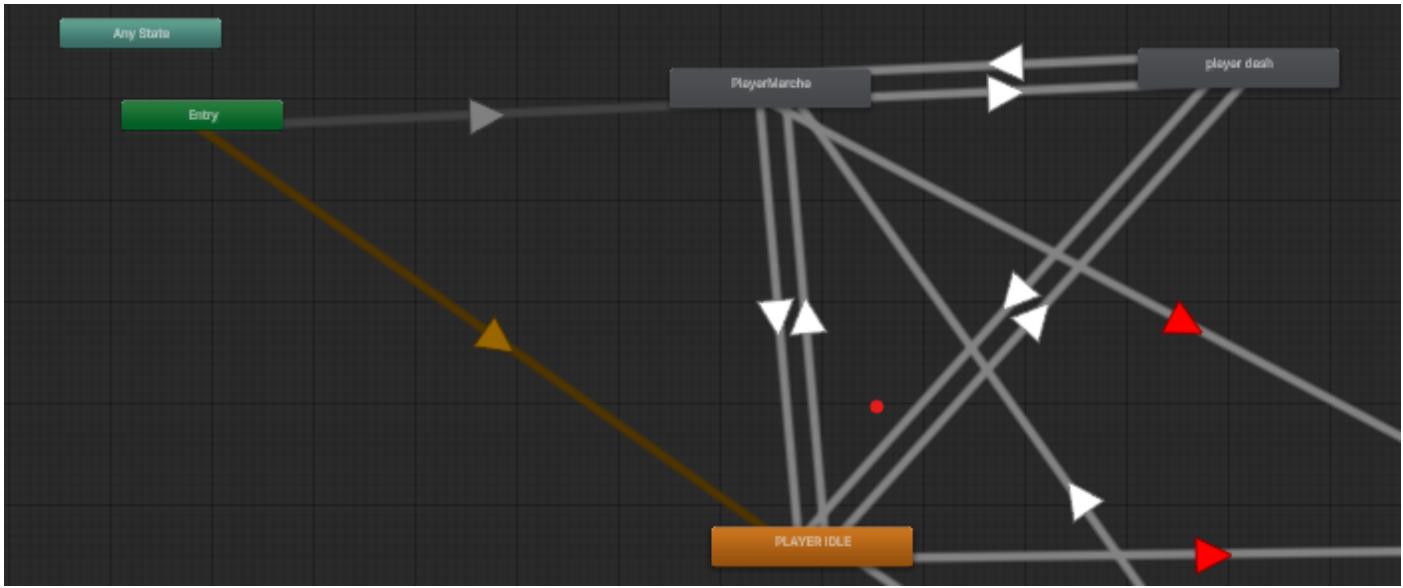


Figure 39: animator du player

3.23 Et le multijoueur dans tout ça ?

Comme si le travail en solo ne suffisait pas, il fallait maintenant le rendre fonctionnel en multijoueur. Grace à Mirror, notre solution de réseau, nous avons pu utiliser des [Command] et [ClientRpc] pour synchroniser les actions. Le dash ne devait pas seulement se déclencher localement, mais être visible sur tous les clients. La difficulté ici était double :

- Assurer que le serveur reçoive la demande de dash sans délai
- Que les autres clients voient le dash avec les mêmes timings et animations

Nous avons donc découpé le processus :

1. Le joueur local appuie sur Left Shift
2. Une commande serveur est envoyée
3. Le serveur valide le dash et appelle un RPC sur tous les clients
4. Chaque client lance l'animation et la coroutine de dash

Cela peut paraître simple, mais chaque micro-désynchronisation (animation trop tôt, mouvement trop tard, etc.) était visible et perturbante. Il a fallu régler finement les timings et les conditions pour obtenir un rendu cohérent et stable.

3.24 Séparer les responsabilités : un modèle plus propre

Afin d'éviter les conflits entre logique de déplacement, gestion d'animations et traitement réseau, nous avons pris la décision de bien cloisonner les différents scripts. `PlayerMovement` gère la logique de déplacement et d'entrée utilisateur. Un script d'animation dédié, `PlayerAnimatorController`, s'occupe de traduire les états en animations. Enfin, `NetworkDashHandler` se charge exclusivement de l'envoi et de la réception des commandes réseau. Cette organisation a permis d'isoler les bugs, de mieux tester chaque module, et surtout de faire évoluer le code plus facilement.

3.25 une petite fonctionnalité, un grand impact

Ce dash, que nous pensions implémenter en une journée, a finalement pris presque une semaines. Mais il nous a permis d'apprendre énormément sur Unity, sur Mirror, sur les animations, et sur l'interaction entre gameplay local et expérience r'eseau. Il est aujourd'hui une mécanique fluide, cohérente, visuellement satisfaisante, et surtout stable en multijoueur. Il représente bien le type de défi que nous avons affronté dans ce projet : transformer une idée simple en fonctionnalité riche, fiable et immersive. Lorsque nous avons commencé ce projet, nous savions dès le départ que l'implémentation d'un mode multijoueur synchronisé, avec des interactions complexes telles qu'un pathfinding dynamique, des plateformes mouvantes ou encore une pince interactive, allait constituer un véritable défi. Pourtant, motivés par l'envie de proposer une expérience coopérative originale, nous nous sommes lancés, conscients que le chemin serait semé d'embûches.

3.26 La pince interactive : conception et réalisation

La pince constitue l'un des éléments emblématiques de notre gameplay. Elle a pour but d'attraper un joueur et de le déplacer selon une trajectoire précise, jusqu'à une destination dépendant de son statut dans le jeu.

Pour gérer son comportement, nous avons défini plusieurs points de passage :

- **A** : Point de départ de la pince (en haut).
- **A2** : Point correspondant à la position basse de la pince.
- **B / B2** : Zone des flammes qui infligent 50 points de dégâts en continu.
- **C / C2** : suite de l'aventure Si le joueur a réussi l'épreuve précédente

Lorsque le joueur a réussi l'épreuve précédente (identifiée par l'activation du "boulet 1"), la pince l'emmène jusqu'à la zone de sortie et le dépose sans danger. Si ce n'est pas le cas, il est délibérément dirigé vers les flammes ou dans le vide. Cette variation ajoute une tension dramatique au gameplay.

Nous avons également mis en place un bouton permettant, une fois l'épreuve réussie, de supprimer les flammes et d'activer une plateforme permanente, afin que les autres joueurs n'aient plus besoin d'utiliser la pince.

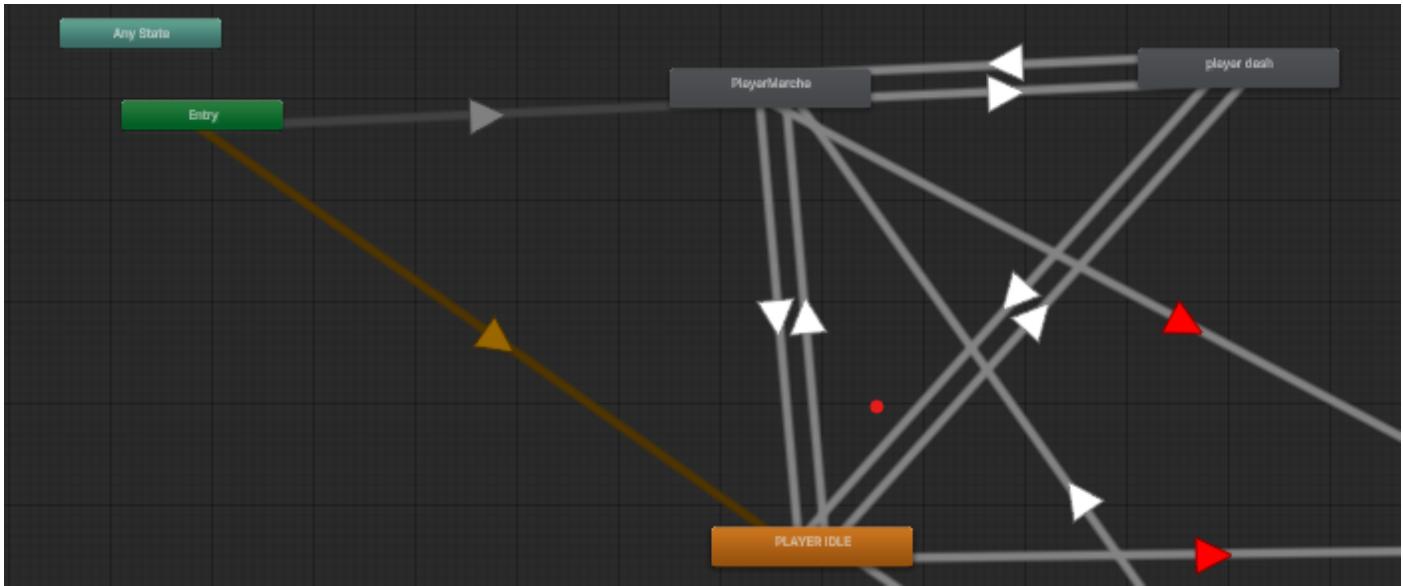


Figure 40: pince

3.27 Plateforme mouvante synchronisée en multijoueur

Autre élément dynamique : la plateforme mouvante. Celle-ci suit un trajet défini entre deux points, de manière continue.

Le défi ici résidait dans la synchronisation parfaite entre le serveur et tous les clients. En effet, une mauvaise gestion de l'authority peut entraîner des différences de positions, des téléportations ou des mouvements erratiques chez les joueurs.

Nous avons opté pour une gestion serveur authoritative : c'est le serveur qui définit la position exacte de la plateforme à chaque frame, et qui l'envoie aux clients via les composants `NetworkTransform`. Cette approche a permis une meilleure stabilité, au prix d'une légère latence perceptible.

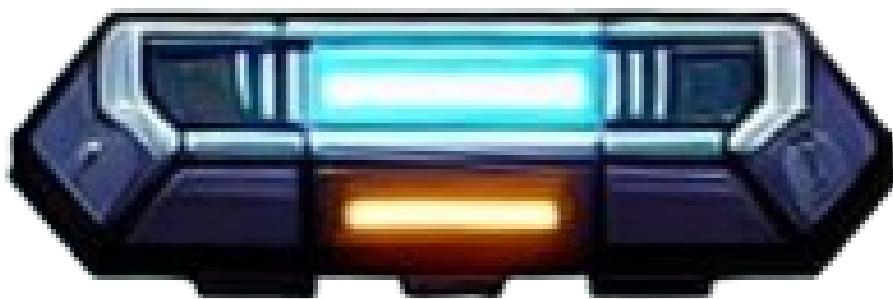


Figure 41: plateforme mouvante

3.28 Pathfinding et IA du monstre

L'un des points les plus techniques du projet a été la réalisation du pathfinding du monstre. Celui-ci doit être capable de se déplacer sur un graphe vers une cible (typiquement un joueur), en évitant les obstacles et en choisissant le chemin optimal.

Nous avons créé un graphe personnalisé en Unity, avec des noeuds placés manuellement. Chaque noeud possède une liste de voisins, et l'algorithme de parcours est inspiré de Dijkstra, bien que simplifié.

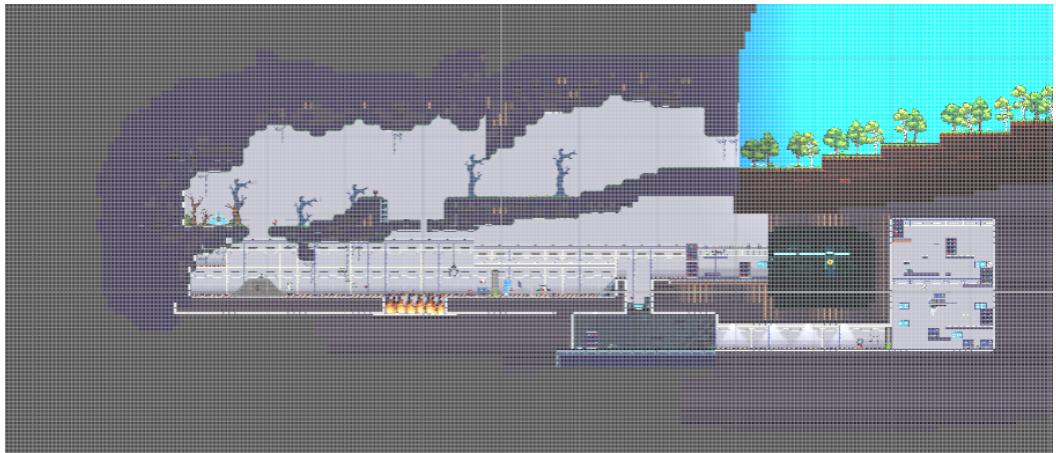


Figure 42: graphe pour pathfinding

Fonctionnement :

- Le monstre identifie sa position actuelle et celle de la cible.
- Il parcourt le graphe en cherchant le chemin le plus court.
- Il suit les noeuds un à un jusqu'à la cible.

Ce système fonctionne bien en local. Cependant, en multijoueur, la synchronisation du comportement du monstre a posé de nombreuses difficultés, notamment sur le choix de la cible (quelle caméra activer ? pour qui ?), et la gestion des autorités entre client et serveur.



Figure 43: monstre dans sa cage

3.29 Scène de crédits

La scène de crédits apparaît dans la dernière zone du jeu, une zone naturelle qui marque la fin de l'aventure. Lorsqu'un joueur entre dans cette zone, le temps est mis en pause afin de permettre un affichage fluide et immersif des crédits.

Affichage du texte déroulant Le texte des crédits est intégré dans un **Canvas**, mais initialement placé en dehors du champ de vision du joueur. Pour obtenir un rendu soigné, nous utilisons le système *Rich Text* de Unity, qui permet de formater le texte à l'aide de balises (****, **<i>**, **<color>**, etc.) afin de modifier la taille, la couleur ou le style du texte.

Nous avons également intégré des images, comme le logo de *LockedInGames* et celui du jeu *ESCAPE*, qui défilent verticalement en même temps que le texte pour renforcer l'aspect visuel de la scène.

Animation de défilement L’effet de défilement est obtenu grâce à une animation. Nous avons enregistré le déplacement du texte depuis le bas de l’écran jusqu’à sa sortie en haut de l’écran. Un *interpolateur linéaire* a été utilisé afin d’assurer une vitesse constante de défilement.

Enfin, un **Animation Event** est déclenché à la fin de l’animation. Celui-ci redirige automatiquement le joueur vers le menu principal du jeu, concluant ainsi l’expérience de manière fluide et cohérente.

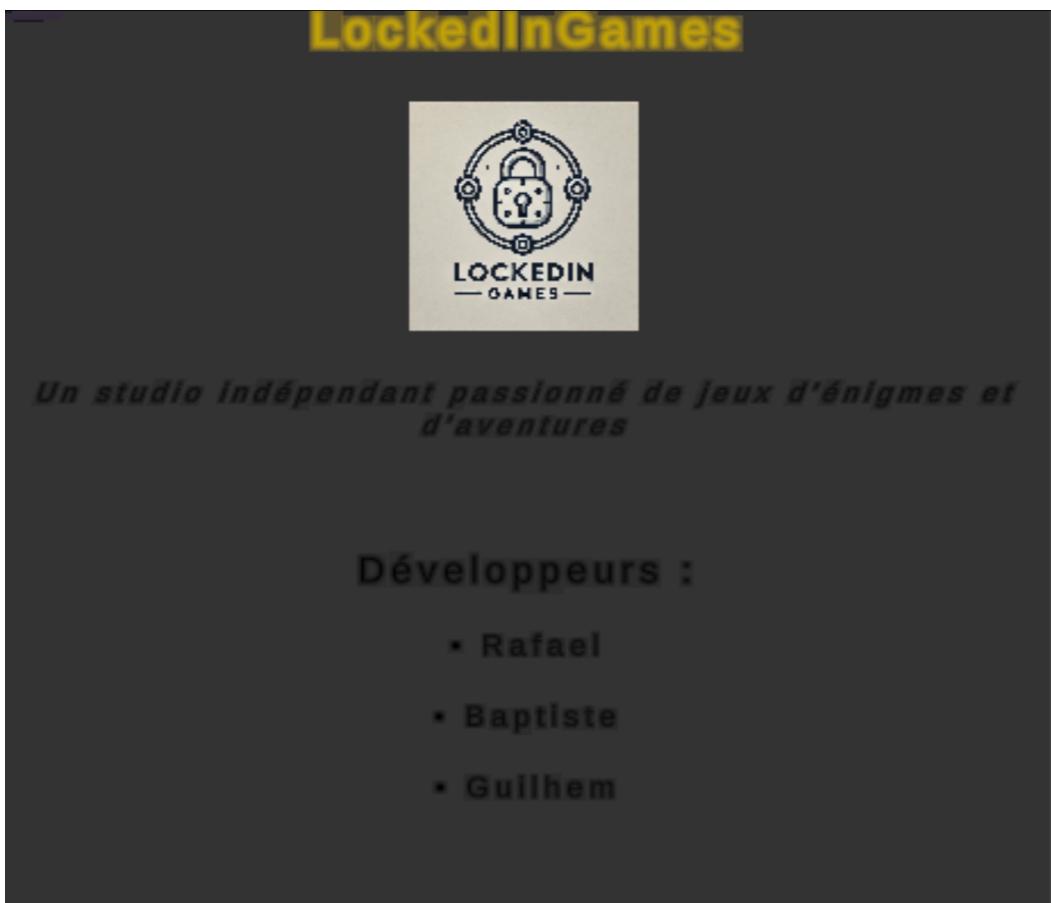


Figure 44: Scene de Crédit

4 Conclusion

4.1 Conclusion

Ce rapport final illustre le chemin parcouru ainsi que les avancées majeures réalisées lors du développement de notre jeu. Nous avons concrétisé de nombreuses étapes techniques, notamment la finalisation du système de déplacement du joueur et l’intégration progressive des énigmes qui structurent le gameplay. Ces mécanismes ont été pensés pour immerger pleinement le joueur dans un univers cohérent et stimulant.

Le travail sur les interactions avec l’environnement, en particulier avec des éléments clés comme la pince mécanique ou la croissance de la plante, a permis de renforcer la richesse et la variété des défis proposés. Parallèlement, l’intégration du mode multijoueur et le développement des aspects sonores et visuels contribuent à rendre l’expérience plus immersive et dynamique.

Le site web de LockedInGames, déjà opérationnel, offre une vitrine professionnelle pour présenter notre équipe, nos objectifs, ainsi qu’une première plongée dans l’univers du jeu. La gestion du code via un dépôt GitHub centralisé a assuré une collaboration fluide malgré la complexité du projet.

Malgré quelques retards liés à la complexité des énigmes et à l'optimisation du gameplay, notre équipe à su resté pleinement engagée et confiante dans la qualité du produit final. Le travail sur l'intelligence artificielle du monstre, le pathfinding et la stabilité générale du jeu à été poursuivie activement afin de garantir une expérience à la fois immersive, réfléchie et accessible.

Nous sommes persuadés que ce projet offre une aventure unique, mêlant réflexion, action et ambiance captivante propre à l'esprit d'un véritable escape game.

4.2 Les joies

La réalisation de ce projet a été avant tout une aventure créative et particulièrement enrichissante. Au fil des réunions, une véritable synergie s'est installée entre les membres de l'équipe. Chacun a apporté ses idées, ses forces, et a contribué à développer un jeu cohérent et engageant. Nous avons découvert à quel point nos compétences se complétaient : certains étaient plus à l'aise avec la partie multijoueur, d'autres avec les mécaniques de gameplay, ou encore la programmation des interfaces.

L'un des plus grands plaisirs a été de voir notre jeu prendre forme, et de constater que les différentes implémentations fonctionnaient et plisaient au groupe. La première fois que le personnage a pu interagir avec un PNJ, déclencher un dialogue ou acheter un objet, c'était un vrai moment de satisfaction. Il y avait une réelle joie à résoudre ensemble des problèmes, à tester nos idées, à itérer et à progresser, étape par étape.

Le plaisir n'était pas uniquement technique. Créer une ambiance, une narration, un univers immersif avec des pièges et des monstres nous a aussi permis de développer notre sens artistique et notre créativité. Ce jeu, c'était un peu comme une œuvre commune, un espace où chacun a pu laisser sa marque.

4.3 Les peines

Comme tout projet, celui-ci n'a pas été exempt de difficultés. Le travail en équipe peut parfois devenir un défi : différences de rythme, de méthodes de travail, voire de vision sur certaines fonctionnalités. Il a fallu apprendre à faire des compromis, à mieux communiquer, et à abandonner des idées bonnes mais irréalisables.

D'un point de vue technique, certains obstacles nous ont donné du fil à retordre. Notamment certains problèmes liés au multijoueur, mais surtout la gestion de Git, qui devait nous permettre de mutualiser notre travail. À cause de conflits fréquents, nous avons dû travailler de manière très coordonnée sur une même branche, en nous informant systématiquement des moments d'utilisation et des modifications apportées.

Par ailleurs, respecter les délais n'a pas toujours été évident. Entre les autres cours, les examens et les imprévus personnels, il a parfois été difficile de se retrouver régulièrement et de maintenir une cadence de travail soutenue. La gestion du temps s'est révélée être un enjeu à part entière.

4.4 Les leçons tirées

Au final, ce projet nous a énormément appris, bien au-delà des compétences techniques. Il nous a permis de comprendre l'importance de la planification, de la clarté dans la répartition des rôles, et surtout de la communication au sein d'un groupe. Travailler sur un jeu vidéo, c'est à la fois du code,

de la logique, de la narration, du graphisme et du design d’interaction. Cette diversité nous a poussés à sortir de notre zone de confort et à nous entraider.

Nous avons aussi compris que les échecs ne sont jamais inutiles : chaque erreur, chaque impasse technique a été une source d’apprentissage, nous obligeant à nous adapter. Ce projet nous a donné confiance dans notre capacité à mener un projet de bout en bout, malgré les aléas.

En résumé, cette réalisation a été un défi stimulant, parfois difficile, mais surtout profondément formateur et humainement enrichissant.

5 Répartition

Figure 45: Cahier des charges techniques

Nom du groupe :	The New Game			
Nom du projet :	Escape			
Noms des membres :				
Nom :	Prénom :	Login :	Classe :	
Batrancourt	Baptiste	baptiste.batrancourt	E1	
Petit	Guilhem	guilhem.petit	E1	
Magalhaes-boulan	Rafael	rafael.magalhaes-boulan	E1	
Said	Georgina	georgina.said	E1	
Dincov	Ioan	ioan.dincov	E1	
Type de jeu :				
Action/Aventure	Battle Royale	Beat them all	Combat	Simulation
FPS	MMORPG	MOBA	Party Games	Survival Horror
Plateforme	Puzzles	Reflexion	Rogue Like	TPS
RPG	RTS	Sandbox	Shoot them up	Course
Autre :				
Caractéristiques générales du jeu :				
IA :	Errer	Attaquer	S'échapper	"Path Finder"
Multijoueurs :	Coopé	Battle (2-4)	Massif	
Réseau :	P2P	Lan	Online	
Caractéristiques graphiques :				
Dimension :	2D	3D	Autres :	
Particularités :	Stéréoscopie	AR	VR	
graphiques :	Perso	Custom	Existant	
Précisions :				
Caractéristiques sonores :				
Musique :	Perso	Custom	Existant	
FX :	Perso	Custom	Existant	
Précisions :				
Autres caractéristiques :				
Site Web :	Perso	Custom	Préfabriqué	

Figure 46: **Diagramme de GANTT**

