



## PROJET S2 EPITA

### RAPPORT DE SOUTENANCE 2

**Noé Cornu ; Théo Gille ; Clément Grégoire ;  
Leandro Tolaini**

*Avril 2022*

# Table des matières

<b>1</b>	<b>Reprise du cahier des charges</b>	<b>1</b>
<b>2</b>	<b>Noé Cornu</b>	<b>3</b>
2.1	Design et Interface . . . . .	3
2.2	IA . . . . .	3
2.3	Son Effet . . . . .	5
2.4	Dialogues . . . . .	6
2.5	Gestion du GIT . . . . .	7
2.6	Ressentis . . . . .	7
<b>3</b>	<b>Leandro Tolaini</b>	<b>8</b>
3.1	Design des personnages . . . . .	8
3.2	Design des niveaux . . . . .	11
3.3	Aspect technique . . . . .	13
3.4	Ressenti . . . . .	14
<b>4</b>	<b>Clément Grégoire</b>	<b>15</b>
4.1	Site Web . . . . .	15
4.2	Les piques . . . . .	17
4.3	Double saut . . . . .	17
4.4	L'attaque du démon . . . . .	18
4.5	Ressentis . . . . .	19
<b>5</b>	<b>Théo Gille</b>	<b>20</b>
5.1	Multijoueurs . . . . .	20
5.2	UI Menu . . . . .	21
5.3	Interaction & Echelles . . . . .	24
5.4	Ressentis . . . . .	27

# Chapitre 1

## Reprise du cahier des charges

Depuis la première soutenance, nous avons travaillé dur pour atteindre nos objectifs. Tout d’abord, nous avons continué à travailler sur le multijoueur, qui est l’un des points clés de notre projet. Nous avons pu approfondir notre compréhension de cette fonctionnalité et avons fait des progrès significatifs dans son développement. Nous avons également avancé sur le design de la map, en créant des environnements plus complexes et détaillés pour les joueurs à explorer. Nous avons également commencé à mettre en place des éléments de gameplay en lien avec le multijoueur, en permettant aux joueurs de collaborer et de combattre des ennemis.

En ce qui concerne les éléments prévus pour la première soutenance, nous avons réalisé des avancées significatives dans plusieurs domaines. Nous avons créé un système de dialogues qui permet d’introduire l’histoire, ce qui renforce l’immersion et ajoute de la profondeur au scénario. Nous avons également commencé à travailler sur le design d’une map, en créant un environnement visuellement intéressant et en introduisant des éléments de gameplay pour que les joueurs puissent les prendre en main et comprendre les mécaniques. Nous avons mis en place un site web qui servira de plateforme pour le téléchargement du jeu et des documents annexes. Nous avons également créé un menu de base pour faciliter la navigation à travers le multijoueur et avons commencé à implémenter les mouvements basiques des joueurs, ce qui nous permettra de créer des interactions plus complexes à l’avenir.

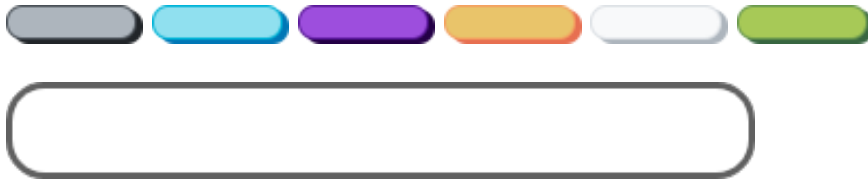
2eme soutenance	Pas commencé	En cours	Fini
Design et interface		X	
UI & Menus		X	
IA		X	
Multijoueurs		X	
Multijoueurs		X	
Map design		X	
Site Web			X
Son effet		X	
Mecanique de niveau		X	
Dialogues			X

## Chapitre 2

# Noé Cornu

### 2.1 Design et Interface

Depuis la première soutenance, nous avons continué à travailler sur le design du jeu en utilisant différents moyens pour atteindre nos objectifs. J'ai personnellement travaillé avec Photoshop pour créer des boutons et une chatbox pour intégrer les dialogues et les indications de gameplay. Cela a permis d'améliorer l'interface utilisateur et d'offrir une expérience plus fluide aux joueurs. Pour le reste du design, nous avons opté pour l'utilisation d'assets trouvés sur Internet, car aucun des membres du groupe ne sait dessiner. Cette solution nous a permis de gagner du temps et de nous concentrer sur d'autres aspects du développement. Nous avons choisi des assets de qualité pour que l'environnement visuel soit cohérent et attractif pour les joueurs. Nous sommes très satisfaits de la façon dont le design s'intègre au gameplay et renforce l'expérience de jeu globale.



### 2.2 IA

L'implémentation d'une IA pour l'enemy l'un des points principaux de notre projet. Après avoir étudié différentes méthodes pour rendre l'enemy intelligent, j'ai finalement opté pour la méthode MoveTowards qui permet de déplacer l'enemy vers un point cible en suivant une ligne droite. Cependant, avant d'utiliser cette méthode, il a fallu prendre en compte certaines considérations pour éviter des erreurs d'exécution. Tout d'abord, il est important de vérifier si au moins un des deux joueurs est présent sur la map et, si oui, lequel est le plus proche de l'enemy. Une fois cette information recueillie, le script va assigner la direction de déplacement de l'enemy vers le joueur le plus proche et rafraîchir cette position à chaque frame pour suivre le joueur en temps réel.

```
transform.position = Vector2.MoveTowards(transform.position, player.position);
```

Afin d'éviter que l'enemy ne tombe dans le vide lorsqu'il suit le joueur, un GroundCheck a été ajouté au Prefab de l'enemy pour vérifier s'il est toujours au sol. Nous avons également ajouté

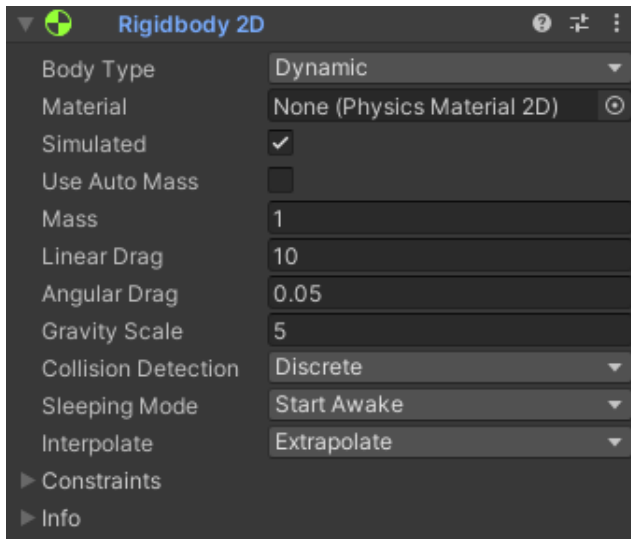
une range de détection pour que lorsque l'enemy n'est plus en vue de la caméra, il ne bouge plus afin de ne pas surcharger les performances du jeu. De plus, cette range permet également d'éviter qu'un enemy à l'autre bout de la map commence à suivre le joueur dès le lancement du niveau.



Enfin, nous avons ajouté la capacité de l'enemy à attaquer le joueur. Lorsque l'enemy est suffisamment proche du joueur, il déclenche son attaque et inflige des dégâts au joueur. Cette fonctionnalité a été ajoutée pour rendre le gameplay plus intéressant et offrir une expérience plus dynamique aux joueurs. Tout cela a demandé beaucoup de temps et d'efforts, mais cela en valait la peine pour améliorer l'expérience de jeu globale.

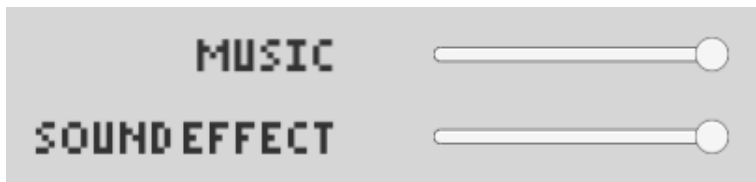
```
if (distPlayerA < distPlayerB && distPlayerA > stoppingDistance && distPlayerA < agroRa
{
    FollowPlayer(targetA);
}
else if (distPlayerB < distPlayerA && distPlayerB > stoppingDistance && distPlayerB < a
{
    FollowPlayer(targetB);
}
else
{
    HandleIdleAndAttack(distPlayerA, distPlayerB);
}
```

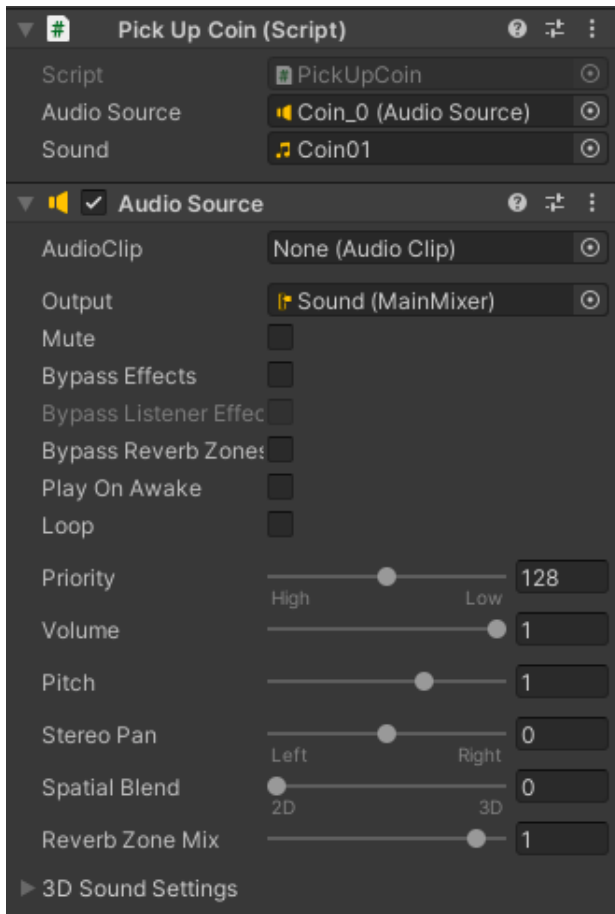
Finalement, la mise en place de l'IA pour que l'enemy suive le joueur linéairement uniquement sur l'axe des x a été une tâche plus difficile que prévue. Nous avons dû passer par plusieurs itérations avant de trouver la solution idéale. En effet, lors des premiers tests, l'enemy avait tendance à voler ou à flotter, ce qui le rendait difficile à utiliser. Nous avons réalisé que le problème venait de la gravité qui était inadaptée à cette situation. Nous avons alors expérimenté plusieurs paramètres de gravité pour trouver le juste équilibre entre une gravité réaliste et suffisamment puissante pour empêcher l'enemy de décoller du sol tout en suivant le joueur sur l'axe des x. Après plusieurs essais, nous avons finalement trouvé la configuration idéale qui a permis à l'enemy de suivre le joueur sans défaillance.



## 2.3 Son Effet

En ce qui concerne la musique du jeu, nous avons voulu donner une ambiance sombre et oppressante qui rappelle l'enfer. Nous avons donc contacté une amie pour qu'elle nous compose une musique originale qui collerait parfaitement à l'univers du jeu. Grâce à elle, nous avons pu ajouter une touche personnelle à notre jeu et rendre l'expérience de jeu encore plus immersive pour les joueurs. Les effets sonores quant à eux sont essentiels pour renforcer l'immersion et donner une véritable dimension sonore à l'univers du jeu. Nous avons cherché des sons gratuits et de qualité sur internet afin de gagner du temps, mais nous avons également apporté des modifications pour les adapter à nos besoins spécifiques. Pour que les joueurs puissent régler le volume sonore à leur guise, nous avons intégré une catégorie "Réglages" dans le menu de démarrage du jeu, où le volume sonore de la musique et des effets sonores peut être ajusté individuellement. Cela permet aux joueurs de personnaliser leur expérience de jeu et de l'adapter à leurs préférences auditives.



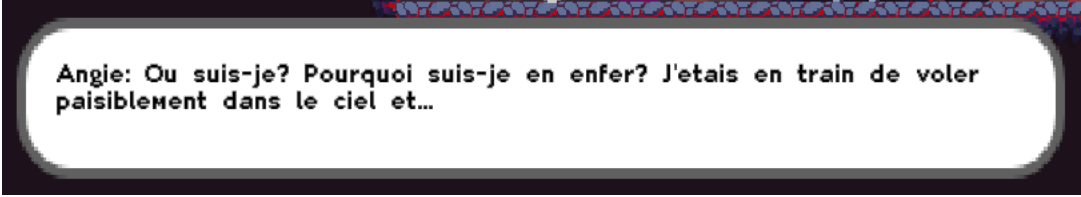


## 2.4 Dialogues

Pour la finalisation des dialogues, il ne restait plus qu'à intégrer les dialogues dans le gameplay et à gérer leur défilement. Heureusement, pour la première soutenance, les dialogues d'introduction avaient déjà été écrits. Nous avons donc travaillé sur la création d'un script qui affiche les dialogues à l'écran. Le script prend en input une liste de chaînes de caractères qu'il parcourt jusqu'à la fin. Pour gérer l'affichage des dialogues, nous avons utilisé la chatbox que nous avions désigné auparavant. Pour activer le défilement des dialogues, nous avons transformé le canvas de la chatbox en un bouton. Ainsi, lorsque le joueur clique sur le canvas de la chatbox, le dialogue passe au suivant. Ce système simple et efficace permet aux joueurs de lire les dialogues à leur propre rythme tout en étant plongé dans l'univers du jeu.

```
void TypingText(string[] sentence)
{
    manager.textDisplay.text = "";
    manager.textDisplay.text = sentence[index];
    if (manager.textDisplay.text == sentence[index])
        manager.continueButton.SetActive(true);
}
```





Angie: Ou suis-je? Pourquoi suis-je en enfer? J'etais en train de voler paisiblement dans le ciel et...

## 2.5 Gestion du GIT

Au cours de l'avancement du projet, nous avons commencé à adopter une méthode de travail plus organisée en utilisant des branches dédiées à chaque membre du groupe. Cela a permis de travailler de manière plus efficace et de gagner en organisation. Cependant, nous avons rencontré des difficultés lors de la gestion des branches, en particulier lors de la résolution des conflits de fusion. Malgré plusieurs tentatives pour les résoudre, ces conflits demeuraient complexes et difficiles à gérer. Pour nous aider à mieux comprendre la gestion des branches, nous avons sollicité l'aide de notre enseignant M. Hervot. Nous avons ainsi réalisé que certaines branches créées ne pointaient pas au bon endroit. Comme nous n'avions pas les autorisations pour supprimer ou déplacer ces branches, nous avons décidé d'abandonner certaines d'entre elles pour en recréer de nouvelles et travailler de manière plus saine. C'est pourquoi les branches "Noe", "Theo", "Leandro", "backtothefutur" et "FixMergeConflictsTheo" sont des branches fantômes que nous n'utiliserons plus, mais que nous ne pouvons pas supprimer. Cette expérience nous a permis de comprendre l'importance de la gestion des branches et de l'organisation de notre projet sur GIT.

## 2.6 Ressentis

Au cours de ce projet, j'ai acquis de nombreuses compétences qui vont me servir dans ma carrière professionnelle, notamment en termes de développement et de gestion de projet. J'ai pu apprendre de nombreuses choses sur Unity, ainsi que sur la gestion d'un projet en utilisant Git. Les problèmes que nous avons rencontrés avec l'implémentation de l'IA sur l'ennemi nous ont permis de mieux comprendre comment fonctionne un ennemi dans un jeu vidéo, ainsi que les différents aspects de la conception de jeux vidéo, qu'il s'agisse du développement en réseau ou en local. Cependant, la chose la plus importante que j'ai apprise est sans aucun doute la gestion du Git. Avant ce projet, je n'avais jamais utilisé les branches pour gérer un projet et je ne connaissais pas vraiment les concepts de base tels que les commits, les branches, etc. Grâce à ce projet, j'ai acquis une solide compréhension de l'outil et je suis maintenant capable de l'utiliser efficacement pour des projets futurs.

## Chapitre 3

# Leandro Tolaini

### 3.1 Design des personnages

La partie design des personnages est focalisée sur l'implémentation des animations maintenant que les assets sont choisis. Lors de la première soutenance, j'ai appris à prendre en main les outils à notre disposition sur Unity, en implémentant la première animation du jeu : l'animation de marche du démon.

Maintenant, afin d'avoir un rendu bien plus abouti, il fallait implémenter les animations d'idle (quand le personnage ne bouge pas), de saut, d'attaque et d'escalade (pour monter aux échelles). Toutes ces animations ont été fournies dans l'asset que nous avons acheté sauf celles d'escalade.

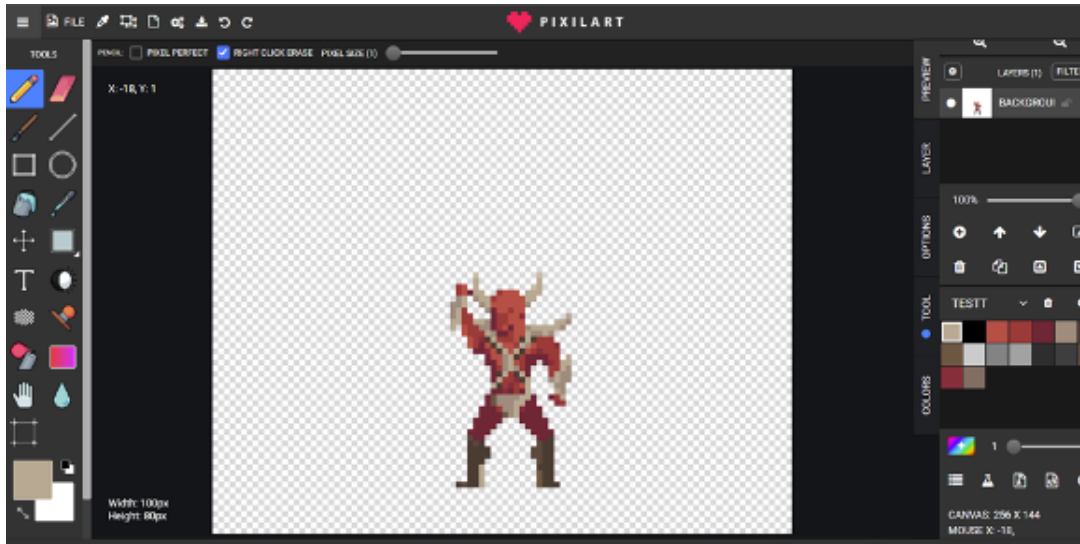
La première étape a donc dû être de créer moi-même les animations manquantes. Pour cela, je suis allé sur un site spécialisé :



De là j'ai pu m'aider d'un modèle de base du pack d'asset :



Et avec mon peu de compétences en dessin et beaucoup plus de temps que j'aurais imaginé j'ai enfin pu arriver à un résultat satisfaisant :

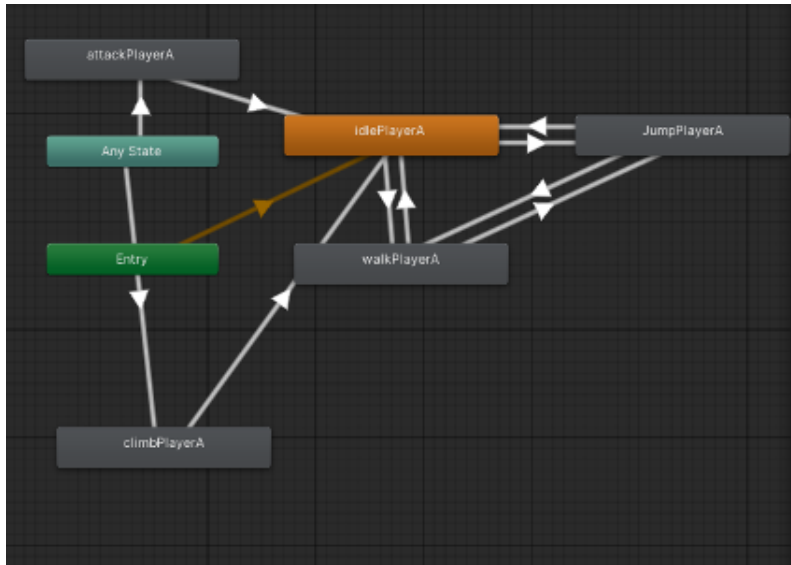


Ensuite il m'a suffit de retourner l'image sur l'axe x pour avoir deux frames qui mise à bout donnaient un rendu satisfaisant pour une animation d'escalade.

J'ai fait de même pour l'ange :



Une fois toutes les animations faites dans Unity, je devais réussir à faire en sorte qu'elles soient jouées au bon moment, en me servant de l'Animator. Le schéma pour du démon, assez similaire à celui de l'ange, ressemble à ça :

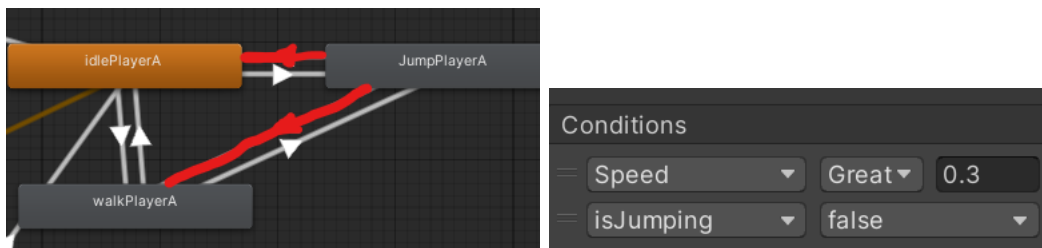


Chaque lien représente une transition possible d'une animation à une autre : par exemple, l'animation d'attaque et celle d'escalade sont liées au bloc « Any State » qui signifie qu'elles peuvent être jouées qu'importe l'animation actuelles. De même l'animation de marche ne peut être jouée qu'après celle de saut ou d'idle.

Ces transitions fonctionnent grâce à des conditions utilisant des variables qui peuvent être de plusieurs types différents. Je me suis servi des conditions suivantes :

Speed	0
isJumping	<input type="checkbox"/>
Attack	<input checked="" type="radio"/>
IsClimbing	<input type="checkbox"/>

Par exemple, j'ai besoin du booléen isJumping et du float Speed pour savoir quelle animation jouer après un saut. Si isJumping est à false, et que la vitesse est supérieure à une certaine valeur, alors l'animation de marche doit être jouer, sinon cela sera l'animation d'idle.

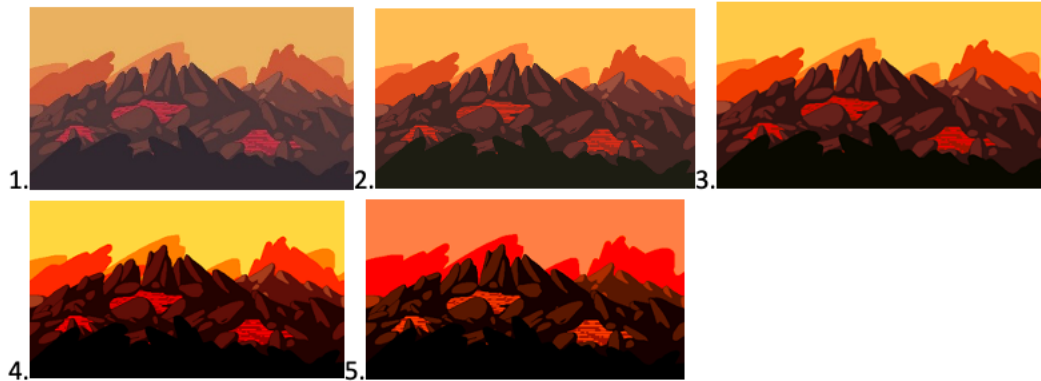


Il suffisait ensuite de modifier les scripts aux bons endroits grâce à ce genre d'instructions :

```
animator.SetBool("isJumping", false);
animator.SetFloat("Speed", Mathf.Abs(moveInput));
```

## 3.2 Design des niveaux

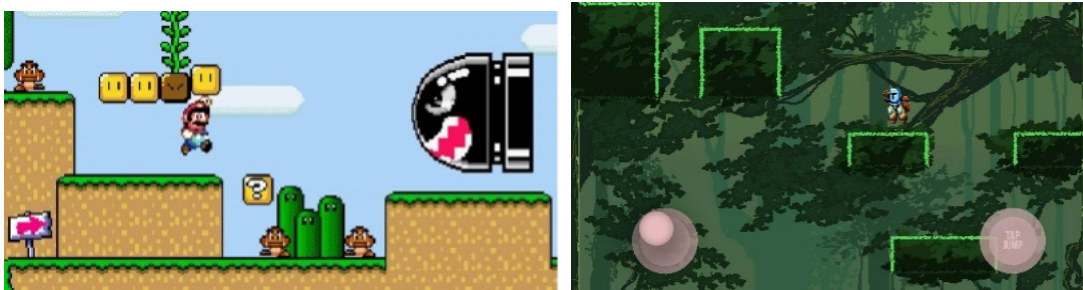
Je voulais donner une identité plus forte au jeu, tout en essayant de « recycler » nos asset. J'ai donc eu l'idée de modifier en peu le background à chaque niveau :



Plus les niveaux avancent, plus l'arrière-plan a de contraste, donnant un effet bien plus vif et cauchemardesque, comme si la température augmentait, ce qui symbolisera aussi l'augmentation de la difficulté des niveaux.

Le premier niveau faisait office de didacticiel, toutes les mécaniques étaient expliquées de manière claire afin de rendre la prise en main du jeu plus simple pour les joueurs, et par conséquent, il n'a pas demandé beaucoup d'imagination à concevoir. Il est très linéaire et simple, et c'était le rendu recherché. En revanche, je veux que les autres niveaux soient en vrai défis, avec des niveaux bien plus réfléchis et utilisant les mécaniques implémentées.

C'est à partir de ce moment que je me suis rendu compte que le game design est un métier à part entière : créer des niveaux attrayants, avec un bon équilibre entre la difficulté et l'imagination n'est pas simple. Il ne faut pas que les niveaux soient linéaires, peu intéressants, assez simples et trop courts. Pour éviter cela je me suis documenté sur le sujet, et je me suis intéressé au travail de succès commerciaux du genre. Par exemple les Mario Bros, pour la diversité de leurs niveaux, Jump King pour un autre style et la verticalité, ou encore Celeste pour la difficulté.

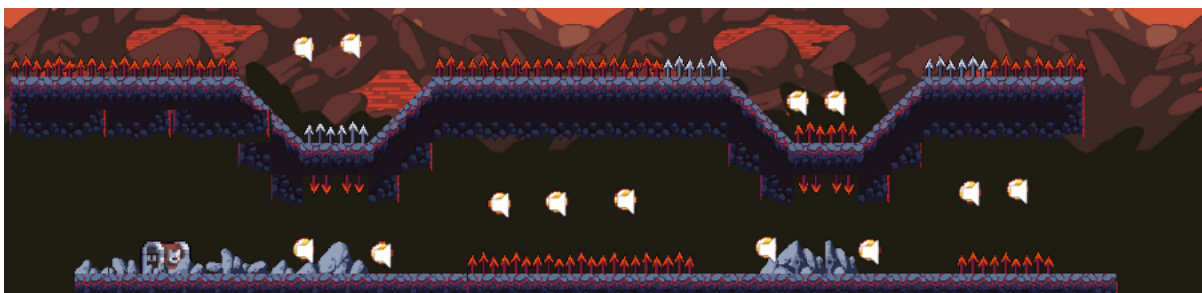




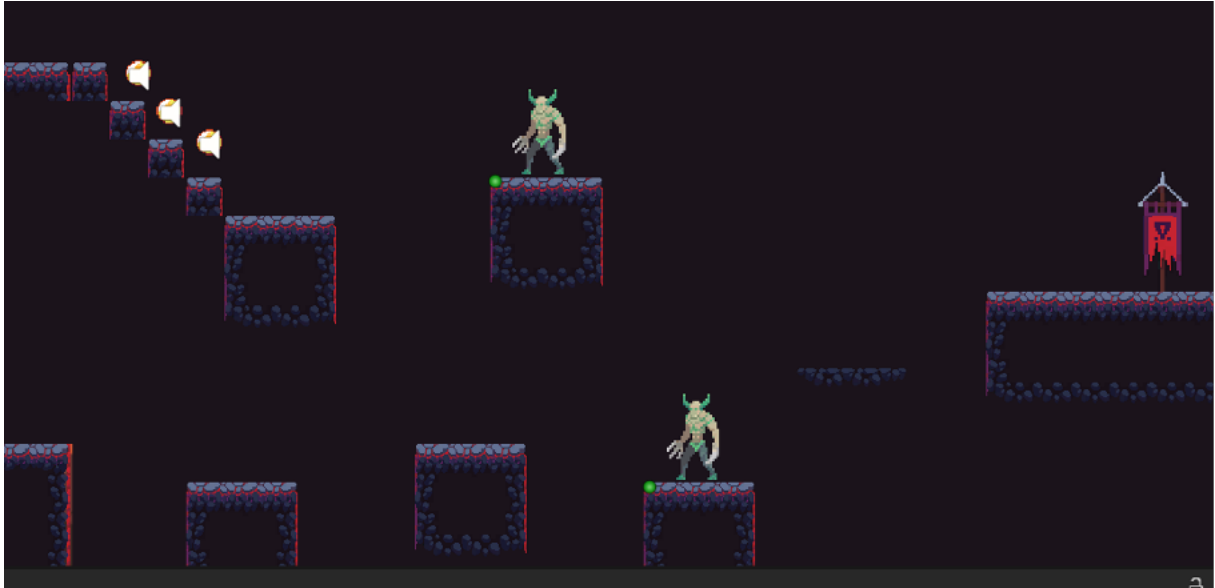
Le niveau 2 se veut comme le premier vrai niveau du jeu, sa conception se voulait plus travaillée. La mécanique des coffres est un bon moyen de cacher des clés dans le niveau, obligeant les joueurs à chercher comme ici :



La clé se trouve dans un passage secret en bas, visible grâce au double saut de l'ange. Le démon devra combattre les ennemis et il faudra aller contre le sens logique du niveau (de gauche à droite) pour atteindre le coffre.



Des parties deviennent plus techniques, comme par exemple en se servant des piques pour atteindre les coffres.



Un genre de grotte est présent dans le niveau, où de l'agilité est nécessaire. Ce genre de passage sera réutilisé et adapté.

Il faut garder en mémoire que le niveau 2 reste assez simple, même si il marque un assez gros contraste avec le premier. Les autres niveaux sont encore en développement, avec de nouveaux ennemis et peut-être de nouvelles mécaniques que j'ai imaginé en plus de nos idées de base, dans le cas où nous ne serions pas en retard.

### 3.3 Aspect technique

J'ai aussi dû implémenter de nouvelles choses qui seront utiles dans tous les niveaux.

Par exemple le spawn des joueurs, le checkpoint leur permettant de ne pas recommencer le niveau s'ils meurent à un certain stade, ou encore les pièces qu'ils peuvent ramasser dans les niveaux.



Toutes ces zones sont symbolisées par des drapeaux animés.

Le fonctionnement de ce genre d'objet peut presque se résumer à une seule fonction, comme pour le checkpoint par exemple :

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("PlayerA") || collision.CompareTag("PlayerB"))
    {
        playerSpawn.position = transform.position;
    }
}
```

Si le Joueur A ou le Joueur B entrent en contact avec le Checkpoint, alors la position du spawn devient celle de l'objet.

Le ramassage et stockage du nombre de pièces est aussi géré par un script résumable en une fonction principale :



```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("PlayerA") || collision.CompareTag("PlayerB"))
    {
        AudioManager.instance.PlayClipAt(sound, transform.position);
        Inventory.instance.AddCoins(1);
        Destroy(gameObject);
    }
}
```

Si le Joueur A ou le Joueur B entrent en contact avec la pièce, on joue un son, on ajoute une pièce à l'inventaire et on détruit cette dernière.

### 3.4 Ressenti

M'imposer une régularité dans mon travail et une organisation dès la fin de la première soutenance m'a beaucoup aidé en me donnant un rythme qui m'a permis de faire ce que je voulais en temps et en heure. Durant cette période la prise en main de Unity a été bien meilleure qu'au début, j'ai pu me détacher de plus en plus des tutos trouvables sur internet et me débrouiller seul à la place. Malgré ceci les difficultés que j'ai rencontrées se trouvent dans la compréhension des scripts fait par les autres membres du groupe au moment des animations, ce qui a mené à quelques bugs notamment sur le saut.

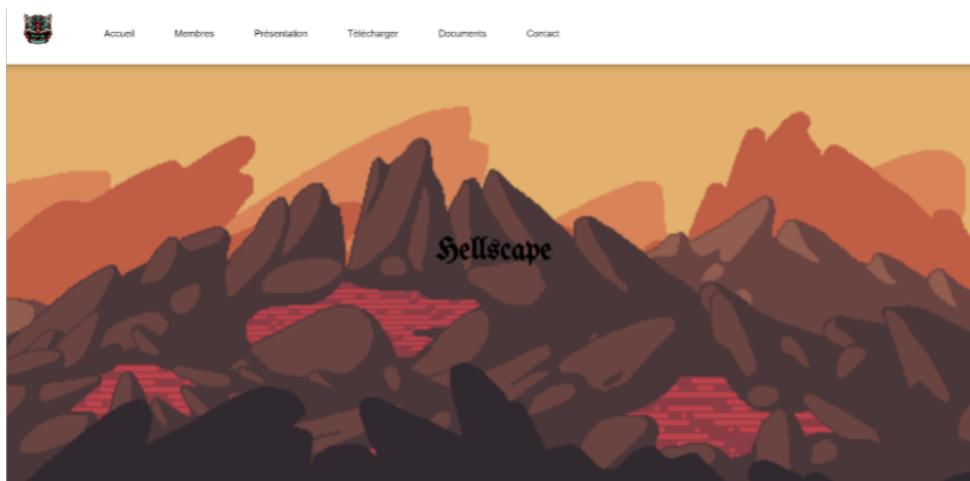


## Chapitre 4

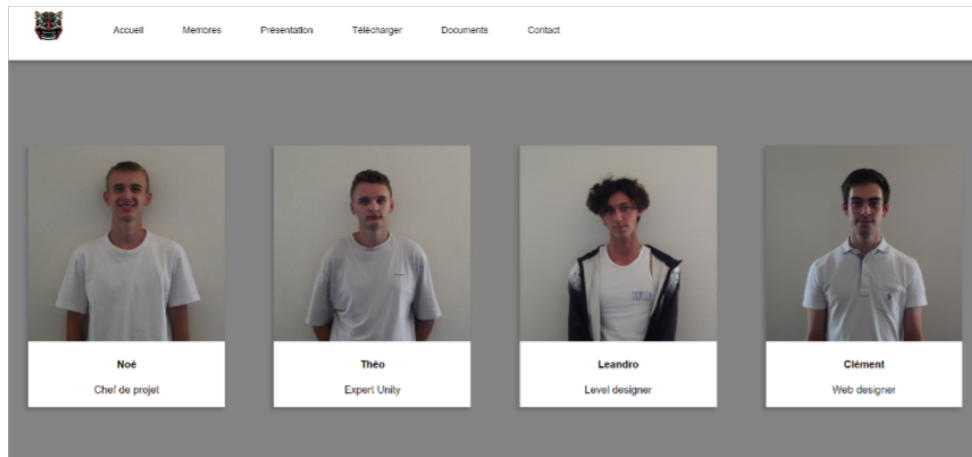
# Clément Grégoire

### 4.1 Site Web

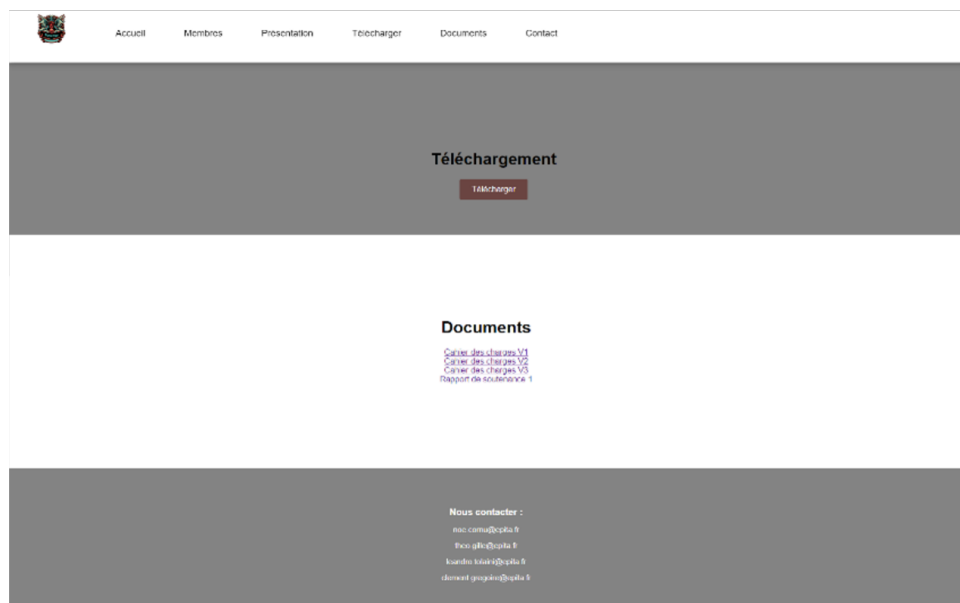
Pour le site internet, nous avons décidé de rajouter le nom de notre jeu sur la page d'accueil. La police utilisée est UnifrakturMaguntia, c'est une police de style gothique qui rappelle l'univers du jeu.



Dans la section membres, nous avons rajouté nos photos. Nous avons opté pour nos photos de profil du cri car elles ont toutes le même format. Nous avons utilisé le site internet <https://imgur.com/> pour mettre les photos en ligne.



Enfin, le reste du Css à été implémenté à la fin du site. Les documents ont été mis à jour. Le bouton téléchargement n'est toujours pas fonctionnel.



Le site est disponible sur internet à l'adresse suivante : <https://noecrn.github.io/t>. Il est hébergé sur GitHub.

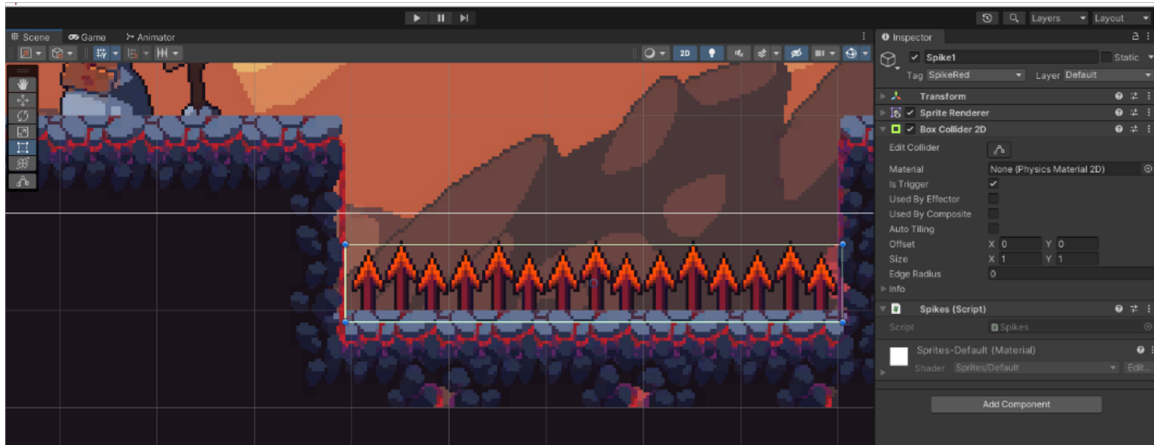
Comme j'avais sous-estimé la conception de ce site avant la première soutenance, j'ai commencé à m'en occuper plus rapidement. J'ai eu plus de facilité à régler les problèmes avec plus de temps.

Le site internet est donc terminé et en ligne. Pour la dernière soutenance, il faudra mettre le jeu disponible avec le bouton téléchargement et mettre les documents à jours.

## 4.2 Les piques

Les piques sont de deux couleurs différentes, rouges ou blancs. Les piques rouges infligent des dégâts uniquement à l'ange et les piques.

Pour implémenter cette mécanique, les piques sont associés à un box collider qui nous permet de détecter les collisions avec les joueurs. Les piques possèdent également un tag qui indique s'ils sont rouges ou blancs.



Pour la partie code, il faut utiliser la méthode `OnTriggerStay2d` et vérifier que le joueur est sur les piques qui lui infligent des dégâts. Si c'est le cas, on appelle la méthode `TakeDamage`.

```
void OnTriggerStay2D(Collider2D col)
{
    if (col.gameObject.tag == "PlayerA" && gameObject.tag == "SpikeWhite")
    {
        PlayerHealth.instance.TakeDamage(1);
    }

    if (col.gameObject.tag == "PlayerB" && gameObject.tag == "SpikeRed")
    {
        PlayerHealth.instance.TakeDamage(100);
    }
}
```

## 4.3 Double saut

La capacité de l'ange est de pouvoir effectuer un double saut.

Cette fonctionnalité n'est pas encore disponible dans le jeu car nous faisons face à un problème de `groundCheck` que nous n'avons pas réussi à résoudre. Cependant, le code suivant devrait permettre à l'ange d'effectuer le double saut quand ce problème sera résolu.

On rajoute deux attributs au script de mouvement de l'ange : `jumpLeft` et `maxJump`. `maxJump` est initialisé à 2 et `JumpLeft` prends la valeur de `maxJump` dans la méthode `Awake`.

Dans la méthode Update, lorsque la touche de saut est pressée, on regarde si jumpLeft est supérieur à 0. Si c'est le cas, on applique le saut à l'ange et on décrémente jumpLeft. Si jumpLeft est toujours supérieur à 0, on peut effectuer un autre saut.

```
if (jumpLeft > 0 && Input.GetKeyDown(KeyCode.Space))
{
    Debug.Log("ca marche");
    if (jumpLeft == 2)
    {
        rb.AddForce(new Vector2(0f, 6), ForceMode2D.Impulse);
    }
    else
    {
        rb.AddForce(new Vector2(0f, 4), ForceMode2D.Impulse);
    }

    isJumping = true;
    jumpLeft -= 1;

    photonView.RPC("Jump", RpcTarget.Others);
}
```

Enfin, si le joueur est au sol, on réassigne jumpLeft à maxJump, il peut à nouveau faire un double saut.

```
if (isGrounded)
{
    jumpLeft = maxJump;
}
```

## 4.4 L'attaque du démon

La capacité du démon est de pouvoir attaquer les ennemis.

Sur le démon nous avons donc rajouté une zone qui correspond à l'endroit où l'attaque prendra effet



Dans le code, nous devons détecter le moment où la touche d'attaque est pressée dans la méthode Update afin de lancer la méthode Attack.

```
if (Input.GetKeyDown(KeyCode.K))
{
    Attack();
}
```

Dans la méthode Attack, on déclenche l'animation d'attaque du démon. Puis on met dans un tableau tous les ennemis qui étaient dans la zone d'attaque lorsque le joueur a appuyé sur la touche d'attaque.

Enfin, on applique des dégâts à tous les ennemis contenus dans le tableau.

```
public void Attack()
{
    animator.SetTrigger("Attack");
    Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(attackPoint.position, attackRa
    foreach (var enemy in hitEnemies)
    {
        if (enemy.gameObject.TryGetComponent<EnemyFollow>(out EnemyFollow enemyHealth))
        {
            enemyHealth.TakeDamage(15);
        }
    }
}
```

Etant donné les dégâts qu'infligent le démon et la vie des ennemies, le joueur est capable de tuer les ennemis en deux coups.

## 4.5 Ressentis

Comme je n'avais pas fait d'Unity avant la première soutenance, j'ai eu du mal à commencer. Les autres avaient déjà avancé sur le jeu, beaucoup de fonctionnalités étaient déjà implémenté ce qui rendait la compréhension difficile pour moi. Cependant, mes tâches n'étaient pas indispensables pour que les autres membres du groupe puissent continuer. J'ai donc pu avancer à mon rythme sans bloquer.

# Chapitre 5

## Théo Gille

### 5.1 Multijoueurs

Pour cette deuxième soutenance, je me suis toujours occupé l'aspect multijoueur du jeu. Ainsi, dans la plupart des cas, je dois m'occuper de repasser sur les scripts pour ajouter la synchronisation entre les joueurs et d'autres choses. Pour permettre cela, il faut faire hériter la classe que l'on souhaite par « MonoBehaviourPunCallbacks » et « IPunObservable ».

- **MonoBehaviourPunCallbacks** : permet d'utiliser les Callbacks du serveur.
- **IPunObservable** : permet la synchronisation des objets entre les clients en ligne.

Lorsqu'on utilise IPunObservable, nous devons ajouter ces méthodes nommées OnPhotonSerializeView et OnPhotonDeserializeView qui sont appelées pour sérialiser et désérialiser les données de l'objet pour la synchronisation.

```
public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
{
    if (stream.IsWriting)
    {
        // Envoi des informations de mouvement aux autres joueurs sur le réseau
        stream.SendNext(transform.position);
        stream.SendNext(rb.velocity);
        stream.SendNext(animotor.GetFloat("Speed"));
        stream.SendNext(animotor.GetBool("IsGrounded"));
    }
    else
    {
        // Réception des informations de mouvement des autres joueurs sur le réseau
        transform.position = (Vector3)stream.ReceiveNext();
        rb.velocity = (Vector2)stream.ReceiveNext();
        animotor.SetFloat("Speed", (float)stream.ReceiveNext());
        animotor.SetBool("IsGrounded", (bool)stream.ReceiveNext());
    }
}
```

Ici, nous voyons la méthode OnPhotonSerializeView utilisée pour le déplacement du joueur, ainsi, on remarque qu'il y a une partie qui va servir pour l'envoi des informations et une autre

partie sur la réception des informations. Il y a aussi d'autres méthodes à ajouter comme notamment `photonView.RPC()` qui permet de synchroniser les actions entre les clients du serveur comme on peut le voir ci-dessous :

```
if (isGrounded && Input.GetKeyDown(KeyCode.Space) && Mathf.Abs(rb.velocity.y) < 0.01f)
{
    rb.AddForce(new Vector2(0f, jumpForce), ForceMode2D.Impulse);
    isJumping = true;
    animator.SetBool("isJumping", true);

    // Envoi d'un appel de fonction RPC pour synchroniser le saut avec les autres j
    photonView.RPC("Jump", RpcTarget.Others);
}
```

Dans ce cas, cela va permettre de synchroniser l'action de sauter dans le jeu entre les joueurs.

Ainsi, dans le cadre du multijoueur, c'est ce que j'ai pu faire entre la première et la deuxième soutenance.

## 5.2 UI Menu

Comme pour la première soutenance, j'ai continué l'implémentation et l'amélioration de tout ce qui est UI du jeu. J'ai tout d'abord corrigé et fini le Menu Principal du jeu. (celui-ci peut toujours être modifié dans le futur.)



Il possède notamment une fenêtre de réglage qui est pour l'instant basique et qui pourrait évoluer au fil du temps :



Ainsi, pour gérer les différents paramètres réglables possibles, il faut passer par un script permettant de gérer et d'adapter chaque scène en fonction de la résolution demandée, du son de la Musique et des sons environnement. (Bruit de pièces, attaque, etc.)

```
public Dropdown resolutionDropdown;

Resolution[] resolutions;

public void Start()
{
    audioMixer.GetFloat("Music", out float musicValueForSlider);
    musicSlider.value = musicValueForSlider;

    audioMixer.GetFloat("Sound", out float soundValueForSlider);
    soundSlider.value = soundValueForSlider;

    resolutions = Screen.resolutions.Select(resolution => new Resolution { width = reso
    resolutionDropdown.ClearOptions();

    List<string> options = new List<string>();

    int currentResolutionIndex = 0;
    for (int i = 0; i < resolutions.Length; i++)
    {
        string option = resolutions[i].width + "x" + resolutions[i].height;
        options.Add(option);

        if(resolutions[i].width == Screen.width && resolutions[i].height == Screen.heigh
        {
            currentResolutionIndex = i;
        }
    }

    resolutionDropdown.AddOptions(options);
```



```

resolutionDropdown.value = currentResolutionIndex;
resolutionDropdown.RefreshShownValue();

Screen.fullScreen = true;
}
public void SetFullScreen(bool isFullScreen)
{
    Screen.fullScreen = isFullScreen;
}

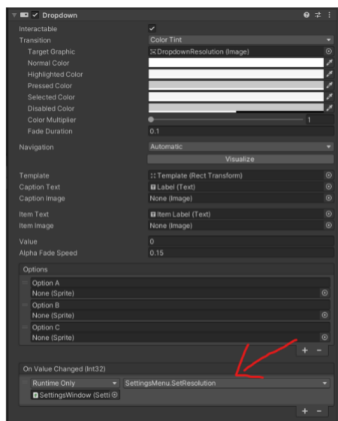
public void SetResolution(int resolutionIndex)
{
    Resolution resolution = resolutions[resolutionIndex];
    Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);
}

```

Ici par exemple, on peut voir la manière dont est géré la résolution dans le DropDown, nous permettant de choisir et d'adapter la fenêtre de jeu en conséquence.

Ainsi, l'ensemble du script permet une action particulière, pour que cela marche, il ne faut pas simplement attacher le script au Canvas demandé, il faut aussi attribuer aux boutons et Slider la commande qui doit être effectuée lorsqu'on appuie dessus :

Par exemple ici, le Dropdown est utilisée pour ajuster la résolution, il est donc seulement relié à cette partie-là du script SettingsWindow.



J'ai pu aussi ajouter deux autres UI qui sont un compteur de pièces et une barre de vie qui fonctionnent :



Ces deux UI sont essentielles puisqu'elles sont liées à des mécaniques de jeu, ce fut donc une priorité à faire.

## 5.3 Interaction & Echelles

Étant donné que mes collègues étaient occupés et j'avais fini mes tâches primaires, j'ai décidé de m'avancer sur certaines tâches, comme notamment un inventaire fonctionnel, interagir avec certains objets.

Dans notre jeu, interagir avec des objets tels que des coffres ou bien des clefs va être quelque chose d'essentiel pour parcourir les différents niveaux. J'ai décidé de m'occuper de cela car tout d'abord ; il y a une grande partie de UI avec l'inventaire notamment :



Cette fenêtre va pouvoir être utilisée pour vérifier notre inventaire, mais surtout pour utiliser certains objets comme notamment les clefs ou des objets nous permettant de récupérer de la vie.

Lorsque nous passons la souris sur un objet en particulier, celui-ci s'affiche sur la fenêtre de gauche et montre à quoi celui-ci peut servir (grâce au texte item Description qui sera remplacé et différent selon chaque objet.). Actuellement, cette fonctionnalité n'est pas fonctionnelle à 100 %, mais le sera pour la prochaine soutenance.

D'autres interactions plus simples vont pouvoir être fait (déjà implémentées) comme notamment le fait de porter ou de jeter des objets :



L'ensemble de ces interactions vont être gérées par plusieurs scripts tels que : Item.cs, InventorySystem.cs et InteractionSystem.cs.

**Item.cs** : permet la création d'objets interactifs et la personnalisation des événements auxquels ils sont associés si on les utilise ainsi que leur type (PickUp, Examine ou GrabDrop)

**InventorySystem** : permet de gérer les différentes fonctionnalités de l'inventaire.

**InteractionSystem** : permet de gérer les interactions entre les objets.

```
public void GrabDrop()
{
    if (isGrabbing)
    {
        photonView.RPC("DropItem", RpcTarget.All);
    }
    else
    {
        photonView.RPC("GrabItem", RpcTarget.All);
    }
}
```

```
[PunRPC]
void GrabItem()
{
    if (isGrabbing)
    {
        isGrabbing = false;

        grabbedObject.transform.parent = null;

        grabbedObject.transform.position =
```

```

        new Vector3(grabbedObject.transform.position.x, grabbedObjectYValue, grabbedObjectYValue);
    }

    grabbedObject = null;
}

else
{
    isGrabbing = true;

    grabbedObject = detectedObject;

    grabbedObject.transform.parent = transform;

    grabbedObjectYValue = grabbedObject.transform.position.y;

    grabbedObject.transform.localPosition = grabPoint.localPosition;
}
}

```

C'est deux scripts viennent tout deux de la classe InteractionSystem, et vont être utilisés pour attraper ou poser des Items. On remarque qu'il y a une fonction où seulement des RPC vont être appelés, et l'autre fonction va contenir l'ensemble du code en ajoutant [PunRPC] au début pour faire comprendre que cette fonction peut être appelée avec un RPC et donc liée à la synchronisation.

L'ensemble de ces interactions ne sont pas fonctionnelles à 100 %.

Pour finir, je me suis occupé de la mécanique des échelles permettant aux joueurs de descendre et monter aux échelles qui sont placés et utiles dans notre jeu. Avant de commencer cette mécanique, je pensais que celle-ci allait être plutôt facile, pourtant, j'ai eu énormément de mal à gérer le comportement de notre personnage pour avoir un comportement prévisible, car je souhaitais que le personnage ne puisse plus du tout bouger horizontalement s'il est sur l'échelle à partir d'une certaine hauteur.

Cette mécanique est gérée par le script Ladder qui va utiliser plusieurs choses :

```

public bool isLadder;
public bool isClimbing;

public Rigidbody2D rb;

public Animator animator;

private CompositeCollider2D collider;

```

**isLadder** : permet de détecter si on est en présence d'une échelle ou non.

**isClimbing** : permet de détecter si l'on grimpe l'échelle.

**Collider :** permet de gérer les collider du TileMap (utilisé ici pour passer les collider en OnTrigger ou non)

## 5.4 Ressentis

Après cette deuxième soutenance, je trouve que j'ai pu acquérir de nombreuses connaissances sur tout ce qui est Collider et Rigidbody, notamment lorsque j'ai fait le script MovePlayer, mais surtout Ladder où le fait de gérer les collider de toute TileMap à été compliqué pour moi. Malgré certains scripts qui ne fonctionnent pas à 100 %, j'ai pu atteindre mes objectifs pour cette deuxième soutenance, et même faire plus avec notamment l'inventaire et les interactions.