DOSSIER EXPLICATIF | M1-DEDI | ALBERTINI-CURGIS-DUCROS

➢ Description du concept :

Concept : Notre jeu s'inscrit dans un style metroid like, c'est-à-dire d'action-aventure à travers un système de plate-formes, qui empruntent fortement à des jeu de séries tel que Metroid, Castlevania ou Rayman, qui ont été nos sources d'inspirations premières. En effet, notre choix s'est porté sur ce type de plateforme 2D, car le joueur est beaucoup plus libre dans ses capacités de mouvements, lui permettant alors de parcourir le niveau du jeu de manière variées.
Par ailleurs, cela nous a également permis d'adopter plus de liberté et de créativité dans la manière d'imaginer et de concevoir l'ensemble du jeu, ainsi que son ambiance. De ce fait, nous avons opté pour un univers de fantaisie, mélangeant des éléments à la fois réalistes et mythiques.

Personnages présents :

- Personnage principal : Ernesto le poisson
- Ennemis : les dragons, et les crocodiles

Règles du jeu : L'objectif principal est de diriger le personnage principal Ernesto vers la fin du niveau, tout en évitant les divers ennemis et obstacles qui croiseront son chemin à travers son aventure. Par ailleurs, la mission du personnage principal est également d'aller chercher une clé, qui lui permettra d'ouvrir un coffre vers la fin du parcours, et ainsi de valider le niveau.

➢ Notice d'utilisation :

Gameplay : Au niveau du gameplay, notre personnage à la possibilité de se déplacer de manière horizontale dans le niveau mais aussi de manière verticale. C'est cette verticalité qui permet au personnage de traverser le niveau en cherchant le coffre et la clé pour l'ouvrir. Le but étant au final de trouver la sortie du niveau qui se fait en interagissant avec le dernier coffre.
Le jeu étant un métroïd-like, le personnage peut revenir sur ses pas s'il lui manque un objet pour déverrouiller le coffre à la fin du niveau. Le game design a donc été créé de manière à ce que les éléments de plateforme puissent être utilisés si le personnage retourne en arrière, de manière à empêcher le joueur de rester bloqué.

Le personnage Ernesto doit donc avancer dans le niveau en évitant de tomber dans l'eau car même si c'est un poisson, il ne sait pas nager, entraînant une perte de points de vie et replaçant le personnage au dernier "checkpoint" qui est un point de sauvegarde. Il doit aussi éviter les ennemis ou bien les éliminer en leur sautant sur la tête. Si les ennemis touchent le personnage, celui-ci perdra un pourcentage de ses points de vie. De la même manière, si le personnage touche les pics, il perdra là aussi quelques points de vie, dont la perte totale entraîne sa mort. Cependant, pour empêcher que notre poisson perde sa vie trop

rapidement, celui-ci obtient une petite période d'invincibilité après la perte de point de vie. Cette invincibilité est symbolisée par un clignotement de quelques secondes du personnage qui lui permet de s'échapper.
Il peut aussi regagner de la vie en ramassant les cœurs qui sont disséminés dans le niveau. Lorsque le personnage perd la totalité de ses points de vie, c'est alors le "Game Over" et il a donc la possibilité de recommencer le niveau ou alors de retourner au menu principal, voire même de quitter le jeu.

Quelques précisions sur les déplacements :

- Pour se déplacer le joueur doit utiliser les croix directionnelles de son clavier comme dans beaucoup de plateformers 2D sur PC. C'est ainsi que la croix directionnelle gauche permet au poisson de se déplacer vers la gauche et la croix directionnelle droite permet à celui-ci d'aller vers la droite.

- La barre espace permet de faire sauter le personnage, lui permettant ainsi d'atteindre les différentes plateformes. Le personnage ne peut pas effectuer de double saut, mais il possède en revanche un saut d'une grande amplitude.

- Pour ramasser les objets, le joueur a juste à déplacer le personnage sur l'objet pour que celui-ci soit collecté.

- Au niveau de la caméra, celle-ci est automatique et suit le personnage sans avoir besoin de la bouger manuellement. La caméra a un petit décalage temporel quand elle suit le joueur, ce qui permet ainsi à celui-ci de ne pas se sentir piégé dans le cadre qui l'entoure.

- La touche E est utilisée pour les interactions avec différents objets quand le message "appuyer sur E" apparaît. On peut ainsi utiliser la touche E pour monter aux échelles mais aussi pour redescendre, et enfin pour ouvrir un coffre.

- La touche ECHAP permet de mettre le jeu en pause. Le joueur a donc la possibilité de reprendre le jeu là où il l'a laissé, ou de retourner au menu principal, mais aussi de pouvoir aller dans les paramètre du jeu pour modifier la résolution, le son et l'utilisation du jeu en plein écran

- Le menu principal est disponible dès qu'on lance le jeu mais aussi à travers le menu échap ou après un game over. Le menu permet de lancer le jeu, mais aussi d'aller dans les paramètres qu'on a vu juste au-dessus, et enfin de quitter le jeu.

- Le dragon vole automatiquement de gauche à droite ou de bas en haut. Pour l'éliminer, notre protagoniste doit sauter sur sa tête. S'il saute sur une autre partie de son corps, il perd alors des points de vie

- Le crocodile est présent sur le sol et dans l'eau, pour l'anéantir, il faut sauter sur son dos. Si le protagoniste saute sur sa tête, il perd des points.

➢ Nos intentions de créations :

Pour concevoir notre interface de jeu, nous avons opté pour **une production complète** concernant les aspects technique et design. De ce fait, **l'ensemble des assets utilisés dans la production et la conception des personnages et des décors ont été entièrement réalisés par nous-mêmes.** Nous les avons réalisés sur Adobe Photoshop.
En effet, nous avons pris le parti de créer notre propre univers graphique afin de pouvoir développer une démarche graphique très créative autour de notre concept fantastique. De plus, cela nous a permis d'appréhender la conception d'un jeu dans l'entièreté de sa production à partir de rien. Ainsi, cela nous a permis de développer nos aptitudes dans l'importation et l'animation de nos propres assets, et de mieux comprendre les aspects techniques qui les accompagnent.

Quelques explications sur nos intentions concernant :

- **Les personnages :** nous avons choisi de concevoir nous même tous les personnages dans un style cartoon et dans une palette de couleurs vives. Ainsi, nous avons opté pour un design assez travaillé dans un univers fantastique.

  Notre protagoniste est un poisson à pattes portant le doux nom d'Ernesto. Il est équipé d'une épée et a la possibilité de marcher et de sauter. Son but ultime est de traverser les différents niveaux en tentant d'éviter les ennemis qui sont sur son chemin. On retrouve donc deux ennemis principaux qui sont : le dragon, il a la possibilité de voler de bas en haut et de gauche à droite et le crocodile, qui glisse sur le sol et qui peut également être présent dans les marécages.

- **Les décors, l'ambiance… :** notre intention était de concevoir une ambiance réaliste autour d'un environnement naturel marécageux/forêt, où les divers personnages fantastiques pourraient être mis en valeur, autant dans leurs mouvements que dans leurs manières d'interagir.
  De plus, nous voulions que le joueur soit le plus possible immergé dans cet univers, afin d'offrir à l'utilisateur une meilleure expérience de jeux c'est donc également pour cette raison que nous avons choisi de réaliser nos propres décors, fonds et accessoires. Notre choix de fond pour le jeu s'est alors porté sur une illustration en parallaxe, dans le but de donner une impression de mouvement.

- **Les sons :** en effet plusieurs mélodies et bruitages divers sont présents dans le jeu et interagissent avec l'environnement de gameplay. Tout d'abord, on peut y entendre 2 musiques différentes de manière extradiégétique, la première dès l'entrée dans le menu, et la seconde durant la partie.

➢ Nos intentions de productions :

Au travers de notre jeu, nous voulions rendre hommage aux plateformers 2D et notamment aux jeux metroid-likes qui ont été popularisés grâce au jeu Metroid. Ainsi, beaucoup de metroid-like fonctionnent de la même manière: on parcourt un niveau en évitant les pièges et les ennemis tout en récoltant des objets qui nous aideront à avancer. Ici, dans notre jeu, cela consiste à récupérer la clé pour ensuite trouver le coffre qui permettra quand on l'ouvre de terminer le niveau.

Beaucoup de metroid-like utilisent un système de tilesets pour la construction de leurs niveaux. Ainsi le monde est représenté comme un quadrillage et les tilesets sont des carrés qui une fois placés à côté les uns des autres, et assemblés, permettent de créer les plateformes, les décors... Nous avons donc choisi ce système là comme les jeux dont nous sommes inspirés.

Pour ces genres de jeu, beaucoup ont misé sur le pixel art comme ces jeux sont plutôt anciens, mais on retrouve encore ce type de graphisme aujourd'hui dans des jeux orientés rétro. Mais, pour beaucoup, les graphismes ont évolué et se sont diversifiés. Nous avons donc créé nos propres tilesets avec un style graphique qui nous est propre pour avoir notre propre univers. Un univers coloré qui plaira donc au plus grand nombre.

Nous n'avons pas de règles en jeu pour expliquer le principe car dans la majorité des metroid-likes, le joueur apparaît dans un univers sans indication de ce qu'il doit faire. Cependant, comme notre jeu est pour tout public, nous avons fait en sorte que les différents éléments qui composent notre jeu permettent à l'utilisateur de comprendre par lui-même. Par exemple, l'utilisateur va facilement comprendre qu'un cœur lui redonne de la vie. Et nous avons mis un système de checkpoints comme ça l'utilisateur pourra expérimenter de lui-même, ce qui arrive quand il tombe dans un piège, sans avoir à tout recommencer depuis le début (à moins qu'il n'ait plus de vie).

Le jeu étant créé pour plaire à tout le monde, le gameplay est pensé pour être facile à prendre en mains avec seulement les touches directionnelles pour se déplacer, la touche Espace pour sauter, la touche E pour interagir et la touche Echap pour mettre sur pause. Il est aussi possible de jouer sur appareils mobiles avec les touches qui sont remplacées par des touches tactiles.

➢ Production & intégration Unity :

→ En ce qui concerne la production et l'intégration, voici la liste des diverses fonctionnalités de Unity utilisées dans cette perspective, que l'on peut retrouver dans le jeu :

- imports de nos propre assets
- utilisation de sons divers : son de mouvement sauter, son de mouvement sauter sur un ennemi, son d'ambiance extradiégétique, son récupérer la clé ou autre éléments
- paramètres des composants : barre de vie, animations, GUI, colliders
- parallaxe

→ Au niveau de la production et de l'intégration sur Unity, nous avons utilisé plusieurs fonctionnalités pour créer notre jeu afin qu'il ressemble à ce que nous imaginions. Ainsi, nous avons importé **nos propres assets** que nous avions dessiné au préalable sur photoshop. L'univers entier, nos personnages, nos montres, … proviennent de notre imagination, avec un style orienté cartoon.
Les seules choses que nous n'avons pas fait nous-même sont les musiques et les sons que nous avons importé après les avoir téléchargé sur un site internet spécialisé dans la création de jeux vidéo: OpenGameArt.

→ Nous avons également des sons qui sont utilisés quand le poisson ramasse une clé, quand il reprend de la vie. Mais aussi quand il prend des dégâts, quand il tombe dans le vide pour informer l'utilisateur qu'il perd des points de vie. Un son se fait aussi entendre quand le joueur élimine un ennemi.
→ L'utilisation de sons extradiégétiques semble importante, nous avons des sons différents pour le game over et la fin du niveau. Au départ, nous voulions mettre des sons lorsque le personnage saute. Cependant, le jeu ayant beaucoup de verticalité, le personnage saute donc tout au long du niveau et entendre un son en permanence n'est pas agréable. Pour ce qui est des musiques, on peut entendre trois musiques différentes dans le jeu. On en a une au menu principal, une pendant la phase de gameplay et une pendant les crédits. Ces trois musiques sont différentes et elles ont été choisies pour compléter notre univers avec ce qu'elles apportent.

→ Plusieurs composants, comme par exemple la barre de vie, ont aussi été dessinés au préalable. La barre de vie a donc été dessinée de manière vide sur photoshop puis en l'intégrant dans unity, nous avons fait en sorte qu'elle augmente ou baisse en fonction de la vie du personnage grâce à des scripts et au GUI de Unity.

→ Par ailleurs, les GUI ont aussi été utilisés pour la création des boutons comme ceux du menu mais aussi pour écrire du texte qu'on peut par exemple voir apparaître quand le personnage peut interagir avec son environnement. Au niveau des animations, nous les avons réalisées grâce à l'animator présent sur Unity. Les personnages et créatures ont été dessinés frame par frame pour pouvoir justement les animer par la suite.
Ainsi, pour le personnage, nous avons des animations quand celui-ci respire, quand il marche, quand il grimpe et quand il meurt. Les créatures présentent moins d'animations, elles ont une animation de marche. Nous avons ensuite des animations plus pour le côté technique, avec un système de fondu au noir quand le personnnage tombe dans le vide pour ensuite le ramener sur la plateforme mais aussi un sytème de crédits où les différents noms des personnes qui ont travaillé sur le projet ou contribué défilent.

→ Pour nos différents éléments présents dans le jeu nous avons donc mis des colliders mais aussi des rigidbody pour que le joueur ait des interactions. Mais pour le fond de notre jeu (le background), nous avons utilisé un système différent de ce que nous avons fait avant. En effet, nous avons utilisé un effet de parallaxe pour avoir quelque chose de bien meilleur à l'œil. Le fond a donc différent niveaux de perspectives et de profondeur avec ce qui se

trouve au premier plan qui avance beaucoup plus vite que ce qui se trouve au dernier plan. Un système de boucle a aussi été mis pour que les décors se répètent une fois qu'on arrive à la fin de l'illustration, donnant l'impression d'un décor en continu.

→ Pour ce qui est des différents scripts que nous avons créé, si on prend cette grille de programmation représentant la complexité et quantité de scripts produits:

1. Variable
2. Fonctions / passages de paramètres
3. Variables privées publiques dans Unity
4. Fonctions standards simples unity (Start / update, oncollision)
5. Boucles (niveau moyen <= capacité à produire ces 3 étapes)
6. Coroutines
7. Fonctions standards plus complexes unity (Start / update)
8. Programmation orientée objet
9. Héritage

→ Nous avons ici tout utilisé sauf la programmation orientée objet ni l'héritage comme notre jeu n'en avait pas besoin pour fonctionner. Ainsi, tous nos scripts présentent des variables avec des fonctions.
Certaines variables sont privées et d'autres sont publiques. Le fait d'utiliser des variables publiques est intéressant car si on prend l'exemple de notre personnage, nous avons un script pour ses mouvements, et le fait de mettre la hauteur de saut en publique, nous a permis de régler directement en jeu cette hauteur pour que le poisson puisse atteindre les plateformes sans sauter ni trop haut ni pas assez haut.
Pour ce qui est des fonctions comme start et update, ce n'est pas ce que nous avons utilisé le plus mais cela nous a été utile à certains moments.

→ Nous avons parfois utilisé la fonction awake qui était plus utile que start au lancement du jeu. Les conditions et les boucles ont été très utiles car on en a dans énormément de nos scriptes, comme les boucles permettent de répéter des actions et les conditions de créer des événements qui ne seront remplis que si certaines choses sont faites.
Enfin, pour les coroutines, nous en avons utilisé à certains moments, comme elles permettent de faire des pauses dans notre programme. Mais cela, on va le voir plus en détail dans une fiche pédagogique brève.

Fiches pédagogique brève :

- **Le point que nous allons expliquer est l'utilisation de la coroutine :**

→ Notre jeu étant un plateformer 2D, il doit alors avoir certains codes du plateformer qu'on retrouve dans la plupart des jeux. Ainsi, nous voulons que quand notre personnage reçoit des dégâts, celui-ci obtienne une période d'invincibilité qui l'empêche de recevoir à nouveau des dégâts pour une durée que nous avons fixée à 2 secondes.

Cependant, nous nous sommes demandés comment le joueur pourrait comprendre qu'il est invincible pendant ces quelques secondes. Nous nous sommes donc inspirés du visuel présent dans des jeux comme Mario ou Metroid, où cette invincibilité est caractérisée par le clignotement du personnage. Ainsi, pour que notre personnage clignote lorsqu'il prend des dégâts, nous avons utilisé un système de coroutines.

Le but de notre script est de jouer sur la couleur du "Sprite Renderer" de notre personnage, et particulièrement sur l'"alpha" qui correspond donc à la translucidité de notre poisson. Nous allons donc alterner pendant la période d'invincibilité entre un personnage opaque et un personnage translucide pendant quelques secondes.

Ainsi dans notre script PlayerHealth qui correspond comme son nom l'indique à la vie de notre personnage, nous avons tout d'abord créé un système d'invincibilité.

```
public class PlayerHealth : MonoBehaviour
{
    public int maxHealth = 100;
    public int currentHealth;

    public float invincibilityTimeAfterHit = 2f;
    public float invincibilityFlashDelay = 0.2f;
    public bool isInvincible = false;
```

Nous avons tout d'abord fait une variable *isInvincible* que nous avons mis par défaut sur *false* comme le personnage n'est pas invincible par défaut. Nous avons ensuite *invincibilityTimeAfterHit* qui permet de choisir la durée d'invincibilité et *invincibilityFlashDelay* qui est utilisée pour le délai entre chaque clignotement.

```
public void TakeDamage(int damage)
{
    if (!isInvincible)
    {
        currentHealth -= damage;
        healthBar.SetHealth(currentHealth);

        //vérifier si le joueur est toujours vivant
        if(currentHealth <= 0)
        {
            Die();
            return;
        }

        isInvincible = true;
        StartCoroutine(InvincibilityFlash());
        StartCoroutine(HandleInvincibilityDelay());
    }
}
```

Avec cette méthode, lorsque le personnage va prendre des dégâts, il va alors passer en statut invincible et cela va activer la coroutine. La coroutine est une méthode qui va pouvoir faire des pauses dans le code: faire une action puis attendre, puis refaire une action.

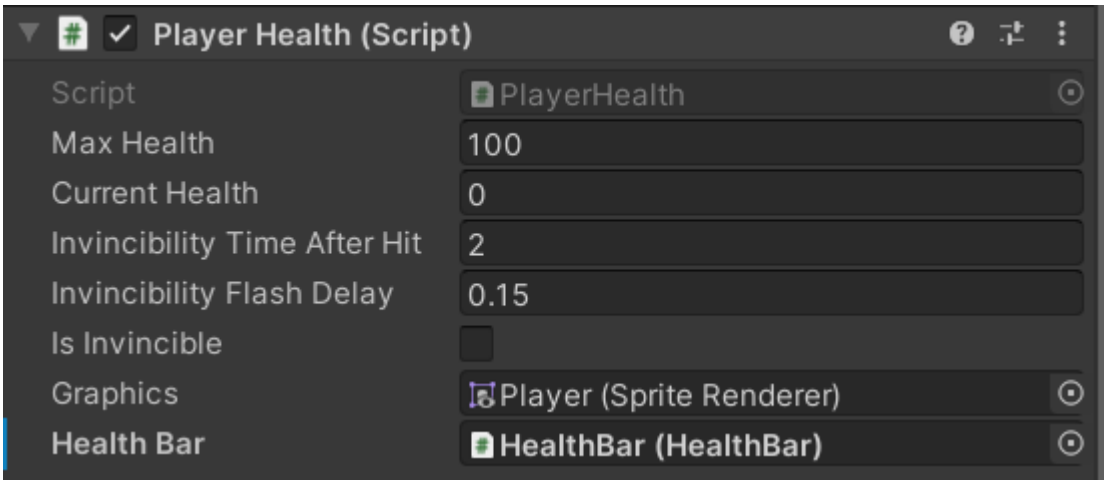
```
//coroutine pour invinciblilité du poisson
public IEnumerator InvincibilityFlash()
{
    while (isInvincible)
    {
        graphics.color = new Color(1f, 1f, 1f, 0f);
        yield return new WaitForSeconds(invincibilityFlashDelay); //utilisation de la coroutine pour les délais
        graphics.color = new Color(1f, 1f, 1f, 1f);
        yield return new WaitForSeconds(invincibilityFlashDelay);
    }
}
```

Nous faisons donc une première coroutine *InvincibilityFlash.* Ce clignotement va s'effectuer tant que le personnage est invincible d'où l'utilisation de la boucle "while".
graphics.color = new Color(1f,1f, 1f, 0f); va permettre de changer la couleur de notre poisson, avec *0f* qui correspond à la couche alpha: la transparence. Le fait de mettre 0 va rendre le poisson totalement transparent. Pour réafficher le poisson, nous utilisons alors *graphics.color = new Color(1f,1f, 1f, 1f);* qui va remettre le poisson opaque comme on a *1f*.
Cependant, si on met seulement ces deux lignes, elles vont alors se lire pratiquement en même temps et on ne verra donc rien. Il faut donc rajouter un délai entre ces lignes. D'où

yield return new WaitForSeconds(invincibilityFlashDelay); Nous utilisons *yield* qui est une classe de unity souvent utilisé avec les coroutine et qui sert pour les délais. *WaitForSeconds* est le temps que nous allons attendre, et nous avons déjà mis une valeur à la variable *invincibilityFlashDelay* de 0.2f. On va donc désactiver le visuel, attendre, puis réactiver le visuel, puis ré-attendre et ainsi de suite. Le problème est que le personnage est maintenant invincible tout le temps à partir du moment où il prend des dégâts. Il faut donc rajouter un système de Timer pour désactiver l'invincibilité au bout de quelques secondes.

```
//coroutine pour durée d'invincibilité
public IEnumerator HandleInvincibilityDelay()
{
    yield return new WaitForSeconds(invincibilityTimeAfterHit);
    isInvincible = false;
}
```

On crée donc une deuxième coroutine pour gérer ce système de Timer, *HandleInvincibilityDelay.* Nous allons donc attendre un certain temps avec *yield return new WaitForSeconds(invincibilityTimeAfterHit);* avec l'utilisation de la variable *invincibilityTimeAfterHit* dont nous avons déjà mis une valeur de 2f ce qui correspond à 2 secondes d'invincibilité. Puis une fois ces 2 secondes passées, *isInvincible* redevient "false". Le personnage peut donc de nouveau reprendre des dégâts.



Les variables apparaissent donc dans l'Inspector de notre poisson comme elles sont en "public" et on peut donc modifier leurs valeurs directement si jamais on s'aperçoit qu'il y a un trop long délai par exemple. Nous avons donc baissé le délai entre chaque flash à 0.15f pour qu'ils clignotent plus vite. Notre Script fonctionne donc bien et notre personne obtient donc une invincibilité avec un clignotement de 2 secondes quand il se fait toucher.