2ème Rapport de Soutenance

Journey Towards Dawn

Home Studio

Charles Delahousse

Maxence Jenn

Lino Develotte

Matthias Couturier

Benjamin Dubois

Table des matières

1 Introduction

Au travers de ce rapport de soutenance, nous, l'équipe Home Studio, vous feront part de nos réalisations conceptuelles et techniques afin d'aboutir notre projet de jeu vidéo Journey Towards Dawn.

De plus, nous vous exposerons les différents problèmes rencontrés par les membres de l'équipe, les solutions mises en place ainsi que les futures implémentation que nous voulons faire au sein du jeu.

2 Généralités

2.1 Rappel du Projet

Le monde humain stagne. Les ressources s'affaiblissent et le monde scientifique peine à faire des découvertes révolutionnaires. Mais un jour, des scientifiques découvrent la présence d'un artefact au fin fond de l'espace. Cet artefact est un mystère et beaucoup théorise sur son origine. Certains chercheurs prétendent que cet objet pourrait être la clé pour sauver l'humanité et le surnomment "Dawn Peak", traduit en "Pic de L'Aube".

C'est pourquoi une expédition spatiale est maintenue pour retrouver cet artefact. Le joueur se verra incarner une IA, du nom de "Wanderer", installée dans le vaisseau qui amènera une équipe de quatre scientifiques vers l'objet mystérieux.

Journey Towards Dawn est donc un jeu de type survie et gestion tour par tour. En effet, après avoir consolidé une équipe de quatre astronautes, le joueur se voit gérer un vaisseau consisté de 6 salles qui abrite son équipe. Des aléas vont menacer son vaisseau et son équipage. Il en est au joueur de prendre les précautions et actions nécessaires pour faire face à ces menaces.

2.2 Répartition des tâches

	Matthias	Charles	Benjamin	Lino	Maxence
Aléas		R	S		
Actions Générales		S	R		
Buff et Débuff		R		S	
Menu	S				R
Santé des Personnages			R		S
Sprite Vaisseau	R	S			
Sprites Personnages	S	R			
Arrière Plan	R			S	
HUD	R	S			
Musiques			S		R
Effets Sonores				R	S
Site Web			S	R	
Intelligence Artificielle	S				R
Multijoueur Actions			R	S	
Multijoueur Connectivité				R	S

2.3 Etat d'avancement du Projet

Lors de la dernière soutenance nous avions beaucoup de retard sur la jouabilité du jeu car nous nous étions trop concentré sur la partie graphique du jeu notamment le Menu. C'est pourquoi, depuis la dernière soutenance, nous nous sommes concentrés exclusivement sur la jouabilité du jeu et nous avons fait un grand pas dans la réalisation de ce projet.

2.4 Outils utilisés

Pour la réalisation de notre projet, nous avons mis en place plusieurs moyens de collaboration et de suivi.

Tout d'abord, nous avons utilisé GitHub afin de travailler en collaboration sur les mêmes fichiers et de résoudre les conflits éventuels. Nous avons également utilisé Discord pour communiquer, échanger des idées et travailler. C'est aussi sur cette plateforme que nous mettons tous les avancements, les problèmes rencontrés et les choses à faire.

3 Menu

3.1 Ajouts de fonctionnalités manquantes

Nous avons ajouté deux sections supplémentaires au menu principal du jeu, nécessaires pour accéder à certaines de ses fonctionnalités.

Nous avions deux approches possibles pour la gestion du menu : soit changer de scène à chaque changement entre ses sections, soit regrouper l'ensemble des sections dans une seule scène et activer/désactiver les GameObjects en fonction des besoins. Pour une meilleure lisibilité et une navigation plus fluide lors du développement, nous avons choisi la seconde option.

De plus, lors de la première soutenance, nous n'avions pas implémenté certaines fonctionnalités dans le Menu. Donc pour ce qui est du Menu, nous avons rajouté quelques fonctionnalités qui n'étaient pas implémenté lors de la première soutenance :

— La case à cocher "Plein Ecran" dans le menu des Paramètres fonctionne en appelant une fonction qui met le jeu plein écran ou non.
— Le bouton "Documentation" dans le Menu Principal est implémenté et ouvre le site web de notre jeu.

Pour la prochaine soutenance, nous devons encore implémenter le menu dans la scène du jeu avec les différents choix (Reprendre la partie, Paramètres, Quitter la partie, ...)

3.2 Logo de Démarrage

Lors de l'ouverture du jeu, avant d'arriver au menu principal, une scène d'introduction s'affiche. Cette scène présente le logo de notre entreprise, Home Studio, qui apparaît puis disparaît grâce à un script Unity. Cette brève apparition permet de mettre en avant l'entreprise responsable du jeu Journey Towards Dawn.

L'implémentation de cette scène a été initialement difficile. En effet, Unity utilise des syntaxes spécifiques que nous ne maîtrisions pas. Cependant, après nous être renseignés sur divers forums et tutoriels vidéo, nous avons découvert et compris les manipulations possibles à l'aide des collections UnityEngine, UnityEngine.UI et UnityEngine.SceneManagement.

Cette scène n'est pas encore finalisée. Nous souhaitons ajouter ultérieurement le nom de notre entreprise sous le logo afin de renforcer l'identité de Home Studio.

4 Développement du Jeu

4.1 Classe Player

Un objet de type Player contient 2 attributs pour l'instant :

— La liste des personnages que le joueur a sélectionnée
— La liste de toutes les salles du vaisseau

Cette classe nous permettra de faciliter l'implémentation des différents script ou fonctionnalités notamment le multijoueur.

4.2 Sélection des Personnages

Après avoir choisi la difficulté du jeu, le joueur accède à la sélection des personnages qu'il devra guider lors de la mission vers le Pic de l'Aube. Le joueur doit choisir un total de quatre personnages parmi une liste de huit astronautes, chacun possédant des statistiques spécifiques.

Cette scène est composée de plusieurs boutons interactifs et de textes qui s'adaptent aux interactions du joueur.

Grâce aux boutons "Suivant" et "Précédent", le joueur peut parcourir la liste des astronautes disponibles. Chaque astronaute est identifié par un nom et un sprite uniques, accompagnés de ses statistiques spécifiques :

— Santé Physique
— Santé Mental
— Faim
— Points de Réparation

Une liste initialement vide est prévue pour enregistrer les personnages sélectionnés. Cette liste constitue l'équipe principale du joueur. Un compteur affiche le nombre de personnages sélectionnés.

Lorsqu'un personnage est ajouté, le compteur augmente d'une unité, le bouton "Sélectionner" est remplacé par "Retirer", et un texte de confirmation s'affiche en haut à gauche du sprite du

personnage. Si le joueur choisit de retirer un personnage, celui-ci est supprimé de la liste, le compteur diminue d'une unité, le texte de confirmation disparaît et le bouton "Sélectionner" réapparaît.

Une fois que quatre personnages sont sélectionnés, le bouton "Jouer" devient actif. En cliquant dessus, la liste des personnages choisis est enregistrée, et le joueur est redirigé vers la scène principale du jeu où son équipage devra faire face aux dangers de l'espace.

La sélection des personnages n'est pas encore complètement finalisée. Nous souhaitons améliorer l'accessibilité des informations en ajoutant une représentation visuelle des personnages sélectionnés.

4.3 Aléas

Les dangers de l'espace prennent la forme d'événements aléatoires. À chaque fin de tour, l'équipage peut être confronté à un événement positif ou négatif. Nous avons créé une énumération de ces événements afin d'y accéder facilement :

— Débris
— Météroïde
— Fuite d'oxygène
— Rumeur négative
— Rumeur positive
— Planète en forme de poulet rôti
— Sonde Spatiale
— Base Spatiale
— Lola

```
public void Easy() //Liste D'aléas Faciles Et Condition Victoire
{
    DifficultyChosen = Difficulty.Easy;

    ListEvents = new List<(Events, double)>() //Les événements Avec Leurs Probabilités D'apparition
    {
        (Events.Debris, 20),
        (Events.Meteroid, 0),
        (Events.Oxygen_Leak, 20),
        (Events.Rumor_Positive, 0),
        (Events.Rumor_Negative, 0),
        (Events.Planet_Chicken_Form, 0),
    };

    EndCampaign = 25;

}
```

FIGURE 1 – Exemple de fonction exécutée

De plus, les effets des aléas sont influencés par la difficulté choisie par le joueur.

Lorsque le joueur sélectionne une difficulté, des fonctions appropriées sont exécutées. Elles initialisent la difficulté, le nombre de tours à survivre ainsi qu'une liste d'événements représentée par des paires (type d'événement, probabilité).

Lorsque le joueur appuie sur le bouton End Day, un compteur de tours augmente. Si ce compteur atteint la valeur EndCampaign, le joueur gagne la partie. Sinon, une condition de défaite est vérifiée :

— Tous les personnages du joueur ont leur santé physique à 0.

— Toutes les salles du vaisseau ont leur durabilité à 0.

Pour faire apparaître les évènements, nous avons implémenté une fonction TryExecuteEvent

```
public void TryExecuteEvent(List<(Events, double)> Events, Player player)
{
    ListEvents.Sort( comparison: (a :(Events,double) , b :(Events,double) ) => b.Item2.CompareTo(a.Item2)); // T

    int probability = Random.Range(1, 101);

    if (ListEvents[0].Item2 >= probability)
    {
        switch (DifficultyChosen) //  Augmente Le % de chance d'apparition de tous les events sa
        {
            case Difficulty.Easy:
                EasyApplyEffects(ListEvents[0].Item1, player);
                ListEvents[0] = (ListEvents[0].Item1, 0);
                IncreasePercentage(ListEvents, start: 1);
                return;
            case Difficulty.Normal:
                NormalApplyEffects(Events[0].Item1, player);
                Events[0] = (Events[0].Item1, 0);
                IncreasePercentage(Events, start: 1);
                return;
            case Difficulty.Hard:
                HardApplyEffects(Events[0].Item1, player);
                Events[0] = (Events[0].Item1, 0);
                IncreasePercentage(Events, start: 1);
                return;
        }
    }
    else
    {
        IncreasePercentage(Events, start: 0); //  Augmente Le % de chance d'apparition de TOUS les
    }
}
```

Cette fonction va tout d'abord trier la liste des évènements dans l'ordre décroissant de leur chance d'apparition. Puis elle génère un nombre aléatoire entre 0 et 100 pour simuler un pourcentage. Ensuite elle regarde si cette valeur est inférieur à la valeur contenue dans

le couple. Si c'est le cas, alors elle appelle une fonction qui va exécuter l'évènement selon la difficulté choisie.

```
public void EasyApplyEffects(Events Alea, Player player) //Effets Aléas Difficulté Facile
{
    switch (Alea)
    {
        case Events.Debris:
            foreach (var Perso in player.CharacterList)
            {
                Perso.Health -= 10;
            }
            foreach (var Salle in player.RoomList)
            {
                Salle.Health -= 10;
            }
            break;
```

```
public void HardApplyEffects(Events Alea, Player player) //Effets Aléas Difficulté Difficile
{
    switch (Alea)
    {
        case Events.Debris:
            foreach (var Perso in player.CharacterList)
            {
                Perso.Health -= 20;
            }
            foreach (var Salle in player.RoomList)
            {
                Salle.Health -= 20;
            }
            break;
```

Si un évènement n'a pas été exécuté, cette fonction appelle une autre fonction IncreasePercentage() afin d'augmenter progressivement les chances d'apparition des événements. Autrement, la chance d'apparition de l'évènement exécuté est remise à 0 et seuls les chances d'apparition des évènements non exécutés seront augmentées.

```
public void IncreasePercentage(List<(Events, double)> events, int start) // Fonction pour augmenter les % des events
{

    double IA = ArtificialIntelligence();
    double progress = 1 + DayCount * 0.01;

    for (int i = start; i < events.Count; i++)
    {
        switch (events[i].Item1)
        {
            case Events.Debris:
                events[i] = (events[i].Item1, events[i].Item2 * progress + IA + 10);
                break;
            case Events.Meteroid:
                events[i] = (events[i].Item1, events[i].Item2 * progress + IA + 5);
                break;
            case Events.Oxygen_Leak:
                events[i] = (events[i].Item1, events[i].Item2 * progress + IA + 10);
                break;
            case Events.Rumor_Positive:
                events[i] = (events[i].Item1, events[i].Item2 * progress + IA + 5);
                break;
            case Events.Rumor_Negative:
                events[i] = (events[i].Item1, events[i].Item2 * progress + IA + 5);
                break;
            case Events.Planet_Chicken_Form:
                events[i] = (events[i].Item1, events[i].Item2 * progress + IA + 5);
                break;
        }
    }
}
```

Cette augmentation suivante se calcule avec la formule suivante :

$$valeur\ de\ base * taux\ de\ progression + IA + pourcentage\ de\ base$$

La valeur de base correspond à la valeur initialement contenue dans le couple.

Le taux de progression représente l'avancement du joueur dans la partie. Par exemple si le joueur est au tour 25, le taux de progression vaudra 1,25.

Le nombre IA correspond au nombre calculé par l'intelligence artificielle de notre jeu qui varie selon la situation du joueur.

Le pourcentage de base correspond à une valeur fixe donné à chaque évènement et qui est plus ou moins grande selon la difficulté de l'évènement.

Ainsi, à chaque fin de tour, les événements deviennent plus fréquents, ce qui rend chaque partie unique et incite le joueur à adapter sa stratégie.

Nous avons rencontré un problème dans l'initialisation de la liste des événements et de la difficulté : les exécutions n'étaient pas correctement enregistrées. Après analyse, nous

avons identifié l'erreur et avons trouvé une solution permettant non seulement de la corriger, mais aussi de regrouper trois fonctions en une seule afin de rendre notre code plus lisible et optimisé.

4.4 Intelligence Artificielle

Dans notre jeu, l'intelligence artificielle s'occupe des chances d'apparition des évènements. Pour ce faire, Elle analyse toutes les statistiques des personnages choisis par les joueurs (les points de vie, la faim et les points de santé mentale) ainsi que les points de vie de toutes les salles pour pouvoir augmenter les chances d'apparition des évènements en fonction de la situation du joueur. Chaque statistique possède une plage de valeurs divisé en quelques intervalles. Selon l'intervalle dans lequel se trouve la somme des valeurs de la statistique, cela va augmenter plus ou moins la valeur IA établie précedemment.

```
// Analyse des HP totaux des personnages

if (CharactersHP >= 100 && CharactersHP < 200)
{
    IA += 1;
}
else if (CharactersHP >= 200 && CharactersHP < 300)
{
    IA += 1.75;
}
else if (CharactersHP >= 300 && CharactersHP <= MaxCharactersHP)
{
    IA += 2.5;
}
```

FIGURE 2 – Exemple d'analyse de statistique

4.5 Buff et Débuff

Un personnage peut être affecté par des effets temporaires appelés buffs ou debuffs, en fonction de leur impact. Ces effets sont également représentés sous forme d'une énumération :

— Inspiré

— Anxieux

— Gourmand

— Affamé

Un personnage peut cumuler plusieurs buffs et debuffs simultanément grâce à une liste paires (type d'effet, durée). Ces effets sont attribués via certains événements aléatoires. Cependant, un personnage ne peut pas recevoir deux fois le même effet consécutivement.

```
case Events.Rumor_Positive:
    int NumberCharactersEffected = Random.Range(1, 5);
    if (NumberCharactersEffected == 4)
    {
        foreach (var Perso in player.CharacterList)
        {
            Perso.Buff.Add((Buffs.Inspired, 3));
        }
    }
    else
    {
        while (NumberCharactersEffected > 0)
        {
            int LuckyAstronaut = Random.Range(0, 4);
            if (!player.CharacterList[LuckyAstronaut].Buff.Contains((Buffs.Inspired, 3)))
            {
                player.CharacterList[LuckyAstronaut].Buff.Add((Buffs.Inspired, 3));
                NumberCharactersEffected--;
            }
        }
    }
    break;
```

Nous avons pris du retard sur l'implémentation des effets, mais nous prévoyons de les gérer à l'aide d'une fonction qui parcourra la liste des buffs et debuffs de chaque personnage. Cette fonction appliquera les effets en fonction de la difficulté et du type de buff, tout en diminuant la durée restante de l'effet. Lorsque cette durée atteint 0, l'effet sera supprimé du personnage.

4.6 Graphismes et HUD

Les Personnages

Chaque personnage possède quatre sprites distincts :

— Un pour l'écran de sélection en début de partie

— Un pour sa représentation dans le vaisseau

— Un pour afficher son portrait dans le HUD

— Un pour la surbrillance lorsqu'il est sélectionné

Une question s'est notamment posée quant à la vue à adopter pour les personnages dans le vaisseau : de dessus ou de face. Afin de mieux les distinguer, nous avons opté pour une vue de face. Nous avons d'ailleurs dû nous y reprendre à deux fois pour créer celui dans le vaisseau, initialement légèrement trop grand.

FIGURE 3 – Avant

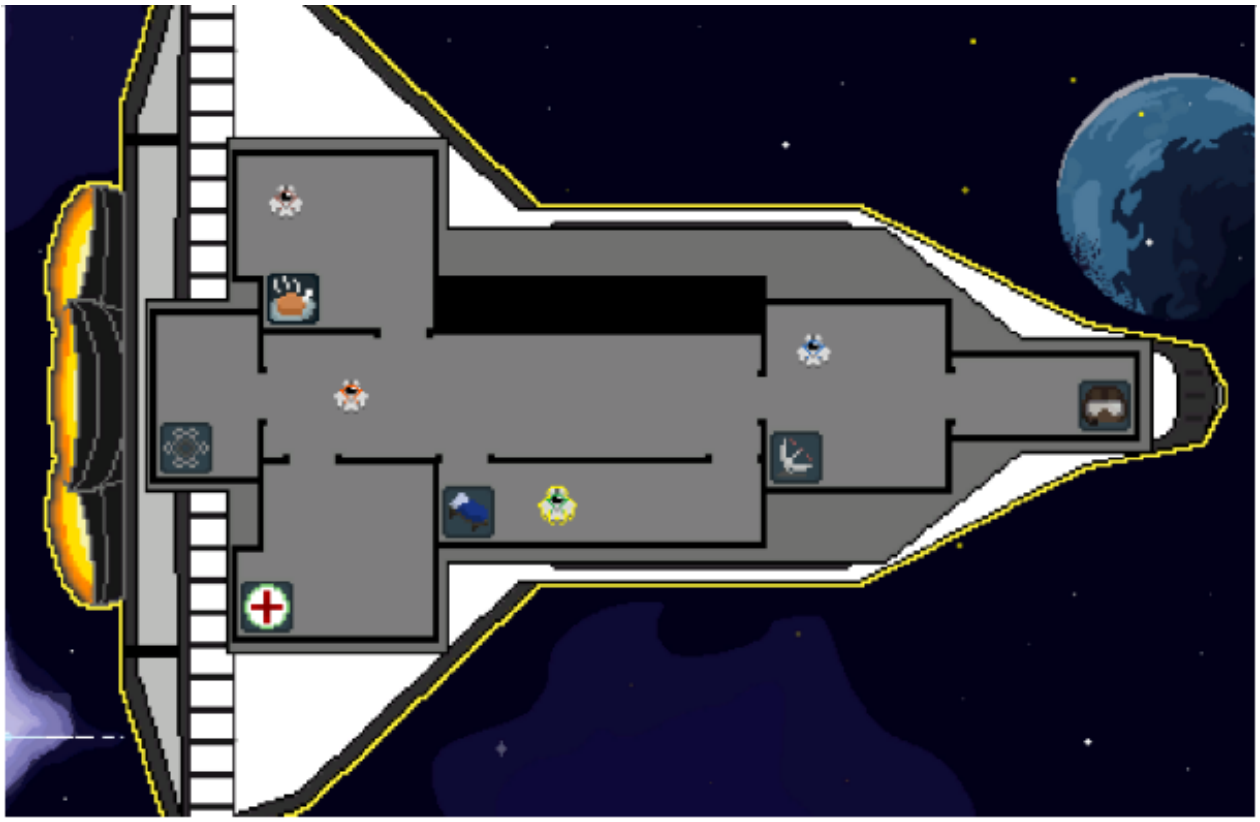
FIGURE 4 – Après

Le sprite de l'écran de sélection n'est pas définitif, mais ne constituant pas une priorité, nous l'utilisons tel quel pour le moment.

Enfin, nous avions initialement prévu d'animer les déplacements des personnages dans le vaisseau avec plusieurs sprites. Faute de temps, cette idée a été mise de côté, et nous avons opté pour un système de glisser-déposer à la place.

Le vaisseau

Pour le vaisseau, nous avons d'abord réalisé que sa taille était trop petite et pas assez lisible, nous l'avons donc grossis et assombrit la couleur des salles permettant de rendre le sprite des personnages visibles. De plus des modifications sur le sprites des salles du vaisseau ont été effectué permettant de les différencier et comprendre le rôle de chacune.



Chaque salle dispose désormais de son propre sprite, distinct de celui du vaisseau. Cette distinction facilite l'intégration de l'interface utilisateur, rendant son implémentation plus fluide et efficace.

HUD

Nous avons d'abord constaté que le HUD occupait trop d'espace à l'écran. Sa taille a donc été réduite et son apparence ajustée en conséquence pour améliorer sa lisibilité.

Nous avons également ajouté les informations relatives au personnage, à savoir sa barre de santé physique, mentale et de faim. Il était initialement prévu d'utiliser des formes différentes du rectangle classique pour représenter ces barres de statistiques. Cependant, nous avons rencontré des difficultés techniques : la jauge de remplissage se déformait lorsque sa valeur était modifiée. Ne souhaitant pas investir trop de temps sur cet aspect, nous sommes finalement revenus à une forme rectangulaire.

Enfin, un bouton permettant de mettre fin à la journée en cours a été ajouté (il sera modifié plus tard pour mieux s'intégrer au design du HUD).

Tous ces éléments sont organisés en plusieurs couches de sprites afin de faciliter l'implémentation de l'UI.

UI

Une fois les personnages sélectionnés et la partie lancée, l'utilisateur doit pouvoir les sélectionner en cliquant dessus.

Dans un premier temps, nous avons attribué à chaque personnages différentes valeurs dont les plus importantes sont :

— Son nom
— Ses statistiques
— Ses différents sprites

Ensuite, il a été nécessaire de créer un système de détection d'interaction pour détecter sur quel objet l'utilisateur effectue son clic. Cette fonctionnalité, au-delà de la sélection des personnages, servira à d'autres aspects du jeu. Pour cela, nous avons ajouté au Canvas, possédant l'intégralité des éléments graphiques du jeu, un script captant tous les clics de l'utilisateur. Après quoi, nous avons attribué à ce script un événement auquel les objets nécessitant l'information « quel élément a été cliqué » peuvent s'abonner.

Lorsqu'un clic est effectué sur un personnage dans le vaisseau, son apparence se change pour lui attribuer son sprite en surbrillance et son portrait s'affiche dans le HUD (voir Graphique – Personnages). Ses statistiques sont alors récupérées pour mettre à jour les trois jauges du HUD en conséquence.

Il a été difficile de relier les informations relatives à un personnage lorsqu'on le sélectionne dans l'écran de sélection à celui dans le jeu principal car ses informations sont stockées sur un script dépourvu de la classe MonoBehavior de Unity. En voulant lui rajouter cette classe, de nombreuses erreurs ont été engendré ce qui a rendu la tâche bien plus longue que prévu.

Par ailleurs, nous avons créé une liste de tous les éléments du jeu qui lorsqu'ils sont cliqués, permettent de conserver la sélection sur le personnage active.

Enfin, le bouton « End Day » (voir Graphique – HUD) déclenche un fondu au noir via un Animator. Durant ce fondu, nous voulions bloquer tout autres clics qui s'effectueraient, par exemple pour éviter de cliquer en boucle sur le bouton ce qui engendrerait des fondus au noir en chaîne. Pour cela, nous avons créé un objet invisible recouvrant tous l'écran qui ne s'active uniquement lorsque l'utilisateur clic sur le bouton « End Day ». Cet objet intercepte tous les clics, bloquant les interactions habituelles avec les autres éléments du jeu.

4.7 Drag and Drop

Dans le développement de Journey Towards Dawn, nous avons d'abord envisagé un système permettant aux joueurs d'assigner les astronautes aux salles du vaisseau à travers une simple sélection. L'idée initiale était de permettre au joueur de cliquer sur une salle, puis sur un astronaute, afin de l'y affecter automatiquement. Ce choix se basait sur une logique de gestion simplifiée et s'appuyait sur l'interaction classique d'un jeu de stratégie où les unités sont déplacées par des commandes directes plutôt que par un glissement manuel.

Cependant, au fil des tests, cette approche s'est révélée peu intuitive et trop rigide. L'interaction manquait de fluidité et obligeait le joueur à effectuer plusieurs actions successives sans véritable feedback visuel immédiat. Nous avons alors opté pour une approche plus naturelle en implémentant un système de drag and drop. Ce dernier permet au joueur de glisser directement les astronautes d'un endroit à un autre, offrant une meilleure ergonomie et un meilleur ressenti d'interactivité.

Le système repose sur deux scripts principaux : DragDrop, qui gère le comportement des astronautes en tant qu'objets déplaçables, et DropZone, qui régule la réception et l'assignation des astronautes aux emplacements dédiés dans chaque salle. L'objectif est de permettre le déplacement libre des astronautes tout en garantissant qu'un slot ne puisse contenir qu'un seul astronaute à la fois.

L'objectif du script DragDrop est de permettre à un astronaute d'être déplacé librement à l'aide de la souris et de s'intégrer correctement à une salle lorsqu'il est relâché. Ce script

repose sur les interfaces IBeginDragHandler, IDragHandler et IEndDragHandler, fournies par Unity, qui permettent de détecter respectivement le début du glissement, le déplacement et le relâchement de l'objet.

Dès le lancement du jeu, chaque astronaute enregistre sa position d'origine et son parent dans la hiérarchie. Cette information est cruciale car elle permet de replacer l'astronaute à sa position initiale s'il n'est pas déposé dans une salle valide.

Le script commence par récupérer plusieurs composants essentiels dans la méthode Start().

```
Event function  56
private void Start()
{
    originalPosition = transform.position;
    originalParent = transform.parent;
    rectTransform = GetComponent<RectTransform>();
    canvas = GetComponentInParent<Canvas>();
    canvasGroup = GetComponent<CanvasGroup>();

    if (canvasGroup == null)
    {
        canvasGroup = gameObject.AddComponent<CanvasGroup>();
    }
}
```

Ici, transform.position enregistre la position initiale de l'astronaute dans l'espace du jeu, tandis que transform.parent sauvegarde son parent hiérarchique. RectTransform est utilisé spécifiquement pour la gestion des éléments UI et permet de manipuler leur position de manière plus précise. Enfin, CanvasGroup est récupéré ou ajouté dynamiquement, ce qui est essentiel pour gérer la visibilité et l'interactivité de l'astronaute pendant son déplacement. Lorsque le joueur commence à déplacer un astronaute, la méthode OnBeginDrag() est appelée.

```
public void OnBeginDrag(PointerEventData eventData)
{
    droppedInRoom = false;

    if (currentDropRoom != null)
    {
        currentDropRoom.RemoveObject(gameObject);
        currentDropRoom = null;
    }

    transform.SetAsLastSibling();
    canvasGroup.blocksRaycasts = false;
}
```

À ce stade, l'astronaute est retiré de la salle dans laquelle il était préceédemment assigné. SetAsLastSibling() assure que l'objet déplacé reste au premier plan, évitant tout problème d'affichage derrière d'autres éléments de l'interface. En désactivant blockRaycasts, on empêche également l'astronaute d'interférer avec la détection des zones de dépôt, garantissant ainsi une meilleure fluidité.

Pendant le déplacement, OnDrag() met à jour la position de l'astronaute en fonction des mouvements de la souris.

```
public void OnDrag(PointerEventData eventData)
{
    rectTransform.anchoredPosition += eventData.delta / canvas.scaleFactor;
}
```

L'utilisation de $eventData.delta$ / $canvas.scaleFactor$ permet d'ajuster le mouvement en fonction du facteur d'échelle du canvas. Sans cette division, l'objet pourrait être déplacé de

manière trop rapide ou trop lente selon la résolution ou le zoom du canvas.

Enfin, lorsque le joueur relâche l'astronaute, OnEndDrag() est appelé pour vérifier si l'objet a bien été déposé dans une salle.

```
public void OnEndDrag(PointerEventData eventData)
{
    canvasGroup.blocksRaycasts = true;

    if (!droppedInRoom)
    {
        ResetPosition();
    }
}

2 usages  56
public void ResetPosition()
{
    transform.position = originalPosition;
    transform.SetParent(originalParent);
}
```

Si aucun slot valide n'a été détecté, l'astronaute est simplement replacé à sa position initiale grâce à ResetPosition(), ce qui empêche les erreurs de placement.

Le script DropZone est responsable de la gestion des salles et de la validation des emplacements disponibles pour chaque astronaute. Chaque salle contient plusieurs slots, qui sont des positions fixes où les astronautes peuvent être assignés.

Lorsqu'un astronaute est relâché, OnDrop() est exécuté pour déterminer si un emplacement est disponible.

```
public void OnDrop(PointerEventData eventData)
{
    if (!isHovered) return;

    DragDrop draggedItem = eventData.pointerDrag.GetComponent<DragDrop>();
    if (draggedItem == null)
        return;

    if (draggedItem.currentDropRoom != null)
    {
        draggedItem.currentDropRoom.RemoveObject(draggedItem.gameObject);
    }

    int freeSlotIndex = -1;
    for (int i = 0; i < dropSlots.Count; i++)
    {
        if (!occupiedSlots.ContainsKey(i))
        {
            freeSlotIndex = i;
            break;
        }
    }

    if (freeSlotIndex == -1)
    {
        draggedItem.ResetPosition();
        return;
    }

    RectTransform dropZoneRect = GetComponent<RectTransform>();
    RectTransformUtility.ScreenPointToLocalPointInRectangle(
        dropZoneRect, screenPoint: eventData.position, eventData.pressEventCamera, out Vector2 localPoint
    );

    occupiedSlots.Add(freeSlotIndex, draggedItem.gameObject);
    draggedItem.droppedInRoom = true;
    draggedItem.currentDropRoom = this;

    draggedItem.transform.position = dropSlots[freeSlotIndex].position;
    draggedItem.UpdateInitialPosition(dropSlots[freeSlotIndex], dropZone: this);
}
```

Ce script commence par vérifier si la salle est bien survolée (isHovered). Ensuite, il cherche un emplacement libre en parcourant la liste des slots. Si aucun emplacement n'est disponible, l'astronaute est replacé à sa position initiale. Sinon, l'astronaute est affecté au slot trouvé, et sa position est mise à jour.

Ce système garantit que chaque salle ne peut accueillir qu'un nombre limité d'astronautes et empêche les placements incorrects.

Lors des premiers tests, nous avons remarqué qu'un slot devenait inutilisable après avoir été occupé une première fois. Même lorsqu'un astronaute était retiré de son emplacement, le slot était toujours marqué comme occupé, empêchant d'y placer un autre astronaute.

Ce problème venait du fait que notre gestion des slots ne mettait pas à jour leur état lorsqu'un astronaute était retiré. Nous avions bien une structure permettant de stocker quel astronaute était dans quel slot, mais nous ne la vidions pas correctement lorsqu'un astronaute était déplacé ailleurs. Nous avons corrigé ce problème en introduisant une méthode RemoveObject(GameObject obj) dans le script DropZone. Cette méthode permet de libérer un slot lorsqu'un astronaute quitte la salle.

```
public void RemoveObject(GameObject obj)
{
    int slotIndex = -1;
    foreach (var pair in occupiedSlots)
    {
        if (pair.Value == obj)
        {
            slotIndex = pair.Key;
            break;
        }
    }

    if (slotIndex != -1)
    {
        occupiedSlots.Remove(slotIndex);
    }
}
```

Avec cette modification, chaque fois qu'un astronaute quittait un slot, celui-ci était immédiatement rendu disponible pour un autre.

Un autre problème rencontré était la possibilité pour plusieurs astronautes de se superposer sur un même slot. Lorsqu'un astronaute était déplacé dans une salle, rien n'empêchait un autre astronaute d'être placé exactement au même endroit.

Ce problème venait du fait que notre gestion des slots ne vérifiait pas si un emplacement était déjà occupé avant d'y assigner un nouvel astronaute. Le dictionnaire occupiedSlots enregistrait quel astronaute était dans quel slot, mais il ne bloquait pas l'ajout d'un nouvel astronaute si un slot était déjà pris.

Nous avons ajouté une vérification avant d'assigner un astronaute à un slot. La méthode OnDrop() a été modifiée pour rechercher un slot libre avant de placer un nouvel astronaute. Si aucun slot n'est disponible, l'astronaute retourne à sa position initiale.

```
int freeSlotIndex = -1;
for (int i = 0; i < dropSlots.Count; i++)
{
    if (!occupiedSlots.ContainsKey(i))
    {
        freeSlotIndex = i;
        break;
    }
}

if (freeSlotIndex == -1)
{
    draggedItem.ResetPosition();
    return;
}
```

Enfin, nous avons remarqué qu'après avoir déplacé un astronaute, celui-ci restait au premier plan dans la hiérarchie des objets, ce qui le rendait visible même lorsqu'il était censé être masqué par d'autres éléments de l'interface.

Ce problème venait du fait que, lors du OnBeginDrag(), nous utilisions transform.SetAsLastSibling(), qui place l'élément en haut de la hiérarchie pour éviter qu'il soit caché pendant le déplacement. Cependant, nous ne remettions pas l'astronaute à sa place une fois le déplacement terminé.

Nous avons ajouté une ligne dans ResetPosition() pour rétablir le parent original.

```
public void ResetPosition()
{
    transform.position = originalPosition;
    transform.SetParent(originalParent);
}
```

Dans nos plans futurs, nous allons poursuivre l'amélioration du système en corrigeant les éventuels bugs restants, notamment ceux liés à la gestion des slots et aux interactions imprévues lors du déplacement des astronautes. Une attention particulière sera portée à l'optimisation du code afin de garantir un fonctionnement fluide et sans erreurs.

Par ailleurs, un affichage des événements en jeu sera implémenté. Ce système informera le joueur des actions effectuées, telles que l'assignation d'un astronaute à une salle ou l'impossibilité d'un déplacement. Cette fonctionnalité renforcera la clarté des interactions et améliorera l'expérience utilisateur.

Enfin, nous intégrerons des pop-ups tutoriels pour expliquer les mécaniques de jeu de manière générale. Ces tutoriels interactifs permettront aux nouveaux joueurs de mieux comprendre le fonctionnement du jeu, en couvrant non seulement le système de drag and drop, mais aussi d'autres aspects essentiels du gameplay.

4.8 Multijoueur

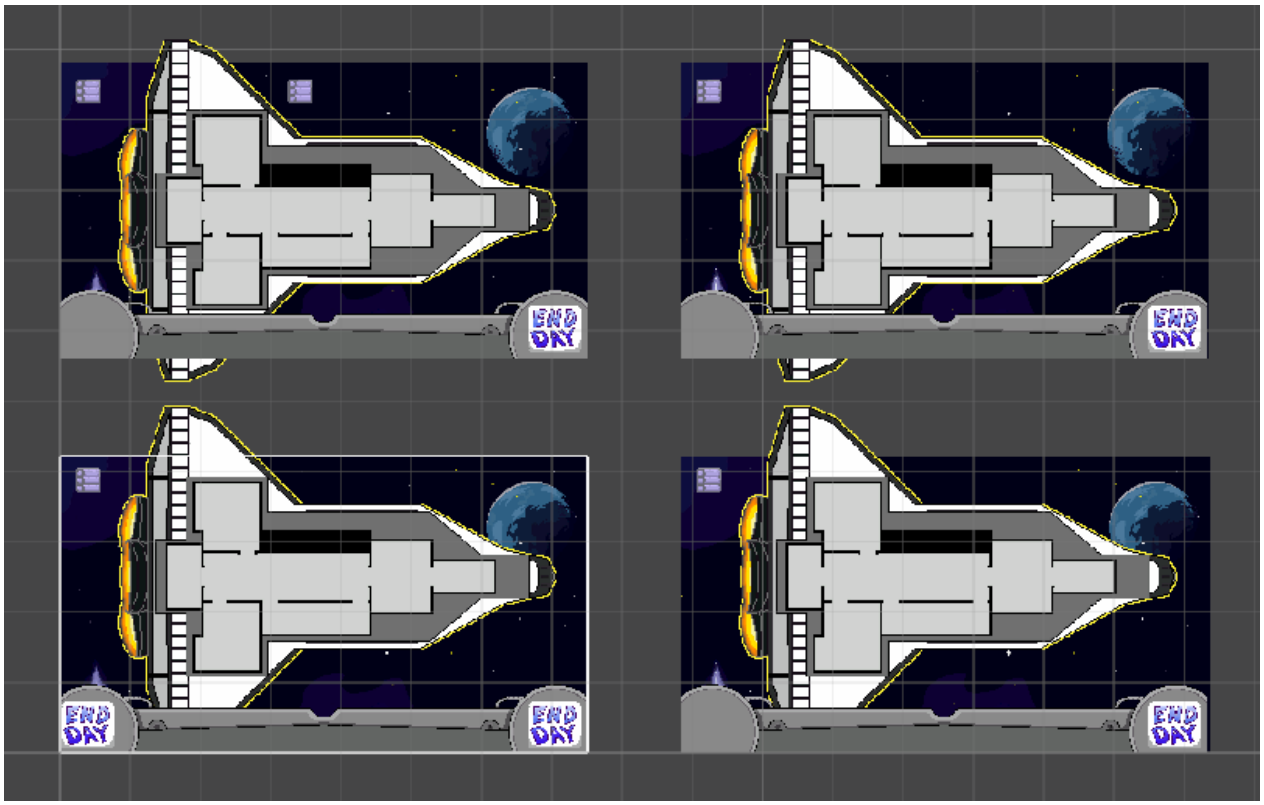
Pour instaurer le multijoueur, nous avons regarder des tutos sur YouTube, notamment celle de « TUTO UNITY FR » qui a opté pour l'utilisation d'un asset multijoueur pas présent de base sur Unity, Mirror. Ainsi, nous avons décidé de faire de même car l'asset nous paraissait d'apparence être plutôt simple d'utilisation et qu'il correspondait à nos critères pour le multijoueur.

Ainsi nous avons tout d'abord créé un « Network manager » qui est l'élément le plus important pour pouvoir joueur à plusieurs avec Mirror. Par la suite, nous avons créé une prefab d'un « game object » avec le component « Network Identity » et une caméra qui sera le joueur et un « game object » avec le component « Network Start Position » qui permettra au « Network Manager » de le reconnaitre comme un point d'apparition de la prefab joueur. Par la suite, nous avons paramétré le « Network Manager » en mettant par exemple le nombres de joueurs max pouvant rejoindre la partie à 4, en choisissant le mode de transport multijoueur (KCP) ou encore en remplissant le compartiment « Player Prefab » par notre joueur.

Finalement, nous avons créé une scène qui est le lobby dans lequel les joueurs choisiront de soit rejoindre un serveur déjà existant ou de créer un serveur à l'aide d'un HUD qui est un component de « Network Manager ».

Cependant lors de la réalisation du multijoueur nous avons rencontré de nombreux problèmes, principalement dû aux canva que nous avons utilisé pour les sprites du vaisseau ce qui nous a fait perdre énormément de temps sur l'avancement de celui-ci.

Par exemple, dans les canvas sur unity, il y a une section « event mode » et une section « render mode » où il faut insérer une caméra et choisir un mode de rendu pour celle-ci. Cependant lorsque qu'une caméra est insérée dans cette emplacement, cela rend impossible l'utilisation de toute autre caméra. Ainsi, lorsqu'un joueur apparaissait, l'écran devenait tout bleu. Malheureusement, nous n'avons pu découvrir la cause de ce problème (le canva) qu'après plusieurs jours de recherche.



Un autre problème que nous avons rencontré mais que nous n'avons toujours pas réglé est l'apparition des joueurs lorsqu'un utilisateur rejoins un serveur. Lorsqu'un joueur rejoint un serveur, il est censé apparaître sur l'un des quatre points d'apparitions. Cependant ils apparaissent tous au même endroit. Nous avons fait des recherches et pensons ainsi que comme le « Network Manager » n'est pas dans la même scène que les points d'apparitions, cela empêche le « Network manager » de les détecter. Nous pensons ainsi à mettre le lobby dans la scène du jeu pour permettre la résolution de ce problème.

5 Site Web

En ce qui concerne le site web, nous avons amélioré son design en remplaçant l'ancien fond beige qui n'était pas en accord avec le thème de notre jeu par un fond bleu foncé qui tend vers le bleu plus clair (dégradé) rappelant ainsi le thème de l'espace. De plus, nous avons complètement remplacé l'ancienne page de présentation des membres en rajoutant notamment le synopsis de notre jeu et les biographies des membres du groupe.

Sur la page Installation, nous avons également changé le bouton d'installation et implanté un carrousel faisant défiler différentes images de notre jeu, donnant ainsi un premier aperçu de notre jeu tout en leur donnant envie de l'installer.

6 Conclusion

Pour conclure, nous avons bien avancé sur le développement du gameplay avec notamment le Drag and Drop et le commencement de l'intelligence artificiel mais également sur le multijoueur. De plus, notre menu, nos graphismes ainsi que notre site web peuvent être considérés comme presque finis avec seulement quelques retouches à faire à certains endroits.

Enfin, Ils nous restent maintenant à faire les sons et musiques du jeu, ainsi qu'à finir le gameplay, pour le mode solo et le mode multijoueur.