

Rapport de soutenance

EPITA- Projet S2
HasBrothers

Mars 2023

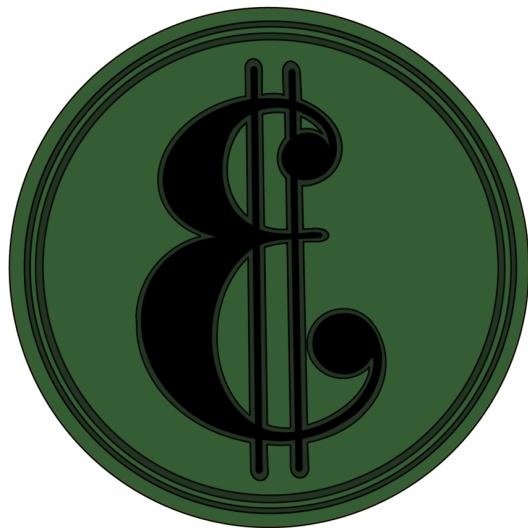


Table des matières

1	Avancements globaux	3
2	Le multijoueur et skin	3
2.1	Le multijoueur	3
2.2	Skin	5
3	Implémentation du dé et déplacement des joueurs	7
4	Menu pause et création des scènes	11
4.1	Menu Pause	11
4.2	Création des scènes	12
5	Passage des scènes	13
6	Gameplay en c# et implémentations dans Unity	15
6.1	Le plateau	15
6.2	Le Gameplay en c#	16
6.3	Implémentation du code dans Unity	17
7	le site	18
8	Conclusion	19
8.1	Nos ressentis	20
8.2	Vers la soutenance finale	20

1 Avancements globaux

Après la première soutenance nous avons tous continué à notre rythme le projet. Si dans un premier temps nous nous sommes essentiellement concentré sur le nouveau tableau des tâches que nous avions élaboré à la fin de la première soutenance, nous avons dû malgré faire preuve de flexibilité et travaillé main dans la main pour résoudre les différents problèmes que l'on a peu rencontré durant le développement de notre jeu.

Néanmoins malgré cela nous sommes fier de pouvoir dire que nous sommes raisonnablement dans les temps par rapport aux différents tâches que nous souhaitions réaliser pour cette deadline.

Wali	Yassine	Anna	Juliette
<ul style="list-style-type: none">— Le multijoueur— implémentation du dé— déplacement des joueurs sur le plateau	<ul style="list-style-type: none">— Le multijoueur— skin— passage des scènes	<ul style="list-style-type: none">— Menu pause— Création des scènes liées aux cases du plateau— passage des scènes	<ul style="list-style-type: none">— Le plateau— GamePlay en c— version 2ieme soutenance du site— implémentation du GamePlay dans Unity

légende :

- Ce qui est fait
- Ce qui sera continuer pour la prochaine soutenance

2 Le multijoueur et skin

2.1 Le multijoueur

Durant la création du projet, Wali et Yassine ont travaillé ensemble sur la mise en place du mode multijoueur en utilisant Photon dans le jeu Unity. Cela a été une expérience enrichissante pour nous, car la mise en place de ce mode peut être un défi technique. Nous avons été ravis de pouvoir mettre nos compétences en développement de jeux à l'épreuve.

Le mode multijoueur est important pour les jeux modernes, car cela permet de jouer en ligne avec d'autres joueurs, apportant ainsi une dimension sociale et compétitive à l'expérience de jeu. En tant que joueur nous-mêmes, nous savons que la mise en place d'un mode multijoueur bien conçu peut faire la différence entre un jeu ordinaire et un jeu extraordinaire.

Nous avons utilisé plusieurs fonctions du code pour gérer la connexion et la création de salles de jeu en utilisant Photon. Par exemple, nous avons utilisé la fonction PhotonNetwork.ConnectUsingSettings pour nous connecter au serveur principal de Photon et la fonction PhotonNetwork.JoinLobby pour rejoindre le lobby où les joueurs peuvent créer ou rejoindre des salles de jeu :

```
private void Awake()
{
    AppSettings appSettings = new AppSettings();
    appSettings.ApidRealtime = "e37002a2-eb64-437e-900c-a5098ae60690";
    appSettings.FixedRegion = "eu";
    PhotonNetwork.ConnectUsingSettings(appSettings);
}

// Event function & lachettebro
private void Start()
{
    UsernameMenu.SetActive(true);
}

// Frequently called (0+1 usages) & lachettebro
public override void OnConnectedToMaster()
{
    PhotonNetwork.JoinLobby(TypedLobby.Default);
    Debug.Log(message: "Connected");
}
```

Nous avons également utilisé les fonctions PhotonNetwork.CreateRoom et PhotonNetwork.JoinOrCreateRoom pour créer et rejoindre des salles de jeu, respectivement :

```
/* Lasset usage: & lachettebro */
public void CreateGame()
{
    if (PhotonNetwork.IsConnectedAndReady)
    {
        PhotonNetwork.CreateRoom(CreateGameInput.text, new RoomOptions() { MaxPlayers = 5 }, TypedLobby.Default);
    }
    else
    {
        Debug.LogError(message: "Cannot create room: not connected to Master Server.");
    }
}

/* Lasset usage: & lachettebro */
public void JoinName()
{
    string roomName = JoinGameInput.text.Trim();
    if (!string.IsNullOrEmpty(roomName.Trim()))
    {
        PhotonNetwork.JoinOrCreateRoom(roomName, new RoomOptions() { MaxPlayers = 5 }, TypedLobby.Default);
    }
    else
    {
        Debug.LogError(message: "Cannot join room: no room name provided.");
    }
}
```

Ces fonctions ont été essentielles pour la gestion des connexions des joueurs et la création de salles de jeu. En fin de compte, l'utilisation de ces fonctions a permis une synchronisation efficace de l'état du jeu entre les différents clients connectés, offrant une expérience de jeu multijoueur incroyable.

Nous avons été impressionné par notre propre compréhension approfondie du fonctionnement de Unity. Nous avons réussi à intégrer Photon de manière transparente dans mon projet Unity, ce qui a permis une synchronisation efficace de l'état du jeu entre les différents clients connectés.

Notre travail a également impliqué la gestion de la connexion des joueurs, en veillant à ce que les joueurs soient connectés en temps réel, tout en gérant les problèmes de synchronisation des connexions des joueurs et la mise en réseau des données entre les clients.

Au final, le travail que nous avons accompli sur la mise en place du mode multijoueur en utilisant Photon dans le jeu Unity a été essentiel pour offrir une expérience de jeu multijoueur incroyable. Notre implication et notre dévouement nous ont permis de contribuer à la création d'un jeu qui apporte une expérience de jeu sociale et compétitive à un tout nouveau niveau.

2.2 Skin

Yassine :

J'ai travaillé sur la création d'une scène non-officielle pour le jeu qui sera implémentée très prochainement. Cette scène permettra aux joueurs de personnaliser leur personnage en choisissant leur apparence parmi une liste de skins disponibles.

Pour créer cette scène, j'ai étudié en détail les différentes options de personnages disponibles dans le jeu, ainsi que leurs caractéristiques et leurs attributs. J'ai également étudié les différentes manières dont les joueurs interagissent avec ces personnages, en particulier en ce qui concerne le choix de leur apparence.

En utilisant les outils de développement fournis par le moteur de jeu, j'ai conçu une scène qui permet aux joueurs de sélectionner leur personnage et de personnaliser leur apparence en utilisant un système d'interface utilisateur simple et intuitif.

Le script SkinManager que j'ai écrit permet aux joueurs de naviguer facilement entre les différentes options de skins disponibles pour leur personnage et de lancer le jeu avec le skin choisi. Les fonctions importantes de ce script comprennent NextOption(),

```
1 asset usage  Yassine SABER
public void NextOption()
{
    selectedSkin = selectedSkin + 1;
    if (selectedSkin == skins.Count)
    {
        selectedSkin = 0;
    }

    sr.sprite = skins[selectedSkin];
}
```

Cette fonction permet de passer au skin suivant dans la liste des skins disponibles pour le personnage.

BackOption()

```
    * asset usage □ Yassine SABER
public void BackOption()
{
    selectedSkin = selectedSkin - 1;
    if (selectedSkin < 0)
    {
        selectedSkin = skins.Count - 1;
    }

    sr.sprite = skins[selectedSkin];
}
```

Cette fonction permet de revenir au skin précédent dans la liste des skins disponibles pour le personnage.

Et PlayGame()

```
    * asset usage □ Yassine SABER +1
public void PlayGame()
{
    GameObject selectedSkinPrefab = Instantiate(playerskin);

    SpriteRenderer prefabSR = selectedSkinPrefab.GetComponent<SpriteRenderer>();
    prefabSR.sprite = sr.sprite;

    SceneManager.LoadScene("MainGame");
}
```

Cette fonction est appelée lorsque le joueur a sélectionné un skin pour son personnage et qu'il souhaite lancer le jeu. Elle instancie un prefab de personnage avec le skin sélectionné et charge la scène principale du jeu.

Ces fonctions permettent aux joueurs de naviguer facilement entre les différentes options de skins disponibles pour leur personnage et de lancer le jeu avec le skin choisi.

Ci-dessous, vous pouvez voir un aperçu de la scène de choix de personnages que j'ai créée :



La création de cette scène de choix de personnages a été un projet passionnant et enrichissant pour moi en tant qu'étudiant en informatique. J'ai pu mettre en pratique mes connaissances en programmation et en conception de jeux tout en offrant aux joueurs une expérience de jeu plus personnalisée et satisfaisante. Cette tâche effectuée pour permettre au jeu de le rendre plus esthétique et plus attractif très prochainement.

3 Implémentation du dé et déplacement des joueurs

MoneyMaker est un jeu de plateau, pour cela nous avons dû concevoir un dé pour que les joueurs puissent avancer sur les cases selon le résultat obtenu sur le dé. La première étape a été de concevoir un plateau de jeu afin de représenter les cases.

Voici une illustration du plateau utilisé pour la conception de la scène de jeu :

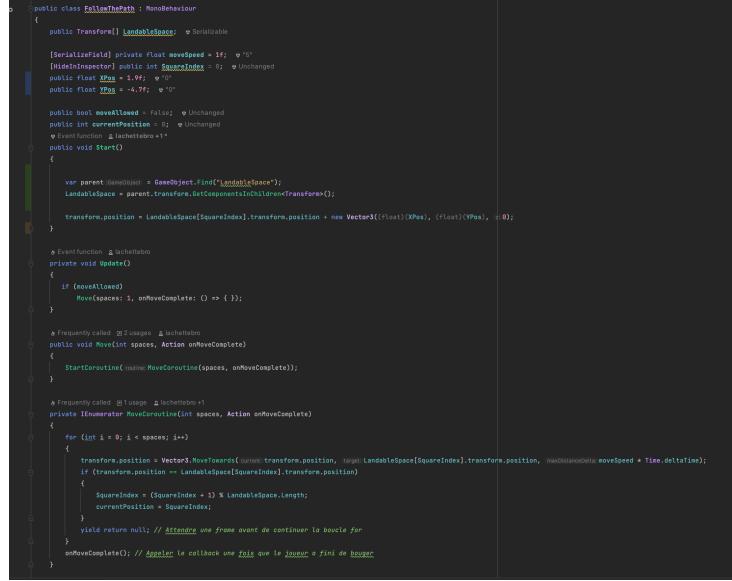


Ainsi, il a fallu représenter sur Unity le parcours que le joueur doit suivre lors du lancer du dé. Pour cela, l'utilisation de GameObject a été nécessaire pour montrer le chemin. Mais également de rédiger un script pour donner l'ordre aux joueurs de suivre un chemin précis. La classe contient plusieurs membres :

- Un tableau de Transform appelé "LandableSpace", qui contient une liste de Transform correspondant aux positions que l'objet doit atteindre.
- Une variable float "moveSpeed" qui détermine la vitesse de déplacement de l'objet le long de la trajectoire.
- Des variables float "XPos" et "YPos" qui déterminent la position initiale de l'objet lorsqu'il commence à suivre la trajectoire.

- Une variable booléenne "moveAllowed" qui détermine si l'objet est autorisé à se déplacer le long de la trajectoire.
- Une variable int "currentPosition" qui stocke l'indice du point sur lequel l'objet se trouve actuellement.
- Les méthodes Start(), Update(), Move() et MoveCoroutine().

Voici une illustration du script permettant ces déplacements :



La méthode Start() est appelée lorsque le script est chargé. Elle recherche un objet appelé "LandableSpace" dans la scène et récupère tous les Transform enfants de cet objet. Elle utilise ensuite l'indice "SquareIndex" pour définir la position initiale de l'objet en ajoutant les valeurs "XPos" et "YPos" à la position du Transform actuel.

La méthode Update() est appelée chaque image de la scène. Elle vérifie si l'objet est autorisé à se déplacer (moveAllowed est true) et appelle la méthode Move() avec un argument "spaces" de 1 et une lambda (expression anonyme) comme deuxième argument.

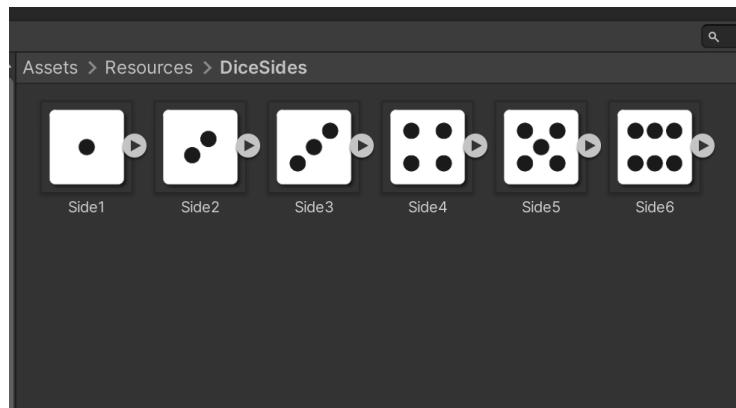
La méthode Move() est une méthode publique qui prend deux arguments : le nombre de "spaces" que l'objet doit avancer le long de la trajectoire et une "onMoveComplete" fonction de rappel qui est appelée une fois que le mouvement est terminé. Cette méthode appelle la méthode MoveCoroutine() avec les mêmes arguments.

La méthode MoveCoroutine() est une coroutine qui déplace progressivement l'objet vers le prochain point de la trajectoire en utilisant la méthode Vec-

tor3.MoveTowards(). Elle utilise une boucle for pour déplacer l'objet "spaces" fois. À chaque itération de la boucle, elle met à jour la position de l'objet en utilisant Vector3.MoveTowards() et vérifie si l'objet est arrivé au prochain point. Si c'est le cas, elle met à jour l'indice "SquareIndex" et "currentPosition" et continue la boucle. Une fois que la boucle est terminée, elle appelle la fonction de rappel "onMoveComplete".

En résumé, ce script permet de déplacer un objet le long d'une trajectoire prédéfinie en utilisant des Transform et la méthode Vector3.MoveTowards(). La méthode Move() est utilisée pour déclencher le déplacement, et la fonction de rappel "onMoveComplete" est appelée une fois que le mouvement est terminé.

L'implémentation du dé a été donc primordial pour les déplacements des joueurs. Pour cela, nous avons dû créer des Sprites représentant les faces du dé :



Pour faire fonctionner ce dé, il a fallu le relier au script FollowThePath également au fonctionnement du joueur. Voici le script qui a permis cela :

```

# 2 asset usages  □ lacheitebro v2
public class GameControl : MonoBehaviour
{
    public GameObject[] players; ↳ Serializable
    public int currentPlayerIndex = 0; ↳ Unchanged
    public FollowThePath[] paths; ↳ Serializable
    private Sprite[] diceSides;
    private SpriteRenderer rend;
    private bool coroutineAllowed = true;

↳ 8 usages  □ Yassine SABER  □ 4 exposing APIs
public enum SceneName
{
    CaseFrance,
    CaseCanada,
    CaseItalie,
    CaseAllemagne
}
↳ Event function  □ lacheitebro
private void Start()
{
    rend = GetComponent<SpriteRenderer>();
    diceSides = Resources.LoadAll<Sprite>(path: "DiceSides/");
}

↳ Event function  □ lacheitebro +1
private void OnMouseDown()
{
    if (coroutineAllowed)
    {
        StartCoroutine(methodName: "RollTheDice");
    }
}

↳ 1 usage  □ lacheitebro
private IEnumerator RollTheDice()
{
    coroutineAllowed = false;
    int randomDiceSide = 0;

    for (int i = 0; i < 20; i++)
    {
        randomDiceSide = Random.Range(0, diceSides.Length - 1);
        rend.sprite = diceSides[randomDiceSide];
        yield return new WaitForSeconds(0.05f);
    }

    int spacesToMove = randomDiceSide + 1;

    yield return StartCoroutine(methodName: "MoveAllPlayers(spacesToMove)");

    currentPlayerIndex++;

    if (currentPlayerIndex >= players.Length)
    {
        currentPlayerIndex = 0;
    }

    coroutineAllowed = true;
}

↳ Frequently called (13 usages  □ lacheitebro +2)
private IEnumerator MoveAllPlayers(int spaces)
{
    for (int i = 0; i < players.Length; i++)
    {
        GameObject currentPlayer = players[i];
        currentPlayer.GetComponent<FollowThePath>().moveAllowed = true;

        for (int j = 0; j < spaces; j++)
        {
            currentPlayer.GetComponent<FollowThePath>().Move(spaces: 1, onMoveComplete: () => ());
            yield return new WaitForSeconds(0.2f);
        }

        currentPlayer.GetComponent<FollowThePath>().moveAllowed = false;
    }
}

```

Ce script est utilisé pour gérer un jeu de plateau avec plusieurs joueurs.

La classe GameControl est attachée à un GameObject et contient plusieurs variables publiques telles que players, currentPlayerIndex et paths.

players est un tableau de GameObjects qui contiennent des scripts Follow-ThePath qui permettent aux joueurs de se déplacer le long du plateau.

currentPlayerIndex est un entier qui représente l'index du joueur en cours.

paths est un tableau de scripts FollowThePath qui contiennent des informations sur les emplacements où les joueurs peuvent atterrir.

Le script utilise également des Sprites pour simuler le lancer de dés.

Lorsque le joueur clique sur le GameObject auquel est attaché le script, la méthode OnMouseDown() est appelée. Cette méthode commence alors la coroutine RollTheDice().

La coroutine RollTheDice() choisit un chiffre aléatoire entre 1 et 6 en faisant défiler les images de dés de manière aléatoire à travers un tableau de Sprites. La valeur sélectionnée est ensuite utilisée pour déterminer le nombre d'espaces que le joueur en cours doit avancer.

La coroutine MoveAllPlayers() est alors appelée pour faire avancer tous les joueurs sur le plateau d'un certain nombre d'espaces. Pour chaque joueur, la méthode Move() du script FollowThePath est appelée pour déplacer le joueur sur l'emplacement suivant du plateau. La coroutine utilise ensuite yield return

pour attendre un court instant avant de faire avancer le joueur suivant.

Enfin, le script passe le tour au joueur suivant en incrémentant la variable currentPlayerIndex. Si le dernier joueur a joué, le script recommence à zéro en réinitialisant la variable currentPlayerIndex à 0. La coroutine RollTheDice() peut alors être appelée à nouveau pour permettre au joueur suivant de jouer.

Toutes ces étapes ont finalement permis d'avoir un gameplay telle que celui-ci :



4 Menu pause et création des scènes

4.1 Menu Pause

Pour cette deuxième soutenance je me suis d'abord occupée de créer un menu pause pour la scène principale.

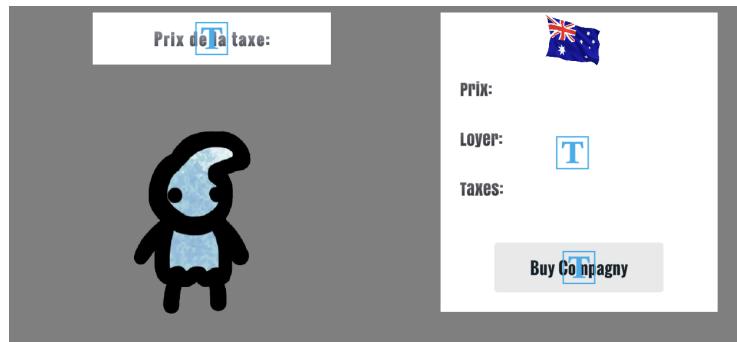
Le menu est sur la même scène que la scène principale afin que le jeu reprenne là où il s'est arrêté. Et pour accéder au menu il faut appuyer sur la touche espace du clavier.



Dans le menu pause on peut « resume » le jeu, aller au menu principal et quitter le jeu.

4.2 Création des scènes

J'ai ensuite fait plusieurs scènes pour chaque case du plateau dans lesquelles il y aura les datas des entreprises et le moyen de les acheter. La scène prison a été faite de manière à ce que quand le joueur apparaît sur la scène, un message s'affiche en disant « tu as atterri en prison »





5 Passage des scènes

Nous avons travaillé, Yassine et Anna pour créer le passage des scènes en fonction des cases.

La tâche consistant à créer le passage des scènes en fonction des cases a été cruciale pour le jeu car elle a permis aux joueurs d'interagir avec l'environnement de manière plus immersive. En effet, grâce à cette fonctionnalité, les joueurs peuvent accéder à des scènes spécifiques en fonction de leur position sur le plateau, ce qui ajoute une dimension supplémentaire à l'expérience de jeu.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using PhotonNetwork;
using Photon.Pun;
using Photon.Pun.Utility;

public class SquareScript : MonoBehaviour
{
    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "Player")
        {
            public void LoadScene()
            {
                Console.WriteLine("CurrentScene");
                SceneManager.LoadScene(currentScene);
            }
            // Start is called before the first frame update
            void Start()
            {
            }

            // Update is called once per frame
            void Update()
            {
            }
        }
    }

    public void OnPlayerMoveComplete(int currentpos)
    {
        // Charger la scène correspondante en fonction de la case sur laquelle le joueur est arrivé
        SceneManager.LoadScene((valCurrentpos), LoadSceneMode.Single);
        PhotonNetwork.LoadLevel((valCurrentpos));
        // Ajouter d'autres conditions pour charger d'autres scènes en fonction des cases
    }
}

```

Nous avons développé cette fonctionnalité en créant un script qui charge les scènes en fonction de la position du personnage sur le plateau du jeu. Ce script est une tâche complexe, mais elle a été réalisée avec succès en collaboration.

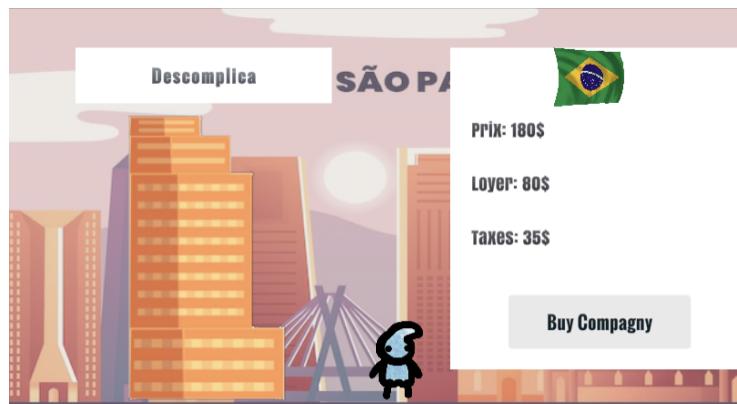
Nous avons utilisé notre expertise technique pour concevoir un algorithme efficace qui permet de charger les scènes rapidement et en douceur, garantissant ainsi une expérience de jeu fluide pour les joueurs.

Nous avons veillé à ce que chaque scène soit correctement chargée en fonction de la position du joueur sur le plateau, ce qui a nécessité un certain nombre d'itérations et de tests. Nous avons également créé une fonction "OnPlayerMoveComplete()" qui permet de charger automatiquement la scène correspondante à la position du joueur sur le plateau. Cette fonction utilise une énumération pour identifier chaque case et charge la scène correspondante en fonction de la position du joueur.

```
 1 usage  Yassine SABER *
public void OnPlayerMoveComplete(int currentpos)
{
    SceneManager.LoadScene(yaya[currentpos], LoadSceneMode.Single);
    PhotonNetwork.LoadLevel(yaya[currentpos]);
}
```

Nous travaillons actuellement sur la création d'un bouton qui permettra aux joueurs de quitter la scène et de revenir sur le plateau. Ce bouton sera un élément essentiel de l'expérience de jeu, car il permettra aux joueurs de revenir facilement à l'action principale du jeu tout en profitant des scènes supplémentaires. Nous sommes en train d'optimiser le code pour garantir que ce bouton sera aussi fluide que possible.

Voici une vue d'ensemble de la scène créée :



En conclusion, la fonction de passage des scènes en fonction des cases a été un élément crucial de la création de notre jeu. Elle ajoute une dimension supplémentaire à l'expérience de jeu et rend l'environnement de jeu plus immersif.

Nous sommes impatients de poursuivre notre travail sur ce projet et de créer une expérience de jeu encore plus immersive pour nos joueurs.

6 Gameplay en c# et implémentations dans Unity

6.1 Le plateau



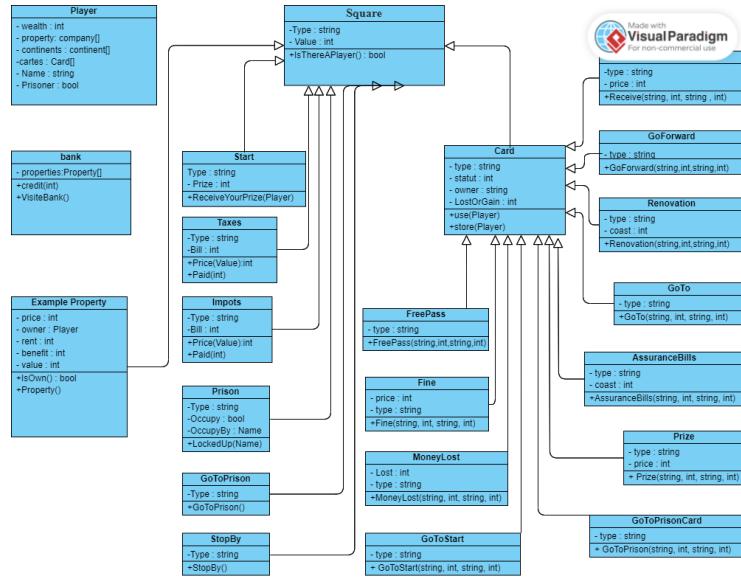
Pour cette seconde soutenance nous avons un peu mis de coté l'aspect design de notre jeu pour se concentrer sur les parties un peu plus technique. Néanmoins, la première étape pour commencer le code du jeu été d'avoir un plateau près à l'emploi.

Ce plateau a été créé en reprenant les codes du Monopoly original, avec des cases ayant chacune leurs attribut et fonction propres.

On différencie deux types de case, les cases pays et les cases spéciales. Chaque case pays se repère par rapport au dégradé de couleur. Un dégradé de couleur orange, jaune, rose, bleu, et vert symbolise le continent auquel appartient cette case et le logo dessus est celui de l'entreprise principal de cette case.

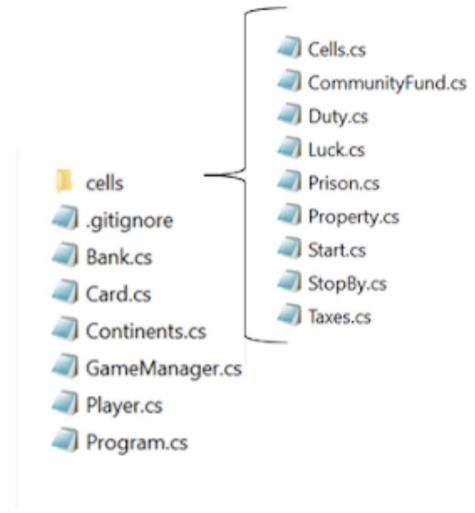
6.2 Le Gameplay en c#

Notre jeu étant essentiellement un jeu de plateau, nous avons dans un premier temps fait le choix d'une première création en c#. Pour cela, nous avons repris l'UML créé pour la première soutenance.



Avec la réalisation du plateau officiel et dans l'intérêt de réalisé quelque chose de plus facile à implémenter pour notre version Unity, il a été créé une classe pour chaque case du jeu (chaque case représentant un pays, les cases chances, les cases impôts et taxes, ...). Il a également été implémenté une classe pour les joueurs, qui plus tard sera associé au multijoueur, une classe pour la banque et une classe pour les cartes. Tout ce code fonctionne grâce à une classe Game-Manager : celle-ci permet la création d'une nouvelle partie. Au lancement du jeu, les joueurs ainsi que chaque case du plateau se créée et sont rangés dans un tableau. La partie se joue alors à partir de ce tableau et des cases qui y sont intégrées.

L'intérêt de coder séparément le jeu en c# été avant tout pour nous une manière d'avoir une vision plus global du jeu et de son implémentation tant sur le fond que sur la forme. Cette version nous couplet au plateau officiel nous permet désormais une vision global de notre jeu et nous permet d'expliquer concrètement l'approche que nous faisons de la création de ce jeu.

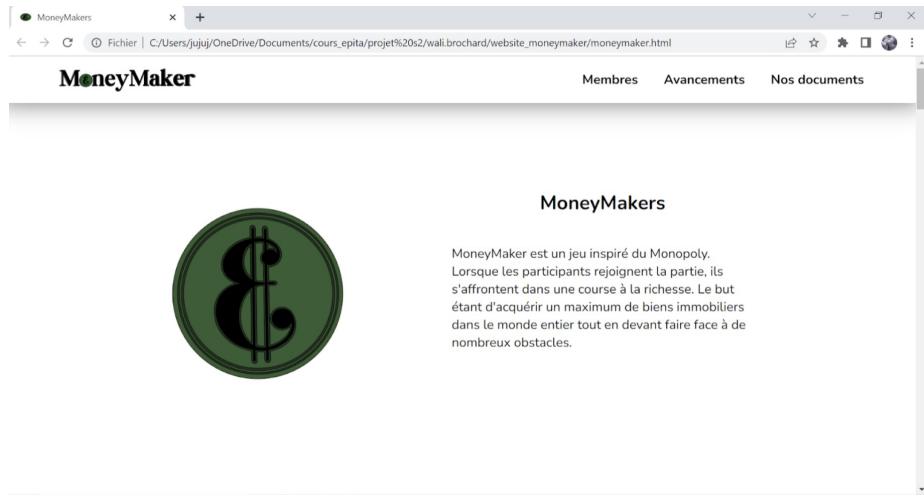


6.3 Implémentation du code dans Unity

Maintenant que nos règles sont définis et créées, il va nous falloir les implémentés à notre jeu Unity, autrement mettre en lien le fond et la forme.

Dit comme cela la tâche paraît simple mais cela va demandé de mettre en lien beaucoup d'éléments du jeu déjà implanté. Il s'agira non seulement de créer les parties en les associant à notre multijoueur mais aussi de relié chacune des scènes représentant une case à son script respectif.

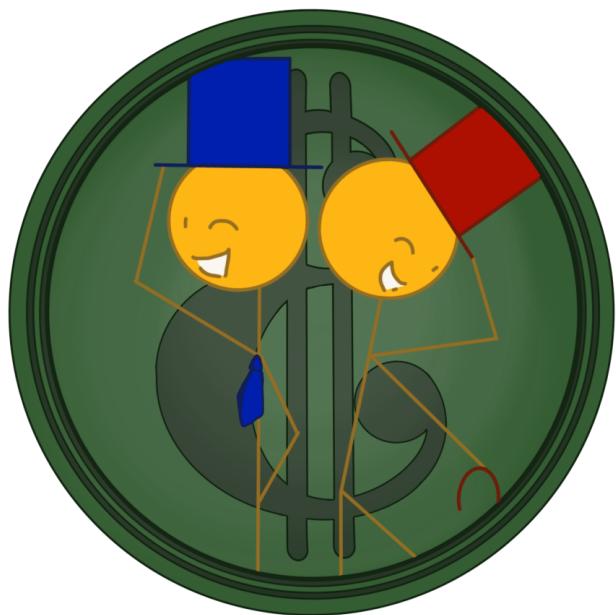
7 le site



Le site a été imaginé comme une présentation marketing de notre jeu : on y retrouve une fiche technique et design et une courte présentation des différents membres du groupe. Dans l'intérêt cette soutenance, il y a été rajouter un tableau d'avancement des tâches par membres et un lien pour télécharger les documents officiels tel que le cahier des charges ou le rapport de soutenance.

D'un point de vue plus technique, il a été réalisé en HTML/CSS. Nous avons souhaité offrir une expérience d'utilisation maximale, malgré la simplicité des designs, notamment avec une barre flottante permettant d'avoir accès aux différentes parties de notre site de manière efficace.

8 Conclusion



Logo Hasbrothers

8.1 Nos ressentis

Depuis cette première soutenance nous avons tous beaucoup progressé, que ce soit d'un point de vue personnel, dans notre capacité à gérer une tâche donné et à faire valoir nos compétences au seins du groupe mais aussi de manière collective, en apprenant comment communiqué de manière efficace les uns avec les autres.

Ce projet est très complet et nous fait travaillé des compétences techniques qui nécessite un travail important et une grande motivation. Mais tout cela reste avant tout un travail de groupe, et si nous doutons parfois face à nos tâches et la manière dont nous devons les aboutir, il est important d'avoir des camarades sur qui compter et avec nous savons être écouté.

8.2 Vers la soutenance finale

Nous avions des objectifs plus ou moins définis à la fin de la première soutenance et le gros de notre planning aura été décidé et acté dans les deux premières semaines qui auront suivis cette première présentation.

Cette remise à niveau de nos objectifs face à nos recherche et au début de certaine grosse partie du projet nous ont poussé à avoir un oeil plus critique sur nos compétence mais aussi et avant tout à nous poussé à donné le meilleur de nous même pour remplir les objectifs annoncé.

Les aspects les plus techniques de notre jeu ont tous commencé à être abordé voir finis et cela nous place en confiance pour la suite.

L'objectif d'avoir un jeu de plateau interactif de concrétise de plus en plus au fur et à mesure de l'avancement du projet. Tout le monde est ainsi dans une bonne dynamique : les prochaines étapes cruciales consisteront principalement à l'implémentation du GamePlay et la création de l'IA. Ayant pour but de présenter un projet techniquement mais aussi esthétiquement aboutit, il nous restera bien sur à reprendre les designs des différentes scènes du jeu et éventuellement celui des personnages.