Documentation technique

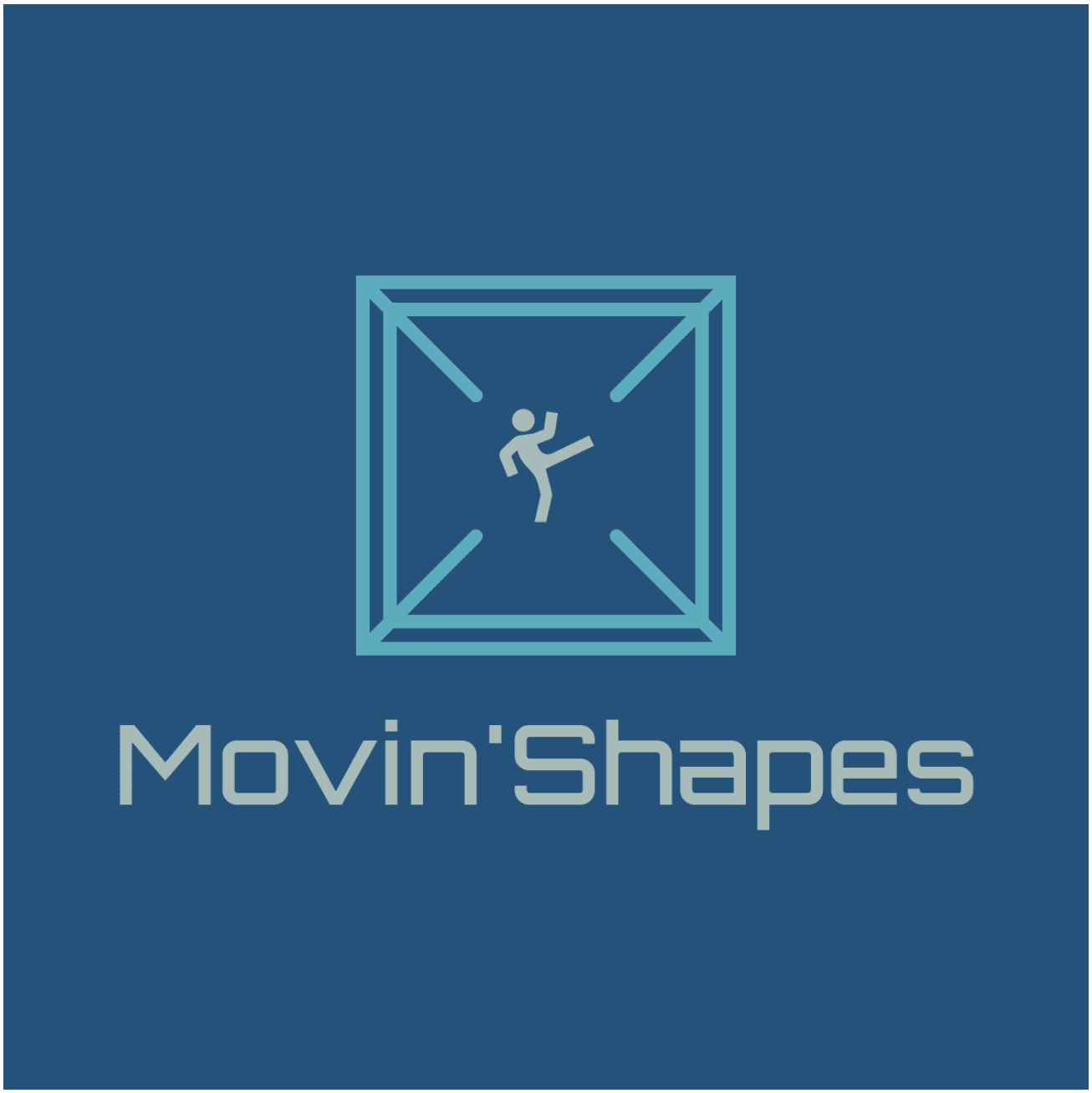


Table des matières

1. Introduction

Notre projet Movin'Shape est en version Unity 2019.4.8f1 et utilise une caméra Kinect V2. Cette documentation montre comment recréer le projet de A à Z et de pouvoir modifier le projet en sachant ce que font les différents objets et scripts.

Ce projet a été imaginé sur le modèle du jeu OShapes VR et inspiré du Dancing Traffic Light de Smart. Deux joueurs peuvent faire équipe et essayer de faire le maximum de points. Au bout d'un certain nombre de vies perdues, le jeu s'arrête et un classement s'affiche.

Il est nécessaire de maîtriser les bases de Unity et du C# pour comprendre les différentes explications.

2. Matériel et Logiciel

2.1 Matériel nécessaire

Voici la liste du matériel nécessaire au fonctionnement du jeu :

- 1 ordinateur avec de Windows 10 à jour
- Une caméra Kinect V2
- Un écran de projection
- Les câbles permettant de faire fonctionner le tout (HDMI, alimentation, etc…)
- Une passerelle pour internet (Utile pour pouvoir télécharger des plugins Unity)

Voici la liste des logiciels et packages Unity nécessaire au fonctionnement du jeu :

- Unity Hub / Unity version 2019.4.8f1
- Kinect-v2-with-MS-SDK-v2.19.2
- Volumetric Lines 3.0.2
- ProBuilder v4.2.3
- Free MatCap Shaders 1.4.0.0

2.2 Kinect

Lorsque la Kinect est branchée au PC pour la première fois, la Kinect installe automatiquement tous les pilotes nécessaires à son fonctionnement. Pour tester son fonctionnement, il faut installer le SDK de Microsoft et lancer le testeur.
Attention, l'installation des pilotes peut prendre beaucoup de temps.

Le branchement et le positionnement de la caméra Kinect se fait selon les schémas suivants :

Schema_branchement -Kinect.vsdx

2.3 Environnement Unity

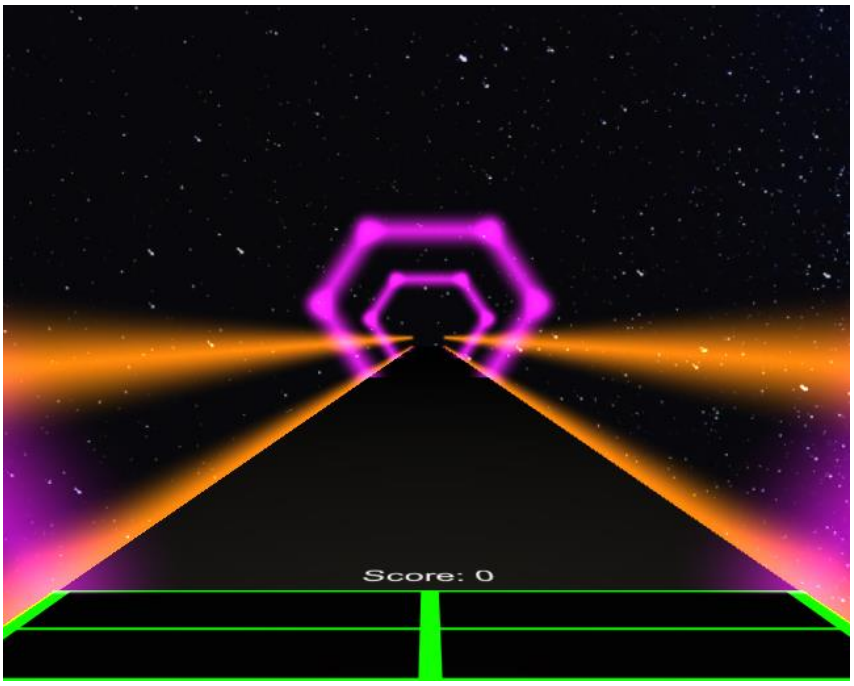
Il suffit de télécharger Unity Hub et installer ensuite Unity version 2019.4.8f1. Si la version n'apparaît pas, on peut la trouver dans les archives des versions Unity. Il faut aussi installer Visual Studio 2019 qui vient normalement directement s'installer avec avec Unity v2019.4.8f1.

Le package Unity de reconnaissance de squelettes Kinect ne fonctionne pas sur les versions plus récentes.

Il faut également se connecter avec un compte Unity ID pour pouvoir télécharger des packages.
Attention, il faut se connecter à un autre réseau que DIVTEC.LOCAL pour le faire.

3. Graphique

3.1 Décor



Le décor se compose en 4 parties :

- Le sol, composé d'un rectangle noir et de lasers orange formant les côtés
- Les arches, composé de lasers violets
- Le fond étoilé
- Le score est juste un texte posé sur le rectangle avec un angle de 90 degrés en X

1. Le rectangle noir faisant le sol est un cube 3D de Unity.
2. Les lasers oranges et violets viennent du package Unity disponible sur l'Asset Store: Volumetric Lines 3.0.2
   Pour faire apparaître les lasers, il suffit de créer un GameObject et d'assigner le

script « Volumetric Line Behavior » et d'ensuite assigner la position de départ et de fin du laser. La couleur étant changé dans le « Line Color ».

3. Le fond étoilé est tiré d'une image venant de Paul Volkmer.

3.2 Couleur du décor

Les couleurs sont présentes dans le dossier Material du projet.

- Vert : #07BF00
- Orange : #FF8400
- Violet : #700B73
- Noir : #000000

3.3 Portes

Pour chaque porte, une fois terminée, a été transformé en Prefab (Un seul objet Unity) contenant tous les différents composants. (La forme et les zones de collisions)

Les portes sont présentes dans le dossier « Portes » du projet.

3.3.1 Création

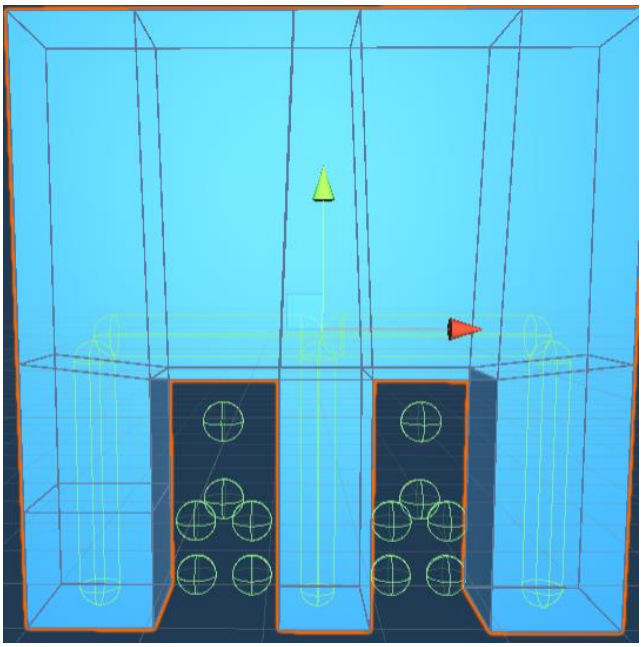
Pour la création des portes, nous avons utilisé le plugin ProBuilder v4.2.3.
Pour débuter la création, nous avons créé un cube auquel nous avons fait des extrusions pour pouvoir donner la forme finale à la porte.

3.3.2 Collisions

Chaque porte est composée des mêmes composants que les autres. Chaque porte est composée de :

- D'un Transform
- D'un Mesh Renderer
- Des collisions des personnages
- Des collisions des portes
- De la couleur

Voici un exemple :



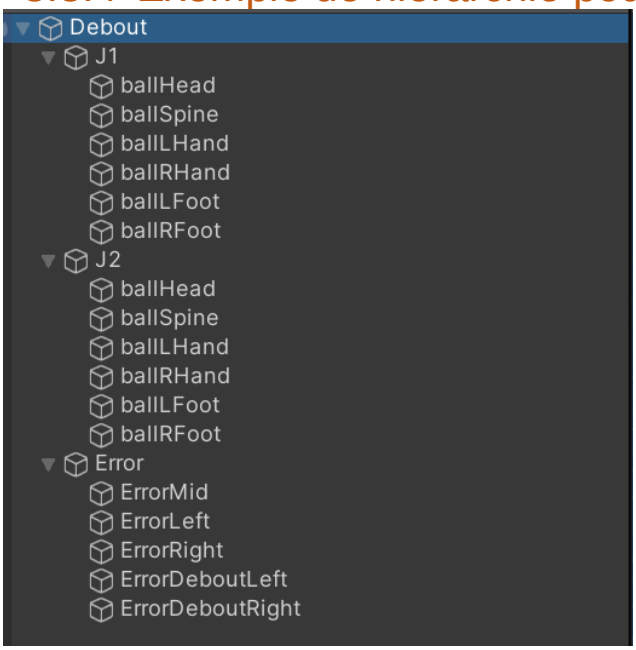
3.3.2.1 Collisions des personnages et des portes

Pour chaque personnage et chaque porte, des Sphere Collider ou Capsule Collider avec le Trigger activé est en place et positionné selon la porte.

3.3.3 Couleur des portes

Bleu : #0087FF

3.3.4 Exemple de hiérarchie pour les portes sur Unity



3.4 Menu principale et de fin

La partie graphique du menu principale est tirée de cette vidéo :
https://www.youtube.com/watch?v=zc8ac_qUXQY

La partie code du menu final est tirée de cette vidéo :
https://www.youtube.com/watch?v=FSEbPxf0kfs

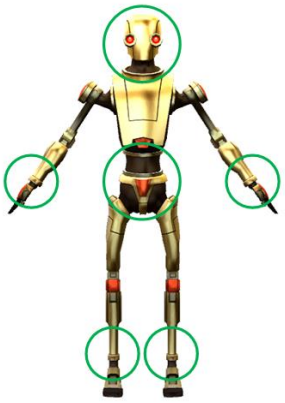
L'exemple se trouve sur GitHub : https://github.com/DapperDino/Scoreboard-Tutorial

4. Personnage

4.1 Collisions

Les personnages sont présents dans le dossier « Characters » du projet.

Les collisions des personnages sont activées sur les poignets, chevilles, bassin et la tête avec l'ajout d'un Rigidbody(avec « is Kinematic » de coché et un Sphere Collider(sans le « is Trigger » de coché) comme sur l'image suivante :



4.2 Positionnement des scripts

Sur chaque partie du corps avec le Sphere Collider présent comme sur l'image, le script OverlappingJ1/OverlappingJ2 (selon le robot du joueur) est présent.

4.3 Couleur des personnages

Les couleurs des personnages viennent d'un package Unity trouvé sur l'Asset Store :

- Free MatCap Shaders 1.4.0.0

5. Programmation

5.1 Scripts

Tous les scripts sont présents dans le dossier « Script » du projet.

5.1.1 AddScore

AddScore est le script permettant l'ajout du score en équipe pour les personnages.

Dans la fonction Start(), on initialise les valeurs et les listes :

```
// Initialisation
textScore.text = "Score: " + GameManager.Score;
listBallToucheJ1 = new List<int>();
listBallToucheJ2 = new List<int>();
```

L'ajout du score se fait en plusieurs étapes dans la fonction Update().

1. On vérifie si la porte n'a pas été touchée et on vérifie que la porte ayant passé la position ne soit pas vide :

```
if (TimerLife.isWallHit == false)
{
	if (InstantiateDoors.doorToDestroy != null)
```

2. On vérifie que le score n'a pas déjà été ajouté et que la position de la porte est au bon endroit et ensuite on compte les points et on ajoute au score :

```
if (addPoints && InstantiateDoors.doorToDestroy.transform.position.z <= POSITION_Z_BEHIND)
{
	for (int pointsJ1 = 0; pointsJ1 < listBallToucheJ1.Count; pointsJ1++)
	{
		nbrePointsJ1 += listBallToucheJ1[pointsJ1];
	}

	for (int pointsJ2 = 0; pointsJ2 < listBallToucheJ2.Count; pointsJ2++)
	{
		nbrePointsJ2 += listBallToucheJ2[pointsJ2];
	}

	score += nbrePointsJ1 + nbrePointsJ2;
```

3. On réinitialise les valeurs :

```
addPoints = false;
listBallToucheJ1.Clear();
listBallToucheJ2.Clear();
nbrePointsJ1 = 0;
nbrePointsJ2 = 0;
```

4. On affiche le score :

```
text.text = "Score: " + score;
```

5.1.2 InstantiateDoors

InstantiateDoors est le script faisant la création aléatoire et la suppression des portes créés.
Le script se déroule en deux étapes principales situé dans les fonctions suivantes :

- CreateDoor(listDoorsInstantiated, listDoors)
- DestroyDoor(listDoorsInstantiated)

En premier, dans la fonction Start(), on initialise les listes contenant les portes et on ajoute les portes à créer dans la bonne liste :

```
listDoors = new List<GameObject>();
listDoorsInstantiated = new List<GameObject>();

// Ajout des portes dans la liste
listDoors.Add(doorAssis);
listDoors.Add(doorDebout);
listDoors.Add(doorDeboutAssis);
listDoors.Add(doorMainLeve);
listDoors.Add(doorPiedLeve);
listDoors.Add(doorTPose);
listDoors.Add(doorX);
listDoors.Add(doorAssisDebout);
listDoors.Add(doorDeboutCoteExt);
listDoors.Add(doorDeboutCoteInt);
```

Dans la fonction Update(), on appelle les deux fonctions CreateDoors() et DestroyDoors().

```
CreateDoor(listDoorsInstantiated, listDoors);
DestroyDoor(listDoorsInstantiated);
```

Pour la création des portes, la fonction appelée « CreateDoor() » est créé :

```
void CreateDoor(List<GameObject> listDoorsInstantiated, List<GameObject> listDoors)
```

Ensuite la création se fait en plusieurs étapes :

1. On instancie le système afin de créer des nombres aléatoires
   ```
   System.Random random = new System.Random();
   ```
2. On parcourt la liste des portes déjà créé s'il y en a jusqu'à atteindre le nombre max de porte :
   ```
   for (int doorsInstantiated = listDoorsInstantiated.Count; doorsInstantiated < maxDoorsToCreate; doorsInstantiated++)
   ```
3. On génère on nombre aléatoire en fonction du nombre de portes disponible et on la crée en l'associant à l'index aléatoire de la liste ainsi que son nom :
   ```
   int indexRandom = random.Next(listDoors.Count);
   GameObject door = null;
   door = Instantiate(listDoors[indexRandom]);
   door.name = listDoors[indexRandom].name;
   ```
4. On compte les portes créées :
   ```
   if ((doorsInstantiated % 4) <= 1)
   {
          TimerLife.doorCounted++;
   }
   ```
5. On met la position de la porte créée :
   ```
   float positionZ = SPACE_BTW_DOORS * (doorsInstantiated + 1);
   door.transform.position = new Vector3(-1.5f, 0f, positionZ);
   ```
6. On lui ajoute le script de mouvement :
   ```
   door.AddComponent<Movement>();
   ```
7. On ajoute la porte à la liste des portes déjà instanciées :

```
listDoorsInstantiated.Add(door);
```

Pour la destruction des portes, la fonction appelée « DestroyDoor() » est créé :

```
void DestroyDoor(List<GameObject> listDoorsInstantiated)
```

1. On parcourt la liste des portes instanciées :
   ```
   for (int indexDoorsToDestroy = 0; indexDoorsToDestroy < listDoorsInstantiated.Count;
   indexDoorsToDestroy++)
   ```
2. On vérifie que la liste des portes créées a 1 élément ou plus et on vérifie si l'index donné est différent de null :
   ```
   if (listDoorsInstantiated.Count >= 1)
   {
          if (listDoorsInstantiated[indexDoorsToDestroy] != null)
   ```
3. On vérifie la position de la porte et si elle a passé le point donné on l'enlève de la liste. On passe aussi l'ajout du score à « Vrai » pour qu'on puisse inscrire le score :
   ```
   if (listDoorsInstantiated[indexDoorsToDestroy].transform.position.z <
   POSITION_Z_PASSED)
          {
                 doorToDestroy = listDoorsInstantiated[indexDoorsToDestroy];
                 listDoorsInstantiated.RemoveAt(indexDoorsToDestroy);
                 AddScore.addPoints = true;
          }
   ```
4. Les dernières étapes sont de vérifier si la porte à détruire n'est pas vide et supprimer la porte quand la position est passée :
   ```
   if (doorToDestroy != null)
   {
          if (doorToDestroy.transform.position.z < POSITION_Z_DETROY)
          {
                 TimerLife.isWallHit = false;
                 Destroy(doorToDestroy);
          }
   }
   ```

5.1.3 Movement

Movement est le script permettant aux portes d'avancer.

Il consiste en une étape dans la fonction Update() :

1. Faire bouger la porte dans le sens qu'on veut selon le temps et la vitesse donnée :
   ```
   speed = TimerLife.doorSpeed;
   //La vitesse des portes
   Vector3 newPosition = new Vector3(6f, 0f, 0f);
   transform.Translate(newPosition * Time.deltaTime * speed);
   ```

5.1.4 OverlappingJ1 et OverlappingJ2

OverlappingJ*1/2* sont les scripts mis sur chaque joueur afin de détecter s'il y a une collision.

On commence par vérifier si un Trigger a été activé en passant en paramètre le « Collider » touché dans la fonction OnTriggerExit :

```
private void OnTriggerExit(Collider other)
```

Une fois activé, voici les étapes :

1. On récupère le « GameObject » parent du « Collider » touché et on vérifie son nom, s'il porte le nom « Error » alors une porte a été touché :

```
GameObject gameObjectOther = other.gameObject;
parentBall = gameObjectOther.transform.parent.gameObject;

//Regarde le nom parent si il est appelé « Error »
if (parentBall.name == "Error")
{
	TimerLife.isWallHit = true;
}
```

2. Sinon on vérifie le tag du Collider et on l'ajoute dans la liste J1 ou J2 suivant le joueur (en exemple, c'est le J1) des zones de collision touchées pour les points :

```
//Regarde le tag de l'objet triggered et ajoute les points
switch (other.tag)
{
	case "LFoot":
		if (!AddScore.listBallToucheJ1.Contains(L_FOOT))
		{
			AddScore.listBallToucheJ1.Add(L_FOOT);
		};
		break;
	case "RFoot":
		if (!AddScore.listBallToucheJ1.Contains(R_FOOT))
		{
			AddScore.listBallToucheJ1.Add(R_FOOT);
		};
		break;
	case "LHand":
		if (!AddScore.listBallToucheJ1.Contains(L_HAND))
		{
			AddScore.listBallToucheJ1.Add(L_HAND);
		};
		break;
	case "RHand":
		if (!AddScore.listBallToucheJ1.Contains(R_HAND))
		{
			AddScore.listBallToucheJ1.Add(R_HAND);
		};
		break;
	case "Spine":
		if (!AddScore.listBallToucheJ1.Contains(SPINE))
		{
			AddScore.listBallToucheJ1.Add(SPINE);
		};
		break;
	case "Head":
		if (!AddScore.listBallToucheJ1.Contains(HEAD))
		{
			AddScore.listBallToucheJ1.Add(HEAD);
		};
		break;
}
```

5.1.5 TimerLife

TimerLife est le script faisant le chronomètre avant la détection des collisions pour le commencement du jeu. Il compte aussi le nombre de fois que les portes ont été touché et mets fin au jeu.

1. Dans la fonction Start(), on démarre le timer et on initialise les valeurs :

```
// Commence le timer automatiquement
timerIsRunning = true;
doorSpeed = 0.5f;
nbreVie = 6;
audioSource = GetComponent<AudioSource>();
```

2. Ensuite on fait tourner le timer jusqu'à 0 dans la fonction Update() :

```
if (timerIsRunning)
{
	// Fonction gérant le timer de début
	RunTimer();
}
```

3. Une fois que le timer est terminée, on utilise la fonction ManageSpeed() :

```
else
{
	// Fonction gérant la vitesse des portes
	ManageSpeed();
}
```

4. On regarde la porte en cours pour le décompte de vie :

```
//Regarde si la porte touchée est la même
doorToGetTL = InstantiateDoors.doorToDestroy;
if (doorToGetTL != ancientDoorTL)
{
	isDoorDifferent = true;
	ancientDoorTL = doorToGetTL;
}
```

5. Ensuite on utilise la fonction ManageLife() pour gérer les vies :

```
ManageLife();
```

Fonction RunTimer() :

1. On fait tourner le timer jusqu'à 0 :

```
public void RunTimer()
{
	// Fait un timer avant le début du jeu
	if (timeRemaining > 0)
	{
		timeRemaining -= Time.deltaTime;
		doorCounted = 0;
	}
	else
	{
		//stop le timer
		UnityEngine.Debug.Log("Time has run out!");
		timeRemaining = 0;
		timerIsRunning = false;
	}
}
```

Fonction ManageSpeed() :

1. On regarde la vitesse des portes et le nombre déjà créé et on augmente la vitesse si besoin :

```
public void ManageSpeed()
{
	//Le timer pour le score est en route
	timeScoring += Time.deltaTime;
	//Contrôle le nombre de porte et la vitesse
	if (!(doorSpeed >= VITESSE_MAX) && doorCreated == doorCounted)
	{
		doorSpeed += ACCELERATION;
		doorCreated += 4;
	}
}
```

Fonction ManageLife() :

1. Si le timer n'est plus en fonction et qu'un mur a été touché le décompte de vie se fait et un son est joué sinon, le jeu s'arrête :

```
if (nbreVie == 0)
{
	SceneManager.LoadScene("End_Screen");
}
//Si ce n'est pas le cas, une vie est enlevée et un son est joué
else if (isDoorDifferent)
{
	audioSource.Play();
	nbreVie--;
	isDoorDifferent = false;
}
```

2. Et ensuite on change la couleur de la porte en rouge :

```
// Regarde le tag de l'objet-enfant de la porte pour changer la couleur
foreach (Transform child in doorToGetTL.transform)
{
	if (child.CompareTag("Child"))
	{
		doorMaterial = child.gameObject.GetComponent<Renderer>().material;
		doorMaterial.color = Color.red;
	}
}
doorMaterial = doorToGetTL.GetComponent<Renderer>().material;
doorMaterial.color = Color.red;
```

5.1.6 MenuPrincipale

MenuPrincipale est le script permettant de naviguer entre les scènes lorsqu'on appuie sur un bouton.

Lors de l'appui du bouton, on appelle la fonction qui est liée sur le bouton grâce à l'action onClick().

Fonction btn_change_scene() :

```
public void btn_change_scene (string scene_name)
{
    GameManager.theName = text.text;

    SceneManager.LoadScene("Scene principale");
}
```

Grâce à *GameManager.theName = text.text ;* on peut récupérer le texte de l'inputfield pour l'afficher sur l'écran de fin.

Pour pouvoir quittez l'application grâce à un bouton, nous avons simplement mis le script suivant.

Fonction btn_quitter_scene() :

```
public void btn_quitter_scene()
{
    Application.Quit();
}
```

5.1.7 AfficheScore

Pour afficher le score personnel sur le classement, il faut récupérer la variable publique qui a été déclarée avant.

La gestion du fichier JSON et de ses données viennent d'un code obtenu sur GitHub et en suivant le tutoriel sur YouTube :

- GitHub : https://github.com/DapperDino/Scoreboard-Tutorial
- YouTube : https://www.youtube.com/watch?v=FSEbPxf0kfs

L'affichage du score est adapté selon ce code :

```
void Start()
{
    if(GameManager.Score != null && GameManager.theName != null)
    {
        ScoreField.text = "" + GameManager.Score;
        TeamFied.text = GameManager.theName;
    }
}
```

5.1.8 ScoreboardSaveData

ScoreboardSaveData Contient le script de la création de la liste pour le classement venant du tutoriel :

```
public class ScoreboardSaveData
{
    public List<ScoreboardEntryData> highscores = new List<ScoreboardEntryData>();
}
```

5.1.9 ScoreboardEntryData

ScoreboardEntryData, venant du tutoriel, permet de stocker le score et le nom d'équipe en jeu à travers toutes les scènes :

```
public string entryName;
public int entryScore;
```

5.1.10 Scoreboard

Scoreboard est le script venant aussi du tutoriel qui permet de gérer le fichier JSON en ajoutant, supprimant ou en mettant à jour les données. Il permet aussi de créer le fichier JSON en lui-même s'il est absent et de le mettre suivant ce chemin :

- C:\Users\Admin\AppData\LocalLow\DefaultCompany\Movin'Shapes

Les données dans le fichier JSON sont automatiquement mis dans l'ordre pour les 10 premiers. (Selon le score)

5.1.11 Programmation externe pour le personnage

La gestion des personnages se fait depuis un code obtenu auprès de M. Filkov qui nous a gentiment partagé un package permettant la gestion des personnages avec une Kinect V2.
Site internet : https://rfilkov.com

Nom du package Unity : Kinect-v2-with-MS-SDK-v2.19.2

2 parties du code ont été modifiées, une partie dans User Avatar Matcher et la seconde dans Avatar Controller.

5.1.11.1 User Avatar Matcher

Cette partie concerne l'ajout d'un deuxième avatar, suivant le nombre de personne détecté, l'avatar est différent :

```
//Change character depending on the number of user detected
switch (i)
{
	case 0: avatarModel = avatarModelJ1;break;
	case 1: avatarModel = avatarModelJ2;break;
	default: avatarModel = avatarModelJ1;break;
}
```

5.1.11.2 Avatar Controller

Cette partie concerne l'inversement de la position de l'axe Z du personnage. En la passant à « `true` » on indique à la Kinect que le personnage doit faire l'inverse de la personne. C'est-à-dire lorsque que le joueur s'approche de la caméra Kinect, le personnage s'éloigne de la caméra du jeu et inversement :

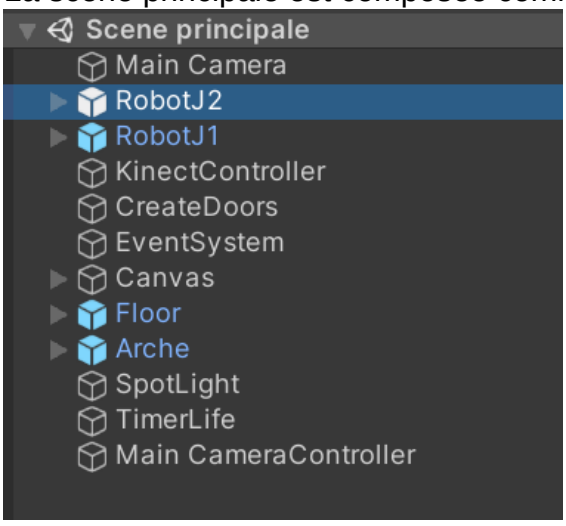
```
[Tooltip("Whether z-axis movement needs to be inverted (Pos-Relative mode only).")]
public bool posRelInvertedZ = true;
```

6. Hiérarchie sur Unity et positionnement des scripts

Les listes suivantes contiennent une description des objets principaux et le positionnement des scripts.

6.1 Hiérarchie de la scène principale

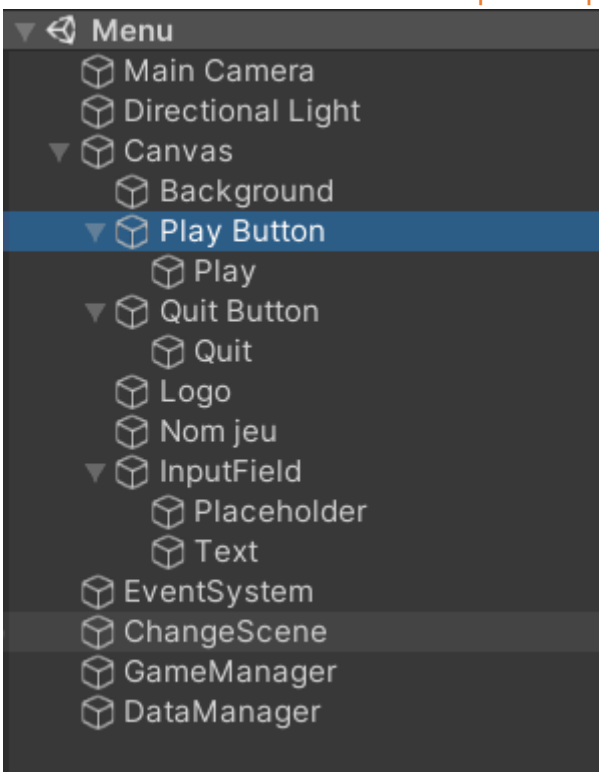
La scène principale est composée comme ci-dessous :



- Main Camera :
  - Cet objet sert de caméra principale pour la scène.
- RobotJ2 et RobotJ1 :
  - Ces objets servent juste de base pour la création des robots.
- KinectController :
  - Cet objet sert de contrôleur principal pour la Kinect.

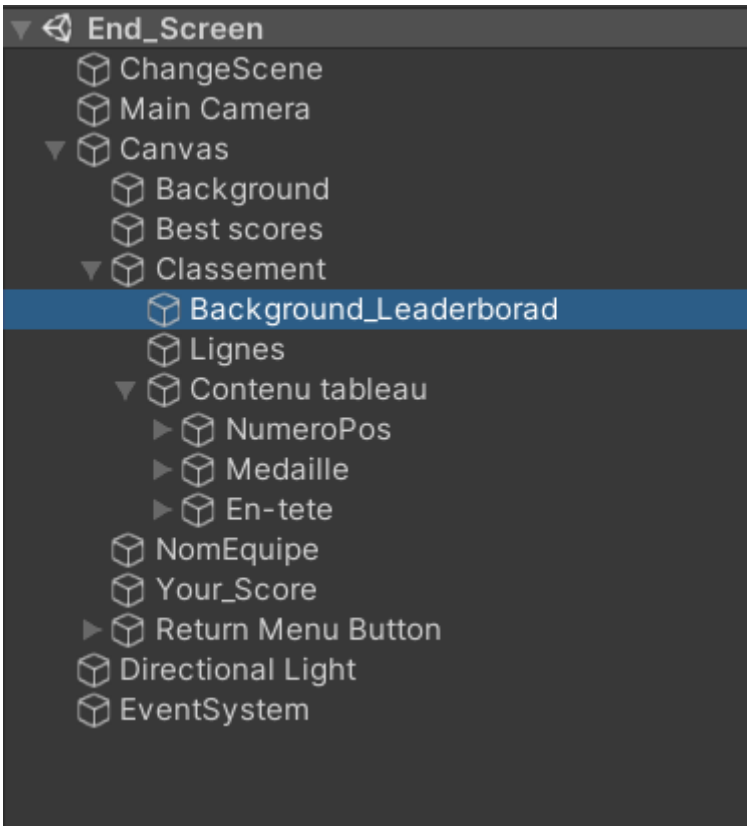
    - Scripts :
        - Kinect Manager (venant du package Unity Kinect-v2-with-MS-SDK-v2.19.2)
        - User Avatar Matcher (venant du package Unity Kinect-v2-with-MS-SDK-v2.19.2)
- CreateDoors :
    - Cet objet sert à séparer le script de création des portes des autres.
    - Script :
        - Instantiate Doors
- Canvas :
    - Cet objet contient l'objet Text et permet de l'afficher sur un plan. L'objet Text est là pour afficher le score et les vies pendant la partie.
    - Script dans l'objet Text :
        - Add Score
- Floor :
    - Cet objet est le sol.
- Arche :
    - Cet objet contient les arches.
- Spotlight :
    - Cet objet est le point de lumière de la scène.
- TimerLife :
    - Cet objet contient juste le script suivant : Timer Life
- Main Camera Controller :
    - Cette caméra est là pour le positionnement des joueurs sur le script Kinect Manager.

6.2 Hiérarchie du Menu principal



- Directional light :
    - Il permet de gérer la lumière qui arrive sur les objets.
- Canvas :
    - Il contient tous les éléments graphiques de la fenêtre.
    - Contient le bouton jouer
        - Il permet de lancer une partie
        - Contient le GameObject ChangeScene
    - Contient le bouton quitter
        - Il permet de quitter le jeu
        - Contient le GameObject ChangeScene
    - Contient un input text
        - Il permet d'entrer le nom de l'équipe
- GameManager :
    - Contient le script où se trouvent les variables qui stockent les données pour le classement
- ChangeScene :
    - Contient le script qui permet de changer de scènes

6.3 Hiérarchie du Menu de fin

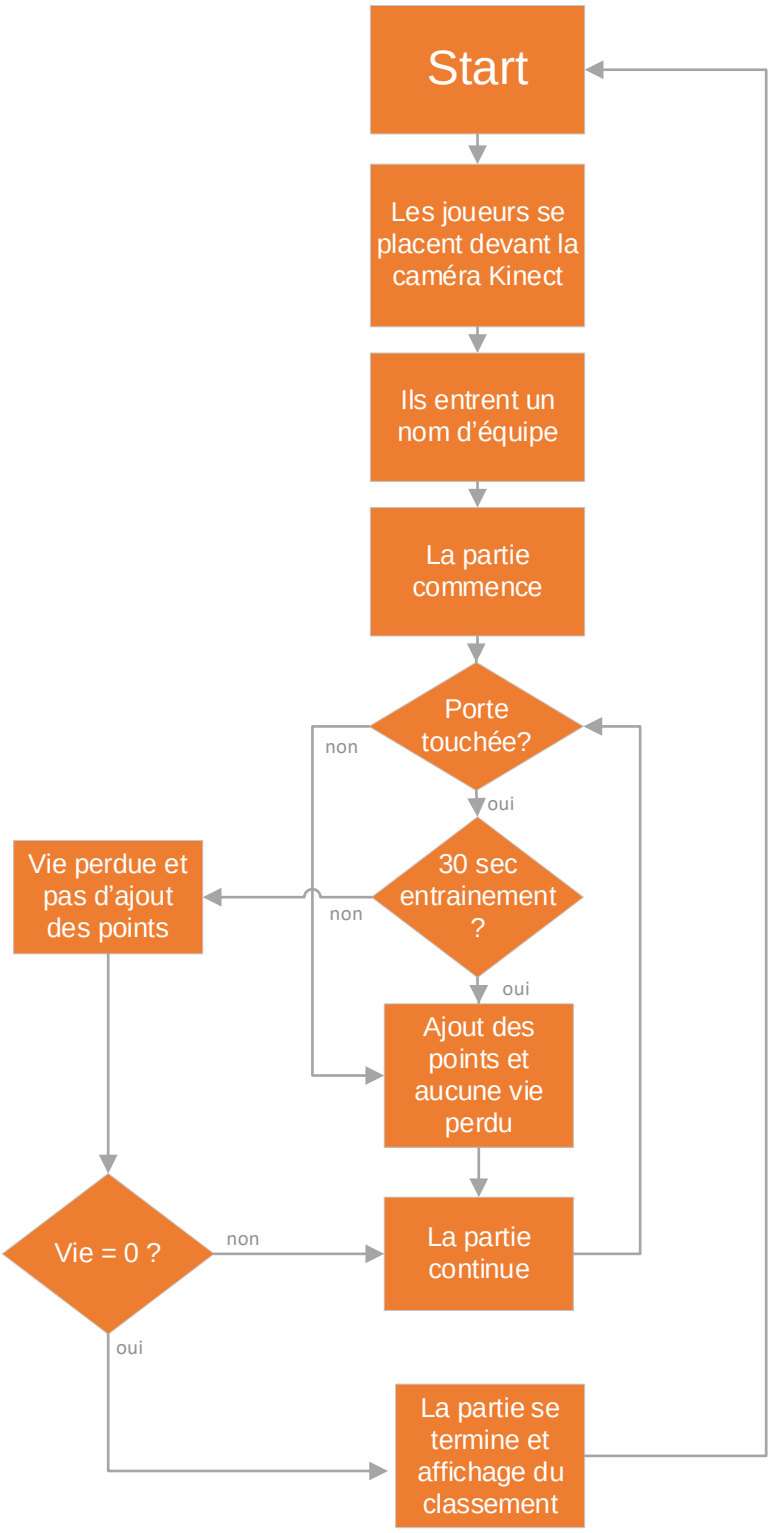


- Canvas :
    - Il contient tous les éléments graphiques de l'écran de fin.
    - Lignes :
        - Il contient les lignes qui seront affichées pour les entrées dans le tableau des scores.
    - Return Menu Button :

- Il contient un bouton pour retourner au menu principal.

7. Déroulement d'une partie

Le déroulement d'une partie se fait comme ci-dessous :



8. Autres

8.1 Bugs mineurs

8.1.1 Positionnement

Lors du jeu, si les personnes ne se mettent pas dans le rectangle prévu ou certaines parties du corps se trouvent cachées par rapport à la caméra (exemple : Bras derrière le dos en faisant face à la caméra) la caméra Kinect essaie de calculer le positionnement, ce qui peut créer des mouvements non-voulus.

Possible solution : Essayer de gérer plusieurs caméra Kinect V2 en même temps

8.1.2 Première porte

Lors du passage de la première porte, il se peut que les points ne soient pas comptés. Le bug est présent mais ce n'est pas toujours le cas, certaines parties, on ne le verra pas et d'autres, aucun point ne sera comptabilisé bien que les 2 joueurs soient passés correctement dans les portes.

Possible solution : Refaire le passage de la première porte et voir où le compte de points fait une erreur.

8.2 Possibles améliorations

8.2.1 En versus

Rajouter un mode versus avec 3 vies par joueurs et le premier à perdre ses 3 vies arrêtent la partie. Ensuite le joueur qui a le plus de points, gagne.

Le choix du mode se ferait dans le menu principal. Le choix se ferait entre le mode versus ou le mode équipe. L'un permettant la compétition entre les deux joueurs et l'autres permettant la coopération. Le classement se ferait en fonction du mode choisi. Un classement pour le mode versus et un autre pour le mode équipe.

8.2.2 Ajout d'une musique

Rajouter une musique de fond sur le jeu afin de donner un peu plus de dynamisme au jeu.

8.2.3 Ajout de portes supplémentaires

Rajouter des portes supplémentaires avec des formes différentes afin d'avoir un réel effet aléatoire dans le choix des portes et ajouter de la diversité dans le jeu.

On pourrait aussi faire que les formes dans les portes soient un peu plus humanoïdes.

9. Conclusion

Le projet a pu respecter les délais et objectifs fixé au début du travail. Il aurait quand même fallu un peu plus de temps pour améliorer la forme des portes et corriger le bug des points en début de partie. Malgré cela, ce projet nous a permis d'apprendre à utiliser Unity et à coder en C#.

L'introduction à l'utilisation de la Kinect a été un peu difficile au début mais avec le package utilisé pour ce projet, ce fut facilité pour toutes les parties du projet, que ce soit la gestion des personnes ou les portes ou encore le décor.

10. Sources

10.1 Unity

- Unity Hub : https://unity3d.com/fr/get-unity/download
- Archives Unity : https://unity3d.com/fr/get-unity/download/archive
- Documentation Unity : https://docs.unity3d.com/Manual/index.html

10.2 Package(s) Unity

- Volumetric Lines 3.0.2 : https://assetstore.unity.com/packages/tools/particles-effects/volumetric-lines-29160
- Probuilder : https://unity3d.com/fr/unity/features/worldbuilding/probuilder
- Free Matcap Shaders 1.4.0.0 : https://assetstore.unity.com/packages/vfx/shaders/free-matcap-shaders-8221
- Mr. Rumen Filkov: https://rfilkov.com/

10.3 Tutoriel suivi

- Menu principale et de fin (Partie graphique) : https://www.youtube.com/watch?v=zc8ac_qUXQY
- Tableau des scores et gestion fichier JSON (Partie programmation) : https://www.youtube.com/watch?v=FSEbPxf0kfs https://github.com/DapperDino/Scoreboard-Tutorial

10.4 Autres

- Fond étoilé : https://unsplash.com/photos/qVotvbsuM_c